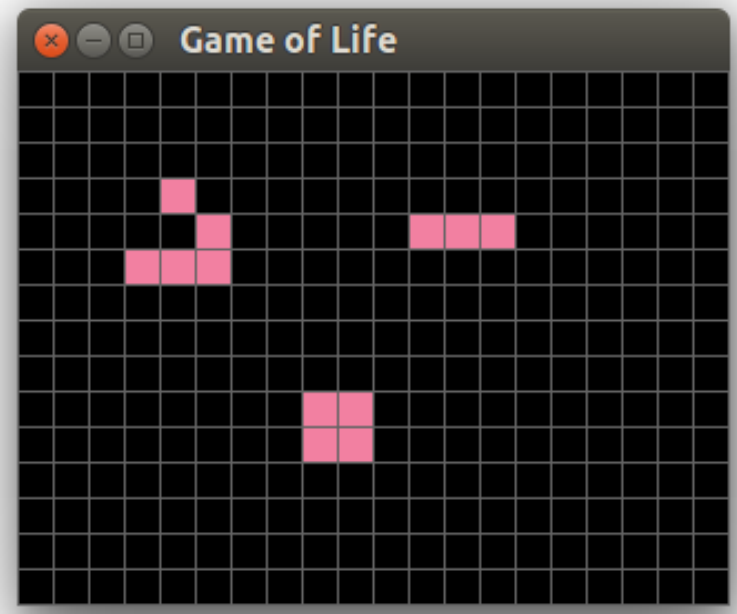
Introduktion till programmering med Scala

Kompendium 2

Andra läsperioden: Modul 8 – 14



Björn Regnell

EDAA45, Lp1-2, HT 2024
Datavetenskap, LTH
Lunds universitet

Kompileringsdatum: 14 mars 2025
http://cs.lth.se/pgk

Editor: Björn Regnell

Contributors in alphabetical order: Anders Buhl, André Philipsson Eriksson, Anna Axelsson, Anna Palmqvist Sjövall, Annie Predel, Anton Andersson, Benjamin Lindberg, Björn Regnell, Casper Schreiter, Cecilia Lindskog, Dag Hemberg, Elias Åradsson, Elliot Bräck, Elsa Cervetti Ogestad, Emelie Engström, Emil Wihlander, Erik Bjäreholt, Erik Grampp, Evelyn Beck, Felix Ohrgren, Fredrik Danebjer, Fritjof Bengtsson, Gustav Cedersjö, Henrik Olsson, Hjalmar Rutberg, Hussein Taher, Jakob Hök, Jakob Sinclair, Johan Ravnborg, Jonas Danebjer, Jos Rosenqvist, Maj Stenmark, Maria Kulesh, Måns Magnusson, Nicholas Boyd Isacsson, Niklas Sandén, Oliver Levay, Oliver Persson, Oscar Sigurdsson, Oskar Berg, Oskar Widmark, Patrik Persson, Per Holm, Philip Sadrian, Sandra Nilsson, Sebastian Hegardt, Simon Persson, Stefan Jonsson, Theodor Lundqvist, Tim Borglund, Tom Postema, Valthor Halldorsson, Viktor Claesson, Wilhelm Wanecek, William Karlsson.

Home: `https://cs.lth.se/pgk`

Repo: `https://github.com/lunduniversity/introprog`

This compendium is on-going work.
Contributions are welcome!
Contact: `bjorn.regnell@cs.lth.se`

Versions:
Scala 3.5.0
JDK 21
introprog-scalalib 1.4.0

Framstegsprotokoll

Genomförda övningar

Till varje laboration hör en övning med uppgifter som utgör förberedelse inför labben. Du behöver minst behärska grunduppgifterna för att klara labben inom rimlig tid. Om du känner att du behöver öva mer på grunderna, gör då även extrauppgifterna. Om du vill fördjupa dig, gör fördjupningsuppgifterna som är på mer avancerad nivå. Kryssa för nedan vilka övningar du har gjort, så blir det lättare för din handledare att anpassa dialogen till de kunskaper du förvärvat hittills.

Övning	Grund	Extra	Fördjupning
`expressions`	☐	☐	☐
`programs`	☐	☐	☐
`functions`	☐	☐	☐
`objects`	☐	☐	☐
`classes`	☐	☐	☐
`patterns`	☐	☐	☐
`sequences`	☐	☐	☐
`matrices`	☐	☐	☐
`lookup`	☐	☐	☐
`inheritance`	☐	☐	☐
`context`	☐	☐	☐
`extra`	☐	☐	☐
`examprep`	☐	☐	☐

Godkända obligatoriska moment

För att bli godkänd på laborationsuppgifterna och projektuppgiften måste du lösa deluppgifterna och diskutera dina lösningar med en handledare. Denna diskussion är din möjlighet att få feedback på dina lösningar. Ta vara på den! Se till att handledaren noterar nedan när du blivit godkänd på respektive obligatorisk moment. Spara detta blad tills du fått slutbetyg i kursen.

Namn: .....

Namnteckning: .....

Lab	kompl+datum,gk+datum	Handl. underskr. + namnförtydl.
`kojo`	.....	.....
`irritext`	.....	.....
`blockmole`	.....	.....
`blockbattle0`	.....	.....
`blockbattle1`	.....	.....
`shuffle`	.....	.....
`life`	.....	.....
`words`	.....	.....
`snake0`	.....	.....
`snake1`	.....	.....
Projektuppgift	.....	.....

☐ `bank` *Om egendef., ge kort beskrivning här:*

☐ `music`

☐ `photo`

☐ *egendefinerad*

Muntligt prov

☐	godkänd	.....	.....

Förord

Detta kompendium innehåller övningar och laborationer och övningslösningar för andra läsperioden i LTH:s grundkurs i programmering för civilingenjörsprogrammet Datateknik.

Vi avslutar första läsperioden med en diagnostisk kontrollskrivning där du får återkoppling på vad du lärt dig hittills. Det är viktigt att du använder dina lärdomar om vad du behöver träna mer på och direkt gör upp en plan för hur du kan befästa din förståelse för begreppen i första läsperioden, så att du hänger med under kommande läsperiod.

Det övergripande målet för den andra läsperioden är att du ska kunna skapa egna program som löser mer omfattande problem än tidigare, genom att kombinera flera abstraktionsmekanismer och begrepp från läsperiod 1. Vi inför även nya abstraktionsmekanismer (t.ex. arv) och nya språkkonstruktioner (t.ex. mönstermatching). Läsperiod 2 avslutas med ett obligatoriskt, individuellt projektarbete. Du ska i slutet av kursen även genomföra ett muntligt prov som kontrollerar att du har de kunskaper som krävs för efterföljande kurs. Du erbjuds även en valfri tentamen som ger möjlighet till ett högre betyg.

Kompendiet distribueras som öppen källkod. Det får användas fritt så länge erkännande ges och eventuella ändringar publiceras under samma licens som ursprungsmaterialet.

I kursens repo github.com/lunduniversity/introprog finns instruktioner om hur du kan bidra till kursmaterialet.

Välkommen till andra halvlek!

Lund, 14 mars 2025, Björn Regnell

Innehåll

Del I

Modulöversikt

W	*Modul*	*Övn*	*Lab*
W01	Introduktion	expressions	kojo
W02	Program och kontrollstrukturer	programs	–
W03	Funktioner och abstraktion	functions	irritext
W04	Objekt och inkapsling	objects	blockmole
W05	Klasser och datamodellering	classes	blockbattle0
W06	Mönster och felhantering	patterns	blockbattle1
W07	Sekvenser och enumerationer	sequences	shuffle
KS	KONTROLLSKRIVN.	–	–
W08	Nästlade och generiska strukturer	matrices	life
W09	Mängder och tabeller	lookup	words
W10	Arv och komposition	inheritance	snake0
W11	Varians och kontextparametrar	context	snake1
W12	Fördjupning, Projekt	extra	Projekt0
W13	Repetition	examprep	Projekt1
W14	MUNTLIGT PROV	Munta	Munta
T	VALFRI TENTAMEN	–	–

W01	**Introduktion**	sekvens, alternativ, repetition, abstraktion, editera, kompilera, exekvera, datorns delar, virtuell maskin, litteral, värde, uttryck, identifierare, variabel, typ, tilldelning, namn, val, var, def, definiera och anropa funktion, funktionshuvud, funktionskropp, procedur, inbyggda grundtyper, println, typen Unit, enhetsvärdet (), stränginterpolatorn s, aritmetik, slumptal, logiska uttryck, de Morgans lagar, if, true, false, while, for
W02	**Program och kontrollstrukturer**	huvudprogram, program-argument, indata, scala.io.StdIn.readLine, kontrollstruktur, iterera över element i samling, for-uttryck, yield, map, foreach, samling, sekvens, indexering, Array, Vector, intervall, Range, algoritm, implementation, pseudokod, algoritmexempel: SWAP, SUM, MIN-MAX, MIN-INDEX
W03	**Funktioner och abstraktion**	abstraktion, funktion, parameter, argument, returtyp, default-argument, namngivna argument, parameterlista, funktionshuvud, funktionskropp, applicera funktion på alla element i en samling, uppdelad parameterlista, skapa egen kontrollstruktur, funktionsvärde, funktionstyp, äkta funktion, stegad funktion, apply, anonyma funktioner, lambda, predikat, aktiveringspost, anropsstacken, objektheapen, stack trace, värdeandrop, namnanrop, klammerparentes och kolon vid ensam parameter, rekursion, scala.util.Random, slumptalsfrö
W04	**Objekt och inkapsling**	modul, singelobjekt, punktnotation, tillstånd, medlem, attribut, metod, paket, filstruktur, jar, classpath, dokumentation, JDK, import, selektiv import, namnbyte vid import, export, tupel, multipla returvärden, block, lokal variabel, skuggning, lokal funktion, funktioner är objekt med apply-metod, namnrymd, synlighet, privat medlem, inkapsling, getter och setter, principen om enhetlig access, överlagring av metoder, introprog.PixelWindow, initialisering, lazy val, typalias
W05	**Klasser och datamodellering**	applikationsdomän, datamodell, objektorientering, klass, instans, Any, isInstanceOf, toString, new, null, this, accessregler, private, private[this], klassparameter, primär konstruktor, fabriksmetod, alternativ konstruktor, förändringsbar, oföränderlig, case-klass, kompanjonsobjekt, referenslikhet, innehållslikhet, eq, ==
W06	**Mönster och felhantering**	mönstermatchning, match, Option, throw, try, catch, Try, unapply, sealed, flatten, flatMap, partiella funktioner, collect, wildcard-mönster, variabelbindning i mönster, sekvens-wildcard, bokstavliga mönster, implementera equals, hashcode

W07	**Sekvenser och enumerationer**	översikt av Scalas samlingsbibliotek och samlingsmetoder, klasshierarkin i scala.collection, Iterable, Seq, List, ListBuffer, ArrayBuffer, WrappedArray, sekvensalgoritm, algoritm: SEQ-COPY, in-place vs copy, algoritm: SEQ-REVERSE, registrering, algoritm: SEQ-REGISTER, linjärsökning, algoritm: LINEAR-SEARCH, tidskomplexitet, minneskomplexitet, översikt strängmetoder, StringBuilder, ordning, inbyggda sökmetoder, find, indexOf, indexWhere, inbyggda sorteringsmetoder, sorted, sortWith, sortBy, repeterade parametrar
KS	KONTROLLSKRIVN.	
W08	**Nästlade och generiska strukturer**	matris, nästlad samling, nästlad for-sats, typparameter, generisk funktion, generisk klass, fri och bunden typparameter, generiska datastrukturer, generiska samlingar i Scala
W09	**Mängder och tabeller**	innehållstest, mängd, Set, mutable.Set, nyckel-värde-tabell, Map, mutable.Map, hash code, java.util.HashMap, java.util.HashSet, persistens, serialisering, textfiler, Source.fromFile, java.nio.file
W10	**Arv och komposition**	arv, komposition, polymorfism, trait, extends, asInstanceOf, with, inmixning supertyp, subtyp, bastyp, override, Scalas typhierarki, Any, AnyRef, Object, AnyVal, Null, Nothing, topptyp, bottentyp, referenstyper, värdetyper, accessregler vid arv, protected, final, trait, abstrakt klass
W11	**Varians och kontextparametrar**	övre- och undre typgräns, varians, kontravarians, kovarians, typjoker, kontextgräns, typkonstruktor, egentyp, typjoker, givet värde (given), kontextparameter (using), ad hoc polymorfism, typklass, api, kodläsbarhet, granskningar
W12	**Fördjupning, Projekt**	välj valfritt fördjupningsområde, påbörja projekt
W13	**Repetition**	träna på extentor, redovisa projekt, träna inför muntligt prov
W14	MUNTLIGT PROV	
T	VALFRI TENTAMEN	

Del II

Moduler

Kapitel 8

Nästlade och generiska strukturer

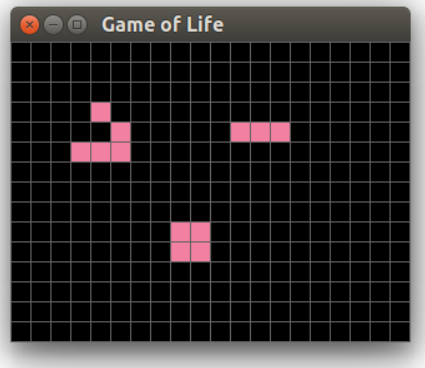
Begrepp som ingår i denna veckas studier:

- ☐ matris
- ☐ nästlad samling
- ☐ nästlad for-sats
- ☐ typparameter
- ☐ generisk funktion
- ☐ generisk klass
- ☐ fri och bunden typparameter
- ☐ generiska datastrukturer
- ☐ generiska samlingar i Scala

8.1 Teori

8.1.1 Veckans labb: `life`

- Universum är en binär matris av **celler** där **levande** celler representeras med `true` och **döda** med `false`.
- Följande regler gäller för **nästa generation** celler i universum:
  - **Fortlevnad**: en levande cell med 2 eller 3 grannar **lever vidare**
  - **Död**: en levande cell med färre än 2 eller fler än 3 grannar **dör**
  - **Födelse**: en död cell med exakt tre grannar föds
- Övning `matrices` uppgift 5: skapa en generisk `case class Matrix[T]`
- På labben: använd `Matrix[Boolean]`



- Du ska simulera *Game of Life* i ett `introprog.PixelWindow`
- Fördjupning: https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

---

8.1.2 Vad är en matris?

- En **matris** inom **matematiken** innehåller **rader** och **kolumner**[1] med tal.
- I en **matematisk** matris har alla rader **lika många** element och
- även alla kolumner har **lika många** element.
- En matris av dimension 2×5 har $2 \cdot 5 = 10$ stycken element.
- Exempel på en matematisk matris av dimension 2×5:

$$M_{2,5} = \begin{pmatrix} 5 & 2 & 42 & 4 & 5 \\ 3 & 4 & 18 & 6 & 7 \end{pmatrix}$$

8.1.3 Indexering i en matris

- En matris av dimension $m \times n$ har $m \cdot n$ stycken element.
- En matris $A_{m,n}$ av dimension $m \times n$ ritas inom matematiken ofta så här:

$$A_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

- Matrisindexering inom matematiken sker ofta från 1, men ofta från 0 i datorprogram.

[1] även kallade *kolonner*

- Vad har talet 42 för index i matrisen $M_{2,5}$ nedan?
  - Inom matematiken?
  - I Scala och Java och många andra språk?

$$M_{2,5} = \begin{pmatrix} 5 & 2 & 42 & 4 & 5 \\ 3 & 4 & 18 & 6 & 7 \end{pmatrix}$$

8.1.4 Hur skapa matriser?

- Inom programmering används ordet **matris** ofta för att beteckna en **nästlad struktur** i två dimensioner. Exempel:
  - **Oföränderliga** sekvenser, t.ex. `Vector[Vector[Int]]`
    `val xss = Vector(Vector(0, 0, 0), Vector(0, 0, 0))` eller enklare:
    `val xss = Vector.fill(2,3)(0)`
  - **Föränderliga** sekvens, t.ex. `Array[Array[Int]]`
    `val yss = Array(Array(0, 0, 0), Array(0, 0, 0))` eller enklare:
    `val yss = Array.fill(2,3)(0)`

8.1.5 Hur indexera i matriser?

En matris med array av arrayer:

```
scala> val xss = Array(Array(5,2,42,4,5),Array(3,4,18,6,7))
xss: Array[Array[Int]] = Array(Array(5, 2, 42, 4, 5), Array(3, 4, 18, 6, 7))
```

Man indexerar i en nästlad sekvens med upprepad `apply`:

```
scala> xss(0)(2)
res0: ???

scala> xss.apply(0).apply(2)
res1: ???

scala> xss(0)
res2: ???
```

Övning: Vad är typ och värde vid `???` ovan?

8.1.6 Hur indexera i matriser?

En matris med array av arrayer:

```
scala> val xss = Array(Array(5,2,42,4,5),Array(3,4,18,6,7))
xss: Array[Array[Int]] = Array(Array(5, 2, 42, 4, 5), Array(3, 4, 18, 6, 7))
```

Man indexerar i en nästlad sekvens med upprepad `apply`:

```
scala> xss(0)(2)
res0: Int = 42

scala> xss.apply(0).apply(2)
res1: Int = 42

scala> xss(0)
res2: Array[Int] = Array(5, 2, 42, 4, 5)
```

Övning: Rita en bild av minnet som referensen `xss` refererar till.

8.1.7 Uppdatering av en förändringsbar nästlad struktur

Man kan förändra en array av arrayer ”på plats” med tilldelning:

```
scala> val xss = Array(Array(5,2,42,4,5),Array(3,4,18,6,7))

scala> xss(0)(0) = 100

scala> xss
res0: ???

scala> xss(0)(2) = xss(0)(2) - 1

scala> xss
res1: ???

scala> xss(1) = Array.fill(5)(-1)

scala> xss
res2: ???
```

8.1.8 Uppdatering av en förändringsbar nästlad struktur

Man kan förändra en array av arrayer ”på plats” med tilldelning:

```
scala> val xss = Array(Array(5,2,42,4,5),Array(3,4,18,6,7))

scala> xss(0)(0) = 100

scala> xss
res0: Array[Array[Int]]=Array(Array(100, 2, 42, 4, 5), Array(3, 4, 18, 6, 7))

scala> xss(0)(2) = xss(0)(2) - 1

scala> xss
res1: Array[Array[Int]]=Array(Array(100, 2, 41, 4, 5), Array(3, 4, 18, 6, 7))

scala> xss(1) = Array.fill(5)(-1)

scala> xss
```

```
16 res2: Array[Array[Int]]=Array(Array(100, 2, 41, 4, 5), Array(-1,-1,-1,-1,-1))
```

8.1.9 Några olika sätt att skapa förändringsbara matriser

Det jobbiga, primitiva sättet:

```
1  scala> val xss = new Array[Array[Int]](2)
2  xss: Array[Array[Int]] = Array(null, null)
3
4  scala> for (i <- xss.indices) {xss(i) = new Array[Int](5)}
5
6  scala> xss
7  res0: Array[Array[Int]] = Array(Array(0, 0, 0, 0, 0), Array(0, 0, 0, 0, 0))
8
9  scala> println(xss)
10 [[I@196a99d0
```

Enklare sätt:

```
1  scala> val xss = Array.ofDim[Int](2,5)
2  xss: Array[Array[Int]] = Array(Array(0, 0, 0, 0, 0), Array(0, 0, 0, 0, 0))
```

Enklare och tydligare sätt, där initialvärdet anges explicit:

```
1  scala> val xss = Array.fill(2,5)(0)
2  xss: Array[Array[Int]] = Array(Array(0, 0, 0, 0, 0), Array(0, 0, 0, 0, 0))
```

8.1.10 Exempel på skapande av oföränderlig nästlad struktur

Om du kan beräkna initialvärde direkt, använd `Vector.fill`:
`def fill[A](n1: Int, n2: Int)(elem: => A): Vector[Vector[A]]`

```
1  scala> Vector.fill(2,5)(scala.util.Random.nextInt(6) + 1)
2  res0:
3    typ???
4    värde???
```

Om du kan beräkna initialvärde ur index, använd `Vector.tabulate`:
`def tabulate[A](n1: Int, n2: Int)(f: (Int, Int) => A): Vector[Vector[A]]`

```
1  scala> Vector.tabulate(5,2)((x,y) => x + y + 1)
2  res1:
3    typ???
4    värde???
```

8.1.11 Exempel på skapande av oföränderlig nästlad struktur

Om du kan beräkna initialvärde direkt, använd `Vector.fill`:

```
def fill[A](n1: Int, n2: Int)(elem: => A): Vector[Vector[A]]
```

```
scala> Vector.fill(2,5)(scala.util.Random.nextInt(6) + 1)
res0: Vector[Vector[Int]] =
  Vector(Vector(1, 2, 6, 2, 1), Vector(1, 4, 3, 3, 2))
```

Om du kan beräkna initialvärde ur index, använd `Vector.tabulate`:

```
def tabulate[A](n1: Int, n2: Int)(f: (Int, Int) => A): Vector[Vector[A]]
```

```
scala> Vector.tabulate(5,2)((x,y) => x + y + 1)
res1: Vector[Vector[Int]] =
  Vector(Vector(1,2), Vector(2,3), Vector(3,4), Vector(4,5), Vector(5,  6))
```

8.1.12 Uppdatering av en oföränderlig nästlad struktur

Uppdatering av endimensionell struktur med `xs.updated`:

```
def updated[A](index: Int, elem: A): Vector[A]
```

```
scala> var xs = Vector.tabulate(5)(x => x + 1)
xs: typ??? = värde???

scala> xs = xs.updated(1, 42)
xs: typ??? = värde???
```

Uppdatering av nästlad struktur i två dimensioner:

```
scala> var xss = Vector.tabulate(2, 5)((x,y) => x + y + 1)
xss:
  typ??? =
  värde???

scala> xss = xss.updated(0, xss(0).updated(1, 42))
xss:
  typ??? =
  värde???
```

8.1.13 Uppdatering av en oföränderlig nästlad struktur

Uppdatering av endimensionell struktur med `xs.updated`:

```
def updated[A](index: Int, elem: A): Vector[A]
```

```
scala> var xs = Vector.tabulate(5)(x => x + 1)
xs: Vector[Int] = Vector(1, 2, 3, 4, 5)

scala> xs = xs.updated(1, 42)
xs: Vector[Int] = Vector(1, 42, 3, 4, 5)
```

Uppdatering av nästlad struktur i två dimensioner:

```
scala> var xss = Vector.tabulate(2, 5)((x,y) => x + y + 1)
xss: Vector[Vector[Int]] =
  Vector(Vector(1, 2, 3, 4, 5), Vector(2, 3, 4, 5, 6))

scala> xss = xss.updated(0, xss(0).updated(1, 42))
xss:
  Vector[Vector[Int]] =
  Vector(Vector(1, 42, 3, 4, 5), Vector(2, 3, 4, 5, 6))
```

8.1.14 Iterera över nästlad struktur

Behandling av nästlade strukturer kräver ofta algoritmer med nästlad iterering.
Exempel: iterera med nästlad **for**-sats för utskrift av denna matris
`val xss = Vector.tabulate(2,5)((x,y) => x + y + 1)`

```
scala> for ??? do
         for ??? do
           print(xss(i)(j))
           print(" ")
         println

1 2 3 4 5
2 3 4 5 6
```

Övning:
Vad ska det stå vid ??? för att alla element ska skrivas ut?

8.1.15 Iterera över nästlad struktur

Behandling av nästlade strukturer kräver ofta algoritmer med nästlad iterering.
Exempel: iterera med nästlad **for**-sats för utskrift av denna matris
`val xss = Vector.tabulate(2,5)((x,y) => x + y + 1)`

```
scala> for xs <- xss do
         for x <- xs do
           print(x)
           print(" ")
         end for
         println()
       end for

1 2 3 4 5
2 3 4 5 6
```

Övning: skriv ut matrisen med nästlad `foreach`

```
xss.foreach { xs =>
  xs.foreach { x => print(x); print(" ") }
  println()
```

```
}
```

8.1.16 Övningsexempel: Yatzy

Skapa en funktion `roll` som ger utfallet av n st tärningskast:

```
scala> import scala.util.Random

scala> def roll(n: Int): Vector[Int] = ???
```

Skapa en funktion `isYatzy` som ger **`true`** om alla utfall är lika:

```
scala> def isYatzy(xs: Vector[Int]): Boolean = ???
```

Du kan anta att xs.length > 0
Tips: använd metoden xs.forall:

```
def forall[A](p: A => Boolean): Boolean
```

8.1.17 Övningsexempel: Yatzy

Skapa en funktion `roll` som ger utfallet av n st tärningskast:

```
scala> import scala.util.Random

scala> def roll(n: Int): Vector[Int] = Vector.fill(n)(Random.nextInt(6) + 1)
```

Skapa en funktion `isYatzy` som ger **`true`** om alla utfall är lika:

```
scala> def isYatzy(xs: Vector[Int]): Boolean = xs.forall(x => x == xs(0))
```

Du kan anta att xs.length > 0
Tips: använd metoden xs.forall:

```
def forall[A](p: A => Boolean): Boolean
```

8.1.18 Iterera över nästlad struktur: for-sats

Iterera med nästlad for-sats: (vad har xss för typ?)

```
scala> val xss = Vector.fill(100)(roll(5))

scala> for i <- ??? do
         for j <- ??? do
           print(s"($i)($j): ${xss(i)(j)} ")
         println(s" YATZY: ${isYatzy(xss(i))}")

(0)(0): 3 (0)(1): 6 (0)(2): 4 (0)(3): 4 (0)(4): 6  YATZY: false
(1)(0): 4 (1)(1): 1 (1)(2): 5 (1)(3): 2 (1)(4): 6  YATZY: false
(2)(0): 1 (2)(1): 3 (2)(2): 5 (2)(3): 6 (2)(4): 2  YATZY: false
(3)(0): 2 (3)(1): 1 (3)(2): 1 (3)(3): 5 (3)(4): 4  YATZY: false
(4)(0): 4 (4)(1): 4 (4)(2): 1 (4)(3): 6 (4)(4): 5  YATZY: false
(5)(0): 3 (5)(1): 3 (5)(2): 2 (5)(3): 3 (5)(4): 6  YATZY: false
```

```
(6)(0): 3 (6)(1): 6 (6)(2): 1 (6)(3): 1 (6)(4): 4  YATZY: false
(7)(0): 6 (7)(1): 2 (7)(2): 4 (7)(3): 4 (7)(4): 3  YATZY: false
(8)(0): 1 (8)(1): 5 (8)(2): 4 (8)(3): 2 (8)(4): 4  YATZY: false
(9)(0): 1 (9)(1): 1 (9)(2): 3 (9)(3): 6 (9)(4): 6  YATZY: false
(10)(0): 2 (10)(1): 5 (10)(2): 2 (10)(3): 4 (10)(4): 5  YATZY: false
(11)(0): 3 (11)(1): 4 (11)(2): 2 (11)(3): 5 (11)(4): 6  YATZY: false
...
```

8.1.19 Iterera över nästlad struktur: for-sats

Iterera med nästlad for-sats: (xss är en `Vector[Vector[Int]]`)

```
scala> val xss = Vector.fill(100)(roll(5))

scala> for i <- xss.indices do
         for j <- xss(i).indices do
           print(s"($i)($j): ${xss(i)(j)} ")
         println(s" YATZY: ${isYatzy(xss(i))}")

(0)(0): 3 (0)(1): 6 (0)(2): 4 (0)(3): 4 (0)(4): 6  YATZY: false
(1)(0): 4 (1)(1): 1 (1)(2): 5 (1)(3): 2 (1)(4): 6  YATZY: false
(2)(0): 1 (2)(1): 3 (2)(2): 5 (2)(3): 6 (2)(4): 2  YATZY: false
(3)(0): 2 (3)(1): 1 (3)(2): 1 (3)(3): 5 (3)(4): 4  YATZY: false
(4)(0): 4 (4)(1): 4 (4)(2): 1 (4)(3): 6 (4)(4): 5  YATZY: false
(5)(0): 3 (5)(1): 3 (5)(2): 2 (5)(3): 3 (5)(4): 6  YATZY: false
(6)(0): 3 (6)(1): 6 (6)(2): 1 (6)(3): 1 (6)(4): 4  YATZY: false
(7)(0): 6 (7)(1): 2 (7)(2): 4 (7)(3): 4 (7)(4): 3  YATZY: false
(8)(0): 1 (8)(1): 5 (8)(2): 4 (8)(3): 2 (8)(4): 4  YATZY: false
(9)(0): 1 (9)(1): 1 (9)(2): 3 (9)(3): 6 (9)(4): 6  YATZY: false
(10)(0): 2 (10)(1): 5 (10)(2): 2 (10)(3): 4 (10)(4): 5  YATZY: false
(11)(0): 3 (11)(1): 4 (11)(2): 2 (11)(3): 5 (11)(4): 6  YATZY: false
...
```

8.1.20 Nästlade for-uttryck

Iterera med **nästlad for-yield**:

```
scala> val xss = for i <- 1 to 2 yield
                   for j <- 1 to 5 yield i + j + 1

val xss: IndexedSeq[IndexedSeq[Int]] =
      ???
```

Om man skriver så här får man en endimensionell struktur:

```
scala> val xs = for { i <- 1 to 2; j <- 1 to 5 } yield i + j + 1
val xs: IndexedSeq[Int] =
    ???
```

8.1.21 Nästlade for-uttryck

Iterera med **nästlad for-yield**:

```
scala> val xss = for i <- 1 to 2 yield
                   for j <- 1 to 5 yield i + j + 1

val xss: IndexedSeq[IndexedSeq[Int]] =
    Vector(Vector(3, 4, 5, 6, 7), Vector(4, 5, 6, 7, 8))
```

Om man skriver så här får man en endimensionell struktur:

```
scala> val xs = for { i <- 1 to 2; j <- 1 to 5 } yield i + j + 1
val xs: IndexedSeq[Int] =
    Vector(3, 4, 5, 6, 7, 4, 5, 6, 7, 8)
```

8.1.22 Nästlade map-uttryck

Iterera med **nästlade map-uttryck**:

```
scala> val xss = (1 to 2).map(i => (1 to 5).map(j => i + j + 1))
xss: IndexedSeq[IndexedSeq[Int]] =
      ???
```

8.1.23 Nästlade map-uttryck

Iterera med **nästlade map-uttryck**:

```
scala> val xss = (1 to 2).map(i => (1 to 5).map(j => i + j + 1))
xss: IndexedSeq[IndexedSeq[Int]] =
      Vector(Vector(3, 4, 5, 6, 7), Vector(4, 5, 6, 7, 8))
```

8.1.24 Fallgrop: likhet av array

```
scala> Vector.fill(5, 2)(42) == Vector.fill(5, 2)(42)
val res0: Boolean = true

scala> Array.fill(5, 2)(42) == Array.fill(5, 2)(42)
val res1: Boolean = false  // AAAARRGH!!! :(
```

Primitiva arrayer har en equals-metod som ger referenslikhet, **inte** innehållslikhet. Och det fungerar följaktligen ej heller på nästlade strukturer.

8.1.25 Kolla likhet mellan två heltalsmatriser (uppfinner hjulet)

```
def isEqual(xss: Array[Array[Int]], yss: Array[Array[Int]]) =
  if xss.length != yss.length then false else
    var i = 0
    var foundUnequal = false
    while i < xss.length && !foundUnequal do
      if xss(i).length != yss(i).length then
        foundUnequal = true
      else
        var j = 0
        while j < xss(i).length && !foundUnequal do
          if xss(i)(j) != yss(i)(j) then foundUnequal = true
          j += 1
        end while
      end if
      i += 1
    end while
    !foundUnequal
  end if
end isEqual
```

8.1.26 Använd INTE `sameElements` på nästlade arrayer

I Scala kan du använda metoden `sameElements` för att kolla innehållslikhet mellan två arrayer, men det funkar **INTE** på djupet i nästlade strukturer.

```
scala> val xs = Array(1,2,3)
xs: Array[Int] = Array(1, 2, 3)

scala> val ys = Array(1,2,3)
ys: Array[Int] = Array(1, 2, 3)

scala> xs.sameElements(ys)                          // xs, ys ej nästlade
res0: Boolean = true                                // innehåll lika!

scala> Array(Array(1)) sameElements Array(Array(1))
res1: Boolean = false                               //AAAARGH!
```

Använd i stället:
`java.util.Objects.deepEquals` eller `java.util.Arrays.deepEquals`
Den senare kräver typkonvertering av argumenten med: `asInstanceOf[Array[Object]]`

8.1.27 Kontroll av innehållslikhet mellan nästlade arrayer

`java.util.Objects.deepEquals` fungerar **på djupet** för godtyckliga referenstyper:

```
scala> java.util.Objects.deepEquals(
         Array(Array("a", Array("b"), 42)),
         Array(Array("a", Array("b"), 42)))
val res0: Boolean = true
```

```
scala> java.util.Objects.deepEquals(
         Array(Array("a", Array("b"), 42)),
         Array(Array("a", Array("b"), 43)))
val res1: Boolean = false
```

`java.util.Objects.deepEquals` kontrollerar om argumenten är arrayer och anropar då i sin tur `java.util.Arrays.deepEquals` efter typkonvertering:

```
scala> java.util.Arrays.deepEquals(
         Array(Array("a", Array("b"), 42)).asInstanceOf[Array[Object]],
         Array(Array("a", Array("b"), 42)).asInstanceOf[Array[Object]])
val res3: Boolean = true
```

https://stackoverflow.com/questions/63686721/best-replacement-of-deep-method-in-scala-2-13

8.1.28 Om veckans övningar

- Träna på att iterera över nästlade strukurer
- Fortsätt jobba med Yatzy-exemplet
- träna på att skapa **imperativa** algoritmer:
  lös `isYatzy` med **`while`**-sats
- Extrauppgift där du ska bygga ett enkelt yatzy-spel i terminalen (kunde varit en tentauppgift...)

8.1.29 Exempel: Icke-generisk case-klass med heltalsmatris

En *icke-generisk* datastruktur har inga obundna typparametrar; alla typer är **konkreta** (alltså specifika).

En icke-generisk case-class med en `Vector[Vector[Int]]`:

```
case class Matrix(data: Vector[Vector[Int]]):
  def apply(x: Int, y: Int): Int = data(x)(y)
```

```
scala> Matrix(Vector(Vector(5, 2, 42, 4, 5),Vector(3, 4, 18, 6, 7)))
res0: Matrix =
  Matrix(Vector(Vector(5, 2, 42, 4, 5), Vector(3, 4, 18, 6, 7)))

```

8.1.30 Exempel: Generisk case-klass med generell matris

En *generisk* datastruktur har minst en **obunden typparameter** som vid användning ska bindas till ett **konkret typargument**.

```
case class Matrix[T](data: Vector[Vector[T]]):
  def apply(x: Int, y: Int): T = data(x)(y)
```

`Matrix` i exemplet ovan är en **generisk** case-class där `T` är obunden, eftersom `T` är en typparameter deklarerad inom `[]` **efter** klassens namn men **före** klassparameterlistan.

Användning där `T` binds till `Int` via kompilatorns typhärledning:

```
scala> Matrix(Vector(Vector(5, 2, 42, 4, 5),Vector(3, 4, 18, 6, 7)))
res1: Matrix[Int] =
  Matrix(Vector(Vector(5, 2, 42, 4, 5), Vector(3, 4, 18, 6, 7)))

```

8.1.31 Vad är en typparameter?

- En **typparameter** gör det möjligt att ge ett **typargument**.
- Detta kallas **parametrisk polymorfism** (eng. *paramteric polymorphism*).
- Exempel: **generisk funktion**:

```
def tnirp[A](x: A):Unit = println(x.toString.reverse)
```

- En **fri** typparameter kan bindas till vilken typ som helst.
- Bindingen av typargument till typparametrar sker vid **kompileringstid**.
- En typparameter är **fri** om den **inte** fått något värde, annars **bunden**.
- Exempel: **generisk klass** med **generiska metoder**:

```
class Cell[A](   // A är här fri men måste bindas vid användning
    var value: A):              // A är bunden vid användning av Cell
  def update(a: A): Unit = value = a    // A är även här bunden vid anv.
  def replaced[B](b: B): Cell[B] = new Cell(b)  // första [B] är fri
```

- **Skuggning kan förekomma**: Om `replaced` i `Cell` hade använt namnet A på sin typparameter hade den **skuggat** klassens typparameter och tolkats som en fri typparameter, alltså en godtycklig typ och **inte** klassens typparameter. (jämför namnöverskuggning vid **lokala** namn i nästlade block)

8.1.32 Exempel: Generisk funktion

Vad händer här?

```

scala> def skrikBaklänges(x: T): String = x.toString.toUpperCase.reverse
1 |def skrikBaklänges(x: T): String = x.toString.toUpperCase.reverse
  |                      ^
  |                      Not found: type T
```

```
                              ^

scala> def skrikBaklänges[T](x: T): String = x.toString.toUpperCase.reverse

scala> skrikBaklänges("gurka är gott")
val res0: String = TTOG RÄ AKRUG
```

Om ingen typparameter deklareras inom hakparenteser efter funktionens namns så vet inte kompilatorn vad `T` är för en typ. Men med en typparameter `[T]` efter funktionsnamnet tolkar kompilatorn funktionen som **generisk** och typen `T` bestäms av argumentets typ **vid anrop** och `T` kan bindas till godtycklig typ.

8.1.33 Exempel: Generisk case-klass

- En generisk klass har en eller flera typparametrar efter klassnamnet:

```
case class Box[A](value: A)
```

- Kompilatorn härleder typparameterarnas typ utifrån givna värden.

```
scala> Box("gurka")
val res1: Box[String] = Box(gurka)
```

- Du kan också ge typpparametern en typ explicit:

```
scala> Box[Int](42)
val res2: Box[Int] = Box(42)
```

- Om typen inte stämmer får du hjälp av kompilatorn att hitta felet:

```
scala> Box[String](42)
-- Error:
1 |Box[String](42)
  |            ^^ Found:    (42 : Int)  Required: String
```

- En generisk klass, här `Box`, kallas också **typkonstruktor** (eng. *type constructor*) då den ”färdiga” typen `Box[Int]` ”konstrueras” på platsen där den används.

8.1.34 Fallgrop: Typradering (eng. *type erasure*)

Informationen om typerna i typparametrar raderas innan kodgenerering för JVM av prestandaskäl och **typparametrar saknas vid runtime** i bytekoden.

```
scala> def isIntVector[T](xs: Vector[T]) = xs.isInstanceOf[Vector[Int]]
-- Warning:
1 |def isIntVector[T](xs: Vector[T]) = xs.isInstanceOf[Vector[Int]]
  |                                     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  |                the type test for Vector[Int] cannot be checked at runtime
def isIntVector[T](xs: Vector[T]): Boolean

scala> isIntVector(Vector("hej"))
res42: Boolean = true  // AAAARGHH!! :(
```

Måste ”packa upp” samlingen och typtesta alla element:

```
scala> def isIntVector[T](xs: Vector[T]) = xs.forall(_.isInstanceOf[Int])

scala> isIntVector(Vector("hej"))
res43: Boolean = false  // FUNKAR :)
```

Typkontroll vid körtid görs oftast hellre med **match**.

8.1.35 Testning och avlusning

- Läs om testning och avlusning (eng. *debugging*) i Appendix D: ”Fixa buggar”
- Träna på println-debugging
- Prova debuggern i VS code

8.2 Övning `matrices`

Mål

- ☐ Kunna skapa och använda matriser med nästlade strukturer av `Vector`.
- ☐ Kunna iterera över elementen i en matris med nästlade **`for`**-satser och **`for`**-**`yield`**-uttryck, samt nästlad applicering av `map` respektive `foreach`.
- ☐ Kunna skapa och använda funktioner som tar matriser som parametrar.
- ☐ Kunna skapa en enkel generisk klass och enkla generiska funktioner med hjälp av en typparameter.
- ☐ Kunna beskriva skillnader och likheter mellan Scala och Java vad gäller indexering och iterering i matriser implementerade med nästlade arrayer.

Förberedelser

- ☐ Studera begreppen i kapitel 8

8.2.1 Grunduppgifter; förberedelse inför laboration

Uppgift 1. *Para ihop begrepp med beskrivning.*

Koppla varje begrepp med den (förenklade) beskrivning som passar bäst:

matris	1	A	indexerbar datastruktur i två dimensioner
radvektor	2	B	har abstrakt typparameter, typen är generell
kolumnvektor	3	C	annat ord för kolumn
kolonn	4	D	konkret typ, binds till typparameter vid kompilering
generisk	5	E	kompilatorn beräknar typ ur sammanhanget
typargument	6	F	matris av dimension $1 \times m$ med m horisontella värden
typhärledning	7	G	matris av dimension $m \times 1$ med m vertikala värden

Uppgift 2. *Skapa matriser med hjälp av nästlade samlingar.* Man kan i ett datorprogram, med hjälp av samlingar som innehåller samlingar, skapa nästlade strukturer som kan indexeras i två dimensioner och på så sätt representera en **matris**.[2]

a) Rita minnessituationen efter tilldelningen på rad 1 nedan. Vad har `m` för typ och värde? Vad har `m` för dimensioner? Hur sker indexeringen i ett datorprogram jämfört med i matematiken?

```
1 scala> val m = Vector((1 to 5).toVector, (3 to 7).toVector)
2 scala> m.apply(0).apply(1)
3 scala> m(1)
4 scala> m(1)(4)
```

b) Vad ger uttrycken på raderna 2, 3 och 4 ovan för värden och typ?

c) Man kan i ett datorprogram mycket väl skapa tvådimensionella, nästlade strukturer där raderna *inte* innehåller samma antal element. Det blir då ingen äkta matris i strikt matematisk mening, men man kallar ofta ändå en sådan struktur för en ”matris”. Vilken typ har variablerna `m2`, `m3`, `m4` och `m5` nedan?

[2] sv.wikipedia.org/wiki/Matris

```
scala> val m2 = Vector(Vector(1,2,3),Vector(4,5),Vector(42))
scala> val m3 = Vector(Vector(1,2), Vector(1.0, 2.0, 3.0))
scala> val m4 = m3(1) +: Vector("a") +: m3
scala> val m5 = Vector.fill(42){ m2(1).map(e => (e * math.random()).toInt) }
```

d) Vilken av variablerna `m2`, `m3`, `m4` och `m5` ovan representerar en äkta matris i matematisk mening? Vilken är dess dimensioner?

Uppgift 3. *Skapa och iterera över matriser.* Du ska skapa matriser där varje rad representerar 5 kast med en tärning i spelet Yatzy.[3]

a) Definiera i REPL en funktion `def throwDie: Int = ???` som returnerar ett slumptal mellan 1 och 6.

b) Skapa nedan heltalsmatris i REPL. Vilken dimension får matrisen?

```
scala> val ds1 = for (i <- 1 to 1000) yield
            for (j <- 1 to 5) yield throwDie
```

c) Man kan också använda nedan varianter för att skapa en heltalsmatris. Vilken av varianterna `ds1` ... `ds6` tycker du är lättast att läsa och förstå? Prova respektive variant i REPL och ange vilken typ på `ds1` ... `ds6` som härleds av kompilatorn.

```
val ds2 = (1 to 1000).map(i => (1 to 5).map(j => throwDie))
val ds3 = (1 to 1000).map(i => Vector.fill(5)(throwDie))
val ds4 = for (i <- 1 to 1000) yield Vector.fill(5)(throwDie)
val ds5 = Vector.fill(1000)(Vector.fill(5)(throwDie))
val ds6 = Vector.fill(1000, 5)(throwDie)
```

d) Definiera en funktion
`def roll(n: Int): Vector[Int] = ???`
som ger en heltalsvektor med n stycken slumpvisa tärningskast. Kasten ska vara sorterade i växande ordning; använd för detta ändamål samlingsmetoden `sorted`.

e) Definera i REPL en funktion `isYatzy(xs: Vector[Int]): Boolean = ???` som testar om alla elementen i en heltalsvektor är samma. Använd samlingsmetoden `forall`.

f) Skapa en funktion
`def diceMatrix(m: Int, n: Int): Vector[Vector[Int]] = ???`
som med hjälp av funktionen `roll` skapar en matris med `m` st vektorer med vardera `n` slumpvisa tärningskast.

g) Skapa en funktion som returnerar en utskriftsvänlig sträng
`def diceMatrixToString(xss: Vector[Vector[Int]]): String = ???`
med hjälp av `map` och `mkString`, som fungerar enligt nedan.

```
scala> val dm2s = diceMatrixToString(diceMatrix(4, 5))
val dm2s: String = 1 4 4 6 6
1 1 2 6 6
2 4 4 5 6
1 1 5 6 6

scala> println(dm2s)
1 4 4 6 6
```

[3] sv.wikipedia.org/wiki/Yatzy

```
1 1 2 6 6
2 4 4 5 6
1 1 5 6 6
```

h) Implementera funktionen
`def filterYatzy(xss: Vector[Vector[Int]]): Vector[Vector[Int]]`
som filtrerar fram alla yatzy-rader i matrisen `xss` enligt nedan. Använd din funktion `isYatzy` och samlingsmetoden `filter`.

```
scala> println(diceMatrixToString(filterYatzy(diceMatrix(10000, 5))))
4 4 4 4 4
6 6 6 6 6
4 4 4 4 4
6 6 6 6 6
4 4 4 4 4
4 4 4 4 4
2 2 2 2 2
```

i) Implementera funktionen
`def yatzyPips(xss: Vector[Vector[Int]]): Vector[Int] = ???`
som ska ge en vektor med de tärningsvärden som gav yatzy, för kasten i matrisen `xss` enligt nedan. Använd din funktion `filterYatzy`.

```
scala> val dm = Vector(Vector(1,2,3,4,5),Vector(4,4,4,4,4),Vector(3,3,3,3,3))
scala> yatzyPips(dm)
val res42: Vector[Int] = Vector(4, 3)
```

Uppgift 4. *En oföränderlig, generisk matris-klass till veckans laboration `life`.* Under veckans laboration ska du simulera en enkel form av ”liv” som består av celler i ett rutnät. För detta ändamål har vi nytta av en matris-klass som du ska implementera steg för steg i denna övning. Skapa case-klassen nedan med en editor i filen `Matrix.scala`. Testa din lösning med hjälp av valfri IDE, t.ex. `scalaide` eller `idea`.

```
case class Matrix(data: Vector[Vector[String]]){
  def apply(row: Int, col: Int): String = data(row)(col)
}
object Matrix {
  def fill(dim: (Int, Int))(value: String): Matrix =
    Matrix(Vector.fill(dim._1, dim._2)(value))
}
```

```
scala> val m = Matrix.fill(3,4)("hej")
scala> val e = m(2, 2)
```

a) Vad får `m` ovan för typ?

b) Vad får `e` ovan för typ?

c) På hur många ställen måste du ändra i `Matrix` ovan för att den i stället ska representera en matris av heltal?

d) Du ska nu med hjälp av en **typparameter** göra `Matrix` **generisk** (eng. *generic*), så att den blir en mer användbar matrisklass som kan innehålla element av vilken typ som helst. Genomför följande ändringar i `Matrix.scala`:

- Lägg till en typparameter `T` inom klammerparenteser efter namnet `Matrix` på alla ställen där det förekommer *utom* efter namnet på kompanjonsobjektet[4].
- Byt ut `String` mot `T` på alla ställen där `String` förekommer.
- Lägg till en typparameter `T` inom klammerparenteser efter `def fill`.

Testa din generiska klass i REPL genom att skapa en boolesk matris:

```
scala> val bm = Matrix.fill(3,4)(false)
scala> val be = bm(0, 0)
```

e) Vad får `bm` ovan för typ?

f) Vad får `be` ovan för typ?

g) Lägg en kodrad i början av klasskroppen som med hjälp av `require` garanterar att alla rader i matrisen är lika långa.

h) Lägg till en medlem `val dim: (Int, Int)` i klasskroppen efter `require`-satsen som ger ett par (alltså en 2-tupel) med antalet rader resp. kolumner i matrisen.

i) Lägg till en metod `def updated(row: Int, col: Int)(value: T): Matrix[T]` som ger en ny matris där element på platsen `(row, col)` har uppdaterats till `value`.

j) Lägg till en metod `def foreachIndex(f: (Int, Int) => Unit): Unit` som för varje index i `data` applicerar funktionen `f`.

k) Lägg till en metod `override def toString` som så att en instans av `Matrix` visas enligt följande:

```
scala> val dm = Matrix.fill(3,4)(42.0)
val dm: Matrix[Double] =
Matrix of dim (3,4):
42.0 42.0 42.0 42.0
42.0 42.0 42.0 42.0
42.0 42.0 42.0 42.0
```

[4] Singelobjekt kan inte ha typparametrar, men deras medlemmar kan.

8.2.2 Extrauppgifter; träna mer

Uppgift 5. *Imperativa matrisalgoritmer.* Imperativa angreppssätt är nödvändiga att kunna när du stöter på samlingar och/eller språk som saknar funktionella metoder och/eller funktionsprogrammeringsmöjligheter. Genom att studera imperativa lösningar till de ofta mer koncisa funktionella lösningarna, får du träning i att skapa algoritmer som använder förändring genom tilldelning vid iterering.

a) Implementera `isYatzy` från uppgift 3e igen, men nu med ett imperativt angreppssätt som använder en **`while`**-sats i stället för funktionella `forall`. Ta hjälp av en variabel `i` som håller reda på index och en variabel `foundDiff` som håller reda på om ett avvikande värde upptäcks. Funktionen kräver ca 9 rader, så det kan vara lämpligt att öppna en editor att skriva i medan du klurar ut lösningen. Börja med att skriva pseudokod, gärna med penna på papper. Prova genom att klistra in i REPL.

b) En imperativ implementation av `diceMatrixToString` från uppgift 3g med hjälp av förändringsbara `StringBuilder`[5] visas nedan. Förklara hur nedan kod fungerar. Vad händer om `xss` är tom? Vad händer om `xss` bara innehåller tomma vektorer? Nämn en fördel och en nackdel med att använda **`val`** `sb: StringBuilder` och `append`, jämfört med en vanlig, oföränderlig **`var`** `s: String` och `+` för tillägg i slutet.

```
def diceMatrixToString(xss: Vector[Vector[Int]]): String =
  val sb = new StringBuilder()
  for(m <- xss.indices) do
    for(n <- xss(m).indices) do
      sb.append(xss(m)(n).toString)
      if n < xss(m).size - 1 then sb.append(" ")
      else if m < xss.size - 1 then sb.append("\n")
    end for
  end for
  sb.toString
```

c) Gör som träning en imperativ implementation av `filterYatzy` med en **`for-do`**-sats (alltså utan att använda `filter`, och utan att använda **`yield`**).

d) Förklara hur nedan funktionella implementation av `filterYatzy` med **`for-yield`**-uttryck fungerar. Tycker du din imperativa lösning är lättare eller svårare att läsa och förstå jämfört nedan funktionella lösning?

```
def filterYatzy(xss: Vector[Vector[Int]]): Vector[Vector[Int]] =
  (for i <- xss.indices if isYatzy(xss(i)) yield xss(i)).toVector
```

Uppgift 6. *Strängtabell med kolumnrubriker.*

a) Implementera case-klassen `Table` enligt specifikationen nedan. Du kan förutsätta att alla rader har lika många kolumner som antalet element i `headings`, samt att alla rubrikerna i `headings` är unika. Parametern `sep` anger det tecken som används för att separera kolumner. Detta förutsätts också gälla för indatafiler som läses in med `fromFile`.

Tips:

- Värdet `indexOfHeading` kan skapas med hjälp av metoden `zipWithIndex` som fungerar på alla sekvenssamlingar, samt metoden `toMap` som fungerar på se-

[5] https://www.scala-lang.org/api/2.12.9/scala/collection/mutable/StringBuilder.html

kvenser av 2-tupler. Undersök först hur metoderna fungerar i REPL och sök upp deras dokumentation.

- Skapa en indatafil som du kan använda för att testa att `Table` fungerar.

```
case class Table(
  data: Vector[Vector[String]],
  headings: Vector[String],
  sep: Char
):
  /** A 2-tuple with (number of rows, number of columns) in data */
  val dim: (Int, Int) = ???

  /** The element in row r and column c of data, counting from 0 */
  def apply(r: Int, c: Int): String = ???

  /** The row-vector r in data, counting from 0 */
  def row(r: Int): Vector[String]= ???

  /** The column-vector c in data, counting from 0 */
  def col(c: Int): Vector[String] = ???

  /** A map from heading to index counting from 0 */
  lazy val indexOfHeading: Map[String, Int] = ???

  /** The column-vector with heading h in data */
  def col(h: String): Vector[String] = ???

  /** A vector with the distinct, sorted values of col with heading h */
  def values(h: String): Vector[String] = ???

  /** Headings and data with columns separated by sep */
  override lazy val toString: String = ???

object Table:
  /** Creates a new Table from fileName with columns split by sep */
  def fromFile(fileName: String, sep: Char = ';'): Table = ???
```

b) Skapa med hjälp av `Table` ett program som kan köras från terminalen med `scala run infile.csv ';'` som ger en utskrift av antalet förekomster av olika värden i respektive kolumn (alltså en variant av registrering).

Uppgift 7. *Skapa ett yatzy-spel för användning i terminalen.* Bygg ett förenklat yatzy-spel i terminalen där användaren kan bestämma vilka tärningar som ska slås om. Börja med något riktigt enkelt och bygg sedan vidare på ditt spel genom att införa fler och fler funktioner.

8.2.3 Fördjupningsuppgifter; utmaningar

Uppgift 8. *Generiska funktioner.* En generisk funktion har (minst) en typparameter inom klammerparenteser efter namnet, till exempel `[T]`. Denna typ förekommer sedan som typ på (någon av) parametrarna i parameterlistan. Kompilatorn härleder en konkret typ vid kompileringstid och ersätter typparametern med denna konkreta typ. På så sätt kan en funktion fungera för många olika typer.

a) Förklara för varje rad nedan vad som händer.

```
scala> def tnirp[T](x: T): Unit = println(x.toString.reverse)
scala> tnirp(42)
scala> tnirp("hej")
scala> case class Gurka(vikt: Int)
scala> tnirp(Gurka(42))
scala> tnirp[String](42)
scala> tnirp[Double](42)
```

b) Man kan kombinera generiska funktioner med funktioner som tar funktioner som parametrar. Det är så `map` och `foreach` är implementerade. Förklara för varje rad nedan vad som händer.

```
scala> def compose[A, B, C](f: A => B, g: B => C)(x: A): C = g(f(x))
scala> def inc(x: Int): Int = x + 1
scala> def half(x: Int): Double = x / 2.0
scala> compose(inc, half)(42)
scala> compose(half, inc)(42)
```

c) Hur lyder felmeddelandet på sista raden ovan? Ändra `inc` och/eller `half` så att typerna passar.

Uppgift 9. *Generiska klasser.* Även klasser kan vara generiska. En generisk klass har (minst) en typparameter inom klammerparenteser efter klassens namn.

a) Testa nedan generiska klass `Cell[T]` i REPL. Skapa instanser av klassen `Cell[T]` där typparametern `T` binds till olika konkreta typer och förklara vad som händer.

```
scala> class Cell[T](var value: T):
         override def toString = "Cell(" + value + ")"

scala> new Cell(42)
scala> new Cell("hej")
scala> new Cell(new Cell(math.Pi))
scala> new Cell[String](42)
scala> new Cell[Double](42)
```

b) Lägg till metoden `def concat[U](that: Cell[U]):Cell[String]` i klassen `Cell` som konkatenerar strängrepresentationerna av de båda cellvärdena.

```
scala> val a = new Cell("hej")
scala> val b = new Cell(42)
scala> a concat b
```

c) Vilken sorts celler kan du konkatenera om du tar bort typparameternamnet `U` i `concat` samtidigt som du använder `Cell[T]` som typ på värdeparametern `that`? Vad ger det för konsekvenser för celler av annan typ än `Cell[String]`?

Uppgift 10. *Implementera fler generiska metoder i* `Matrix[T]`. Bygg vidare på uppgift 4 och implementera nedan specifikation. Skapa egna tester som kontrollerar att alla metoder fungerar som förväntat.

Specification **`Matrix[T]`**

```
/** En oföränderlig, generisk Matris-klass. */
case class Matrix[T](data: Vector[Vector[T]]):
  require(???)  // garantera att alla rader har lika många kolumner

  /** Ger ett par med antal rader och kolumner. */
  val dim: (Int, Int) = ???

  /** Ger elementet på plats (row, col). */
  def apply(row: Int, col: Int): T = ???

  /** Ger en ny matris där elementet på plats (row, col) har värdet value. */
  def updated(row: Int, col: Int)(value: T): Matrix[T] =  ???

  /** Applicerar f på alla element. */
  def foreach(f: T => Unit): Unit = ???

  /** Applicerar f på alla index. */
  def foreachIndex(f: (Int, Int) => Unit): Unit = ???

  /** Ger en ny matris med resultaten av elementvis applicering av f. */
  def map[U](f: T => U): Matrix[U] = ???

  /** Ger en ny matris med resultaten av applicering av f på varje index. */
  def mapIndex[U](f: (Int, Int) => U): Matrix[U] = ???

  /** Ger en utskriftsvänlig strängrepresentation av matrisen. */
  override def toString = ???

object Matrix:
  /** Ger en matris med dimension dim där alla element har värdet value. */
  def fill[T](dim: (Int, Int))(value: T): Matrix[T] = ???
```

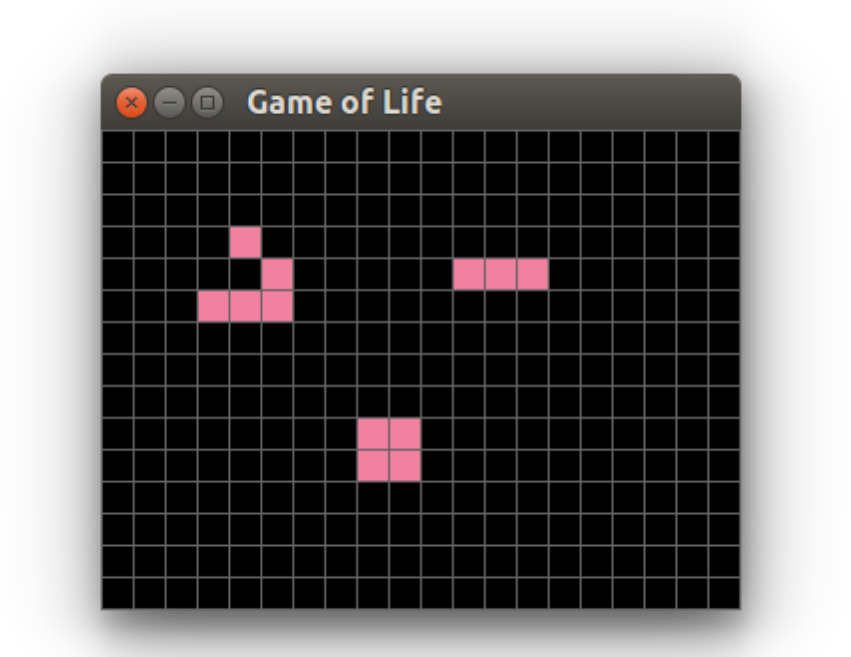
8.3 Laboration: `life`

Mål

- ☐ Kunna skapa och använda matriser med hjälp av en generisk datatyp.
- ☐ Kunna iterera över alla element i en matris.
- ☐ Träna på algoritmkonstruktion.
- ☐ Träna på hantering av både oföränderliga och förändringsbara objekt.
- ☐ Prova på att använda en avlusare (eng. *debugger*) i en integrerad utvecklingsmiljö (IDE), t.ex. VS code.

Förberedelser

- ☐ Gör övning `matrices` i kapitel 8, speciellt uppgift 4.
- ☐ Läs igenom hela laborationen och studera den givna koden[6].
- ☐ Läs appendix D om avlusning (eng. *debugging*).
- ☐ Hämta given kod via kursen github-plats eller via hemsidan under Download.



Figur 8.1: Ett binärt, mörkt datauniversum av dimension 15×20. Cellkolonin innehåller tre cellgrupper: ett rymdskepp av typen *glider*, en *blinker* och ett *block*.

8.3.1 Bakgrund

Game of Life simulerar en koloni av encelliga organismer som lever, förökar sig och dör i en matris, enligt några enkla men väl valda regler som konstruerades av matematikern John Horton Conway på 1970-talet. Spelet går ut på att simulera flera generationer utifrån en startkonfiguration, även kallad *cellkoloni*, där varje enskild cells överlevnad beror på dess omgivning. Spelet har inga medvetna spelare och om reglerna följs så kommer slutresultatet enbart bero på startkonfigurationen.

[6] https://github.com/lunduniversity/introprog/tree/master/workspace/w08_life

I *Game of Life* består universum av en matris med celler som är antingen levande eller döda. Varje cell har 8 stycken *grannar*, som utgörs av de närmsta omgivande cellerna vertikalt, horisontellt och diagonalt. Varje cells tillstånd i nästa generation bestäms av följande regler:

1. **Fortlevnad**. Om en levande cell har två eller tre grannar så lever den vidare.
2. **Död**. Om en levande cell har färre än två eller mer än tre grannar så dör den av underpopulation respektive överpopulation.
3. **Födelse**. Om cellen är död och har exakt tre grannar så föds den och dess tillstånd ändras till levande, annars fortsätter den vara död.

Flera cellkolonier uppvisar ett ”levande” beteende där cellmatrisen koloniseras på intressanta vis när en sekvens av generationer visualiseras. Detta är ett exempel på *emergent* beteende där komplexa, självorganiserade strukturer kan uppstå ur enkla förutsättningar.

Läs mer om *Game of Life* på Wikipedia:

- https://en.wikipedia.org/wiki/Conway's_Game_of_Life
- https://sv.wikipedia.org/wiki/Game_of_Life

8.3.2 Obligatoriska krav

Följande funktionella krav ska uppfyllas av ditt program:

- ☐ Levande celler ska ha den vackra rosa[7] RGB-färgen `(242, 128, 161)`.
- ☐ Döda celler ska vara svarta som rymden.
- ☐ Detta mörka universum med binära dataceller ska ritas i ett rutnät bestående av smala, stilfulla linjer, så som visas i fig. 8.1.
- ☐ Tangenttryckningar och musklick ska fungera enligt följande hjälptext, som ska skrivas ut då programmet startas:

```
val help = """
  Welcome to GAME OF LIFE!

  Click on cell to toggle.
  Press ENTER for next generation.
  Press SPACE to toggle play/stop.
  Press R to create random life.
  Press BACKSPACE to clear life.
  Close window to exit.
"""
```

  Då *play* aktiveras med blankstegstangenten ska en kontinuerlig simulering av universum fortgå där varje ny generation visualiseras med en lagom fördröjning emellan generationer, tills simuleringen stoppas, t.ex. genom tryck ånyo på blankstegstangenten. Vid varje *Enter*-tryck visas *en* efterkommande generation och ev. pågående simulering stoppas. Vid musklick på en cell ska livstillståndet växlas från levande till död eller vice versa. Ett tryck på R ska ge slumpmässigt liv. Ett tryck på backstegstangenten ska rendera alla universums cellers död.

Din kod ska utformas enligt dessa design-krav:

- ☐ Alla klasser och singelobjekt ska ligga i paketet `life`.
- ☐ Det ska finnas en oföränderlig case-klass `Life` som representerar ett celluniversum med hjälp av en `Matrix[Boolean]` från uppgift 4 i veckans övning.

[7] https://www.dsek.se/aktiva/grafiskprofil/farg.php

☐ Det ska finnas en klass `LifeWindow` som visualiserar en instans av klassen `Life` i ett `introprog.PixelWindow` så som i fig. 8.1.

8.3.3 Valbara krav – välj minst ett

Du ska implementera minst ett (gärna flera) av dessa krav:

☐ Cellerna ska färgläggas i olika färger i enlighet med reglerna för nästa generation. Fortlevnad ska fortfarande vara vackert rosa och fortvarig död svart. Följande färger föreslås men välj andra om du tycker det blir finare:

```
val UnderPopulated = java.awt.Color.cyan  // en giftig färg
val OverPopulated  = java.awt.Color.red   // rödklämd av trängsel
val WillBeBorn     = new java.awt.Color(40, 0, 0)  // snart levande
```

Ge dessutom `LifeWindow` en klassparameter `isMultiColor` som gör det möjligt att välja om det ska bli mångfärgade celler eller om det bara ska finnas rosa och svart som i grundkraven.

☐ Om man trycker på `S` för *Save* ska `introprog.Dialog.file("Save Life")` visas och, om användaren inte trycker **Cancel**, det aktuella livet sparas med hjälp av
`introprog.IO.saveString` i en textfil via metoden `toString` i `Life`.

☐ Om man trycker på `O` för *Open* ska `introprog.Dialog.file("Open Life")` anropas och ett nytt universum läsas in från textfil enligt lämpligt format. Inläsningen ska ske med hjälp av `introprog.IO.loadString` och tolkas till en `Life`-instans av en metod i kompanjonsobjektet med detta huvud:

```
def fromString(s: String, rowDelim: String="\n", alive: Char='0'): Life
```

Testa med filen `glider-gun.txt` som ska ha följande innehåll på de första 11 raderna och totalt 32 rader där alla rader efter elfte raden innehåller tomt liv:

```
> head -11 glider-gun.txt
------------------------------------------
-------------------------0----------------
-----------------------0-0----------------
-------------00------00------------00-----
------------0---0----00------------00-----
-00--------0-----0---00------------------
-00--------0---0-00----0-0---------------
-----------0-----0-------0---------------
------------0---0-------------------------
-------------00-------------------------
------------------------------------------
```

☐ Universum ska vara cirkulärt, d.v.s grannen vid kanten finns på andra sidan genom att indexeringen börjar om (eng. *wrapped*) enligt modulo-räkning. Inför en klassparameter `isWrapped` i `Life` och en variabel `wrapped: Boolean` i kompanjonsobjektet `Life` som styr om fabriksmetoderna skapar ett universum som är cirkulärt eller ej, så att du lätt kan konfigurera detta. *Tips:* Du har stor nytta av att använda `java.lang.Math.floorMod` i `apply`-metoden i `Life`; metoden `floorMod` räknar på lämpligt sätt med negativa värden, se dokumentationen för `Math`-paketet i JDK8.

☐ Läs om varianter till `Game of Life` på Wikipedia och implementera alternativa regler som görs valbara genom konfigurering via `args`-parametern i `main`.

☐ Skapa en klass `LifeStatistics` som genom väldigt många simuleringar ska ta reda på sannolikheten att en slumpmässig cellkoloni efter n generationer fortfarande utvecklas, respektive är helt dött. Ingen visualisering med `PixelWindow` ska ske; endast antalet celler som lever vid generation n och antalet celler som ändrades sedan generation $n-1$ behöver registreras.

8.3.4 Tips och förslag

1. Här är ett förslag på hur du kan utforma klassen `Life`:

```
package life

case class Life(cells: Matrix[Boolean]):

  /** Ger true om cellen på plats (row, col) är vid liv annars false.
    * Ger false om indexeringen är utanför universums gränser.
    */
  def apply(row: Int, col: Int): Boolean = ???

  /** Sätter status på cellen på plats (row, col) till value. */
  def updated(row: Int, col: Int, value: Boolean): Life = ???

  /** Växlar status på cellen på plats (row, col). */
  def toggled(row: Int, col: Int): Life = ???

  /** Räknar antalet levande grannar till cellen i (row, col). */
  def nbrOfNeighbours(row: Int, col: Int): Int = ???

  /** Skapar en ny Life-instans med nästa generation av universum.
    * Detta sker genom att applicera funktionen rule på cellerna.
    */
  def evolved(rule: (Int, Int, Life) => Boolean = Life.defaultRule):Life =
    var nextGeneration = Life.empty(cells.dim)
    cells.foreachIndex( (r,c) =>
      ???
    )
    nextGeneration

  /** Radseparerad text där 0 är levande cell och - är död cell. */
  override def toString = ???

object Life:
  /** Skapar ett universum med döda celler. */
  def empty(dim: (Int, Int)): Life = ???

  /** Skapar ett unviersum med slumpmässigt liv. */
  def random(dim: (Int, Int)): Life = ???

  /** Implementerar reglerna enligt Conways Game of Life. */
  def defaultRule(row: Int, col: Int, current: Life): Boolean = ???
```

Du har nytta av metoden `nbrOfNeighbours` när du ska implementera `defaultRule`. Vid implementation av `random` är metoden `foreachIndex` i `Matrix[T]` smidig att använda. Om du som i förslaget ovan låter `evolved` ta uppdateringsregeln som en funktionsparameter blir det lättare att konfigurera vilka regler som ska gälla och därmed blir det även lättare att skapa varianter av *Game of Life* genom att

införa nya regler i kompanjonsobjektet (se en av de valfria uppgifterna med vidare hänvisning till Wikipedia).

2. Här är ett förslag på hur du kan utforma klassen `LifeWindow`:

```
package life

import introprog.PixelWindow
import introprog.PixelWindow.Event

object LifeWindow:
  val EventMaxWait = 1 // milliseconds
  var NextGenerationDelay = 200 // milliseconds
  // lägg till fler användbara konstanter här tex färger etc.

class LifeWindow(rows: Int, cols: Int):
  import LifeWindow.* // importera namn från kompanjon

  var life = Life.empty(rows, cols)
  val window: PixelWindow = ???
  var quit = false
  var play = false

  def drawGrid(): Unit = ???

  def drawCell(row: Int, col: Int): Unit = ???

  def update(newLife: Life): Unit =
    val oldLife = life
    life = newLife
    life.cells.foreachIndex{ ??? }

  def handleKey(key: String): Unit = ???

  def handleClick(pos: (Int, Int)): Unit = ???

  def loopUntilQuit(): Unit = while !quit do
    val t0 = System.currentTimeMillis
    if play then update(life.evolved())
    window.awaitEvent(EventMaxWait)
    while window.lastEventType != PixelWindow.Event.Undefined do
      window.lastEventType match
        case Event.KeyPressed  =>  handleKey(window.lastKey)
        case Event.MousePressed => handleClick(window.lastMousePos)
        case Event.WindowClosed => quit = true
        case _ =>
      window.awaitEvent(EventMaxWait)
    val elapsed = System.currentTimeMillis - t0
    Thread.sleep((NextGenerationDelay - elapsed) max 0)

  def start(): Unit = { drawGrid(); loopUntilQuit() }
```

3. **Dra nytta av din IDE.** Det finns många användbara finesser i en integrerad utvecklingsmiljö (eng. *Integrated Development Environment (IDE)*), så som Microsoft

VS Code[8] med tillägget Metals[9] eller JetBrains IntelliJ IDEA[10] med Scala-plugin[11]. Läsa på nätet om din IDE och lär dig om sådant du inte kände till som verkar användbart. Sök speciellt upp listan med kortkommandon[12] [13] och lär dig några valfria kortkommandon som kan hjälpa dig att snabba upp sådant du gör ofta.

4. Studera dokumentationen om avlusaren (debuggern) in din IDE. [14] [15] [16]

[8] https://code.visualstudio.com/
[9] https://scalameta.org/metals/docs/editors/vscode
[10] https://www.jetbrains.com/idea/
[11] https://www.jetbrains.com/help/idea/discover-intellij-idea-for-scala.html
[12] https://code.visualstudio.com/docs/getstarted/keybindings
[13] https://www.jetbrains.com/idea/resources/
[14] https://scalameta.org/metals/docs/editors/vscode#running-and-debugging-your-code
[15] https://code.visualstudio.com/docs/editor/debugging
[16] https://www.jetbrains.com/help/idea/debugging-code.html

Kapitel 9

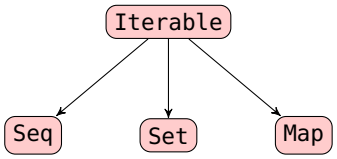
Mängder och tabeller

Begrepp som ingår i denna veckas studier:

- ☐ innehållstest
- ☐ mängd
- ☐ Set
- ☐ mutable.Set
- ☐ nyckel-värde-tabell
- ☐ Map
- ☐ mutable.Map
- ☐ hash code
- ☐ java.util.HashMap
- ☐ java.util.HashSet
- ☐ persistens
- ☐ serialisering
- ☐ textfiler
- ☐ Source.fromFile
- ☐ java.nio.file

9.1 Teori

9.1.1 Hierarki av samlingstyper i `scala.collection` v2.13



`Iterable` har metoder som är implementerade med hjälp av:

```
def foreach[U](f: Elem => U): Unit
def iterator: Iterator[A]
```

`Seq`: ordnade i sekvens
`Set`: unika element
`Map`: par av (nyckel, värde)

Samlingen **`Vector`** är en `Seq` som är en `Iterable`.
De konkreta samlingarna är uppdelade i dessa paket:

`scala.collection.immutable` där flera är **automatiskt** importerade
`scala.collection.mutable` som **måste importeras** explicit
(undantag: primitiva förändringsbara `scala.Array` är automatiskt synlig)

9.1.2 Metoden `iterator` ger en ”engångs-iterator”

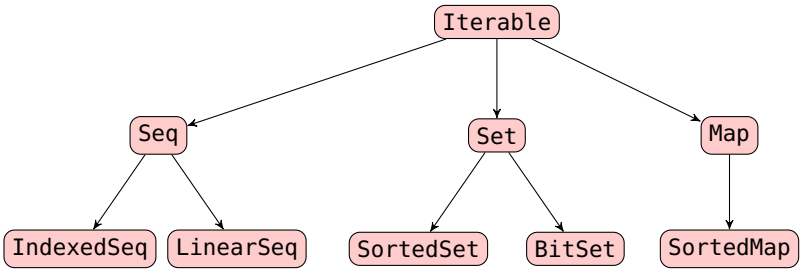
Med `iterator` kan man iterera med **`while`**, men endast **en gång**; sedan är iteratorn ”förbrukad”. (Men man kan be om en ny.) Används ”under huven” i samlingsbiblioteket för att implementera andra metoder.

```
scala> val xs = Vector(1,2,3,4)
val xs: Vector[Int] = Vector(1, 2, 3, 4)

scala> val it = xs.iterator
val it: Iterator[Int] = <iterator>

scala> while it.hasNext do print(it.next)
1234

scala> it.hasNext
val res0: Boolean = false

scala> it.next
java.util.NoSuchElementException: next on empty iterator
```

Normalt behöver man **inte** använda `iterator`: det finns oftast färdiga metoder som gör det man vill, till exempel `foreach`, `map`, `sum`, `min` etc.

9.1.3 Mer specifika samlingstyper i `scala.collection`

Det finns **mer specifika subtyper** av `Seq`, `Set` och `Map`:



Vector är en **IndexedSeq** medan **List** är en **LinearSeq**.

docs.scala-lang.org/overviews/collections-2.13/overview.html

9.1.4 Några oföränderliga och förändringsbara sekvenssamlingar

```
scala.collection.immutable.Seq.
                              IndexedSeq.
                                        Vector
                                        Range
                              LinearSeq.
                                        List
                                        Queue
  scala.collection.mutable.Seq.
                              IndexedSeq.
                                        ArrayBuffer
                                        StringBuilder
                              LinearSeq.
                                        ListBuffer
                                        Queue
```

Fördjupning: Studera samlingars prestanda-egenskaper här:
docs.scala-lang.org/overviews/collections-2.13/performance-characteristics.html

---

9.1.5 Några användbara metoder på samlingar

Iterable	`xs.size`	antal elementet
	`xs.head`	första elementet
	`xs.last`	sista elementet
	`xs.take(n)`	ny samling med de första n elementet
	`xs.drop(n)`	ny samling utan de första n elementet
	`xs.foreach(f)`	gör `f` på alla element, returtyp `Unit`
	`xs.map(f)`	gör `f` på alla element, ger ny samling
	`xs.filter(p)`	ny samling med bara de element där p är sant
	`xs.groupBy(f)`	ger en `Map` som grupperar värdena enligt f
	`xs.mkString(",")`	en kommaseparerad sträng med alla element
	`xs.zip(ys)`	ny samling med par (x, y); ”zippa ihop” xs och ys
	`xs.zipWithIndex`	ger en `Map` med par (x, index för x)
	`xs.sliding(n)`	ny samling av samlingar genom glidande ”fönster”
Seq	`xs.length`	samma som `xs.size`
	`xs :+ x`	ny samling med x sist efter xs
	`x +: xs`	ny samling med x före xs

Prova fler samlingsmetoder ur snabbreferensen: http://cs.lth.se/quickref

Minnesregel för `+:` och `:+` **Colon on the collection side**
Digga denna: https://youtu.be/Lm9JWlEMHjo?si=sNdn_ZDaORlGr3lt

---

9.1.6 Repetition: Vad är en sekvens?

- En sekvens är en **följd av element** som
  - är **numrerade** (t.ex. från noll), och
  - är av en viss **typ** (t.ex. heltal).
- En sekvens kan innehålla **dubbletter**.
- En sekvens kan vara **tom** och ha längden noll.
- Exempel på en icke-tom sekvens med dubbletter:

```
scala> val xs = Vector(42, 0, 42, -9, 0, 13, 7)
val xs: Vector[Int] = Vector(42, 0, 42, -9, 0, 13, 7)
```

- **Indexering** ger ett element via dess ordningsnummer:

```
scala> xs(2)
val res0: Int = 42

scala> xs.apply(2)
val res1: Int = 42
```

9.1.7 En sträng är också en `IndexedSeq[Char]`

Det sker vid behov **implicit konvertering** från `String` till `IndexedSeq[Char]`.

```
scala> val x: IndexedSeq[Char] = "hej"
val x: IndexedSeq[Char] = hej
```

Detta gör att **alla samlingsmetoder på `Seq` även funkar på strängar** och även flera andra smidiga strängmetoder erbjuds **utöver** de som finns i `java.lang.String` genom klassen `StringOps`.

```
scala> "hej".  //tryck på TAB och se alla strängmetoder
JLine: do you wish to see all 248 possibilities (42 lines)?
```

Detta är en stor fördel med Scala jämfört med många andra språk, som har strängar som inte kan allt som andra sekvenssamlingar kan.

9.1.8 Konvertera mellan olika samlingstyper

- För vanligt förekommande konverteringar finns metoderna `toVector`, `toList`, `toArray`, `toBuffer`, `toMap`, `toSeq`, `toIndexedSeq`, `toSet`, `toString`
- Metoden `to` (ny från Scala 2.13) tar ett **kompanjonsobjekt** ur samlingsbiblioteket som argument och kan användas för konvertering till godtycklig samlingstyp.

- Detta kräver kopiering om underliggande representation är olika och samlingen är förändringsbar.
- Kan användas för att t.ex. konvertera mellan oföränderlig och förändringsbar samling:

```
scala> val ms = Set(1,2,3).to(collection.mutable.Set)
val ms: scala.collection.mutable.Set[Int] = HashSet(1, 2, 3)
```

9.1.9 Vad är en mängd?

- En **mängd** är en samling **unika** element av en viss **typ**.
- En mängd kan alltså inte innehålla dubbletter:

```
scala> Set(1,1,2,2,3,3,4,4,5,5)
val res0: Set[Int] = HashSet(5, 1, 2, 3, 4)
```

- En mängd är **inte** en sekvens: du kan inte utgå från att elementen ligger i någon viss ordning, t.ex. den ordning som de ges vid konstruktion; en mängd har ej längd, men en **storlek**; metoden `size` ger antalet element men metoden `length` saknas.
- En mängd kan vara **tom** och har då storleken `0`.
- Man kan gå igenom element i **någon** ordning (exakt vilken är ej def.), med till exempel `xs.map(f)` eller `for (x <- xs) yield f(x)`
- Det går **inte** att indexera i en mängd med `apply`, som i stället ger **innehållstest**: `Set(1,2,3).apply(3) == true`
- En mängd `Set[T]` med element av typen `T` kan således ses som ett **predikat för innehållstest**: alltså en funktion `T => Boolean` som är `true` om elementet finns annars `false`

9.1.10 Oföränderlig mängd

- **Skapa**:

```
scala> var xs = Set("gurka", "tomat", "banan", "pingvin")
```

- **Läsa**: avgöra medlemskap

```
scala> xs("gurka")
val res1: Boolean = true
```

- **Uppdatera**: lägg till element (händer inget om redan finns)

```
scala> xs = xs + "jordekorre" // en ny, delvis förändrad
```

- **Ta bort**: (om finns, annars händer inget)

```
scala> xs = xs - "gurka" // en ny, delvis förändrad
```

`SLUT` = Skapa, Läsa, Uppdatera, Ta bort `CRUD` = Create, Read, Update, Delete

---

9.1.11 Mysteriet med de försvunna elementen

Vad händer här?

```
scala> val xs1 = Vector(1,2,3,4,5,6)
scala> xs1.map(_ % 2).count(_ == 0)
val res0: Int = 3                          // antalet jämna tal
scala> val xs2 = Set(1,2,3,4,5,6)
scala> xs2.map(_ % 2).count(_ == 0)
val res1: Int = 1                          // varför?
```

Mängdegenskaper ger att `xs2.map(_ % 2) == Set(0, 1)`
Fundera alltid noga på om du **riskerar att förlora duplikat** som du egentligen hade velat behålla!
Använd `toSeq` på mängd om du behöver sekvensegenskaper:

```
scala> xs2.toSeq.map(_ % 2).count(_ == 0)
val res1: Int = 3         // med toSeq blir det som vi ville
```

9.1.12 Förändringsbar mängd

Med en **förändringsbar** mängd kan man stegvis utöka på plats.

```
1 scala> val mängd = scala.collection.mutable.Set.empty[Int]
2
3 scala> for i <- 1 to 1_000_000 do mängd.addOne(i)
4
5 scala> mängd.contains(-1)   // samma som mängd(-1) eller mängd.apply(-1)
```

En **mängd** är **snabb** på att avgöra om ett element **finns eller inte** i mängden. Ingen linjärsökning krävs eftersom den smarta implementationen av datastrukturen medger snabb uppslagning (eng. *lookup*) av ett element.
Men i en sekvens krävs linjärsökning vid innehållstest:

```
1 scala> val sekvens = (1 to 1_000_000).toVector
2
3 scala> sekvens.contains(-1)   // kräver linjärsökning ända till slutet
```

Övning: Testa själv att mäta tidsskillnaden med hjälp av:

```
def nanos(b: => Unit) = { val t0 = System.nanoTime; b; System.nanoTime - t0 }
```

9.1.13 Speciella metoder på förändringsbar mängd

Förändringsbara mängder har metoder som ändrar på plats:

```
scala> val s = scala.collection.mutable.Set.empty[Int]

scala> s.addOne(1)     // finns även under namnet += om du gillar operator-notion
val res0: scala.collection.mutable.Set[Int] = HashSet(1)
```

```
scala> s.addOne(2).addOne(3).addOne(3).addOne(42) // addOne returnerar this
val res1: scala.collection.mutable.Set[Int] = HashSet(1, 2, 3, 42)

scala> res0.eq(res1)  // samma instans av mutable.Set (ingen ny har skapats)
val res2: Boolean = true

scala> s.addAll(Vector(3, 4, 5))  // finns även ++= om du gillar operator-notion
val res3: scala.collection.mutable.Set[Int] = HashSet(1, 2, 3, 4, 5, 42)

scala> s.subtractOne(1).subtractAll(List(1,2,3))  // finns även -= och --=
val res4: scala.collection.mutable.Set[Int] = HashSet(4, 5, 42)

scala> s.filterInPlace(_ > 4)
val res5: scala.collection.mutable.Set[Int] = HashSet(5, 42)
```

`addOne`, `addAll`, `subtractOne`, `subtractAll`, `filterInPlace` returnerar `this` så du kan ändra på plats med kedjade anrop med punktnotation.

9.1.14 Vad är en nyckel-värde-tabell?

- En **nyckel-värde-tabell** är en samling element som är **par** med:
  en **nyckel** av någon typ `K` och ett **värde** av någon typ `V`.
- En sådan tabell kan skapas ur en sekvens av par `(k, v)`
  där `k` är en nyckel och `v` är ett värde:

```
scala> val ålder = Map("Björn" -> 42, "Sandra" -> 35, "Kim" -> 19)
val ålder: Map[String, Int] = Map(Björn -> 42, Sandra -> 35, Kim -> 19)
```

- Tabellens nycklar utgör en mängd som ges av metoden `keySet`;
  nycklarna är **unika**.
- Elementen utgör **inte en sekvens** och har ingen speciell ordning;
  en nyckel-värde-tabell har ej längd, men en **storlek**;
  metoden `size` ger antalet element.
- En tabell kan ses som en uppslagsfunktion (eng. *dictionary*):
  alltså en funktion `K => V` som ger ett värde givet en nyckel.

9.1.15 Den fantastiska nyckel-värde-tabellen `Map`

- En **nyckel-värde-tabell** (eng. *key-value table*) är en slags generaliserad vektor där man kan ”indexera” med godtycklig typ.
- Kallas även **hashtabell** (eng. *hash table*), **lexikon** (eng. *Dictionary*) eller **mapp** (eng. *Map*) (det blir lätt sammanblandning med metoden `map`).
- Om man vet nyckeln kan man slå upp värdet **snabbt**, på liknande sätt som indexering sker snabbt i en vektor givet heltalsindex.
- Denna datastruktur är **mycket användbar** och fungerar som en slags databas i kombination med filtrering, registrering, etc.

9.1.16 Oföränderlig nyckel-värde-tabell

- **Skapa**: ge par till metoden `apply`

```
scala> var födelse = Map("C" -> 1972,  "C++" -> 1983, "C#" -> 2000,
  "Scala" -> 2004, "Java" -> 1995, "Javascript" -> 1995, "Python" -> 1991)
```

- **Läsa**: slå upp ett värde med hjälp av en nyckel

```
scala> val year = födelse.apply("Scala")
val year: Int = 2004
```

- **Uppdatera**: lägga till ett par, ersätta ett par

```
scala> födelse = födelse + ("Kotlin" -> 2011)
födelse: Map[String, Int] = HashMap(Scala -> 2004, C# -> 2000, Python -> 1991,
Javascript -> 1995, C -> 1972, C++ -> 1983, Kotlin -> 2011, Java -> 1995)
```

- **Ta bort** ett par via nyckeln (om finns, annars händer inget)

```
scala> födelse = födelse - "Python"
födelse: Map[String, Int] = HashMap(Scala -> 2004, C# -> 2000,
Javascript -> 1995, C -> 1972, C++ -> 1983, Kotlin -> 2011, Java -> 1995)
```

9.1.17 Fler exempel nyckel-värde-tabell

Några ofta förekommande metoder på tabeller:

- `xs.keySet` ger en mängd av alla nycklar
- `xs.map(f)` kör funktionen f på alla par (key, value) i **någon** ordning
- `xs.map((k, v) => k -> f(v))` kör funktionen f på alla **värden**

```
scala> val färg = Map("gurka" -> "grön", "tomat"->"röd", "aubergine"->"lila")
val färg: Map[String, String] =
  Map(gurka -> grön, tomat -> röd, aubergine -> lila)

scala> färg("gurka")
val res0: String = grön

scala> färg.keySet
val res1: Set[String] = Set(gurka, tomat, aubergine)

scala> val ärGrönSak = färg.map((k,v) => (k, v == "grön"))
val ärGrönSak: Map[String,Boolean] =
  Map(gurka -> true, tomat -> false, aubergine -> false)

scala> val baklängesFärg = färg.map((k, v) => k -> v.reverse)
val baklängesFärg: Map[String,String] =
  Map(gurka -> nörg, tomat -> dör, aubergine -> alil)
```

9.1.18 Från sekvens av par till tabell

```
scala> val xs = Vector(("Kim",42), ("Pam", 42), ("Kim", 50), ("Pam", 50))
val xs: Vector[(String, Int)] =
  Vector((Kim,42), (Pam,42), (Kim,50), (Pam,50))

scala> xs.toMap
val res0: Map[String, Int] =
  Map(Kim -> 50, Pam -> 50) // inga dublettnycklar

scala> val grupperaEfterNamn = xs.groupBy(_._1)
grupperaEfterNamn: Map[String,Vector[(String, Int)]] =
  Map(Kim -> Vector((Kim,42), (Kim,50)), Pam -> Vector((Pam,42), (Pam,50)))

scala> val grupperaEfterÅlder = xs.groupBy(_._2)
grupperaEfterÅlder: Map[Int,Vector[(String, Int)]] =
  Map(50 -> Vector((Kim,50), (Pam,50)), 42 -> Vector((Kim,42), (Pam,42)))
```

9.1.19 Övning: Implementera en `Multimap`

- Om du lägger till ett värde i en *vanlig* `Map` så ersätts värdet:

```
scala> val m = Map(1 -> 2, 1 -> 3, 2 -> 1, 2 -> 2)
val m: Map[Int, Int] = Map(1 -> 3, 2 -> 2)   //senaste värdet gäller
```

  ...men ibland vill vi i stället lagra alla tillagda värden.
- En **multimap** är en speciell nyckel-värde-tabell där värdena utgör en samling (ofta en mängd).
- En multimap samlar alla värden som har samma nyckel.

```
scala> val mm = Multimap(1 -> 2, 1 -> 3, 2 -> 1, 2 -> 2)
val mm: Multimap[Int, Int] = Multimap(1 -> Set(2, 3), 2 -> Set(1, 2))
```

Övning: Implementera en multimap som fungerar som ovan, med hjälp av en case-klass med attributet `toMap` som är en oföränderlig nyckel-värde-tabell där värdena är en mängd.
Tips: Använd `groupBy`

9.1.20 Lösning: `Multimap`

```
case class Multimap[K, V] private (toMap: Map[K,Set[V]]):
  def apply(k: K): Set[V] = toMap(k)

  def +(kv: (K, V)): Multimap[K, V] = kv match
    case (k, v) if toMap.isDefinedAt(k) => Multimap(toMap.updated(k, toMap(k) + v))
    case (k, v) => Multimap(toMap + (k -> Set(v)))

  override def toString = toMap.mkString("Multimap(",", ",")")

object Multimap:
```

```
  def apply[K, V](kvs: (K,V)*): Multimap[K, V] =
    new Multimap(kvs.groupBy(_._1).map((k,xs) => k -> xs.map(_._2).toSet))
```

9.1.21 Speciella metoder på förändringsbar tabell

Både `Set` och `Map` finns i **förändringsbara** varianter med extra metoder för uppdatering av innehållet ”på plats” utan att nya samlingar skapas.

```
scala> import scala.collection.mutable

scala> val mm = mutable.Map.empty[String, String]
val mm: scala.collection.mutable.Map[String, String] = HashMap()

scala> mm.addOne("hej" -> "svejs")
val res0: scala.collection.mutable.Map[String, String] =
  HashMap(hej -> svejs)

scala> mm.addAll(Seq("abra" -> "kadabra", "ada" -> "lovelace"))
val res1: scala.collection.mutable.Map[String, String] =
  HashMap(hej -> svejs, abra -> kadabra, ada -> lovelace)

scala> mm("abra")
val res2: String = kadabra
```

Metoden ++= samma som addAll; används gärna med operator-notation:
`mm ++= Seq("hej" -> "san", "abra" -> "kada", "bra" -> "scala")`

9.1.22 Övning: Förändringsbar lokalt, returnera oföränderlig

Om du vill implementera en imperativ algoritm med en föränderlig samling:
Gör gärna detta **lokalt** i en **förändringsbar** samling och returnera sedan en **oföränderlig** samling, genom att köra t.ex. `toSet` på en mängd, eller `toMap` på en hashtabell, eller `toVector` på en `ArrayBuffer` eller `Array`.

Exempel där lösningen har nytta av lokal förändring på plats:

```
def kastaTärningTillsAllaUtfallUtomEtt(sidor: Int = 6): (Int, Set[Int]) = ???
```

```
scala> kastaTärningTillsAllaUtfallUtomEtt()
val res0: (Int, Set[Int]) = (13,HashSet(5, 1, 6, 2, 3))
```

9.1.23 Övning: Förändringsbar lokalt, returnera oföränderlig

Om du vill implementera en imperativ algoritm med en föränderlig samling:
Gör gärna detta **lokalt** i en **förändringsbar** samling och returnera sedan en **oföränderlig** samling, genom att köra t.ex. `toSet` på en mängd, eller `toMap` på en hashtabell, eller `toVector` på en `ArrayBuffer` eller `Array`.

```
def kastaTärningTillsAllaUtfallUtomEtt(sidor: Int = 6): (Int, Set[Int]) =
  /*
    låt s vara en tom förändringsbar heltalsmängd
    låt n vara noll
    så länge mängden s är mindre än sidor - 1 gör:
      lägg till ett nytt tärningskast i s
      uppdatera n så att vi räknar hur många slumptal som dragits
  */
  (n, s.toSet)   // notera toSet som ger oföränderlig mängd
```

```
scala> kastaTärningTillsAllaUtfallUtomEtt()
val res0: (Int, Set[Int]) = (13,HashSet(5, 1, 6, 2, 3))
```

9.1.24 Lösning: Förändringsbar lokalt, returnera oföränderlig

Om du vill implementera en imperativ algoritm med en föränderlig samling:
Gör gärna detta **lokalt** i en **förändringsbar** samling och returnera sedan en **oföränderlig** samling, genom att köra t.ex. `toSet` på en mängd, eller `toMap` på en hashtabell, eller `toVector` på en `ArrayBuffer` eller `Array`.

```
def kastaTärningTillsAllaUtfallUtomEtt(sidor: Int = 6): (Int, Set[Int]) =
  val s = scala.collection.mutable.Set.empty[Int] //förändringsbar lokalt
  var n = 0
  while s.size < sidor - 1 do
    s.addOne(util.Random.nextInt(sidor) + 1)
    n += 1
  (n, s.toSet)   // notera toSet som ger oföränderlig mängd
```

```
scala> kastaTärningTillsAllaUtfallUtomEtt()
val res0: (Int, Set[Int]) = (13,HashSet(5, 1, 6, 2, 3))
```

I veckans uppgifter används detta i en s.k. **builder**: Först bygga upp en förändringsbar struktur i `FreqMapBuilder` steg för steg, och sedan, då alla tillägg är gjorda, övergå till oföränderlig struktur `Map[String, Int]`.

9.1.25 Metoden `sliding`

Metoden `sliding(n)` skapar med ett ”glidande fönster” en sekvens av delsekvenser av längd `n` genom att ”svepa fönstret” från början till slut:

```
scala> val xs = "fem myror är fler än fyra elefanter".split(' ').toVector
val xs: Vector[String] = Vector(fem, myror, är, fler, än, fyra, elefanter)

scala> xs.sliding(2).toVector
val res0: Vector[Vector[String]] =
  Vector(Vector(fem, myror), Vector(myror, är), Vector(är, fler),
      Vector(fler, än), Vector(än, fyra), Vector(fyra, elefanter))

```

```
scala> xs.sliding(3).toVector
val res1: Vector[Vector[String]] =
  Vector(Vector(fem, myror, är), Vector(myror, är, fler),
    Vector(är, fler, än), Vector(fler, än, fyra),
      Vector(än, fyra, elefanter))
```

Denna metod har du nytta av på veckans laboration!
(se fler exempel på övning)

9.1.26 Metoderna zipWithIndex, groupBy

```
scala> val kort = Vector("Knekt", "Dam", "Kung", "Äss")

scala> val kortIndex = kort.zipWithIndex.toMap
kortIndex: Map[String,Int] = Map(Knekt -> 0, Dam -> 1, Kung -> 2, Äss -> 3)

scala> kortIndex("Kung") > kortIndex("Knekt")
res0: Boolean = true

scala> kortIndex.map(p => p._1 -> (p._2 + 11))

scala> val tärningskast = Vector(1,2,3,4,5,6,2,4,6)

scala> val grupperaStörreÄnFyra = tärningskast.groupBy(_ > 4)
grupperaStörreÄnFyra: Map[Boolean,Vector[Int]] =
  Map(false -> Vector(1, 2, 3, 4, 2, 4), true -> Vector(5, 6, 6))

scala> val grupperaLika = tärningskast.groupBy(x => x)
grupperaLika: Map[Int,Vector[Int]] = Map(5 -> Vector(5), 1 -> Vector(1),
  6 -> Vector(6, 6), 2 -> Vector(2, 2), 3 -> Vector(3), 4 -> Vector(4, 4))

scala> val frekvens = tärningskast.groupBy(x => x).map((k,v) => k -> v.size)
frekvens: Map[Int,Int] = Map(5 -> 1, 1 -> 1, 6 -> 2, 2 -> 2, 3 -> 1, 4 -> 2)
```

9.1.27 Fler användbara samlingsmetoder

Exempel att öva på: räkna bokstäver i ord.
Undersök vad som händer i REPL:

```
val ord = "sex laxar i en laxask sju sjösjuka sjömän"
val uppdelad = ord.split(' ').toVector
val ordlängd = uppdelad.map(_.length)
val ordlängdMap = uppdelad.map(s => (s, s.size)).toMap
val grupperaEfterFörstaBokstav = uppdelad.groupBy(s => s(0))
val bokstäver = ord.toVector.filter(_ != ' ')
val antalX = bokstäver.count(_ == 'x')
val grupperade = bokstäver.groupBy(ch => ch)
val antal = grupperade.map(p => p._1 -> p._2.size)
//samma som ovan men utnyttjar "parameter untupling":
val antal2 = grupperade.map((k,v) => k -> v.size)
val sorterat = antal.toVector.sortBy(_._2)
```

```
val vanligast = antal.maxBy(_._2)
```

9.1.28 Serialisering och deserialisering

- Att **serialisera** innebär att **koda objekt** i minnet till en avkodningsbar **sekvens av symboler**, som kan lagras t.ex. i en fil på din hårddisk.
- Att **de-serialisera** innebär att **avkoda en sekvens av symboler**, t.ex. från en fil, och **återskapa objekt** i minnet.

9.1.29 Läsa text från fil och URL

I paketet `scala.io` finns singelobjektet `Source` med metoderna `fromFile` och `fromUrl` för läsning från fil resp. från URL, alltså Universal Resource Locator, som börjar t.ex. med `http://`

```
def läsFrånFil(filnamn: String): String =
  val s = scala.io.Source.fromFile(filnamn)
  try s.mkString finally s.close // säkerställ stängning även vid krasch

def läsRaderFrånFil(filnamn: String): Vector[String] =
  val s = scala.io.Source.fromFile(filnamn)
  try s.getLines.toVector finally s.close

def läsFrånWebbsida(url: String): String =
  val s = scala.io.Source.fromURL(url)
  try s.mkString finally s.close

def läsRaderWebbsida(url: String, kodning: String = "UTF-8"): Vector[String] =
  val s = scala.io.Source.fromURL(url, kodning) // läs med given teckenkodning
  try s.getLines.toVector finally s.close
```

Se vidare veckans övning. Exempel på annan teckenkodning: `"ISO-8859-1"`

9.1.30 Serialisering i modulen `introprog.IO`

- I kursens kodbibliotek `introprog` finns ett singelobjekt `IO` som samlar smidiga funktioner för serialisering och de-serialisering.
- Se api-dokumentation här:
  `http://cs.lth.se/pgk/api/`
  Sök på IO och klicka på singelobjektet.
- Se koden här:
  `https://github.com/lunduniversity/introprog-scalalib/blob/master/src/main/scala/introprog/IO.scala`
- Om du vill får du gärna använda `introprog.IO` istället för `scala.io.Source` på labben.

9.2 Övning `lookup`

Mål

- ☐ Kunna skapa och använda tupler som parametrar och returvärden.
- ☐ Känna till och kunna använda grundläggande metoder på samlingar.
- ☐ Kunna skapa och använda både oföränderliga och föränderliga mängder.
- ☐ Förstå skillnader och likheter mellan en mängd och en sekvens.
- ☐ Kunna beskriva hur algoritmen linjärsökning fungerar.
- ☐ Kunna skapa och använda både oföränderliga och föränderliga nyckel-värde-tabeller.
- ☐ Kunna använda nyckel-värde-tabeller för att implementera registrering.
- ☐ Förstå likheter och skillnader mellan en nyckel-värde-tabell och en sekvens.
- ☐ Kunna spara och läsa data till/från textfiler på disk.

Förberedelser

- ☐ Studera begreppen i kapitel 9

9.2.1 Grunduppgifter; förberedelse inför laboration

Uppgift 1. *Para ihop begrepp med beskrivning.*

Koppla varje begrepp med den (förenklade) beskrivning som passar bäst:

mängd	1	A	leta i sekvens tills sökkriteriet är uppfyllt
nyckel-värde-tabell	2	B	avkoda symbolsekvens och återskapa objekt i minnet
mappning	3	C	en unik identifierare
nyckel	4	D	egenskapen att finnas kvar efter programmets avslut
persistens	5	E	koda objekt till avkodningsbar sekvens av symboler
serialisera	6	F	oordnad samling av mappningar med unika nycklar
de-serialisera	7	G	`nyckel -> värde`
linjärsöka	8	H	oordnad samling med unika element

Uppgift 2. *Vad är en mängd?* Förklara vad som händer nedan. Varför hamnar elementen i en ”konstig” ordning? Varför ”försvinner” det element?

```
scala> val xs = Vector(1,2,3,1,2,3,4,5,7).toSet
xs: scala.collection.immutable.Set[Int] = Set(5, 1, 2, 7, 3, 4)
scala> xs.foreach(print)
512734
```

Uppgift 3. *Använda mängder.*

Para ihop varje uttryck till vänster med ett uttryck till höger som har samma värde:

`Set(1, 2) ++ Set(1, 2)`	1	A	`3`
`(1 to 3).toSet`	2	B	`Set(3)`
`Vector.fill(3)(1).toSet`	3	C	`6`
`Set(1, 2, 3) diff Set(1, 2)`	4	D	`error: ...`
`(1 to 7).toSet.apply(8)`	5	E	`true`
`Set(1, 2, 3).sorted`	6	F	`Set(1, 2) - 2`
`Set(2,4) subsetOf (1 to 7).toSet`	7	G	`Set(1) + 2 + 3`
`Set(1, -1, 2, -2).map(_.abs).sum`	8	H	`false`
`Set(1, 1, 1, 1, 1, 5).sum`	9	I	`Set(1, 2)`

Uppgift 4. *Räkna unika ord med hjälp av en mängd.* På veckans laboration ska vi göra automatisk språkbehandling av långa texter som vi delar upp i ord. Med metoden `s.split(' ').toVector` kan du dela upp en sträng `s` i en sekvens av ord, där `s` blivit uppdelad i många strängar vid varje blanktecken och alla blanktecken är borttagna.

a) Använd metoderna `split` och `toSet` för skapa ett uttryck som beräknar hur många unika ord det finns i strängen `hej` nedan:

```
scala> val hej = "hej hej hemskt mycket hej"
```

b) Mängder är snabba på att kolla om ett element finns i mängden men du kan inte förvänta dig att elementen finns i någon viss ordning. Det finns en sekvenssamlingsmetod som skapar en sekvens med unika element ur en sekvens och behåller den ursprungliga ordningen. Vad heter metoden?
Tips: Leta i snabbreferensen eller sök på nätet. Metoden fungerar på alla samlingar som är av typen `Seq` och har ett namn som börjar med bokstäverna `di`.

Uppgift 5. *Skapa 2-tupler med metoden `->` som kan uttalas ”mappas till”.* Vi har tidigare sett hur två olika värden kan samlas i en 2-tupel, till exempel `(0, true)`. Par kan även skapas med hjälp av metoden `->` enligt nedan. Testa detta i REPL:

```
1 scala> ("Skåne", "Lund")          // ett strängpar med vanlig 2-tupel
2 scala> "Skåne" -> "Lund"           // operatornotation med ->
3 scala> "Skåne".->("Lund")         // punktnotation med -> (inte alls vanligt)
```

Metoden `->` fungerar med alla typer och är en fabriksmetod för par. Metodnamnet liknar en högerpil och illustrerar en mappning från första till andra värdet.

a) Fungerar det på par skapade med `->` att använda metoderna `_1` och `_2`?

b) Deklarera en variabel `val huvudstad: Vector[(String, String)]` som innehåller mappningar mellan geografiska områden och deras huvudstäder enligt tabellen nedan.

Sverige	Stockholm
Danmark	Köpenhamn
Grönland	Nuuk
Skåne	Lund

c) Skriv ett uttryck som plockar fram `"Lund"` ur `huvudstad`.

Uppgift 6. *Linjärsöka efter nyckel i sekvens av mappningar.*

a) Implementera funktionen `lookupIndex` nedan med hjälp av samlingsmetoden `indexWhere` så att linjärsökning sker efter index för ett par i sekvensen där `key` finns på första platsen i paret.

```
def lookupIndex(xs: Vector[(String, String)])(key: String): Int = ???
```

b) Testa din funktion i REPL genom att slå upp index för Skånes huvudstad i sekvensen `huvudstad` från föregående uppgift.

Uppgift 7. *Nyckel-värde-tabell.* En nyckel-värde-tabell är en smart datastruktur som gör att du kan slå upp det värde som en nyckel mappar till *utan* att linjärsökning behöver ske. Värdet plockas fram direkt på en konstant tid, d.v.s. tiden att slå upp ett värde beror *inte* på antalet element i samlingen, utan sker med mycket liten fördröjning.

I Scala heter nyckelvärdetabeller `Map` med stort M och är praktiska att använda i många olika sammanhang. `Map` finns i både en oföränderlig och en förändringsbar variant. Det går med metoder på formen `toXXX` lätt att omvandla mellan en `Map` och en sekvens av par av typen `XXX[(Nyckeltyp, Värdetyp)]`.

a) Deklarera mappen `telnr` nedan i REPL och använd `apply` för att ta reda på telefonnumret till Fröken Ur.

b) Vad har `telnr` för typ?

c) Vad har `telnr.toVector` för typ?

```
val telnr = Map(
  "Anna"     -> 46462229812L,
  "Björn"     -> 46462229009L,
  "Sandra"    -> 46462220368L,
  "Fröken Ur" -> 4690510L,
)
```

En uppsättning `Map`-instanser, vid behov nästlade, kan med fördel användas för att bygga upp en i-minnet-databas där inbyggda samlingsmetoder, t.ex. `map`, `filter`, och `for`-`yield`-uttryck, ger flexibla och effektiva sökmöjligheter. På veckans laboration ska du göra detta.

Samlingen `Map` är en generalisering av en sekvens, där man kan ”indexera”, inte bara med ett heltal, utan med vilken typ av värde som helst, t.ex. en sträng. Datastrukturen `Map` kallas också *associativ array*[1] och är implementerad som en s.k. *hashtabell*[2], men du får vänta till fördjupningskursen innan vi går igenom hur en sådan datastruktur implementeras.

Uppgift 8. *Använda nyckel-värdetabell.*

a) Skapa nedan variabler i REPL.

```
val follow = for i <- 2 to 16 by 2 yield (i, i + 1)
val xs = follow.toMap
val ys = xs.toVector
```

[1] https://en.wikipedia.org/wiki/Associative_array
[2] https://en.wikipedia.org/wiki/Hash_table

Hamnar mappningarna i `ys` i samma ordning som `follow`? Varför?

b) Med `xs` och `ys` deklarerade i REPL enligt ovan, para ihop yttryck till vänster med rätt resultat till höger. Om du är osäker på de sammansatta uttrycken, prova enklare uttryck i REPL och undersök värde och typ hos delresultat.

`xs(2) + xs(4)`	1	A	`8`
`ys(0)`	2	B	`7`
`xs(0)`	3	C	`(10, 11)`
`(xs + (0 -> 1)).apply(0)`	4	D	`1`
`xs.keySet.apply(2)`	5	E	`(16, 17)`
`xs isDefinedAt 0`	6	F	`-9`
`xs.getOrElse(0, 7)`	7	G	`true`
`xs.maxBy(_._2)`	8	H	`false`
`xs.map(p => p._1 -> -p._2)(8)`	9	I	`NoSuchElementException`

Uppgift 9. *Registrering i förändringsbar nyckel-värde-tabell.* I denna uppgift ska du implementera en hjälpklass för registrering i en frekvenstabell som du sedan ska använda på veckans laboration. Klassen ska heta `FreqMapBuilder` som efter upprepade anrop av metoden `add(s: String): Unit` kan skapa frekvenstabeller av typen `Map[String, Int]`, där nyckel-värde-paren i tabellen anger antalet förekomster av en viss sträng. Du ska utgå från koden nedan.

Klassen använder en förändringsbar tabell internt. Efter att man har lagt till många strängar kan man med metoden `toMap` få en oföränderlig tabell för uppslagning av frekvenser för specifika strängar. Läs i snabbreferensen om vilka extra metoder för uppdatering som erbjuds av `mutable.Map[K, V]`.

```
class FreqMapBuilder:
  private val register = collection.mutable.Map.empty[String, Int]
  def toMap: Map[String, Int] = register.toMap
  def add(s: String): Unit = ???

object FreqMapBuilder:
  def apply(xs: String*): FreqMapBuilder = ???
```

Implementera och testa `FreqMapBuilder`. *Tips:* Du kan t.ex. använda `mutable.Map`-metoderna `addOne` och `getOrElse`.

Uppgift 10. *Metoden `sliding`.* I veckans laboration kommer du att ha nytta av metoden `sliding`, som ger en iterator för speciella delsekvenser av en sekvens, vilka kan liknas vid ”utsikten” i ett ”glidande fönster”.

a) Kör nedan i REPL och beskriv vad som händer.

```
1 scala> val xs = Vector("fem", "gurkor", "är", "fler", "än", "fyra", "tomater")
2 scala> xs.sliding(2).toVector
3 scala> xs.sliding(3).toVector
4 scala> xs.sliding(10).toVector
```

b) Använd `xs.sliding(2)` och omvandla varje element i resultatet till ett par. Gör sedan om sekvensen av par till en nyckel-värde-tabell. Vad kan tabellen användas till?

Uppgift 11. *Läsa text från fil och webbservrar.* På laborationen ska du bygga upp tabeller från data i textformat. Då har du nytta av att kunna läsa text från filer och från webben. Testa detta i REPL:

```
scala> val url = "https://fileadmin.cs.lth.se/pgk/europa.txt"
scala> val xs = io.Source.fromURL(url, "UTF-8").getLines.toVector
scala> val data = xs.map(_.split(';').toVector)
scala> data.head
scala> data.foreach(println)
```

a) Skapa dessa tabeller ur sekvensen `data`:

```
val populationOf: Map[String, Int]    = ???  // länders invånarantal
val sizeOf:       Map[String, Int]    = ???  // länders yta i km^2
val capitalOf:    Map[String, String] = ???  // länders huvudstäder
```

Testa tabellerna i REPL.

b) Spara ner data i en textfil `europa.txt`. Läsa in data från filen med metoden `Source.fromFile(filnamn, teckenkodning)` på liknande sätt som med `fromURL` ovan. Om du kör i en Linux-terminal kan du enkelt ladda ner en fil så här (notera att det är stora bokstaven O och inte en nolla i optionen `-sLO`):

```
> curl -sLO https://fileadmin.cs.lth.se/pgk/europa.txt
```

Skriv ut alla raderna i `europa.txt` med hjälp av `Source.fromFile` i REPL.

9.2.2 Extrauppgifter; träna mer

Uppgift 12. *Skapa ett textspel med hjälp av tabeller.* Gör ett enkelt spel för att träna på olika fakta om Europas länder och huvudstäder genom att läsa data från URL:en: https://fileadmin.cs.lth.se/pgk/europa.txt
Där finns text kodad i UTF-8 med följande innehåll (endast de första raderna visas):

```
Land;Invånarantal;Storlek(km^2);Huvudstad
Albanien;3581655;28748;Tirana
Andorra;71201;468;Andorra la Vella
Belgien;10584534;30528;Bryssel
Bosnien-Hercegovina;4590310;51129;Sarajevo
Bulgarien;7385367;110910;Sofia
Cypern;854000;9250;Nicosia
Danmark;5475791;43094;Köpenhamn
Estland;1324333;45226;Tallinn
Finland;5315280;338145;Helsingfors
Frankrike;61538322;551695;Paris
Färöarna;48344;139574;Torshamn
Grekland;10964021;131940;Aten
// ... etcetera för alla Europas länder.
```

Låt till exempel användaren svara på slumpvisa frågor av typen:

- Har Andorra fler invånare än Cypern?
- Vad heter huvudstaden i Bulgarien?
- Har Danmark större yta än Finland?

Använd oföränderliga tabeller med lämpliga nycklar och värden. Du kan använda en mängd med länder/huvudstäder som användaren hittills svarat rätt på för att kunna förhindra att dessa återkommer igen.

9.2.3 Fördjupningsuppgifter; utmaningar

Uppgift 13. *Registrering med `groupBy`.* Vi ska nu utnyttja ett riktigt listigt trick för att via en enda kodrad implementera registrering med hjälp av samlingsmetoderna `groupBy` och `map`.

a) Läs om metoden `groupBy` i snabbreferensen. Du hittar den under rubriken *"Methods in trait `Iterable[A]`"* eftersom `groupBy` fungerar på alla samlingar. Testa `groupBy` enligt nedan och beskriv vad som händer.

```
scala> val xs = Vector(1, 1, 2, 2, 4, 4, 4).groupBy(x => x > 2)
scala> val ys = Vector(1, 1, 2, 2, 4, 4, 4).groupBy(x => x)
```

b) Skapa en funktion `freq` med nedan funktionshuvud som returnerar en tabell med antalet förekomster av olika heltal i `xs`. Testa `freq` på en sekvens av 1000 slumpvisa tärningskast och förklara hur funktionen `freq` fungerar. *Tips:* Gör först `groupBy(???)` och sedan `map(???)`.

```
def freq(xs: Vector[Int]): Map[Int, Int] = ???

def kasta(n: Int): Vector[Int] =
  Vector.fill(n)(scala.util.Random.nextInt(6) + 1)
```

Uppgift 14. *Skriva till fil.* Som hjälp när du skapar egna intressanta applikationer eller bygger vidare på kursens laborationer och övningar med frivilliga extrauppgifter, kan du använda funktionerna i singelobjektet `IO` nedan, som finns i kursens scala-bibliotek introprog.[3]

IO-modulen använder `scala.io.Source` för att serialisera och de-serialisera strängar till och från vanliga textfiler. IO-modulen använder även paketet `java.io` för att erbjuda funktioner som gör det enkelt att serialisera/de-serialisera godtyckliga objekt skapade med hjälp av serialserbara klasser till/från binärfiler. Case-klasser i Scala blir automatiskt serialiserbara.

I implementationen av `IO` används `try ... finally` för att säkerställa att filer inte lämnas öppnade även om något går fel under den läs/skriv-process som sköts av det underliggande operativsystemet.

a) Kompilera och resta nedan med `introprog` på classpath, t.ex. med hjälp av `sbt`.

```
import introprog.IO

case class Player(name: String)

@main def run(): Unit =
  println("Test of output/input objects to/from disk:")
  val highscores = Map(Player("Sandra") -> 42, Player("Björn") -> 5)
  IO.saveObject(highscores,"highscores.ser")
```

[3] Källkoden finns här och även på sidan **??**:
https://github.com/lunduniversity/introprog-scalalib/blob/master/src/main/scala/introprog/IO.scala

```
val highscores2 = IO.loadObject[Map[Player, Int]]("highscores.ser")
val isSameContents = highscores2 == highscores
val testResult = if (isSameContents) "SUCCESS :)" else "FAILURE :("
println(testResult)
```

b) Använd IO-modulen för att spara användarens poängresultat i ditt spel om Europas länder och städer, i extrauppgiften ovan. Implementationen av `introprog.IO` finns här: https://github.com/lunduniversity/introprog-scalalib/blob/master/src/main/scala/introprog/IO.scala

9.3 Laboration: `words`

Mål

- ☐ Kunna skapa och använda nyckel-värde-tabeller med samlingstypen `Map`.
- ☐ Kunna skapa och använda mängder med samlingstypen `Set`.
- ☐ Förstå skillnader och likheter mellan en sekvens och en mängd.
- ☐ Förstå likheter och skillnader mellan en sekvens av par och en nyckel-värde-tabell.
- ☐ Kunna implementera algoritmer som använder nästlade strukturer.

Förberedelser

- ☐ Gör övning `lookup` i avsnitt 9.2
- ☐ Läs igenom hela laborationen.
- ☐ Hämta och läs given kod via kursen github-plats eller via cs.lth.se/pgk/download

9.3.1 Bakgrund

Denna uppgift handlar om analys av naurligt språk (eng. *Natural Language Processing, NLP*). Språkanalys bygger ofta på statistik över förekomsten av olika ord i långa texter. Du ska skriva kod, som utifrån en lång text, till exempel en bok, kan hjälpa dig att svara på denna typ av frågor:

- Hur vanligt är ett visst ord i en given text?
- Vilket är det vanligaste ordet som följer efter ett visst ord?
- Hur kan man generera ordsekvenser som liknar ordföljden i en given text?

För att kunna svara på sådana frågor ska du skapa frekvenstabeller och även så kallade *n-gram*; sekvenser av n ord som förekommer i följd i en text. Exempel på några 2-gram (kallas även *bigram*) som finns i föregående mening: (för, att), (att, kunna), (kunna, svara), (svara, på), (på, sådana), och så vidare.[4]

9.3.2 Obligatoriska uppgifter

Du ska bygga ditt program med en editor, t.ex. VS `code`, och kompilera och köra din kod i terminalen med hjälp av `scala-cli` i *watch mode* med det arbetssätt som beskrivs i appendix F avsnitt F.2.1. Medan du steg för steg utvecklar ditt program, ska du parallellt göra experiment i REPL för att undersöka hur du kan använda samlingsmetoder för att lösa uppgifterna. Kod att utgå ifrån finns här: `https://github.com/lunduniversity/intropro g/tree/master/workspace/w09_words`

Dessa ofärdiga kodfiler ligger i paketet `nlp`:

- `FreqMapBuilder.scala` innehåller ett skelett till en klass för att, ord för ord, bygga en nyckel-värde-tabell som registrerar antalet förekomster av olika ord. Att implementera denna ingick i övningen du gjorde tidigare i veckan.
- `Text.scala` innehåller ett skelett till en klass som kan göra textbehandling genom att analysera ord i en text.
- `Main.scala` innehåller ett ofärdigt huvudprogram som du kan använda i laborationens senare del.

[4] Du kan undersöka olika n-gram i en stor mängd böcker med hjälp av Googles n-gram-viewer: `https://books.google.com/ngrams/`

Uppgift 1. *Skapa frekvenstabeller*. Du ska använda `FreqMapBuilder` från veckans övning för att skapa frekvenstabeller av typen `Map[String, Int]`, där nyckel-värde-paren i tabellen anger antalet förekomster av en viss sträng.

a) Lägg klassen `FreqMapBuilder` i ett paket som heter `nlp` och kompilera.

```
1  package nlp
2
3  class FreqMapBuilder:
4    private val register = collection.mutable.Map.empty[String, Int]
5    def toMap: Map[String, Int] = register.toMap
6    def add(s: String): Unit = ???
7
8  object FreqMapBuilder:
9    /** Skapa ny FreqMapBuilder och räkna strängarna i xs */
10   def apply(xs: String*): FreqMapBuilder = ???
```

b) Testa noga så att din `FreqMapBuilder` fungerar korrekt. Exempel på test i REPL:

```
1  scala> import nlp._
2
3  scala> val fmb = FreqMapBuilder("hej", "på", "dej")
4  fmb: nlp.FreqMapBuilder = nlp.FreqMapBuilder@458f85ef
5
6  scala> fmb.add("hej")
7
8  scala> fmb.toMap
9  res0: Map[String,Int] = Map(på -> 1, hej -> 2, dej -> 1)
10
11 scala> (1 to Short.MaxValue).foreach(i => fmb.add(i.toString))
12
13 scala> fmb.toMap.size
14 res1: Int = 32770
15
16 scala> fmb.toMap
17 res2: Map[String,Int] =
18   Map(10292 -> 1, 19125 -> 1, 26985 -> 1, 29301 -> 1, 5451 -> 1, 4018 -> 1, 312
```

I kommande uppgifter ska du steg för steg skapa och testa case-klassen `Text`.

```scala
package nlp

case class Text(source: String):
  lazy val words: Vector[String] = ???  // dela upp source i ord

  lazy val distinct: Vector[String] = words.distinct

  lazy val wordSet: Set[String] = words.toSet

  lazy val wordsOfLength: Map[Int, Set[String]] = wordSet.groupBy(_.length)

  lazy val wordFreq: Map[String, Int] = ???  // använd FreqMapBuilder

  def ngrams(n: Int): Vector[Vector[String]] = ???  // använd sliding

  lazy val bigrams: Vector[(String, String)] =
    ngrams(2).map(xs => (xs(0), xs(1)))

  lazy val followFreq: Map[String, Map[String, Int]] = ??? //nästlad tabell

  lazy val follows: Map[String, String] =
    followFreq.map( (key, followMap) =>
      val maxByFreq: (String, Int) = followMap.maxBy(_._2)
      val mostCommonFollower: String = maxByFreq._1
      (key, mostCommonFollower)
    )
    //eller kortare med samma resultat: (lättare eller svårare att läsa?)
    //  followFreq.map((k, v) => k -> v.maxBy(_._2)._1)

object Text:
  def fromFile(fileName: String, encoding: String = "UTF-8"): Text =
    val source = scala.io.Source.fromFile(fileName, encoding)
    val txt = try source.mkString finally source.close()
    Text(txt)

  def fromURL(url: String, encoding: String = "UTF-8"): Text =
    val source = scala.io.Source.fromURL(url, encoding)
    val txt = try source.mkString finally source.close()
    Text(txt)
```

Uppgift 2. *Dela upp en sträng i ord*. Du ska implementera medlemmen `words`. Den ska innehålla en vektor med alla ord i `source`, utan andra tecken än bokstäver. Detta åstadkommer du genom att utgå ifrån strängen `source` och i tur och ordning göra följande:

1. byta ut alla tecken i `source` för vilka `isLetter` är falskt mot `' '`
2. dela upp strängen från föregående steg i en array av strängar med `split(' ')`
3. filtrera bort alla tomma strängar
4. gör om alla bokstäver i alla strängar till små bokstäver
5. gör om arrayen till en sekvens av typen `Vector[String]`.

Testa så att `words`, och de värden som använder `words`, fungerar i REPL:

```
scala> val t = Text("Gurka är ingen tomat, men gurka är en grönsak.")

scala> t.words
res1: Vector[String] =
  Vector(gurka, är, ingen, tomat, men, gurka, är, en, grönsak)

scala> t.distinct
res2: Vector[String] =
  Vector(gurka, är, ingen, tomat, men, en, grönsak)

scala> t.wordSet
res3: Set[String] = Set(grönsak, är, gurka, men, ingen, tomat, en)

scala> t.wordsOfLength(5)
res4: Set[String] = Set(gurka, ingen, tomat)
```

Uppgift 3. Du ska nu skapa ordfrekvenstabellen `wordFreq` genom att registrera ordförekomster med hjälp av `FreqMapBuilder`. Tabellen `wordFreq` ska bestå av nyckel-värdepar `w -> f` där `f` är antalet gånger ordet `w` förekommer i `words`. Testa `wordFreq` genom att ladda ner boken ”Skattkammarön” skriven av Robert Louis Stevenson[5] och undersök frekvensen för olika vanliga ord. Vilket ord är vanligast? Näst vanligast?

```
scala> val piratbok = Text.fromURL("https://fileadmin.cs.lth.se/pgk/skattkammaron.txt")
val piratbok: nlp.Text = Text(Herr Trelawney, doktor Livesey och de övriga herrarna har

scala> piratbok.words.size
val res0: Int = 69438

scala> piratbok.wordFreq("pirat")
val res1: Int = 7
```

Länkar till böcker i UTF-8-format som du kan använda i dina tester:

- ”Skattkammarön” av R. L. Stevenson:
  `https://fileadmin.cs.lth.se/pgk/skattkammaron.txt`
- ”Inferno” av August Stringberg:
  `https://fileadmin.cs.lth.se/pgk/inferno.txt`
- ”Pride and Prejudice” av Jane Austen:
  `https://fileadmin.cs.lth.se/pgk/prideandprejudice.txt`
- Projekt Gutenberg med många fritt tillgängliga böcker i textformat:
  `https://www.gutenberg.org/`

Uppgift 4. Implementera metoden `ngrams` som ger en sekvens med alla ordföljder i n steg. *Tips:* På veckans övning ingick att undersöka hur metoden `sliding` fungerar, med vilken du kan skapa n-gram. Gör `toVector` på resultatet från `sliding`. Testa noga så att `ngrams` och `bigrams` fungerar korrekt innan du går vidare.

```
scala> piratbok.ngrams(3).take(2)
val res1: Vector[Vector[String]] =
```

[5] Copyright för denna bok har gått ut, så du gör dig inte skyldig till piratkopiering (i juridisk mening).

```
  Vector(Vector(herr, trelawney, doktor), Vector(trelawney, doktor, livesey))

scala> piratbok.bigrams.take(2)
val res2: Vector[(String, String)] =
  Vector((herr,trelawney), (trelawney,doktor))
```

Uppgift 5. Implementera `followFreq`, som ska innehålla en nyckel-värde-tabell där värdet i sin tur är en frekvenstabell över de ord som kommer efter nyckeln.

Genom att analysera alla ordpar kan vi få fram vilket som är det vanligaste ordet som följer efter ett givet ord. Metoden `bigrams` ger oss alla ordpar `(w1, w2)` där `w2` följer efter `w1`. Vi kan spara statistiken över efterföljande ord i en nyckel-värdetabell med mappningarna `w -> f` där nyckeln `w` är ett ord och värdet `f` är en frekvenstabell av typen `Map[String, Int]`. I frekvenstabellen lagrar vi frekvensen för alla de ord som följer efter `w`. Du ska alltså bygga en nästlad tabell av typen `Map[String, Map[String, Int]]`. Rita en bild av den nästlade strukturen. ✎

Implementera metoden followFreq genom att utgå från nedan pseudokod:

```
val result = collection.mutable.Map.empty[String, FreqMapBuilder]
for (key, next) <- bigrams do
  if /* key finns redan definierad i result */ then
    /* på "platsen" result(key): lägg till next i frekvenstabellen */
  else
    /* lägg till (key -> ny frekvenstabell med next) i result*/
end for
result.map(p => p._1 -> p._2.toMap).toMap // toMap ger oföränderlig Map
```

Gör utskrifter för att ta reda på följande frågor. Skriv ner svaren och var redo att redovisa dem i samband med kontrollfrågorna (se avsnitt 9.3.3). ✎

a) Vilka ord brukar följa efter *han* respektive *hon* i Stevensons ”Skattkammarön”?

b) Vilka ord brukar följa efter *han* respektive *hon* i Stringbergs ”Inferno”?

c) Vilka ord brukar följa efter *he* respektive *she* i Austens ”Pride and Prejudice”?

Uppgift 6. Skapa ett huvudprogram som rapporterar valfria, intressanta mått om orden i en text. Programmet ska ta textens källa som argument, givet som en URL eller ett filnamn. Skriv huvudprogrammet i filen `Main.scala` i ett singelobjekt med namnet `Main`. Exempel på en rapport som ditt huvudprogram kan generera finns nedan. Här ges även ett heltal som argument som styr topplistornas längd.

```
> scala run . -- https://fileadmin.cs.lth.se/pgk/skattkammaron.txt 13

Källa: https://fileadmin.cs.lth.se/pgk/skattkammaron.txt

*** Antal ord: 69438

*** De 13 vanligaste orden och deras frekvens:
(och,3089), (jag,2007), (att,1594), (det,1382), (en,1262),
(i,1244), (som,1132), (på,1068), (han,1063), (var,990),
(med,854), (den,774), (av,740)

*** De 13 längsta orden och deras längd:
(besättningsmedlemmarnas,23), (befästningsanordningar,22),
(temperamentsuppvisning,22), (undsättningsexpedition,22),
```

```
(besättningsmedlemmarna,22), (försiktighetsåtgärder,21),
(undsättningsfartyget,20), (sjukdomsframkallande,20),
(husföreståndarinnans,20), (sjömansterminologin,19),
(parlamentärsflaggan,19), (bregravningsplatsen,19),
(tidvattenströmmarna,19)
```

Exempel på huvudprogram som kan skapa ovan utskrift:

```
package nlp

object Main:
  val defaultUrl = "https://fileadmin.cs.lth.se/pgk/skattkammaron.txt"
  val defaultN = 10

  def top(n: Int, freqMap: Map[String, Int]): Vector[(String, Int)] = ???

  def report(text: Text, from: String, n: Int): String =
    val longestWordsWithLength =
      top(n, text.distinct.map(w => (w, w.length)).toMap).mkString(", ")
    s"""
    |Källa: $from
    |
    |*** Antal ord: ${text.words.size}
    |
    |*** De $n vanligaste orden och deras frekvens:
    |${top(n, text.wordFreq).mkString(", ")}
    |
    |*** De $n längsta orden och deras längd:
    |$longestWordsWithLength
    """.stripMargin

  def main(args: Array[String]): Unit =
    val location = if args.isEmpty then defaultUrl else args(0)
    val n = if args.length < 2 then defaultN else args(1).toInt
    val text =
      if location.startsWith("http") then Text.fromURL(location)
      else Text.fromFile(location)

    println(report(text, location, n))
```

Uppgift 7. Para ihop dig med en annan student och planera hur ni tillsammans kan med hjälp av `https://cs.lth.se/pgk/muntabot` kan träna inför det muntliga provet där ni ömsesidigt agerar ”låtsasexaminator”. Gör en plan för när ni ska testa varandra på vilka veckor. Visa er plan för handledare och diskutera vad det innebär att vara en bra ”låtsasexaminator”.

9.3.3 Kontrollfrågor

1. Vilket är dina svar på uppgift 5 a) b) c) på sidan 64?
2. I vilken ordning hamnar elementen om man anropar `distinct` på en sekvens?
3. Om man itererar över en mängd, i vilken ordning behandlas elementen?
4. Ge exempel på när är det lämpligt att använda en mängd i stället för en sekvens av distinkta värden?
5. Är alla nycklar i en nyckel-värde-tabell garanterat unika?

6. Är alla värden i en nyckel-värde-tabell garanterat unika?
7. LTH-teknologen Oddput Clementin vill summera längden på varje sträng i en mängd och skriver:

```
scala> Set("hej", "på", "dej").map(_.length).sum
res0: Int = 5
```

Varför blir det fel? Hur kan Oddput åtgärda problemet?

9.3.4 Frivilliga uppgifter

Uppgift 8. Bygg vidare på klassen `Text` och implementera nedan metod som ska ge ett slumpmässigt ord ur `wordSet`. Varje ord ska förekomma med lika stor sannolikhet.

```
def randomWord: String = ???
```

Uppgift 9. Med NLP kan man generera slumpmässiga meningar som statistiskt sett liknar ”riktiga”, människoskapade meningar.

Implementera metoden `randomSeq(firstWord, n)` nedan i klassen `Text`. Den ska ge en sekvens $w_1, w_2, ..., w_n$ där w_1 är `firstWord` och w_{i+1} är något slumpmässigt ord som är draget bland de ord som följer efter w_i. Detta kan du åstadkomma genom att varje efterföljande ord w_{i+1} väljs ur `keys.toVector` för den `followFreq`-tabell som hör till w_i. Orden ska dras ur efterföljandemängden, med lika stor sannolikhet.

```
def randomSeq(firstWord: String, n: Int): Vector[String] = ???
```

Uppgift 10. För att dina datorgenererade meningar verkligen ska likna mänskligt språk kan vi skapa de mest sannolika meningarna av olika längder ur vår analys av ordfrekvenser.

Lägg till metoden `mostCommonSeq` i klassen `Text` enligt nedan:

```
def mostCommonSeq(firstWord: String, n: Int): Vector[String] = ???
```

a) Implementera metoden så att resultatet blir en sekvens med `n` ord. Sekvensen ska börja med `firstWord` och därefter följas av det ord som är det *vanligaste* efterföljande ordet efter `firstWord`, och därpå det vanligaste efterföljande ordet efter det, etc. *Tips:* Använd en lokal variabel `val result` som är en ArrayBuffer till vilken du i en `while`-loop lägger de efterföljande orden.

b) Jämför de slumpmässiga sekvenserna med sekvenser genererade med `randomSeq` i uppgift 9. Vilka sekvenser liknar mest ”riktiga” meningar?

Uppgift 11. Använd befintliga samlingsmetoder i stället för `FreqMapBuilder` för att registrera efterföljande ord.

a) Undersök i REPL hur metoden `groupBy(x => x)` fungerar då den appliceras på en samling med strängar. Sök efter och studera dokumentationen för `groupBy`.

b) Inför värdet `lazy val wordFreq2`. Den ska ge samma resultat som `wordFreq` men men implementeras med hjälp av `groupBy` och `map` i stället för `FreqMapBuilder`.

c) Jämför prestanda mellan `wordFreq2` och `wordFreq`. Vilken är snabbast för stora texter? Är skillnaden stor? ★

d) Inför värdet `lazy val followsFreq2`. Den ska ge samma resultat som `followsFreq` men implementeras med hjälp av `groupBy` och `map` i stället för `FreqMapBuilder`. Denna uppgift är ganska knepig. Experimentera dig fram i REPL, och bygg upp en lösning steg för steg. *Tips:*

```
bigrams
  .groupBy(???)
  .map(p => p._1 -> p._2.map(???).groupBy(???).map(???))
```

★ e) Jämför prestanda mellan `followsFreq2` och `followsFreq`. Vilken är snabbast för stora texter? Är skillnaden stor?

Uppgift 12. *Gör `FreqMapBuilder` generisk.* Generiska strukturer, alltså sådana som har typparametrar, är ofta väsentligt mycket mer användbara. Om du gör `FreqMapBuilder` generisk genom att införa en typparameter i stället för att hårdkoda typen till `String` så kan du använda `FreqMapBuilder` med godtycklig elementtyp.

a) Studera `FreqMapBuilder` och identifiera allt i den klassen som är specifikt för typen `String`.

b) Inför en typparameter `A` inom hakparenteser efter klassnamnet och använd sedan `A` i stället för `String` i alla metoder.

c) Testa så att din generiska frekvenstabellbyggare fungerar på sekvenser som innehåller annat än strängar.

Detta funkar eftersom inget i `FreqMapBuilder` egentligen förutsätter att elementen som ska räknas är av sträng-typ (det räcker att det finns en vettig `equals` och `hashcode`).

Kapitel 10

Arv och komposition

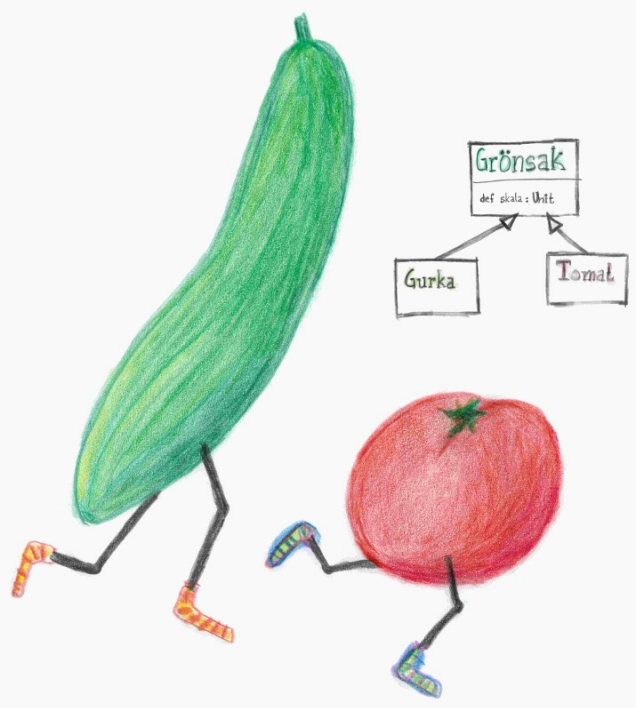
Begrepp som ingår i denna veckas studier:

- ☐ arv
- ☐ komposition
- ☐ polymorfism
- ☐ trait
- ☐ extends
- ☐ asInstanceOf
- ☐ with
- ☐ inmixning supertyp
- ☐ subtyp
- ☐ bastyp
- ☐ override
- ☐ Scalas typhierarki
- ☐ Any
- ☐ AnyRef
- ☐ Object
- ☐ AnyVal
- ☐ Null
- ☐ Nothing
- ☐ topptyp
- ☐ bottentyp
- ☐ referenstyper
- ☐ värdetyper
- ☐ accessregler vid arv
- ☐ protected
- ☐ final
- ☐ trait
- ☐ abstrakt klass

10.1 Teori

10.1.1 Vad är arv?

Arv (eng. *inheritance*) beskriver relationen
X **är en** *Y*

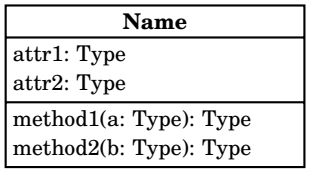


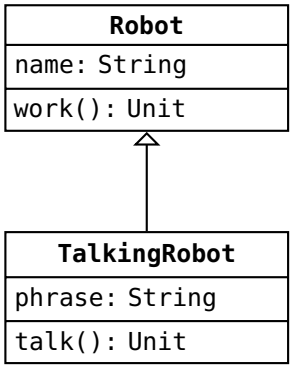
10.1.2 Varför behövs arv?

- Man kan använda arv för att dela upp kod i:
  - **generella** (gemensamma) delar och
  - **specifika** (specialanpassade) delar.
- Man kan åstadkomma **kontrollerad flexibilitet**:
  - Klientkod kan **utvidga** (eng. *extend*) ett givet API med egna specifika tillägg.
- Man kan använda arv för att deklarera en gemensam **bastyp** så att variabler och generiska samlingar kan ges en lagom specifik elementtyp.
  - Det räcker att man vet bastypen för att kunna nå gemensamma medlemmar för alla element i samlingen.
  - Exempel: Alla grönsaker har en vikt.

10.1.3 Klassdiagram med UML (Unified Modeling Language)

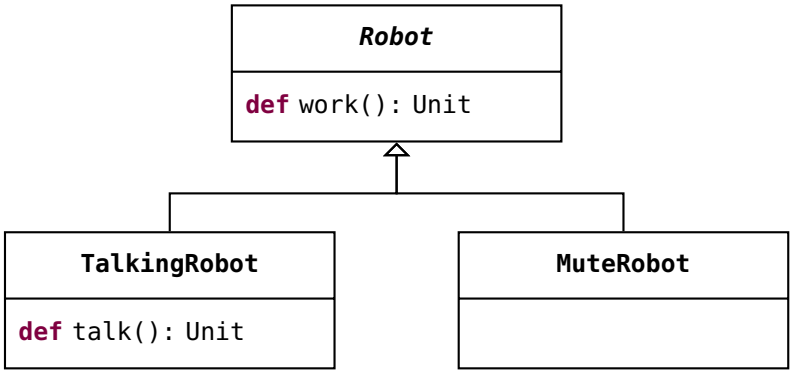
UML Klassdiagram:





Mer om UML-diagram i senare kurser.
en.wikipedia.org/wiki/Class_diagram

10.1.4 Exempel: Robot som bastyp för två subtyper

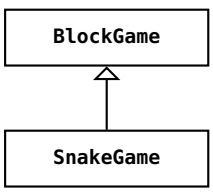


(Ibland utelämnar man attribut och/eller metoder i ett UML-diagram för att minska antalet detaljer i modellen och ge en överblick.)

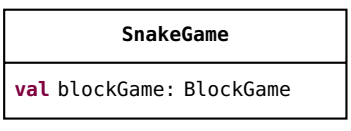
10.1.5 Alternativ till arv: komposition

- Som alternativ till att klassen X ärver klassen Y kan man i stället använda **komposition** (eng. *composition*), som innebär att klassen X **har ett attribut** som **refererar** till klassen Y.

Exempel på arv i snake-labben:
SnakeGame **är** ett BlockGame



Ett alternativ vore **komposition**:
SnakeGame **har** ett BlockGame



- Komposition där **X har en Y** är ofta ett bättre alternativ än arv om:
  1. det *inte* finns en tydlig **X-är-en-Y**-relation
  2. det *inte* behövs en **gemensam bastyp**
  3. det *inte* är önskvärt ärva och exponera *alla* Y:s medlemmar via X

10.1.6 Exempel på komposition i snake-labben

En `Player` **har en** [1] `snake`.

Player
`val snake: Snake`

```
class Player(val snake: Snake)
```

10.1.7 Behovet av gemensam bastyp

```
scala> case class Gurka(vikt: Int)

scala> case class Tomat(vikt: Int)

scala> val gurkor = Vector(Gurka(200), Gurka(300))
val gurkor: Vector[Gurka] = Vector(Gurka(200), Gurka(300))

scala> gurkor.map(_.vikt)
res0: Vector[Int] = Vector(200, 300)

scala> val grönsaker = Vector(Gurka(200), Tomat(42))
val grönsaker: Vector[Gurka | Tomat] = Vector(Gurka(200), Tomat(42))

scala> grönsaker.map(_.vikt)
-- [E008] Not Found Error: -------------------------
1 |grönsaker.map(_.vikt)
  |              ^^^^^^
  |              value vikt is not a member of Gurka | Tomat
```

Hur får vi detta att fungera som vi vill?
→ Skapa en bastyp **bastyp** med gemensamt attribut `vikt`!

10.1.8 Varför syns inte gemensam medlem i en typunion?

```
scala> val grönsaker = Vector(Gurka(200), Tomat(42))
val grönsaker: Vector[Gurka | Tomat] = Vector(Gurka@15f11bfb,Tomat@16a499d1)
```

[1] Läs mer om komposition och de relaterade begreppen aggregering, association, ägarskap etc. här: https://en.wikipedia.org/wiki/Object_composition

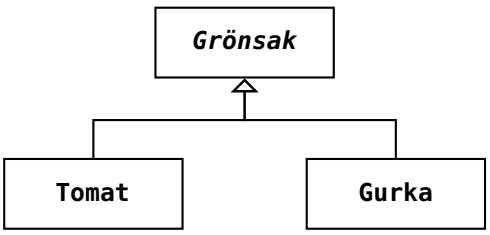
```
scala> grönsaker.map(_.vikt)
-- [E008] Not Found Error: ------------------------
1 |grönsaker.map(_.vikt)
  |              ^^^^^^
  |              value vikt is not a member of Gurka | Tomat
```

- Typerna `Grönsak` och `Tomat` är **orelaterade**. (`AnyRef` saknar `vikt`)
- En medlem som råkar ha samma namn har inte alltid samma innebörd.
- I Scala måste du explicit relatera medlemmarna genom **arv** av bastyp med gemensam medlem, eller så får du **matcha** på unionens delar.
- Du kan erbjuda en sådan matchning som en **extensionsmetod**:

```
scala> extension (gEllerT: Gurka | Tomat) def vikt = gEllerT match
         case g: Gurka => g.vikt
         case t: Tomat => t.vikt

scala> val vikter = grönsaker.map(_.vikt)
val vikter: Vector[Int] = Vector(200, 42)
```

10.1.9 Skapa en gemensam bastyp med arv

Typen ***Grönsak*** är en **bastyp** i nedan arvshierarki:



Pilen  betecknar **arv** och utläses ”**är en**”

Typerna `Tomat` och `Gurka` är **subtyper** typen `Grönsak`.
Bastyp som ej ska instansieras direkt är **abstrakt** (visas med *kursiv* stil).

10.1.10 Skapa en gemensam bastyp med `trait` och `extends`

Med en **`trait`** `Grönsak` kan klasserna `Gurka` och `Tomat` få en gemensam abstrakt **bastyp** genom att båda **subtyperna** gör **`extends`** `Grönsak`:

```
1 scala> trait Grönsak
2
3 scala> case class Gurka(vikt: Int) extends Grönsak
4
5 scala> case class Tomat(vikt: Int) extends Grönsak
```

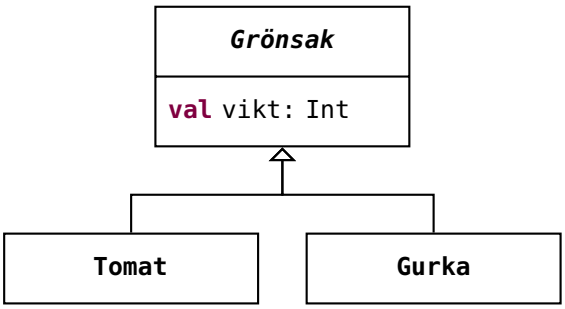
```
scala> val grönsaker = Vector(Gurka(200), Tomat(42))
val grönsaker: Vector[Grönsak] = Vector(Gurka(200), Tomat(42))
```

Men det är ännu **inte** som vi vill ha det:

```
scala> grönsaker.map(_.vikt)
-- [E008] Not Found Error: -------------------------
1 |grönsaker.map(_.vikt)
  |              ^^^^^^
  |              value vikt is not a member of Grönsak
```

10.1.11 En gemensam bastyp med gemensamma delar

Placera gemensamma medlemmar i bastypen:



- Alla grönsaker har nu attributet `val vikt`.
- Det specifika värdet på vikten definieras **inte** i bastypen.
- Medlemen `vikt` kallas **abstrakt** eftersom den **saknar implementation** och kan därför inte instansieras direkt.

10.1.12 Placera gemensamma delar i bastypen

Vi inkluderar det gemensamma attributet `val vikt` som en **abstrakt medlem** i bastypen:

```
trait Grönsak:
  val vikt: Int    // implementation saknas, inget =

case class Gurka(vikt: Int) extends Grönsak

case class Tomat(vikt: Int) extends Grönsak
```

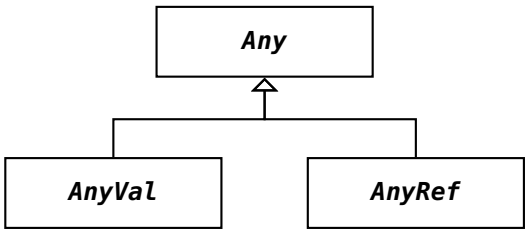
Nu har du explicit sagt till kompilatorn att du vill att alla grönsaker har en vikt:

```
scala> val grönsaker = Vector(Gurka(200), Tomat(42))
val grönsaker: Vector[Grönsak] = Vector(Gurka(200), Tomat(42))

scala> grönsaker.map(_.vikt)
val res0: Vector[Int] = Vector(200, 42)
```

Den abstrakta medlemmen `vikt` i den abstrakta typen `Grönsak` **implementeras** i en konkret subklass, här som en klassparameter.

10.1.13 Scalas typhierarki

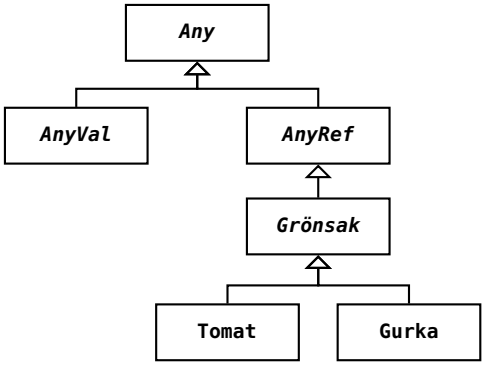
En förenklad bild av den översta delen av typhierarkin i Scala:



- De numeriska typerna `Int`, `Double`, etc är subtyper till **`AnyVal`** och kallas **värdetyper** och lagras på ett speciellt, effektivt sätt i minnet.
- Alla dina egna klasser är subtyper till **`AnyRef`** och kallas **referenstyper** och kan (direkt eller indirekt) konstrueras med **`new`**.
- `AnyRef` motsvaras av **`java.lang.Object`** i JVM.
- (Det finns även `Matchable` som är subtyp till `Any` och supertyp till `AnyRef` och `AnyVal`. Typen `Matchable` behövs för att skilja mellan typer som kan undersökas med mönstermatchning och andra s.k. **opaka typer** (överkurs).)

10.1.14 Implicita supertyper till dina egna klasser

Alla dina egna typer ingår underförstått i Scalas typhierarki:



10.1.15 Vad är en trait?

- **Trait** betyder **egenskap**.
- En trait liknar en klass, **men** speciella regler gäller:
  - den **kan** innehålla delar som **saknar implementation**
  - den **kan mixas** med flera andra traits så att olika koddelar kan kombineras på flexibla sätt.
  - den **kan inte** instansieras direkt.
  - den **kan** ha **parametrar**[2] på samma sätt som klasser.

10.1.16 Vad används en trait till?

En `trait` kan användas för att skapa en bastyp som kan vara hemvist för gemensamma delar hos subtyper:

```
trait Bastyp { val x = 42 }                  // Bastyp har medlemmen x
class Subtyp1 extends Bastyp { val y = 43 } // Subtyp1 ärver x, har även y
class Subtyp2 extends Bastyp { val z = 44 } // Subtyp2 ärver x, har även z
```

```
scala> val a = new Subtyp1
val a: Subtyp1 = Subtyp1@51016012

scala> a.x
val res0: Int = 42

scala> a.y
val res1: Int = 43

scala> a.z
-- Error:
  value z is not a member of Subtyp1

scala> new Bastyp
--Error:
  Bastyp is a trait; it cannot be instantiated
```

10.1.17 En trait kan ha abstrakta medlemmar

```
trait X { val x: Int }   // x är abstrakt, d.v.s. saknar implementation
class A extends X { val x = 42 }   // x ges en implementation
class B extends X { val x = 43 }   // x ges en annan implementation
```

```
scala> val a = new A
val a: A = A@5faeada1

scala> val b = B()    // fungerar utan new men då behövs ()
val b: B = B@cb51256
```

[2] I gamla Scala 2 kan traits ej ha parametrar

```
scala> val xs = Vector(a,b)
val xs: Vector[X] = Vector(A@5faeada1, B@cb51256)

scala> xs.map(_.x)
val res0: Vector[Int] = Vector(42, 43)

scala> class Y { val y: Int }
-- Error:
  class Y needs to be abstract, since val y: Int in class Y is not defined
```

10.1.18 En trait kan ha parametrar

```
trait X(val x: Int)
class A extends X(42)
class B(y: Int) extends X(y) // värdet av y blir argument till x i X
```

```
scala> val a = A()
val a: A = A@5faeada1

scala> val b = B(43)
val b: B = B@cb51256

scala> val xs = Vector(a,b)
val xs: Vector[X] = Vector(A@5faeada1, B@cb51256)

scala> xs.map(_.x)
val res0: Vector[Int] = Vector(42, 43)

scala> b.y
--  Error:
  value y cannot be accessed as a member of (b : B)
```

Hur kan vi göra medlemmen y synlig?
Lägg till **val** framför parameternamnet, eller använd en **case**-klass.

10.1.19 Abstrakta och konkreta medlemmar

```
object exempelVego1:

  trait Grönsak:
    var vikt: Double              // abstrakt medlem, saknar implementation
    var ärSkalad: Boolean = false // konkret medlem, har implementation

    def skala(): Unit             // abstrakt medlem, saknar implementation

  class Gurka(var vikt: Double) extends Grönsak:
    def skala(): Unit =           // implementation specifik för Gurka
      if !ärSkalad then
        println("Skalas med skalare.")
```

```
13          vikt = 0.99 * vikt
14          ärSkalad = true
15
16    class Tomat(var vikt: Double) extends Grönsak:
17      def skala(): Unit =           // implementation specifik för Tomat
18        if !ärSkalad then
19          println("Skållas.")
20          vikt = 0.99 * vikt
21          ärSkalad = true
```

10.1.20 Undvika kodduplicering med hjälp av arv

```
1  object exempelVego2:
2
3    trait Grönsak:  // innehåller gemensamma delar; hjälper oss undvika upprepning
4      val skalningsmetod: String      // abstrakt
5      val skalfaktor = 0.99           // konkret
6      var vikt: Double                // abstrakt
7      var ärSkalad: Boolean = false   // konkret
8
9      def skala(): Unit = if !ärSkalad then // konkret
10       println(skalningsmetod)
11       vikt = skalfaktor * vikt
12       ärSkalad = true
13
14   class Gurka(var vikt: Double) extends Grönsak: // det som är speciellt för gurkor
15     val skalningsmetod = "Skalas med skalare."
16
17   class Tomat(var vikt: Double) extends Grönsak: // det som är speciellt för tomater
18     val skalningsmetod = "Skållas."
```

10.1.21 Varför kan kodduplicering orsaka problem?

- Mer att skriva (inte jättestort problem)
- Fler kodrader att läsa och förstå
- Fler kodrader som påverkas vid tillägg
- Fler kodrader att underhålla:
  - Om man rättar en bug på ett ställe måste man komma ihåg att göra **exakt samma ändring** på alla de ställen där kodduplicering förekommer → **risk för nya buggar**
- Principen på engelska: `def DRY = "Don't Repeat Yourself!"`

Det *finns* tillfällen när **kodduplicering faktiskt är att föredra**: t.ex. om man vill att olika delar av koden ska vara **helt oberoende** av varandra.

10.1.22 Subtypspolymorfism och dynamisk bindning

```
trait Robot { def work(): Unit }

case class CleaningBot(name: String) extends Robot:
  def work(): Unit = println(" Städa Städa")

case class TalkingBot(name: String) extends Robot:
  def work(): Unit = println(" Prata Prata")
```

Polymorfism betyder ”många former”. Referenserna r och bot nedan kan ha olika ”former”, d.v.s de kan referera till olika sorters robotar.
Dynamisk bindning innebär att körtidstypen avgör vilken metod som körs.

```
scala> def robotDoWork(bot: Robot) = { print(bot); bot.work() }

scala> var r: Robot = CleaningBot("Wall-E")

scala> robotDoWork(r)
CleaningBot(Wall-E) Städa Städa

scala> r = TalkingBot("C3PO")

scala> robotDoWork(r)
TalkingBot(C3PO) Prata Prata
```

10.1.23 Exempel: Överskuggning och `override`

```
object exempelVego3:

  trait Grönsak:
    val skalningsmetod: String      // abstrakt
    val skalfaktor = 0.99           // konkret
    var vikt: Double                // abstrakt
    var ärSkalad: Boolean = false   // konkret

    def skala(): Unit = if !ärSkalad then
      println(skalningsmetod)
      vikt = skalfaktor * vikt
      ärSkalad = true

  class Gurka(var vikt: Double) extends Grönsak:
//nyckelordet override behövs ej om abstrakt medlem i supertyp
    val skalningsmetod = "Skalas med skalare."

  class Tomat(var vikt: Double) extends Grönsak:
//nyckelordet override behövs ej om abstrakt medlem i supertyp men tillåtet:
    override val skalningsmetod = "Skållas."
//  override val skalningmetod = "Skållas." //kompilatorn hittar stavfelet!
```

```
22      override val skalfaktor = 0.95  // override måste anges vid ändrad impl.
```

10.1.24 En final medlem kan ej överskuggas

```
1  object exempelVego4:
2
3    trait Grönsak:
4      val skalningsmetod: String
5      final val skalfaktor = 0.99  // en final medlem kan ej överskuggas
6      var vikt: Double
7      var ärSkalad: Boolean = false
8
9      def skala(): Unit = if !ärSkalad then
10       println(skalningsmetod)
11       vikt = skalfaktor * vikt
12       ärSkalad = true
13
14   class Gurka(var vikt: Double) extends Grönsak:
15     val skalningsmetod = "Skalas med skalare."
16
17   class Tomat(var vikt: Double) extends Grönsak:
18     val skalningsmetod = "Skållas."
19 //  override val skalfaktor = 0.95
20 //  ger KOMPILERINGSFEL: "cannot override final member"
```

10.1.25 Protected ger synlighet begränsad till subtyper

```
scala> trait Super:
         private val minHemlis = 42
         protected val vårHemlis = 42

scala> class Sub extends Super { def avslöjad = minHemlis }
- Error: Not found: minHemlis

scala> class Sub extends Super { def avslöjad = vårHemlis }

scala> val s = Sub()
val s: Sub = Sub@2eee9593

scala> s.avslöjad
val res0: Int = 42

scala> s.minHemlis
-- Error:
  value minHemlis is not a member of Sub - did you mean s.vårHemlis?

scala> s.vårHemlis
-- Error:
  Access to protected value vårHemlis not permitted because enclosing object
  is not a subclass of trait Super where target is defined
```

10.1.26 Filnamnsregler och -konventioner

- I flera språk, t.ex. Java, gäller dessa regler (men **inte** i Scala):
  - Det går bara ha **en enda publik klass per kodfil**.
  - Kodfilen måste ha **samma namn** som den publika klassen, t.ex. `KlassensNamn.java`
- Scala är **friare**:
  - I Scala får man ha **så många** icke-privata klasser/traits/singelobjekt i samma kodfil **som man vill**.
  - I Scala får man döpa kodfilerna **oberoende** av deras innehåll.
- Dessa **konventioner** brukar användas i Scala:
  - Om en kodfil bara innehåller **en enda** klass/trait/singelobjekt ge filen samma namn som innehållet, t.ex. `KlassensNamn.scala`
  - Om en kodfil innehåller **flera** saker, döp filen till något som återspeglar hela innehållet och använd **liten begynnelsebokstav**, t.ex. `drawing-utils.scala` eller `bastypensNamn.scala`
- Scala 3 varnar vid arv utanför samma kodfil (se öppna klasser senare).

10.1.27 Klasser, arv och klassparametrar

Klasser kan ärva andra typer (klasser och traits). Om supertypen har parametrar så **måste** subtypen ge argument efter `extends`.

```
1  object personExample1:
2
3    class Person(val namn: String)
4
5    class Akademiker(
6      override val namn: String,
7      val universitet: String) extends Person(namn)
8
9    class Student(
10     override val namn: String,
11     override val universitet: String,
12     val program: String) extends Akademiker(namn, universitet)
13
14   class Forskare(
15     override val namn: String,
16     override val universitet: String,
17     val titel: String) extends Akademiker(namn, universitet)
18
19   def main(args: Array[String]): Unit =
20     val kim = new Student("Kim Robinsson", "Lund", "Data")
21     println(s"${kim.namn} ${kim.universitet} ${kim.program}")
```

10.1.28 Statisk och dynamisk typ

```
var p: Person = Forskare("Robin Smith", "Lund", "Professor Dr")
```

- Den **statiska typen** för `p` är `Person` vilket gör att vi sedan kan låta `p` referera till *andra* instanser som är av typen Person.

  ```
  p = Student("Kim Robinson", "Lund", "Data")
  ```

- Med ”statisk typ” menas den typinformation som finns vid **kompileringstid**.
- Den **dynamiska typen**, även kallad **körtidstypen** som gäller vid exekvering kan vara mer specifik: den dynamiska typen för `p` är nu efter ovan tilldelning Student, men också Akademiker och Person.
- Man kan undersöka om den dynamiska typen för `p` är `EnVissTyp` med `p.isInstanceOf[EnVissTyp]` (men använd hellre `match`)
- Man kan säga åt kompilatorn: **”jag garanterar att p är av typen `EnVissTyp` så du kan omforma den statiska typen till `EnVissTyp` och så får jag stå ut med körtidsfel om jag ljuger”** genom att göra **typomvandling** (eng. *type casting*) med `p.asInstanceOf[EnVissTyp]` (detta är mycket ovanligt i normal Scala-kod, eftersom typtest + typomvandling görs säkrare med `match`)

10.1.29 Inmixning

Man kan ärva flera traits. Detta kallas **inmixning** (eng. *mix-in*) och görs med en komma-separerad typ-lista efter `extends`.

```
1  object personExample2:
2
3    trait Person     { val namn: String }
4    trait Akademiker { val universitet: String }
5    trait Examinerad { val titel: String }
6
7    class Student(val namn: String,
8                  val universitet: String,
9                  val program: String) extends Person, Akademiker
10
11   class Forskare(val namn: String,
12                  val universitet: String,
13                  val titel: String) extends Person, Akademiker, Examinerad
14
15   def main(args: Array[String]): Unit =
16     val f = new Forskare("B. Regnell", "Lunds universitet", "Professor Dr")
17     println(s"${f.titel} ${f.namn}, ${f.universitet}")
```

10.1.30 `isInstanceOf` och `asInstanceOf`

Testa körtidstyp med `isInstanceOf[Typ]`. Lova kompilatorn (och ta själv ansvar för) att det är en viss körtidstyp med `asInstanceOf[Typ]`. OBS! Använd hellre `match`.

```
object personExample3:

  trait Person     { val namn: String }
  trait Akademiker { val universitet: String }
  trait Examinerad { val titel: String }

  class Student(val namn: String,
                val universitet: String,
                val program: String) extends Person, Akademiker

  class Forskare(val namn: String,
                 val universitet: String,
                 val titel: String) extends Person, Akademiker, Examinerad

  def main(args: Array[String]): Unit =
    var p: Person = new Forskare("B. Regnell", "Lunds universitet", "Professor Dr")
    if p.isInstanceOf[Akademiker] then println(p.namn + " är akademiker")
    p = new Student("Kim Robinson", "Lund", "Data")  // går det att göra p.program?
    if p.isInstanceOf[Student] then println(p.asInstanceOf[Student].program)
  // Ovan görs hellre med match
```

10.1.31 Anonym klass

Om man har en abstrakt typ med saknade implementationer kan man fylla i det som fattas i dessa i ett extra block som ”hängs på” vid instansiering:

```
scala> trait Grönsak { val vikt: Int }
// defined trait Grönsak

scala> new Grönsak    // eller Grönsak()
-- Error:
1 |new Grönsak
  |    ^^^^^^^
  |    Grönsak is a trait; it cannot be instantiated

scala> new Grönsak { val vikt = 42 }
val res0: Grönsak = anon1@4e3f2908
```

Man får då vad som kallas en **anonym klass**. (I detta fall en ganska konstig grönsak som inte är någon speciell sorts grönsak, men som ändå har en vikt.)

Den allra enklaste (och mest meningslösa) anonyma klassen är:

```
scala> new {}
val res0: Object = anon1@5bb37371
```

10.1.32 Hur förhindra subtypning?

Du kan förhindra subtypning på dessa sätt:

- Med `final` skapar du en final klass som ej går att ärva:

```
final class Person(name: String)
```

```
scala> object Björn extends Person("Björn")
-- Error:
1 |object Björn extends Person("Björn")
  |       ^
  |       object Björn cannot extend final class Person
```

- Med **sealed** kan du försegla en hel typhierarki:

```
scala> sealed trait T  // tryck Alt+TAB för att vänta med evaluering
     | final class A extends T
     | final class B extends T
     |
scala> new T{}
-- Error:
1 |new T{}
|^
|Cannot extend sealed trait T in a different source file
```

Med en förseglad typhierarki kan du bara ärva bastypen inom samma kodfil.

10.1.33 Förseglade typer med `sealed`

Med en **sealed** kan du skapa en **förseglad** uppräkning:

```
sealed trait Färg(val toInt: Int)
object Färg:
  val values = Vector(Spader, Hjärter, Ruter, Klöver)
  case object Spader  extends Färg(0)
  case object Hjärter extends Färg(1)
  case object Ruter   extends Färg(2)
  case object Klöver  extends Färg(3)
```

Nyckelordet **sealed förhindrar** vidare subtypning av `Färg` i **annan kodfil** och **ger varning** om matchning är ofullständig – tips för att undvika körtidsfel.

```
scala> Färg.values(0) match { case Färg.Spader => "hej" }
-- Warning:
1 |Färg.values(0) match { case Färg.Spader => "hej" }
  |^^^^^^^^^^^^^^
  |match may not be exhaustive.
  |
  |It would fail on pattern case: Hjärter, Ruter, Klöver
val res0: String = hej
```

Använd hellre **enum** så får du både **sealed** och mer godis på köpet!

10.1.34 Öppen klass signalerar uppmuntrad subtypning

- Om man ärver en klass får man tillgång till alla medlemmar som inte är privata och kan byta till godtycklig implementation om typerna stämmer.

- Detta kan vara riskabelt om den som skrivit klassen inte planerat för detta och noga dokumenterat hur klassen är tänkt att användas vid arv.
- Genom att skapa **öppna klasser** med nyckelordet **open** signalerar du att klassen är tänkt att vara en supertyp vid arv.

```
open class Gurka(val vikt: Int, val pris: Double):
  /** Gör override på denna om du vill ha annat alternativ. */
  def alternativ: String = s"Det går precis lika bra med selleri!"
```

```
scala> object StorGurkan extends Gurka(1000, 1_000_000): // ärv på bäst du vill!
         override def alternativ = "kan kanske också funka med en betongpelare"
```

- **open** krävs om du vill tysta varning vid **arv från en annan kodfil**.
- Du kan även komma undan varningen med **import** `scala.language.adhocExtensions`

https://youtu.be/aFmIS5qeetA?t=206

10.1.35 Trait eller abstrakt klass?

Nyckelordet **abstract** behövs framför **class** om abstrakta medlemmar:

```
scala> class X { val x: Int }
1 |class X { val x: Int }
  |      ^
  |      class X needs to be abstract, since val x: Int in class X is not defined

scala> abstract class X { val x: Int }  // fungerar!
```

Men går det inte lika bra med en trait? Det går ofta precis lika bra med en trait.

Använd en **trait** om...

- ...du är osäker på vilket som är bäst. (Du kan alltid ändra till en abstrakt klass senare.)
- ...du vill kunna mixa in din trait tillsammans med andra traits.
- ...du vill göra din trait, om inmixad, transparent vid typhärledning.

Använd en (abstrakt) **klass** om...

- ...du vill begränsa inmixning – man kan bara ärva från en enda klass.
- ...du vill vidarebefordra parameter till supertyp (se nästa bild).
- ...du vill ärva din klass i Javakod.
- ...du vill minimera tid för omkompilering vid ändringar (spar tid vid stora projekt).

10.1.36 En trait får ej vidarebefordra parametrar

En trait får inte skicka vidare parametrar till en supertyp (det skulle bli knepiga problem annars vid inmixning):

```
scala> trait X(x: Int)
// defined trait X
```

```
scala> trait Y(y: Int) extends X(y)
-- Error:
1 |trait Y(y: Int) extends X(y)
  |                         ^^^^
  |  trait Y may not call constructor of trait X
```

Men det funkar fint med en **abstract class** eller en **class** (som ju **inte** får vara inmixningar):

```
scala> abstract class Y(y: Int) extends X(y)
// defined class Y
```

Mer detaljer här: dotty.epfl.ch/docs/reference/other-new-features/trait-parameters.html

---

10.1.37 Medlemmar, arv och överskuggning

Olika sorters överskuggningsbara medlemmar i **Scala**:

- **def**
- **val**
- **lazy val**
- **var**

Olika sorters överskuggningsbara instansmedlemmar i **Java**:

- variabel
- metod

Medlemmar som är **static** kan ej överskuggas (men döljas) vid arv.

- När man överskuggar (eng. *override*) en medlemmen med en annan medlem med samma namn i en subtyp, får denna medlem en (ny) implementation.
- Singelobjekt kan ej ärvas (och medlemmar i singelobjekt kan därmed ej överskuggas).
- När man konstruerar ett objektorienterat språk gäller det att man definierar sunda överskuggningsregler vid arv. Detta är förvånansvärt knepigt.
- Alla knepiga regler för överskuggning är svåra att lära sig utantill, men som tur är så hjälper kompilatorns felmeddelande dig att följa reglerna :)

10.1.38 Fördjupning: Regler för överskuggning i Scala

En medlem M1 i en supertyp får överskuggas av en medlem M2 i en subtyp, enligt dessa regler: (se även https://stackoverflow.com/questions/40057337)

1. M1 och M2 ska ha samma namn och typerna ska matcha.
2. **def** får bytas ut mot: **def**, **val**, och **lazy val**, och under speciella förutsättningar **var** (mer om att byta **def** mot **var** snart)
3. **val** får bytas ut mot **val**. Om medlemmen i M1 som överskuggas är abstrakt så får en **val** även bytas mot en **lazy val**.
4. En abstrakt **var** får bara bytas ut mot en **var**. En konkret **var** får ej bytas ut.
5. **lazy val** får bara bytas ut mot en **lazy val**.

6. Om en medlem i en supertyp är abstrakt *behöver* man inte använda nyckelordet **override** i subtypen. (Men det är bra att göra det ändå så att kompilatorn hjälper dig att kolla att du verkligen överskuggar något.)
7. Om en medlem i en supertyp är konkret *måste* man använda nyckelordet **override** i subtypen, annars ges kompileringsfel.
8. M1 får inte vara **final**.
9. M1 får inte vara **private**, men kan vara **private**[X] om M2 också är **private**[X], eller **private**[Y] om X är (en del av) Y.
10. Om M1 är **protected** måste även M2 vara det.

10.1.39 Fördjupning: Överskugga var med var

Det krävs speciella förutsättningar för att överskugga en **var** med en **var**.

- En konkret medlem får **inte** bytas ut mot en **var** – inte ens en konkret **var**:

```
scala> trait ConcreteVar { var x = 42 }
scala> trait Sub extends ConcreteVar { override var x = 43 }
-- Error:
1 |trait Sub extends ConcreteVar { override var x = 43 }
  |                                              ^
  |             error overriding variable x in trait ConcreteVar of type Int;
  |               variable x of type Int cannot override a mutable variable
```

- En abstrakt **var**-medlem får bytas ut mot en **var** om du **inte** skriver **override**:

```
scala> trait AbstractVar { var x: Int }
scala> trait Sub extends AbstractVar { override var x = 43 }
-- Error:
1 |trait Sub extends AbstractVar { override var x = 43 }
  |                                              ^
  |             error overriding variable x in trait AbstractVar of type Int;
  |               variable x of type Int cannot override a mutable variable
scala> trait Sub extends AbstractVar { var x = 43 }   // funkar utan override
// defined trait Sub
```

Undvik abstrakta **var**-medlemmar – oftast bättre med abstrakt **def**.

10.1.40 Fördjupning: Överskugga def med var

- En abstrakt **def**-medlem får bytas ut mot en **var** om du **inte** skriver **override**:

```
scala> trait Super { def x: Int }
scala> trait Sub extends Super { override var x = 43 }
-- Error:
1 |trait Sub extends Super { override var x = 43 }
  |                                              ^
  |                                              setter x_= overrides nothing

scala> trait Sub extends Super { var x = 43 }   // funkar om ej override
```

  Den abstrakta **def**-medlemmen blir då implementerad av en konkret getter.

- Egentligen är en publik **var**-medlem en kombination av en getter och en setter. Du kan skapa konkret getter+setter och överskugga gettern explicit med **override** (notera att settern inte kan göra **override**, eftersom superklassen inte har någon motsvarande metod att byta ut – jämför felmeddelande ovan):

```
scala> trait Sub2 extends Super:
         private var myPrivateValue = 42
         override def x: Int = myPrivateValue
         def x_(newValue: Int): Unit = myPrivateValue = newValue
```

Ovan fungerar fint eftersom vi nu har en giltig kombination av getter+setter.

10.1.41 Att skilja på mitt och ditt med `super`

```
1  scala> class X { def gurka = "super pepino" }
2
3  scala> class Y extends X:
4           override val gurka = ":("
5           val sg = super.gurka
6
7  scala> val y = new Y
8  y: Y = Y@26ba2a48
9
10 scala> y.gurka
11 res0: String = :(
```

Super Pepinos to the rescue:


```
scala> y.sg
res1: String = super pepino
```

https://youtu.be/NPhjiXskz34

10.1.42 Fördjupning: Intersektionstyp

Vid inmixning blir supertypen en intersektionstyp (eng. *intersection type*), vilket indikeras med typoperatorn & i exempel nedan:

```
trait Grönsak(val vikt: Int)
trait HarSmak(val smak: String)

object Gurka extends Grönsak(42), HarSmak("vattning")
object Tomat extends Grönsak(43), HarSmak("syrlig")
```

```
scala> Vector(Gurka, Tomat)
val res0: Vector[Grönsak & HarSmak] =
  Vector(Gurka@560742f7, Tomat@3cc82e23)
```

Det går att skapa egna intersektionstyper med **with**:

```
scala> class X { val x = 42 }
scala> trait Y { val y = "hej" }
scala> val a: X & Y = new X         // -- Error: Found: X Required: X & Y
scala> val b: X & Y = new X with Y  // Funkar! En anonym klass av typen X & Y
```

10.1.43 Fördjupning: Transparent trait

Du kan göra din trait **genomskinlig** vid typhärledning av supertypen för inmixningen med nyckelordet `transparent`:

```
trait Grönsak(val vikt: Int)
transparent trait HarSmak(val smak: String)

object Gurka extends Grönsak(42), HarSmak("vattning")
object Tomat extends Grönsak(43), HarSmak("syrlig")
```

```
scala> Vector(Gurka, Tomat)
val res0: Vector[Grönsak] =
  Vector(Gurka@560742f7, Tomat@3cc82e23)
```

Vilken typ hade visats utan `transparent`?
`Vector[Grönsak & HarSmak]`

10.1.44 Fördjupning: Typunioner med eller-operator

Typunioner skapas typ-operatorn | mellan typer:

```
scala> var x: Int | String = 42
var x: Int | String = 42

scala> x = "hej"
x: Int | String = hej

scala> type IntOrErr = Int | String
// defined alias type IntOrErr = Int | String

scala> def div(nom: Int, denom: Int): IntOrErr =
          if denom != 0 then nom / denom else "div. by zero"
```

Fördel jämfört med klass: rudimentärt enkelt.
Nackdelar: kan inte deklarera parametrar, medlemmar och allt annat som en klass kan; i viss mån kan detta kompensera med `extension` och `match`.
Läs mer här: https://dotty.epfl.ch/docs/reference/new-types/union-types.html

---

10.1.45 Terminologi och nyckelord vid arv

subtyp	en typ som ärver en supertyp
supertyp	en typ som ärvs av en subtyp
bastyp	en typ som är rot i ett arvsträd
abstrakt medlem	en medlem som saknar implementation
konkret medlem	en medlem som ej saknar implementation
abstrakt typ	en typ som kan ha abstrakta medlemmar; kan ej instansieras
konkret typ	en typ som ej har abstrakta medlemmar; kan instansieras
anonym klass	en namnlös klass som skapas direkt vid instansiering
`class`	en konkret typ som **kan ej ha abstrakta medlemmar**
`abstract class`	en abstrakt typ som **kan ha abstrakta medlemmar**
`trait`	är en abstrakt typ som **kan mixas in**
`extends`	står före en supertyp, medför arv av supertypens medlemmar
`override`	en medlem överskuggar (byter ut) en medlem i en superttyp
`protected`	gör en medlem synlig i subtyper till denna typ (jmf `private`)
`final def gurka`	gör medlemen gurka final: förhindrar överskuggning
`final class`	gör klassen final: förhindrar vidare subtypning
`sealed`	förseglad trait/klass: enbart subtyper i denna kodfil, koll av match
`open class`	berätta att den är tänkt att ärvas, `open` krävs för arv i annan kodfil
`transparent trait`	gör typen ”genomskinlig” (alltså osynlig) vid typhärledning
`super.gurka`	refererar till supertypens medlem `gurka` (jmf `this`)

10.1.46 Vad är en algebraisk datatyp?

Algebraisk datatyp (eng. *algebraic data type*), förk. ADT:

- En datatyp som består av (en kombination av):
- **Produkt**-typer, **Summa**-typer:
  - En produkt-typ är en ”och”-typ (ä.k. record, struct), exempel:
    `case class Person(namn: String, ålder: Int)`
    består av attributen namn **OCH** ålder.
  - En summa-typ är en ’Eller”-typ (ä.k. disjunkt union, co-produkt).
  - Exempel: `enum Färg { case Röd, Svart}`
    Färg kan vara antingen Röd **ELLER** Svart.
    En `Grönsak` är antingen en `Gurka` **ELLER** en `Tomat`

```
sealed trait Grönsak { val vikt: Int }
case class Gurka(vikt: Int) extends Grönsak
case class Tomat(vikt: Int) extends Grönsak
```

https://en.wikipedia.org/wiki/Algebraic_data_type

10.1.47 En case-klass är en produkt.

Klassen `Person` nedan har ett namn **och** en ålder.

```
1 scala> case class Person(namn: String, ålder: Int)
2
```

```
scala> Person("Kim",42)
val res0: Person = Person(Kim,42)

scala> res0.isInstanceOf[Product]
val res1: Boolean = true

scala> res0.product   // Tryck TAB
productArity          productElement    productElementName
productElementNames   productIterator   productPrefix

scala> res0.productElementNames.toVector
val res2: Vector[String] = Vector(namn, ålder)

scala> res0.productElement
val res3: Int => Any = Lambda1981/0x00000008408f2840@44498af0

scala> res0.productElement(0)
val res4: Any = björn
```

10.1.48 Algebraisk datatyp, kombinerad produkt och summa

Med trait, case-klass och objekt:

```
sealed trait PersonId // summan av två subtyper (Number eller Missing)
object PersonId:
  case class Number(n: Long) extends PersonId  // produkt med ett element
  case object Missing extends PersonId         // ett enkelt värde
```

Motsvarande med **enum**:

```
enum PersonId:
  case Number(n: Long)
  case Missing
```

10.1.49 Algebraisk datatyp med typparameter

Denna ADT har ett fall som är en **generisk** case-klass:

```
sealed trait Kanske[+A]   // +A gör typen flexibel, ä.k. "kovariant" mer om det senare
object Kanske:
  case class Någon[A](a: A) extends Kanske[A]
  case object Ingen extends Kanske[Nothing]
```

Motsvarande med **enum**: (kompilatorn fyller i **extends** ... automatiskt)

```
enum Kanske[+A]:
  case Någon(a: A)
  case Ingen
```

Känner du igen denna? Jämför inbyggda typen `Option`

Implementation av `getOrElse` med extensionsmetod, tips: använd **`match`**

```
extension [A](k: Kanske[A]) def getOrElse(default: A):A = k match
  case Kanske.Någon(a) => a
  case Kanske.Ingen => default
```

10.2 Övning `inheritance`

Mål

- ☐ Kunna deklarera och använda en arvshierarki i flera nivåer med nyckelordet **`extends`**.
- ☐ Känna till synlighetsregler vid arv och nyttan med privata och skyddade medlemmar och nyckelorden **`private`** och **`protected`**.
- ☐ Kunna deklarera och använda överskuggade medlemmar och nyckelordet **`override`**.
- ☐ Kunna deklarera och använda en hierarki av klasser där konstruktorparametrar överförs till superklasser.
- ☐ Kunna deklarera och använda uppräknade värden med case-objekt och gemensam bastyp.
- ☐ Känna till reglerna som gäller vid överskuggning av olika sorters medlemmar.
- ☐ Känna till nyttan med finala klasser och finala attribut och nyckelordet **`final`**.
- ☐ Kännedom om dessa begrepp: bastyp, supertyp, subtyp, körtidstyp, dynamisk bindning, polymorfism, trait, inmixning, överskuggad medlem, anonym klass, skyddad medlem, abstrakt medlem, abstrakt klass, referenstyp, värdetyp.

Förberedelser

- ☐ Studera begreppen i kapitel 10

10.2.1 Grunduppgifter; förberedelse inför laboration

Uppgift 1. *Para ihop begrepp med beskrivning.*

Koppla varje begrepp med den (förenklade) beskrivning som passar bäst:

bastyp	1	A	har supertypen `AnyRef`, allokeras i heapen via referens
supertyp	2	B	kan ha många former, t.ex. en av flera subtyper
subtyp	3	C	klass utan namn, utvidgad med extra implementation
körtidstyp	4	D	en typ som är mer specifik
dynamisk bindning	5	E	kan ha parametrar, kan ej instansieras, kan ej mixas in
polymorfism	6	F	saknar implementation
trait	7	G	har supertypen `AnyVal`, lagras direkt på stacken
inmixning	8	H	tillföra egenskaper med **`with`** och en trait
överskuggad medlem	9	I	är abstrakt, kan mixas in, kan ha parametrar
anonym klass	10	J	kan vara mer specifik än den statiska typen
skyddad medlem	11	K	är endast synlig i subtyper
abstrakt medlem	12	L	körtidstypen avgör vilken metod som körs
abstrakt klass	13	M	medlem i subtyp ersätter medlem i supertyp
förseglad typ	14	N	den mest generella typen i en arvshierarki
referenstyp	15	O	en typ som är mer generell
värdetyp	16	P	subtypning utanför denna kodfil är förhindrad

Uppgift 2. *Gemensam bastyp.* Man vill ofta lägga in objekt av olika typ i samma samling.

```
scala> class Gurka(val vikt: Int)
scala> class Tomat(val vikt: Int)
scala> val gurkor = Vector(Gurka(100), Gurka(200))
scala> val grönsaker = Vector(Gurka(300), Tomat(42))
```

a) Om en samling innehåller objekt av flera olika typer försöker kompilatorn härleda den mest specifika typen som objekten har gemensamt. Vad blir det för typ på värdet `grönsaker` ovan?

b) Försök ta reda på summan av vikterna enligt nedan. Vad ger andra raden för felmeddelande? Varför?

```
scala> gurkor.map(_.vikt).sum     // fungerar
scala> grönsaker.map(_.vikt).sum  // fungerar inte
```

c) Du ska nu göra så att du kan komma åt vikten på alla grönsaker genom att ge gurkor och tomater en gemensam bastyp som de olika konkreta grönsakstyperna utvidgar med nyckelordet **`extends`**. Det heter att subtyperna `Gurka` och `Tomat` **ärver** egenskaperna hos supertypen `Grönsak`.

Skapa en bastyp `Grönsak` med ett abstrakt attribut `vikt`. Låt sedan de konkreta grönsakerna ärva bastypen:

```
scala> trait Grönsak { val vikt: Int }
scala> class Gurka(val vikt: Int) extends Grönsak
scala> class Tomat(val vikt: Int) extends Grönsak
scala> val gurkor = Vector(Gurka(100), Gurka(200))
scala> val grönsaker = Vector(Gurka(300), Tomat(42))
```

När sker initialisering av attributet `vikt`?

d) Vad blir det nu för typ på variabeln `grönsaker` ovan?

e) Går det nu att summera vikterna i `grönsaker` med uttrycket nedan? Varför?
`grönsaker.map(_.vikt).sum`

f) En trait liknar en klass, men man kan inte instansiera den direkt. Vad blir det för felmeddelande om du försöker skapa en instans av en trait enligt nedan?

```
scala> trait Grönsak { val vikt: Int }
scala> new Grönsak
```

g) Traiten `Grönsak` har en abstrakt medlem `vikt`. Den sägs vara abstrakt eftersom den saknar implementation – medlemmen har bara ett namn och en typ men inget värde. Du kan instansiera den abstrakta traiten `Grönsak` om du fyller i det som ”fattas”, nämligen ett värde på `vikt`. Man kan fylla på det som fattas i genom att ”hänga på” ett block efter typens namn vid instansiering. Man får då vad som kallas en **anonym klass**, i detta fall en ganska konstig grönsak som inte är någon speciell sorts grönsak med som ändå har en vikt.

Vad får `anonymGrönsak` nedan för typ och strängrepresenation?

```
scala> val anonymGrönsak = new Grönsak { val vikt = 42 }
```

h) Vad blir felmeddelandet om du skapar en anonym klass `Grönsak` med en kropp som saknar definition av vikt?

Uppgift 3. *Polymorfism vid arv, s.k. subtypspolymorfism.* Polymorfism betyder ”många skepnader”. I samband med arv innebär det att flera subtyper, till exempel `Ko` och `Gris`, kan hanteras gemensamt som om de vore instanser av samma supertyp, så som `Djur`. Subklasser kan implementera en metod med samma namn på olika sätt. Vilken metod som exekveras bestäms vid körtid beroende på vilken subtyp som instansieras. På så sätt kan djur komma i många skepnader.

a) Implementera funktionen `skapaDjur` nedan så att den returnerar antingen en ny `Ko` eller en ny `Gris` med lika sannolikhet.

```
scala> trait Djur { def väsnas: Unit }
scala> class Ko   extends Djur { def väsnas = println("Muuuuuuu") }
scala> class Gris extends Djur { def väsnas = println("Nöffnöff") }
scala> def skapaDjur(): Djur = ???
scala> val bondgård = Vector.fill(42)(skapaDjur())
scala> bondgård.foreach(_.väsnas)
```

b) Lägg till ett djur av typen Häst som väsnas på lämpligt sätt och modifiera `skapaDjur` så att det skapas kor, grisar och hästar med lika sannolikhet.

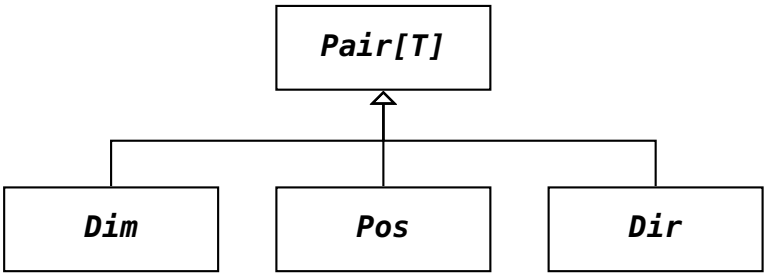
Uppgift 4. *Olika typer av heltalspar till laborationen `snake0`.* **OBS! Gör denna uppgift *innan* du kollar på given kod i labben så att du inte spojlar uppgiften.**

Under veckans laboration ska du använda olika typer av par som representerar riktning och position på en tvådimensionell spelplan, samt spelplanens storlek. I stället för att använda en vanlig 2-tupel till dessa tre olika typer av par ska du skapa egna, specifika typer som alla ärver bastypen `Pair[T]`. Dessa typer ska alla ligga i filen `pairs.scala` i `package snake`.

```
// detta är en skiss på filen pairs.scala
package snake

trait Pair[T]:
  def x: T
  def y: T
  // uppgift a) lägg till den konkreta metoden tuple

// efterföljande deluppgifterna implementerar dessa subtyper till Pair:
//   case klass Dim beskriver en 2-dimensionell ytas storlek
//   case klass Pos beskriver en position på en yta av Dim storlek
//   enum Dir beskriver förflyttning mot North, South, East, West
```

Skillnaden mellan `Pair[T]` och en vanlig 2-tupel är att medlemmarna `x` och `y` garanterat är av *samma* typ, medan en 2-tupel kan innehålla element av olika typ.

I fig. 10.1 visas en bild av klasshierarkin som du steg-för-steg ska utveckla i efterföljande uppgifter. Fördelen med att ha olika typer av par är att det är mer typsäkert (eng. *type safe*): vi får hjälp av kompilatorn att upptäcka om vi av misstag förväxlar t.ex. en position med en riktning.



Figur 10.1: Arvshierarki med `Pair[T]` som bastyp.

a) Öppna en editor och koda `trait Pair[T]` i en fil `pairs.scala`. Lägg dessutom till en konkret metod `tuple` i `Pair[T]` som returnerar en 2-tupel med de båda elementen i paret, så att det vid behov går att omvandla `Pair`-instanser till 2-tupler. Använd REPL för att testa att detta fungerar:

```
scala> val p = new Pair[Int] { override val x = 10; override val y = 20 }
p: Pair[Int]{val x: Int; val y: Int} = $anon$1@784223e9

scala> p.tuple
val res0: (Int, Int) = (10,20)
```

b) Fungerar koden ovan även utan nyckelordet `override` (testa i REPL)? Varför? När *måste* `override` användas? Vad är fördelen resp. nackdelen med att använda `override` även när det inte är nödvändigt?

c) Skapa en case-klass `Dim` som ärver `Pair[Int]`. Instanser av denna klass kommer du att använda under veckans laboration för att representera en spelplans storlek genom att låta `x` ange antalet horisontella positioner och `y` antalet vertikala positioner.

Lägg även till ett kompanjonsobjekt `Dim` med en `apply`-metod som kan skapa `Dim`-instanser givet en 2-tupel. Testa i REPL enligt nedan.

```
scala> Dim(50, 60)
val res1: Dim = Dim(50,60)

scala> Dim((60, 50))
val res2: Dim = Dim(60,50)

scala> res2.tuple
val res3: (Int, Int) = (60,50)
```

d) Lägg till en case-klass `Pos` som ärver `Pair[Int]` som representerar en position med en `x`-koordinat och en `y`-koordinat, båda klassparametrar. Kordinaterna ska hållas inom en spelplansstorlek som ges av klassparametern `dim` av typen `Dim`. Kordinatpositionerna är heltal och räknas från `0` till (men inte med) `dim.x` resp. `dim.y`.

Gör primärkonstruktorn i case-klassen `Pos` **privat**, genom att skriva nyckelordet `private` efter klassnamnet men före klassparameterlistan, så att det inte går att skapa instanser via primärkonstruktorn utanför klasskroppen och kompanjonsobjektet.

Implementera metoderna `+` och `-` i case-klassen `Pos`. Båda metoderna ska ta en parameter `p` av typen `Pair[Int]` och returnera en ny `Pos`, där `p.x` resp. `p.y` är adderat resp. subtraherat från aktuell position. Observera att du inte ska skriva `new` när du skapar en ny instans, eftersom dessa alltid ska skapas via kompanjonsobjektets `apply`-metod, som är en ”smart” fabriksmetod som garanterar håller koordinaterna inom

spelplanen.

Lägg till ett kompanjonsobjekt `Pos` med en `apply`-metod som skapar en ny `Pos`-instans som ser till att koordinaterna alltid är inom `dim`. Aritmetiken ska ske modulo storleken `dim`, d.v.s en position ska aldrig kunna hamna utanför spelplanen; i stället så börjar man om på andra sidan (se exempel i REPL nedan).
Tips: Använd `java.lang.Math.floorMod` som hanterar negativa argument så att resultatet blir positivt (till skillnad från modulo-operatorn %).

Lägg även till fabriksmetoden `random` som kan skapa nya slumpmässiga positioner inom `dim`. *Tips:* Använd `scala.util.Random.nextInt`.

Testa att det fungerar enligt nedan:

```
scala> Pos(-1,20,Dim(10,20))
val res4: Pos = Pos(9,0,Dim(10,20))

scala> new Pos(-1,20,Dim(10,20))  // förbjuds med privat primärkonstruktor
-- Error:
1 |new Pos(-1,20,Dim(10,20))
  |    ^^^
  |constructor Pos cannot be accessed as a member of Pos

scala> Pos(0,0,Dim(5,5)) + Pos(6,12, Dim(5,5))
val res5: Pos = Pos(1,2,Dim(5,5))

scala> Pos(0,0,Dim(5,5)) - Pos(1,2, Dim(5,5))
val res6: Pos = Pos(4,3,Dim(5,5))

scala> for (_ <- 1 to 3) yield Pos.random(Dim(10,10))
val res7: IndexedSeq[Pos] =
  Vector(Pos(8,8,Dim(10,10)), Pos(2,6,Dim(10,10)), Pos(3,7,Dim(10,10)))
```

e) Vad händer om du glömmer skriva **`new`** när du anropar den privata konstruktorn i din `apply`-metod? Varför finns inte detta problem i `apply`-metoden för `Dim`?

f) Lägg till en **`enum`** `Dir` som ärver `Pair[Int]` och har två **`val`**-parametrar `x` och `y`. Lägg också till fyra fall med **`case`** som alla ärver `Dir` och som representerar en enstegsförflyttning i de fyra väderstrecken, genom att ge parametrarna `x` resp. `y` något av värden 1, −1 eller 0. Norrut ska anges med x-koordinaten 0 och y-koordinaten −1, etc. Verifiera i REPL att enumerationen fungerar.

Lägg till en **`export`** som gör så att det räcker att importera `snake.*` för att få alla fyra riktningar synliga direkt (annars behövs även import av `Dir.*` på alla ställen där riktning används i och utanför paketet `snake`)

Uppgift 5. *Supertyp med parameter.* Utbildningsdepartementet vill med sitt nya datasystem hålla koll på vissa personer och skapar därför en klasshierarki enligt nedan. Skriv in koden i en editor och testa i REPL med `sbt`.

```
class Person(val namn: String)

class Akademiker(
  namn: String,
  val universitet: String) extends Person(namn)

class Student(
  namn: String,
```

```
  universitet: String,
  program: String) extends Akademiker(namn, universitet)

class Forskare(
  namn: String,
  universitet: String,
  titel: String) extends Akademiker(namn, universitet)
```

a) Deklarera fyra olika `val`-variabler med lämpliga namn som refererar till olika instanser av alla olika klasser ovan och ge attributen valfria initialvärden genom olika parametrar till konstruktorerna.

b) Skriv två satser: en som först stoppar in instanserna i en `Vector` och en som sedan loopar igenom vektorn och skriv ut alla instansers `toString` och `namn`.

c) Utbildningsdepartementet vill att det inte ska gå att instansiera objekt av typerna `Person` och `Akademiker`. Det kan åstadkommas genom att placera nyckelordet `abstract` före `class`. Uppdatera koden i enlighet med detta. Vilket blir felmeddelande om man försöker instansiera en `abstract class`? Går det lika bra med en `trait`?

d) Utbildningsdeparetementet vill slippa implementera `toString`. Gör därför om typerna `Student` och `Forskare` till case-klasser. *Tips:* För att undkomma ett kompileringsfel (vilket?) behöver du använda `override val` på lämpligt ställe. Skapa instanser av de nya case-klasserna `Student` och `Forskare` och skriv ut deras `toString`.

e) Använd abstrakta attribut i stället för parametrar för typerna som är abstrakta, så att du inte behöver skriva `override val` i klassparametrarna till de konkreta case-klasserna. Du ska också införa en case-klass `IckeAkademiker` som ska användas i olika statistiska medborgarundersökningar. Dessutom förser man alla personer med ett personnummer representerat som en `Long`. Hur ser utbildningsdepartementets kod ut nu, efter alla ändringar? Skriv ett testprogram som skapar några instanser och skriver ut deras attribut.

10.2.2 Extrauppgifter; träna mer

Uppgift 6. *Bastypen `Shape` och subtyperna `Rectangle` och `Circle`.* Du ska i denna uppgift skapa ett litet bibliotek för geometriska former med oföränderliga objekt implementerade med hjälp av case-klasser. De geometriska formerna har en gemensam bastyp kallad `Shape`. Utgå från koden nedan.

```
case class Point(x: Double, y: Double):
  def move(dx: Double, dy: Double): Point = Point(x + dx, y + dy)

trait Shape:
  def pos: Point
  def move(dx: Double, dy: Double): Shape

case class Rectangle(pos: Point, width: Double, height: Double) extends Shape:
  def move(dx: Double, dy: Double): Rectangle = copy(pos = pos.move(dx, dy))

case class Circle(pos: Point, radius: Double) extends Shape:
  def move(dx: Double, dy: Double): Circle = copy(pos = pos.move(dx, dy))
```

a) Instansiera några cirklar och rektanglar och gör några relativa förflyttningar av dina instanser genom att anropa `move`.

b) Lägg till en konkret metod `moveTo` i `Point` som gör en absolut förflyttning till koordinaterna `x` och `y`. Lägg till en abstrakt metod `moveTo` `Shape` som implementeras i subklasserna. Testa med REPL på några instanser av `Rectangle` och `Circle`.

c) Lägg till metoden `distanceTo(that: Point): Double` i case-klassen `Point` som räknar ut avståndet till en annan punkt med hjälp av `math.hypot`. Klistra in i REPL och testa på några instanser av `Point`.

d) Lägg till en konkret metod `distanceTo(that: Shape): Double` i traiten `Shape` som räknar ut avståndet till positionen för en annan Shape. Testa i REPL på några instanser av `Rectangle` och `Circle`.

e) Gör så att `distanceTo` kan anropas med operatornotation.

10.2.3 Fördjupningsuppgifter; utmaningar

Uppgift 7. *Inmixning.* Man kan utvidga en klass med multipla traits med en kommaseparerad lista. På så sätt kan man fördela medlemmar i olika traits och återanvända gemensamma koddelar genom så kallad **inmixning**, så som nedan exempel visar.

En alternativ fågeltaxonomi, speciellt populär i Skåne, delar in alla fåglar i två specifika kategorier: Kråga respektive Ånka. Krågor kan flyga men inte simma, medan Ånkor kan simma och oftast även flyga. Fågel i generell, kollektiv bemärkelse kallas på gammal skånska för Fyle.[3]

```
trait Fyle:
  val läte: String
  def väsnas: Unit = print(läte * 2)
  val ärSimkunnig: Boolean
  val ärFlygkunnig: Boolean

trait KanSimma       { val ärSimkunnig = true }
trait KanInteSimma   { val ärSimkunnig = false }
trait KanFlyga       { val ärFlygkunnig = true }
trait KanKanskeFlyga { val ärFlygkunnig = math.random() < 0.8 }

class Kråga extends Fyle, KanFlyga, KanInteSimma:
  val läte = "krax"

class Ånka extends Fyle, KanSimma, KanKanskeFlyga:
  val läte = "kvack"
  override def väsnas = print(läte * 4)
```

a) En flitig ornitolog hittar 42 fåglar i en perfekt skog där alla fågelsorter är lika sannolika, representerat av vektorn `fyle` nedan. Skriv i REPL ett uttryck som undersöker hur många av dessa som är flygkunniga Ånkor, genom att använda metoderna `filter`, `isInstanceOf`, `ärFlygkunnig` och `size`.

```
scala> val fyle =
         Vector.fill(42)(if math.random() > 0.5 then new Kråga else new Ånka)
scala> fyle.foreach(_.väsnas)
scala> val antalFlygånkor: Int = ???
```

[3] www.klangfix.se/ordlista.htm

b) Om alla de fåglar som ornitologen hittade skulle väsnas exakt en gång var, hur många krax och hur många kvack skulle då höras? Använd metoderna `filter` och `size`, samt predikatet `ärSimkunnig` för att beräkna antalet krax respektive kvack.

```
scala> val antalKrax: Int = ???
scala> val antalKvack: Int = ???
```

Uppgift 8. *Finala klasser.* Om man vill förhindra att man kan göra **`extends`** på en klass kan man göra den final genom att placera nyckelordet **`final`** före nyckelordet **`class`**.

a) Eftersom klassificeringen av fåglar i uppgiften ovan i antingen Ånkor eller Krågor är fullständig och det inte finns några subtyper till varken Ånkor eller Krågor är det lämpligt att göra dessa finala. Ändra detta i din kod.

b) Testa att ändå försöka göra en subklass `Simkråga` **`extends`** `Kråga`. Vad ger kompilatorn för felmeddelande om man försöker utvidga en final klass?

Uppgift 9. *Accessregler vid arv och nyckelordet* ***`protected`***. Om en medlem i en supertyp är privat så kan man inte komma åt den i en subklass. Ibland vill man att subklassen ska kunna komma åt en medlem även om den ska vara otillgänglig i annan kod.

```
trait Super:
  private val minHemlis = 42
  protected val vårHemlis = 42

class Sub extends Super:
  def avslöja = minHemlis
  def kryptisk = vårHemlis * math.Pi
```

a) Vad blir felmeddelandet när klassen `Sub` försöker komma åt `minHemlis`?

b) Deklarera `Sub` på nytt, men nu utan den förbjudna metoden `avslöja`. Vad blir felmeddelandet om du försöker komma åt `vårHemlis` via en instans av klassen `Sub`? Prova till exempel med `(new Sub).vårHemlis`

c) Fungerar det att anropa metoden `kryptisk` på instanser av klassen `Sub`?

Uppgift 10. *Användning av* ***`protected`***. Den flitige ornitologen från uppgift 7 ska ringmärka alla 42 fåglar hen hittat i skogen. När hen ändå håller på bestämmer hen att även försöka ta reda på hur mycket oväsen som skapas av respektive fågelsort. För detta ändamål apterar den flitige ornitologen en Linuxdator på allt infångat fyle. Du ska hjälpa ornitologen att skriva programmet.

a) Inför en **`protected var`** `räknaLäte` i traiten `Fyle` och skriv kod på lämpliga ställen för att räkna hur många läten som respektive fågelinstans yttrar.

b) Inför en metod `antalLäten` som returnerar antalet krax respektive kvack som en viss fågel yttrat sedan dess skapelse.

c) Varför inte använda **`private`** i stället for **`protected`**?

d) Varför är det bra att göra räknar-variabeln oåtkomlig från ”utsidan”?

Uppgift 11. *Inmixning av egenskaper.* Det visar sig att vår flitige ornitolog från uppgift 7 på sidan 99 sov på en av föreläsningarna i zoologi när hen var nolla på

Natfak, och därför helt missat fylekategorin `Pjodd`. Hjälp vår stackars ornitolog så att fylehierarkin nu även omfattar Pjoddar. En Pjodd kan flyga som en Kråga men den `ÄrLiten` medan en Kråga `ÄrStor`. En Pjodd kvittrar dubbelt så många gånger som en Ånka kvackar. En Pjodd `KanKanskeSimma` där simkunnighetssannolikheten är 0.2. Låt ornitologen ånyo finna 42 slumpmässiga fåglar i skogen och filtrera fram lämpliga arter. Undersök sedan hur dessa väsnas, i likhet med deluppgift 7b.

Uppgift 12. *Arvshierarki med matematiska tal.* Studera den djupa arvshierarkin i paketet `numbers` i koden på efterföljande sidor. Paketet `numbers` modellerar olika sorters tal i matematiken, med syftet att erbjuda ett s.k. DSL [4], alltså ett specialspråk för en viss applikationsdomän[5], här: domänen matematiska tal.

Du kan ladda ner koden för `numbers` här:
github.com/lunduniversity/introprog/blob/master/compendium/examples/numbers.scala
Notera speciellt metoden `reduce` som reducerar ett tal till sin enklaste form. Metoden `reduce` överskuggas på lämpliga ställen med relevant reduktion.

a) Rita en bild över typhierarkin, t.ex. som ett upp-och-nedvänt träd med bastypen `Number` som rot.

b) Skriv kod som använder de olika konkreta klasserna i `package numbers`.

```
scala> numbers.  // Tryck Tab
AbstractComplex   AbstractNatural    AbstractReal   Frac    Nat      Polar
AbstractInteger   AbstractRational   Complex        Integ   Number   Real

scala> numbers.Integ(12)
res0: numbers.Integ = Integ(12)

scala> import numbers.Syntax._
import numbers.Syntax._

scala> 42.j
res1: numbers.Complex = Complex(Real(0),Real(42))

scala> 42.42.j
res2: numbers.Complex = Complex(Real(0),Real(42.42))
```

c) Ändra på metoden + i `trait Number` så att den blir abstrakt och implementera den i alla konkreta klasser.

d) Implementera fler räknesätt och bygg vidare på koden så som du finner intressant.

e) Gör så att metoden `reduce` i klassen `AbstractRational` använder algoritmen Greatest Common Divisor (GCD)[6] så som beskrivs här:
www.artima.com/pins1ed/functional-objects.html#6.8
så att täljare och nämnare blir så små som möjligt.

```
package numbers

trait Number:
  def reduce: Number = this
  def isZero: Boolean
  def isOne: Boolean
  def +(that: Number): Number = ???
```

[4] https://en.wikipedia.org/wiki/Domain-specific_language
[5] https://stackoverflow.com/questions/49216312/what-is-dsl-in-scala
[6] https://sv.wikipedia.org/wiki/St%C3%B6rsta_gemensamma_delare

```
8
9  object Number:
10   def Zero = Nat(0)
11   def One  = Nat(1)
12   def Im(im: BigDecimal) = Complex(Real(0), Real(im))
13   def Im(im: Real)       = Complex(Real(0), im)
14   def j                  = Complex(Real(0), Real(1))
15   def Re(re: BigDecimal) = Complex(Real(re), Real(0))
16   def Re(re: Real)       = Complex(re, Real(0))
17
18 trait AbstractComplex extends Number:
19   def re: AbstractReal
20   def im: AbstractReal
21   def abs = Real(math.hypot(re.decimal.toDouble,im.decimal.toDouble))
22   def fi =  Real(math.atan2(im.decimal.toDouble, re.decimal.toDouble))
23   override def isZero: Boolean = re.decimal == 0 && im.decimal == 0
24   override def isOne: Boolean = abs.decimal == 1.0
25   override def reduce: AbstractComplex = if im.decimal == 0 then re.reduce else this
26
27 final case class Complex(re: Real, im: Real) extends AbstractComplex
28
29 object Complex:
30   def apply(re: BigDecimal, im: BigDecimal) = new Complex(Real(re), Real(im))
31
32 final case class Polar(
33     override val abs: Real,
34     override val fi: Real
35   ) extends AbstractComplex:
36   override def re = Real(abs.decimal.toDouble * math.cos(fi.decimal.toDouble))
37   override def im = Real(abs.decimal.toDouble * math.sin(fi.decimal.toDouble))
38
39 object Polar:
40   def apply(abs: BigDecimal, fi: BigDecimal) = new Polar(Real(abs), Real(fi))
41
42 trait AbstractReal extends AbstractComplex:
43   def decimal: BigDecimal
44   override def isZero = decimal == 0
45   override def isOne = decimal == 1
46   override def re = this
47   override def im = Number.Zero
48   override def reduce:  AbstractReal =
49     if decimal == 0 then Number.Zero else if decimal == 1 then Number.One else this
50
51 final case class Real(decimal: BigDecimal) extends AbstractReal
52
53 trait AbstractRational extends AbstractReal:
54   def numerator: AbstractInteger
55   def denominator: AbstractInteger
56   override def decimal = BigDecimal(numerator.integ)
57   override def isOne = numerator.integ == denominator.integ
58   override def reduce: AbstractRational =
59     if denominator.isOne then numerator.reduce else this // should use GCD
60
61 final case class Frac(numerator: Integ, denominator: Integ) extends AbstractRational:
62   require(denominator.integ != 0, "denominator must be non-zero")
63
```

```
64 object Frac:
65   def apply(n: BigInt, d: BigInt) = new Frac(Integ(n), Integ(d))
66
67 trait AbstractInteger extends AbstractRational:
68   def integ: BigInt
69   override def numerator = this
70   override def denominator = Number.One
71   override def isZero = integ == 0
72   override def isOne = integ == 1
73   override def decimal: BigDecimal = BigDecimal(integ)
74   override def reduce: AbstractInteger =
75     if isZero then Number.Zero else if isOne then Number.One else this
76
77 final case class Integ(integ: BigInt) extends AbstractInteger
78
79 trait AbstractNatural extends AbstractInteger
80
81 final case class Nat(integ: BigInt) extends AbstractNatural:
82   require(integ >= 0, "natural numbers must be non-negative")
83
84 extension (i: Int)    def j = Number.Im(i)
85 extension (d: Double) def j = Number.Im(d)
```

10.3 Grupplaboration: `snake0`

Mål

- ☐ Kunna använda arv.
- ☐ Kunna göra överskuggning av medlemmar i en supertyp.
- ☐ Kunna förklara begreppet dynamisk bindning.
- ☐ Kunna använda abstrakta klasser och skapa en klasshierarki.

Förberedelser

- ☐ Gör övning `lookup` i kapitel 9.2, speciellt uppgift 4.
- ☐ Läs dokumentationen för `introprog.BlockGame`.
- ☐ Hämta given kod via kursen github-plats eller via hemsidan under Download.
- ☐ Läs igenom hela laborationen och förbered dig inför första gruppmötet.
- ☐ Diskutera i din samarbetsgrupp hur ni ska dela upp koden mellan er i flera olika delar, som ni kan arbeta med var för sig. En sådan del kan vara en klass, en trait, ett objekt, ett paket, eller en funktion.
- ☐ Varje del ska ha en **huvudansvarig** individ.
- ☐ Arbetsfördelningen ska vara någorlunda jämnt fördelad mellan gruppmedlemmarna.
- ☐ Den som är huvudansvarig för en viss del redovisar den delen.
- ☐ Ni ska ta fram en gruppgemensam checklista för kodgranskning. Varje gruppmedlem ska granska minst en annan gruppmedlems kod enligt checklistan.
- ☐ Grupplaborationen görs över **två veckor** uppdelat på två delredovisningar. Vid första redovisningen ska arbetsupplägget och pågående utveckling redovisas. Vid andra tillfället ska de färdig lösningarna presenteras av respektive huvudansvarig individ.
- ☐ Vid första redovisningen ska du redogöra för handledaren hur ni delat upp koden och vem som är huvudansvarig för vad och vad ditt ansvar omfattar, samt hur ni jobbar praktiskt med att synkronisera er utveckling.
- ☐ Grupplaborationen är en **extra stor uppgift** och grupparbetet behöver ledtid för att ni ska hinna koordinera er sinsemellan. Du behöver därför planera för att arbeta med något i grupplabben i stort sett varje dag under de tillgängliga veckorna, och vara redo att bidra i diskussioner.
- ☐ Träffas i din samarbetsgrupp och diskutera ert arbetssätt utifrån följande frågor:
  - Vilka krav ska ni implementera?
  - Hur ska ni jobba med gemensamma koddelar?
  - Hur ska ni dela med er av de koddelar som ni utvecklar var för sig?

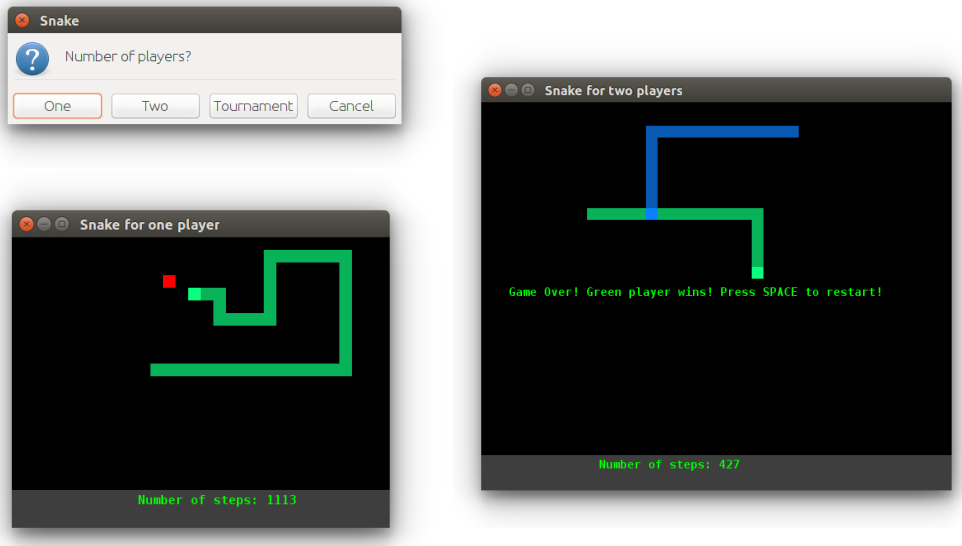
10.3.1 Bakgrund

Spelet *Snake*[7] blev mäkta populärt i Sverige redan på 1980-talet, ofta spelat på den legendariska datorn ABC80. Spelet finns i flera varianter, både för en spelare och som duell mellan två spelare. Varje spelare styr en mask med huvud och svans som hela tiden rör sig framåt. Det gäller att undvika att köra in i en masksvans och att samla poäng t.ex. genom att äta äpple.

Figur 10.2 visar en startdialog där man kan välja antal spelare, samt ett exempel på spel med en och två spelare. I varianten för en spelare närmar sig maskens huvud äpplet och lyckas kanske äta det om spelaren styr rätt. I varianten för två spelare

[7] Även kallat ”masken”. https://sv.wikipedia.org/wiki/Snake

vinner grön mask eftersom den blåa masken råkade köra in i den gröna maskens svans.



Figur 10.2: Spelet snake för en spelare med äpple och för två spelare utan äpple.

10.3.2 Obligatoriska funktionella krav

Följande funktionella krav ska uppfyllas av ert program om ni är sex personer i gruppen. Om ni är färre ingår de obligatoriska krav som visas i tabell 10.1.

- ☐ **`Player`**. Det ska finnas spelare som motsvarar mänskliga användare och som har ett namn och fyra tangenter som den kan spela med. Varje spelare har en egen orm som den kan styra med sina tangenter.
- ☐ **`Snake`**. Det ska finnas ormar. En orm består av ett antal block, där det främsta blocket kallas huvud och resten av blocken kallas svans. Huvudet har en ljusare färg än kroppen. Svansens längd ökar under spelets gång. En orm rör sig i en viss riktning och varje spelare kan ändra riktningen på sin orm med sina tangenter, i en av fyra riktningar `North`, `South`, `East` eller `West`.
- ☐ **`Apple`**. Det ska finnas (minst ett) äpple. Ett äpple består av ett rött block och finns på en slumpvis position. Ett äpple kan ätas av en orm om ormens huvud träffar äpplet. Varje gång ett äpple äts upp av en orm så teleporteras äpplet till en ny position och kan ätas igen.
- ☐ **`Banana`**. Det ska finnas (minst en) banan. En banan består av tre vertikala gula block och finns på en slumpvis position. En banan äts upp av en orm om ormens huvud träffar bananen. Varje gång en banan äts upp av en orm så teleporteras bananen till en ny slumpvis position och kan ätas igen.
- ☐ **`Monster`**. Det ska finnas (minst ett) monster. Ett monster består av fem rosa block i kryssform. Ett monster föds på en slumpvis position och rör sig i en riktning som bestäms vid monstrets födelse. Ett orm blir uppäten och dör om ormens huvud nuddar ett monsterblock .
- ☐ **`OneplayerGame`**. Det ska gå att spela ensam. I varianten med en spelare finns en orm och minst ett äpple (och ev. även bananer och monster). Varje gång användarens orm lyckas äta en frukt får användaren poäng. När ormen ätit ett visst antal äpplen, eller om ormen blivit uppäten av ett monster, är spelet slut

och poängen visas. En ormsvans ska bli längre vid jämna tidsintervall eller om den äter frukt.

- ☐ **`TwoplayerGame`**. Det ska gå att spela två och två. I varianten med två spelare finns två ormar. Det finns också äpplen, bananer och monster. Om en orm äter en banan blir dess svans längre. När ormen ätit ett visst antal äpplen, eller om ormen blivit uppäten av ett monster, är spelet slut och poängen visas. En ormsvans ska bli längre vid jämna tidsintervall eller om den äter frukt.
- ☐ **`Settings`**. Inställningar för spelet ska vara konfigurerbara genom en textfil som laddas i början av spelet. Inställningar ska vara en kontextparameter.

Tabell 10.1: Krav som minst ska implementeras vid respektive gruppstorlek. Om du har särskilda skäl kan du efter godkännande från kursansvarig göra labben enskilt.

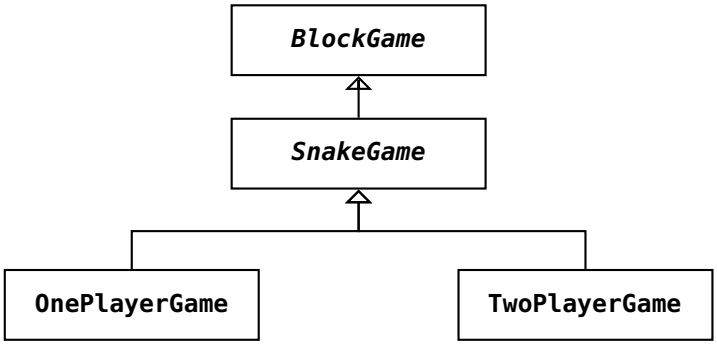
Krav / Antal personer	1	2	3	4	5	6
`Player`	√	√	√	√	√	√
`OnePlayerGame`	√				√	√
`TwoPlayerGame`		√	√	√	√	√
`Snake`	√	√	√	√	√	√
`Apple`	√		√	√	√	√
`Banana`				√		√
`Monster`			√			√
`Settings`	√	√	√	√	√	√

10.3.3 Obligatoriska design-krav

- ☐ Snake-spel ska gå att starta med huvudprogrammet nedan. Huvudprogrammet får ändras vid behov i enlighet med minimikrav vad gäller gruppstorlek i tabell 10.1, samt valbara extrakrav i avsnitt 10.3.4, och era egna ideer.

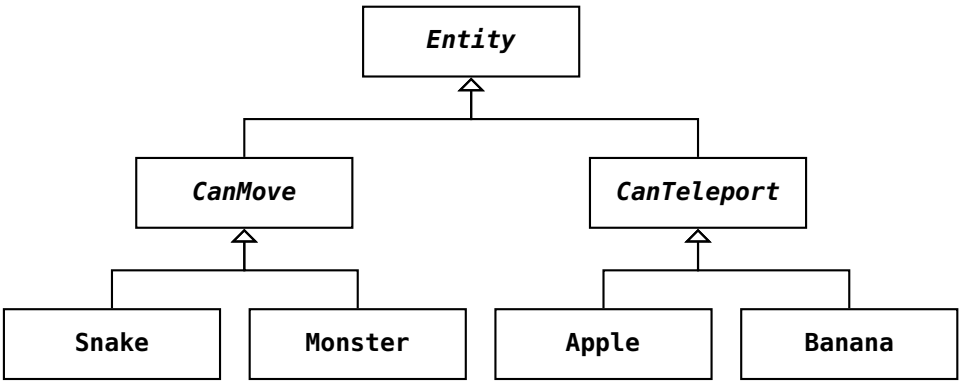
```
package snake

def createOnePlayerGame(): Unit =
  Settings.default.windowTitle = "Snake: One Player"
  OnePlayerGame().play()

def createTwoPlayerGame(): Unit =
  Settings.default.windowTitle = "Snake: Two Player"
  TwoPlayerGame().play()

@main
def run: Unit =
  val buttons =
    Seq("One", "Two", "Competition", "Tournament", "Cancel")
  val selected =
    introprog.Dialog.select("Number of players?", buttons, "Snake")
  selected match
    case "One"         => createOnePlayerGame()
    case "Two"         => createTwoPlayerGame()
    case "Competition" => println(s"TODO: $selected")
    case "Tournament"  => println(s"TODO: $selected")
    case  _            => println("Goodbye!")
```

☐ Spelet ska bygga vidare på `introprog.BlockGame` enligt typhierarkin i fig. 10.3.



Figur 10.3: Arvshierarki med klassen `introprog.BlockGame` som bastyp.

☐ Ormar, monster och frukt ska utgå från bastypen `Entity` enligt typhierarkin i 10.4.



Figur 10.4: Arvshierarki med klassen `Entity` som bastyp.

☐ `Entity` representerar en varelse i ett spel och ska se ut så här:

```
package snake

trait Entity:
  def draw(): Unit

  def erase(): Unit

  def update(): Unit

  def reset(): Unit

  infix def isOccupyingBlockAt(p: Pos): Boolean
```

Metoderna `draw` resp. `erase` anropas vid ritning resp. radering. Metoden `reset` återställer ursprungstillståndet. Metoden `update` anropas en gång i varje runda i spel-loopen. Predikatet `isOccupyingBlockAt` ger sant om positionen `p` finns bland de block som varelsen ockuperar på skärmen.

☐ `CanMove` representerar en entitet som kan röra sig i en viss hastighet, enligt:

```
package snake

trait CanMove extends Entity:
  def move(): Unit

  var movesPerSecond: Double = 20.0

  final def millisBetweenMoves: Int =
    (1000 / movesPerSecond).round.toInt max 1

  private var _timestampLastMove: Long = System.currentTimeMillis

  final def timestampLastMove = _timestampLastMove

  override final def update(): Unit =
    // flytta om tiden har gått millisBetweenMoves
    if System.currentTimeMillis >
         _timestampLastMove + millisBetweenMoves
    then
      _timestampLastMove = System.currentTimeMillis
      move()
```

☐ `CanTeleport` representerar en entitet som finns på en viss plats men som efter ett visst antal uppdateringar utan förvarning teleporterar sig till en ny position:

```
package snake

trait CanTeleport extends Entity:
  private var _pos = teleport()

  def pos: Pos = _pos

  protected var nbrOfStepsSinceLastTeleport = 0

  def teleportAfterSteps: Int

  def teleport(): Pos

  def update(): Unit =
    nbrOfStepsSinceLastTeleport += 1
    if nbrOfStepsSinceLastTeleport > teleportAfterSteps
    then reset()

  def reset(): Unit =
    nbrOfStepsSinceLastTeleport = 0
    _pos = teleport()
```

☐ Det ska finnas en enumeration `State` i singelobjektet `SnakeGame` som representerar spelets övergripande tillstånd enligt följande:

```
package snake

object SnakeGame:
  enum State:
    case Starting, Playing, GameOver, Quitting
  export State.* // gör alla tillstånd synliga i SnakeGame
```

☐ Vid varje runda i spelloopen ska följande logik exekveras. Denna kod placeras förslagsvis i `gameLoopAction`, se vidare `SnakeGame` i avsnitt 10.3.5.

```
    if state == Playing && !isPaused then
      _iterationsSinceStart += 1
      entities.foreach(_.erase())
      entities.foreach(_.update())
      entities.foreach(_.draw())
      onIteration()
      if isGameOver then enterGameOverState()
```

☐ Det ska finnas ett singelobjekt `Colors` där alla färger som används i spelet samlas.

☐ Filen `pairs.scala` ska enligt laborationsförberedelser i övningsuppgift 4 på sidan 95 innehålla `Pair[T]`, `Dim`, `Pos`, `Dir`, `North`, `South`, `East`, `West`. Se workspace här: https://github.com/lunduniversity/introprog/tree/master/workspace/

☐ Klassen `Player` ska se ut som följer:

```
package snake

class Player(
  var name: String,
  var keyMap: Player.KeyMap,
  val snake: Snake,
  var points: Int = 0,    // TODO: count points when e.g. eating apple
):
  def handleKey(key: String): Unit =
    ??? // om key ingår i keyMap så uppdatera snake.dir

object Player:
  enum KeyMap(left: String, right: String, up: String, down: String):
    val dir = Map(left -> West, right -> East, up -> North, down -> South)
    case Letters extends KeyMap("a", "d", "w", "s")
    case Arrows extends KeyMap("Left", "Right", "Up", "Down")
```

10.3.4 Valbara krav – varje person ska välja minst ett

Varje person i gruppen ska implementera *minst ett* (gärna flera) av kraven nedan. Vid implementation av flera av dessa krav blir spelet väsentligt roligare.

☐ **`Points`**. Inför ett poängsystem, där poängen beror på t.ex. längden på svansen, antalet steg, antalet svängar, antal uppätna äpplen, etc.

- ☐ **`Highscore`**. Spelet ska visa en lista med de spelare som fått flest poäng.
- ☐ **`Äpple`**. Om inte redan ingår bland obl. krav enl. 10.1.
- ☐ **`Monster`**. Om inte redan ingår bland obl. krav enl. 10.1.
- ☐ **`Banan`**. Om inte redan ingår bland obl. krav enl. 10.1.
- ☐ **`SelfTailCrash`**. Om en spelare kör in i sin egen orms svans så är spelet förlorat. (Om detta krav ej implementeras så *får* man köra igenom sin egen svans utan att något händer.)
- ☐ **`BoundaryCrash`**. Om en spelare kör utanför spelplanen så är spelet förlorat. (Om detta krav ej implementeras så ska ormen fortsätta på andra sidan spelplanen när man når kanten.)
- ☐ **`EnterPlayerName`**. Spelare kan ange sitt namn, t.ex. via en dialog eller genom argument till `main`. Namnet används i meddelandefältet vid poängräkning och i meddelanden om vem som vunnit.
- ☐ **`OnePlayerGame`**. Du kan välja att implementera `OnePlayerGame` om det inte redan ingår i de obligatoriska kraven.
- ☐ **`TwoPlayerComp extends Competition`**. Två spelare ska kunna tävla i en bäst-av-n-matcher-tävling i en sekvens av `TwoPlayerGame.play`, där den som vinner flest matcher blir blir totalvinnare.
- ☐ **`SinglePlayerComp extends Competition`**. Flera spelare ska kunna tävla i en-persons-Snake, där den som får flest poäng av n `OnePlayerGame`-spel blir totalvinnare.
- ☐ **`Tournament extends Competition`**. Många spelare ska kunna spela en turnering.[8] Namnen på spelarna läses in från en textfil. Valbara varianter:
  - ☐ **`KnockOut extends Tournament`**. Det ska gå att spela en utslagsturnering, som avslutas med final efter semi-final, etc., beroende på antal spelare.
  - ☐ **`RoundRobin extends Tournament`**. Det ska gå att spela en alla-möter-alla-turnering, där alla möjliga par av spelare möts i slumpvis ordning.

10.3.5 Tips och förslag

I detta stycke presenteras skisser till några av de klasser som behövs i enlighet med designkraven. Det är tillåtet att ändra, ta bort och lägga till, så länge de obligatoriska designkraven uppfylls. Koden finns här:
https://github.com/lunduniversity/introprog/tree/master/workspace/

Här följer en skiss på klassen `Snake`:

```
package snake

import java.awt.Color

class Snake (
  val initPos: Pos,
  val initDir: Dir,
  val headColor: Color,
  val tailColor: Color,
 )(using ctx: SnakeGame, settings: Settings) extends CanMove:
  var dir: Dir = initDir
  val initBody: List[Pos] = List(initPos + initDir, initPos)
  val body: scala.collection.mutable.Buffer[Pos] = initBody.toBuffer

  val initLength: Int = settings.snake.initLength
```

[8] https://en.wikipedia.org/wiki/Tournament

```
  val growEvery: Int = settings.snake.growEvery
  val startGrowingAfter: Int = settings.snake.startGrowingAfter

  private var _nbrOfSteps = 0
  def nbrOfSteps: Int = _nbrOfSteps

  private var _nbrOfApples = 0
  def nbrOfApples: Int = _nbrOfApples

  def reset(): Unit = ???  // återställ starttillstånd, ge rätt svanslängd

  def grow(): Unit = ??? // växa i rätt riktning med extra svansposition

  def shrink(): Unit = ??? // krymp svansen om kroppslängden är större än 2

  def isOccupyingBlockAt(p: Pos): Boolean = ??? // kolla om p finns i kroppen

  def isHeadCollision(other: Snake): Boolean = ??? // kolla om huvudena krockar

  def isTailCollision(other: Snake): Boolean = ??? // mitt huvud i annans svans

  private var _isEatenByMonster: Boolean = false
  def isEatenByMonster: Boolean = _isEatenByMonster
  def eatenByMonster(): Unit = ???

  def move(): Unit = ???
    // växa och krymp enl. regler
    // åtgärder om äter frukt eller blir uppäten av monster

  override def toString = // bra vid println-debugging
    body.map(p => (p.x, p.y)).mkString(">:)", "~", s" going $dir")

  def draw(): Unit = ???

  def erase(): Unit = ???
```

Här följer en skiss på den abstrakta klassen `SnakeGame` med de abstrakta metoderna `isGameOver` och `play` som överskuggas i de efterföljande underklasserna `OnePlayerGame` och `TwoPlayerGame`:

```
package snake

object SnakeGame:
  enum State:
    case Starting, Playing, GameOver, Quitting
  export State.*

abstract class SnakeGame(settings: Settings) extends introprog.BlockGame(
  title                = settings.windowTitle,
  dim                  = settings.windowSize,
  blockSize            = settings.blockSize,
  background           = settings.background,
  framesPerSecond      = settings.framesPerSecond,
  messageAreaHeight    = settings.messageAreaHeight,
  messageAreaBackground = settings.messageAreaBackground
):
  // exempel på olika synlighet (diskutera val av synlighet utifrån användning)
```

```
var entities: Vector[Entity] = Vector.empty
protected var players: Vector[Player] = Vector.empty
private var isPaused = false

import SnakeGame.*

protected var state: State = Starting
private var _iterationsSinceStart = 0
def iterationsSinceStart = _iterationsSinceStart

def enterStartingState(): Unit = ??? //sudda, meddela "tryck space för start"

def enterPlayingState(): Unit = ??? //sudda, för varje entitet: nollställ & rita

def enterGameOverState(): Unit = ??? // meddela "game over"

def enterQuittingState(): Unit =
  println("Goodbye!")
  pixelWindow.hide()
  state = Quitting

def randomFreePos(): Pos =
  ??? // dra slump-pos tills ledig plats, används av frukt, monster

override def onKeyDown(key: String): Unit =
  println(s"""key "$key" pressed""")
  state match
    case Starting => if key == " " then enterPlayingState()

    case Playing =>
      if key == "Esc" then
        println(s"Toggle pause: isPaused == $isPaused")
        isPaused = !isPaused
      else
        players.foreach(_.handleKey(key))

    case GameOver =>
      if key == " " then enterPlayingState()
      else if key == "Esc" then enterQuittingState()

    case _ =>

override def onClose(): Unit =
  println("Window Closed!")
  enterQuittingState()

/** Implement this with logic for when to end the game */
def isGameOver: Boolean

/** Override this if you want to add game-logic in gameLoopAction
 *  Call super.onIteration() if you want to keep the step counter.
 */
def onIteration(): Unit =
  clearMessageArea()
  drawTextInMessageArea(s"Number of steps: $iterationsSinceStart", 10, 2)

override def gameLoopAction(): Unit =
```

```
    if state == Playing && !isPaused then
      _iterationsSinceStart += 1
      entities.foreach(_.erase())
      entities.foreach(_.update())
      entities.foreach(_.draw())
      onIteration()
      if isGameOver then enterGameOverState()

  final def start(ps: Player*)(es: Entity*): Unit =
    players = ps.toVector
    entities = es.toVector
    isPaused = false
    pixelWindow.show()  // möjliggör omstart även om fönstret stängts...
    enterStartingState()
    gameLoop(stopWhen = state == Quitting)

  /** Implement this with a call to start with specific players and entities. */
  def play(playerNames: String*): Unit
```

Om gruppen funderar på att använda git och github:

- Diskutera i gruppen om alla har kunskaper nog för att köra git och github, samt för- och nackdelar med det.
- Om inte alla är bekväma med git och github så överväg om ni vill göra manuell versionshantering med kopiering av nya filer via USB-minne, ssh eller upp- och nedladdning via molnlagring. Efter en konkret upplevelse av manuell versionshantering så får du en djupare förståelse för behovet av verktygsstöd för versionshantering och det blir extra motiverande att lära sig git.
- Diskutera arbetssätt. Hur ska ni använda github issues, git branch, etc? Eller ska alla pusha till main branch? Ska ni använda github pull requests, github reviews, etc.?
- Kolla så att du har en `.gitignore` innan du gör push, så att inte t.ex. maskinkodsfiler hamnar i ert repo, vilket kan medföra knepigt städjobb och onödiga merge-konflikter. Exempel på en lämplig `.gitignore` finns här: https://github.com/lunduniversity/introprog/blob/master/workspace/w10_snake/.gitignore
- **Var noga med att göra ert github-repo privat!** Det är inte tillåtet att dela labblösningar på internet – då kan du efter disciplinärende dömas som skyldig till medhjälp till fusk och du kan bli avstängd från dina studier.

Kapitel 11

Varians och kontextparametrar

Begrepp som ingår i denna veckas studier:

- ☐ övre- och undre typgräns
- ☐ varians
- ☐ kontravarians
- ☐ kovarians
- ☐ typjoker
- ☐ kontextgräns
- ☐ typkonstruktor
- ☐ egentyp
- ☐ typjoker
- ☐ givet värde (given)
- ☐ kontextparameter (using)
- ☐ ad hoc polymorfism
- ☐ typklass
- ☐ api
- ☐ kodläsbarhet
- ☐ granskningar

11.1 Teori

11.1.1 Typparameter, generisk struktur, typkonstruktor

- Med hjälp av **typparametrar** (eng. *type parameters*) kan du skapa **generiska strukturer** (eng. *generic structures*).
- Typparametrar skrivs inom hakparenteser, exempelvis: `[T, U]`
- En generisk struktur fungerar för *godtycklig* typ, **okänd** vid **deklaration**.
- Kompilatorn säkerställer **korrekt användning** redan vid *kompilering*.
- På användningsplatsen i koden (vid anrop eller instansiering) **binds** typparametrar till ”verkliga” typer enligt typsystemets regler.

```
case class Pair[A, B](a: A, b: B):  // A och B är obundna (fria) inom []
  def swap: Pair[B, A] = Pair(b, a) // A och B bundna då swap saknar []
```

- Skriver du inte ut typerna försöker kompilatorn **härleda** (eng. *infer*) dem:

```
scala> val p = Pair("hej", 42)
val p: Pair[String, Int] = Pair(hej,42)

scala> p.swap
val res0: Pair[Int, String] = Pair(42,hej)
```

- Klassen `Pair` kallas **typkonstruktor** (eng. *type constructor*) då den ”färdiga” typen `Pair[String, Int]` ”konstrueras” vid användning.

Övning: Ändra klassen `Pair` ovan så att båda elementen i paret har samma typ.

11.1.2 Olika sätt att begränsa generiska typer

Det finns i Scala flera olika sätt att att begränsa vilka typer du vill tillåta – kompilatorn att hjälper dig att kontrollera detta.

- **Övre gräns** (eng. *upper bound*) med `A <: B`
- **Undre gräns** (eng. *lower bound*) med `A >: B`
- **Egentyp** (eng. *self type*) begränsar hur du kan mixa in en trait.
  Ge ett godtyckligt namn följt av en typanotering och en raket i klasskroppen:

```
trait MinTrait:
  self: ÄrDennaTyp =>

  // Kod här kan utgå från att denna instans är av typen ÄrDennaTyp
  // namnet, här self, är valfritt, men self är vanligaste valet
```

  Kompilatorn kontrollerar att inmixing sker med `ÄrDennaTyp`.
- Det finns också något som heter **kontextgräns** (eng. *context bound*) där `[A: B]` gör så att typkonstruktorn `B` blir en kontextparameter `B[A]`
  – mer om det senare i avsnittet om kontextuella abstraktioner.

11.1.3 Övre och undre typgräns

Med typoperatorerna `<:` och `>:` går det att begränsa vilka typer som kan bindas till en typparameter i en generiska struktur.

- Minnesregel för typgränser: **kolon på slutet**.

Antag att `T` är en **obunden** typparameter, medan `U` och `L` är **bundna**.

- med `T <: U` blir `U` en övre gräns (eng. *upper bound*) för `T`
  `U` är ”högsta” möjliga typ som `T` får vara (jämför ”mindre eller lika med”)
- med `T >: L` blir `L` en undre gräns (eng. *lower bound*) för `T`
  `U` är ”lägsta” möjliga typ som `T` får vara (jämför ”större eller lika med”)
- För alla typer `A` gäller att `A >: Nothing` och `A <: Any`

Exempel:

```
trait Grönsak { def vikt: Int }

def f[T <: Grönsak](x: T): Int = x.vikt
```

Kompilatorn använder den övre typgränsen för att konstatera att metoden `vikt` är tillgänglig via den generiska parametern `x`.

11.1.4 Exempel på övre och undre typgräns

```
class Djur
class Katt extends Djur
class Hund extends Djur
class Robothund extends Hund

def testUpperBound[T <: Hund](x: T) = println(x)
def testLowerBound[T >: Hund](x: T) = println(x)
```

```
scala> testUpperBound[Katt](Katt())
-- Error:
1 |testUpperBound[Katt](Katt())
  |               ^
  |               Type argument Katt does not conform to upper bound Hund

scala> testLowerBound[Robothund](Robothund())
-- Error:
1 |testLowerBound[Robothund](Robothund())
  |               ^
  |               Type argument Robothund does not conform to lower bound Hund
```

11.1.5 Vad är varians?

Är en kattbur också en djurbur??

Om vi tillåter **varians** så blir generiska strukturer mer **flexibla**.

11.1.6 Varför behövs varians?

```
trait Djur
case class Katt() extends Djur
case class Hund() extends Djur

case class Bur[A](a: A)
```

Nedan fungerar inte! Buren ovan är **invariant** (oflexibel i sin typparameter).

```
scala> val djurbur: Bur[Djur] = Bur[Katt](Katt())
-- Error:
1 |val djurbur: Bur[Djur] = Bur[Katt](Katt())
  |                  ^^^^^^^^^^^^^^^^
  |                  Found:    Bur[Katt]
  |                  Required: Bur[Djur]
```

Varför fungerar detta??

```
scala> val djur: Vector[Djur] = Vector[Katt](Katt())
val djur: Vector[Djur] = Vector(Katt())
```

`Vector` är deklarerad som **kovariant** och därmed mer flexibel!

11.1.7 Kovarians (eng. *covariance*)

- För en **kovariant** typkonstruktor `F` gäller att:
  om `T <: U` **så** `F[T] <: F[U]` (subtypsflexibel ”på samma håll”)

- I Scala kan du få en kovariant typkonstruktor med + före typparametern:

```
trait Djur
case class Katt() extends Djur
case class Hund() extends Djur

case class Bur[+A](a: A)  // kovariant tack vare + före A
```

- Nu funkar det :)

```
scala> val djurbur: Bur[Djur] = Bur[Katt](Katt())
val djurbur: Bur[Djur] = Bur(Katt())
```

- `Bur[Katt]` är nu en **subtyp** till `Bur[Djur]`.
- **Oföränderliga** samlingar är ofta kovarianta, t.ex Vector, Option, List.
- Generiska enumerationer **behöver** vara kovarianta för att de olika fallen ska få en flexibel typparameter. `Ledig` är en `Toalett[Nothing]` här:

```
enum Toalett[+T]:
  case Upptagen(x: T)
  case Ledig
```

11.1.8 Kontravarians

Är en kattveterinär också en djurveterinär?

Ibland vill vi ha variansen på andra hållet: En veterinär som bara kan behandla katter ska inte få behandla vilket djur som helst.

11.1.9 Kontravarians (eng. *contravariance*)

- För en **kontravariant** typkonstruktor `F` gäller att:
  om `T <: U` **så** `F[U] <: F[T]` (subtypsflexibel ”på fel håll”)

- Du skapar en kontravariant typkonstruktor med - före typparametern:

```
trait Djur
case class Katt() extends Djur
case class Hund() extends Djur

class Veterinär[-A]:    // kontravariant med -
  def behandla(x: A) = println(s"$this har behandlat $x")
```

- Nu funkar det ”baklänges” (men inte på andra hållet):

```
scala> val kattveterinär: Veterinär[Katt] = Veterinär[Djur]()
val kattveterinär: Veterinär[Katt] = Veterinär@77b6d94c

scala> val kattveterinär: Veterinär[Djur] = Veterinär[Katt]()
-- Error:
1 |val kattveterinär: Veterinär[Djur] = Veterinär[Katt]()
  |                                     ^^^^^^^^^^^^^^^^^
  |                                     Found:    Veterinär[Katt]
  |                                     Required: Veterinär[Djur]
```

11.1.10 Variansproblem – tack kompilatorn!

- En typkonstruktor kan **inte** vara kovariant om typparametern används som parametertyp för metoder (så kallad *kontravariant position*):

```
trait Djur
case class Katt() extends Djur
case class Hund() extends Djur
```

```
scala> case class Bur[+A](a: A):
         def bytTill(x: A): Bur[A] = Bur(x)
-- Error:
2 |  def bytTill(x: A): Bur[A] = Bur(x)
  |          ^^^^
  |  covariant type A occurs in contravariant position in type A of parameter x
```

- Här måste typen tillåtas variera ”uppåt” med hjälp av en **undre gräns**:

```
case class Bur[+A](a: A):
  def bytTill[B >: A](x: B): Bur[B] = Bur(x)
```

```
scala> Bur[Katt](Katt()).bytTill(Hund())
val res1: Bur[Djur] = Bur(Hund())
```

11.1.11 När använda vilken slags varians?

- Oföränderliga generiska klasser som har metoder som är **producenter** av nya oföränderliga generiska typer passar ofta bäst som **kovarianta**, t.ex. `Vector[+T]`, `Option[+T]` .
- En typkonstruktor av `T`, som har metoder som är **konsumenter** av `T`, kan vara **kontravariant**, t.ex. `Veterinär[-T]`.
  Den kan också vara **kovariant** om konsumerande metoder **vidgar** inparametertypen till `B` där `B >: T`
- En typkonstruktor med flera parametrar kan vara **både** kontravariant **och** kovariant, t.ex. `Function1[-A, +B]`
  ett parametervärde `:A` *konsumeras* och ett returtvärde `:B` *produceras*
  (`Function1[-A, +B]` är egentliga typen av ”syntaktiska sockret” `A => B`
- **Förändringsbara** strukturer måste vara **invarianta**, annars väntar **kaos**!
  Antag att det finns en `Bur` som kan ändras på plats via metoden `bytTill` *och samtidigt* vore kovariant (varning för känsliga kodare):

```
ejscala> val kattBur: Bur[Katt] = Bur(Katt())

ejscala> val djurBur: Bur[Djur] = kattBur  // en kovariant referens till samma Bu

ejscala> djurBur.bytTill(Hund())           // ändra på plats

ejscala> val katt: Katt = kattBur.släppUt  // KAOS! typosäkert hundkatt-monster :
```

11.1.12 Typjoker: varning för gränslösa typer

Invarianta klasser har oflexibel typparameter, vilket begränsar användningen.

```
1 scala> class Box[A](val value: A)
2
3 scala> val b: Box[Any] = Box[Int](42)
4 -- Error:
5 1 |val b: Box[Any] = Box[Int](42)
6   |                  ^^^^^^^^^^^
7   |                  Found:    Box[Int]
8   |                  Required: Box[Any]
```

Tveksam räddning: En **typjoker** (eng. *wildcard type*) skrivs med ett frågetecken och ger en helt okänd typ som **ej kontrolleras av kompilatorn**.

```
1 scala> var whatever: Box[?] = Box[Int](42)
2 var whatever: Box[?] = Box@70d7a49b
3
4 scala> whatever = Box("hej")
5 whatever: Box[?] = Box@43120a77
```

Använd bara typpjoker om det *verkligen* behövs.

11.1.13 Mer om varians för den nyfikne

- https://docs.scala-lang.org/scala3/book/types-variance.html
- https://en.wikipedia.org/wiki/Covariance_and_contravariance_(computer_science)
- https://www.artima.com/pins1ed/type-parameterization.html

11.1.14 Egentyp + anonym klass för att ”injektera” beroenden

- Det går att få ett explicit, valfritt namn, t.ex. `self`, på ”mig själv”, alltså denna instans, genom att skriva `self =>` i kroppen på en trait eller klass.
- En typbegränsning på den egna instansen kallas **egentyp** (eng. *self type*).
- Användbart vid inmixning: du kan begränsa med vad denna trait får mixas in.

```
trait Grönsak { def vikt: Int }

trait KanSkalas:
  self: Grönsak =>
  def viktEfterSkalning = 0.99 * vikt
```

- Notera att *även* om `KanSkalas` *inte* gör **`extends`** så är *ändå* `vikt` tillgänglig i dess kropp, eftersom vi med egentypen kräver att `KanSkalas` är en `Grönsak`.
- Du kan du kombinera anonym klass och dynamisk inmixning med **`with`**:

```
scala> val g = new Grönsak with KanSkalas { val vikt = 100 }
val g: Grönsak & KanSkalas = anon1@70a91d72

scala> g.viktEfterSkalning
val res0: Double = 99.0
```

- Detta är ett elegant sätt att **injektera beroenden** (eng. *dependency injection*). https://en.wikipedia.org/wiki/Dependency_injection

---

11.1.15 Vad är fördelen med egentyper i stället för arv?

Arv tillåter **inte** ömsesidiga (cykliska) beroenden...

```
scala> trait Tulpan extends Ros; trait Ros extends Tulpan
-- [E110] Syntax Error: ----------------------------------------------------------
1 |trait Tulpan extends Ros; trait Ros extends Tulpan
  |                      ^^^
  |                      Cyclic inheritance: trait Tulpan extends itself
  |
```

...medan egentyper *kan* vara **ömsesidigt beroende**:

```
trait Tulpan:
  self: Ros =>

trait Ros:
  self: Tulpan =>
```

```
scala> val tulipanaros = new Tulpan with Ros
val tulipanaros: Tulpan & Ros = anon1@1c63b39a
```

https://sv.wikipedia.org/wiki/Tulipanaros

11.1.16 Vad är ett bra api?

- Ett api (eng. *Application Programming Interface*) är ett gränssnitt för applikationsprogrammering.
- Med ”gränssnitt” menas att det (i koden) finns en gräns mellan vad som syns utåt och vad som finns innanför ”under huven”.
- Det finns många kvalitetsaspekter som är önskvärda för ett api:
  - Enkelt att använda på utsidan även om insidan är komplex.
  - Vadå ”enkelt att använda”?
    - Lätt att lära
    - Lätt att komma ihåg
    - Lätt att begripa
    - Najs upplevelse (subjektivt)
  - Löser ett (generellt) problem på ett bra sätt. Vadå ”bra”?
    - Snabbt (hög prestanda)
    - Snålt (effektiv användning av resurser, t.ex. minne)
    - ...
- Intressant föredrag av Joshua Bloch (Google), om god api-design:
  - https://research.google.com/pubs/archive/32713.pdf
  - https://youtu.be/aAb7hSCtvGw

11.1.17 Api-desgin med Scala

- Kombinera paradigm OO+FP för att välja bäst lämpade lösningen.
- Avancerade abstraktionsmekanismer som kan vara utmanande för api-konstruktören men samtidigt bli enkelt för api-användaren
- Gör det möjligt att skapa ett api som fungerar i flera körmiljöer:
  - På desktop och i back-end med JVM och Graal VM
  - I front-end i webbläsaren med Scala JS
  - Direkt till plattformsspecifik maskinkod med Scala Native
- Avancerade saker som vi inte gått in på i kursen men som kan hjälpa api-konstruktörer att göra lättanvänt api:
  - Kontextfunktioner `?=>`
  - Opaka typer `opaque type`
  - Typklassderivering `derives`
  - Metaprogrammering `inline`
  - Typ-lambda och match-typer `=>>`
  - ... (forskning pågår)

11.1.18 Sammanhanget är avgörande när du kodar!

- **Kontexten**, alltså sammanhanget, styr vilka namn som syns var.
- **Objektorientering** (OO) skapar sammanhang med:
  - **Namnrymder**
  - **Tillstånd**
  - **Subtypspolymorfism**
- **Funktionsprogrammering** (FP) skapar sammanhang med:
  - **Parametrar**
  - **Funktionsvärden**
  - **Parametrisk polymorfism**
- **Kombinationen** av OO och FP är **extra** kraftfull och flexibel!
- Men det finns **svårigheter**:
  - OO: ibland svårt att resonera om ändrade tillstånd och dynamisk bindning
  - FP: kan bli många parametrar som måste upprepas vid nästlade anrop
- Till vår räddning: fler **coola grejer** i Scala:
  - **Kontextparametrar**: möjliggör att **givna** (implicita) värden kan framkallas automatiskt av kompilatorn.
  - **Ad hoc polymorfism**: möjliggör olika implementationer beroende på **statisk** typ utan att det krävs en arvsrelation eller dynamisk bindning.

11.1.19 Repetition: default-argument

- Vi har tidigare sett att kompilatorn, med **default-argument**, kan **fylla i** värden, som vi inte måste skriva, vid funktionsanrop:

```
scala> def f(x: Int, y: Int = 42) = x + y
def f(x: Int, y: Int): Int

scala> f(1, 2)        // explicit y-värde
val res0: Int = 3

scala> f(1)          // vi kan också skippa y-värdet
val res1: Int = 43   // då används default-argumentet
```

11.1.20 Repetition: uppdelade parameterlistor

- Vi har tidigare sett uppdelade parameterlistor, som möjliggör stegvis applicering (s.k. Curry-funktioner):

```
scala> def f(x: Int)(y: Int = 42) = x + y
def f(x: Int)(y: Int): Int

scala> val g = f(1)  // en ny funktion utan default-arg
val g: Int => Int = Lambda1352/0x08406d0840@dbc7e0a
```

```
scala> f(1)()   // y-värdet fylls i av kompilatorn
val res4: Int = 43
```

- Kan vi ta detta ett steg till och **frikoppla** deklarationen av funktionen **från** det givna värdet, och ge detta på annan plats baserat på typen?
- JA! I Scala 3 görs detta med **givna** värden och **kontextparametrar**, som skapas med nyckelorden **given** respektive **using**
  (i Scala 2 användes gamla nyckelordet **implicit**)

11.1.21 Givna värden + kontextparameter

Exmpel på användning av **given** och **using**:

```
case class Default(value: Int)

object Default:
  given d: Default = Default(0)  // Använd detta om inget annat givet värde finns lokalt
```

```
scala> def f(x: Int)(using d: Default) = x + d.value

scala> f(1)(using Default(2))    //explicit värde används alltid först om sådant finns
val res0: Int = 3

scala> f(1)  // kompilatorn framkallar ett givet värde för Default ur kompanjonsobjekte
val res1: Int = 1

scala> given d: Default = Default(42)  // låt d vara givna värdet i denna kontext
lazy val given_Default: Default

scala> f(1)       // kompilatorn framkallar nu det givna värdet i denna lokala kontext
val res2: Int = 43
```

Man kan utelämna namnet på värdet, eftersom det oftast inte behövs:
`given Default = Default(42)` eller ännu kortare: `given Default(42)`

11.1.22 Går det inte lika bra att ha en global variabel?

Varning för känsliga kodare!

- Ett försök att skapa ett default-värde med en global **var**-variabel:

```
scala> var globalVar = Default(42)
var ctx: Int = 42

scala> def f(x: Int, d: Default = globalVar) = x + d.value

scala> f(1)
val res3: Int = 43

scala> globalVar = Default(0) // FÖRÄNDRINGEN SLÅR GLOBALT I HELA DITT PROGRAM!!

scala> f(1)
val res4: Int = 1
```

- Här utgörs ”kontexten” av referensen till ett föränderligt värde som bestäms vid **körtid** i den globala namnrymden. Funktionen `f` är ej äkta!
- Denna lösning kan **inte** erbjuda **olika** kontextberoende default-värden; alla funktionsanrop använder **ett** globalt föränderliga värde. **Buggrisk**!

Givna värden med **`given`** härleds istället vid **kompileringstid** ur en speciell **implicit namnrymd** (eng. *implicit scope*) enligt speciella regler.

11.1.23 Import av kontextparameter

```
object EnNamnrymd:
  given enGivenSträng: String = "ett givet värde i EnNamnrymd"
  def framkalla(using s: String) = s     //kontextparametrar märks med using
```

Det är den **lokala** kontexten vid **användning** som styr vad som kan framkallas (och inte den namnrymd där kontextparametern deklarerades):

```
scala> EnNamnrymd.framkalla
-- [E172] Type Error: ----------------------------------------------------------
1 |EnNamnrymd.framkalla
  |                   ^
  | No given instance of type String was found for parameter s of method framkalla
  | in object EnNamnrymd. The following import might fix the problem:
  | import EnNamnrymd.enGivenSträng
```

Du kan importera givna värden med speciell syntax där typen anges istället för namnet:

```
scala> import EnNamnrymd.given String    // speciell import-syntax baserat på typ

scala> EnNamnrymd.framkalla
val res0: String = ett givet värde i EnNamnrymd
```

11.1.24 Framkalla värde med `summon`

I standardbiblioteket för Scala 3 finns en **generisk** variant av metoden `framkalla` i föregående exempel, som är definierad så här:

```
def summon[T](using x: T) = x
```

Funktionen `summon` kan användas för att testa vilket värde kompilatorn framkallar i en viss kontext:

```
object EnAnnanNamnrymd:
  given String = "tagen för givet"    // namn behövs ej, det räcker med typ
```

```
scala> summon[String]
-- Error:
1 |summon[String]
  |             ^
  | no implicit argument of type String was found for parameter x of
```

```
  | method summon in object Predef. The following import might fix the problem:
  | import EnAnnanNamnrymd.given_String

scala> import EnAnnanNamnrymd.given    // importerar alla givna värden i MinKontext

scala> summon[String]
val res0: String = tagen för givet
```

11.1.25 Prioritetsordning vid framkallning av givna värden

- Det kan finnas flera **olika** givna värden av **samma** typ om de finns i olika namnrymder.
- Det får **inte** vara **tvetydigt** vilket värde som ska framkallas.
- Därför finns det speciella prioriteringsregler:
  1. **Explicita** argument till kontextparametrar märkta med **using**
  2. **given** och **import given** ... i aktuell namnrymd (eng. *current scope*)
  3. **given**-värden i **kompanjonsobjekt** för den använda typen.
  4. ... (fler regler i speciella fall som vi inte går in på här)
- *Specialregel:* Om flera givna värden kan framkallas för typer som ingår i en gemensam **arvshierarki** så väljer kompilatorn det givna värdet som är av den **mest specifika** typen.

Framkallning av givna värden har flera namn på engelska:
to summon, eller *term inference*, eller *implicit resolution*.

11.1.26 Ad hoc polymorfism

- Med så kallad **Ad hoc polymorfism** kan du ha *olika* implementationer av en funktion för *olika* typer *utan* att du behöver använda arv och dynamisk bindning.
- Detta uppfanns i språket ML och vidareutvecklades i språket Haskell.
- I Haskell skapas Ad hoc polymorfism med en s.k. ”**typklass**” (eng. *type class*).
- Detta görs i Scala genom att kombinera typparametrar och kontextparametrar.
- En ”typklass” i Scala är en tillståndslös **trait** med minst en typparameter och minst en abstrakt metod.

```
trait Parser[T]: //en typklass för tolkning och omvandling av strängar till godtycklig typ
  def fromString(value: String): Option[T]

object Parser:            //implementationer för en viss typ kan erbjudas som givna värden
  given Parser[Int] with        //nyckelordet with behövs då abstrakt medlem implementeras
    def fromString(value: String): Option[Int] = value.toIntOption
```

- Den specifika implementationen framkallas vid anrop med kontextparameter:

```
scala> def användParser[T](s: String)(using p: Parser[T]) = p.fromString(s)

scala> användParser[Int]("12")        // hittar givet värde i kompanjonsobjektet
val res0: Option[Int] = Some(12)
```

11.1.27 Hur få typklassen `Parser` att funka för fler typer?

```
scala> användParser[java.awt.Color]("Color(120,10,0)")
-- Error:
1 |användParser[java.awt.Color]("Color(120,10,0)")
  |                             ^
  | no implicit argument of type Parser[java.awt.Color] was found
  | for parameter p of method användParser
```

```
given Parser[java.awt.Color] with
  def fromString(value: String): Option[java.awt.Color] =
    if !value.startsWith("Color(") then None else
      val trimmed = value.trim.stripPrefix("Color(").stripSuffix(")")
      trimmed.split(",").map(_.toIntOption) match
        case Array(Some(r),Some(g),Some(b)) => Some(java.awt.Color(r, g, b))
        case _ => None
```

```
scala> användParser[java.awt.Color]("Color(120,10,0)")
val res1: Option[java.awt.Color] = Some(java.awt.Color[r=120,g=10,b=0])
```

Detta ger **flexibilitet**: Jag kan ge givna värden för mina **egna** typer!

11.1.28 Namnet på kontextparametrar kan utelämnas

- Namnet på det givna värdet behövs ofta inte – det är ju *typen* som är det viktiga.
- Därför är det tillåtet att utelämna parameternamnet vid `using`.
- Det givna värdet kan istället framkallas med `summon` vid behov:

```
def användParser[T](s: String)(using Parser[T]) =
  summon[Parser[T]].fromString(s)
```

11.1.29 Kontextgräns

Denna form av `using`-parameter med en typkonstruktor...

```
def användParser[T](s: String)(using Parser[T])
```

...är så vanlig i Scala att det finns kortare skrivsätt som kallas **kontextgräns**:

```
def användParser[T: Parser](s: String)
```

Om `F[A]` är en typkonstruktor och du skriver `[T: F]` så blir `F` en så kallad **kontextgräns** (eng. *context boundary*). Kompilatorn expandera detta automatiskt till kontextparametern (`using F[T]`)

11.1.30 Ännu smidigare typklass med extensionsmetod

Det får att kombinera kontextgräns med extensionsmetoder:

```
extension [T: Parser](s: String)
  def parseOrElse(default: T): T =
    summon[Parser[T]].fromString(s).getOrElse(default)
```

Nu har vi smidig punktnotation:

```
scala> "12".parseOrElse(default = 42)
val res2: Int = 12

scala> "gurka".parseOrElse(default = 42)
val res3: Int = 42

scala> "Color(100,100,0)".parseOrElse(default = java.awt.Color.black)
val res4: java.awt.Color = java.awt.Color[r=100,g=100,b=0]

scala> "fel färg".parseOrElse(default = java.awt.Color.black)
val res5: java.awt.Color = java.awt.Color[r=0,g=0,b=0]
```

11.1.31 Sortera samlingar med given ordning

```
scala> case class Gurka(namn: String, vikt: Int)

scala> val xs = Vector(Gurka("a", 100), Gurka("b", 50), Gurka("c", 100))
val xs: Vector[Gurka] = Vector(Gurka(a,100), Gurka(b,50), Gurka(c,100))

scala> xs.sorted
-- Error:
1 |xs.sorted
  | ^
  | No implicit Ordering defined for B
  | where: B is a type variable with constraint >: Gurka
```

Detta kan fixas genom att tillhandahålla en given ordning för `Gurka`:

```
given Ordering[Gurka] with
  def compare(x: Gurka, y: Gurka): Int =
    if (x == y) then 0
    else if x.vikt < y.vikt then -1
    else 1
```

```
scala> xs.sorted   // nu funkar det :)
val res0: Vector[Gurka] = Vector(Gurka(b,50), Gurka(a,100), Gurka(c,100))
```

11.1.32 Sortera samlingar med ännu smidigare given ordning

- Det är vanligt att man vill definiera egna ordningsrelationer.

- Därför finns en smidig hjälpmetod i kompanjonsobjektet för typklassen `Ordering` som heter `fromLessThan`:

```
given Ordering[Gurka] =
  Ordering.fromLessThan((g1, g2) => g1.vikt < g2.vikt)
```

- Den tar som inparameter en funktion som tar två instanser som ska ordnas och ger **`true`** om den första ska anses **mindre** (alltså kommer **före**) enligt valfri definition.
- `fromLessThan` returnerar en `Ordering` som du kan låta vara givet.
- Prova gärna detta på veckans fördjupningsövningar.
- Läs mer om typklasser i Scala här: https://docs.scala-lang.org/scala3/reference/contextual/type-classes.html

---

11.1.33 Förslag på användning av kontextparameter i snake-labben

Antag att klassen `Snake` har två kontextparametrar:

```
class Snake (
  val initPos: Pos,
  val initDir: Dir,
  val headColor: Color,
  val tailColor: Color,
  )(using ctx: SnakeGame, settings: Settings) extends CanMove
```

- Var erbjuda default-värden för kontext-paramterar?
  - Ett bra ställe att placera `given default: Settings = ???` är i kompanjonsobjekt till klassen `Settings`.
  - Du kan i kroppen på klassen `SnakeGame` ha en `given SnakeGame = this` Då kommer kompilatorn använda aktuell instans av `SnakeGame` som kontext-parameter när spelet i sin tur skapar instanser av `Snake`.

Notera denna text i labben, avsnitt *Tips och förslag*: ”Det är tillåtet att ändra, ta bort och lägga till, så länge de obligatoriska designkraven uppfylls.”

11.1.34 Översikt av kursens avslutning

Återstående moment:

- W11: `snake` — Alla ska vara aktiva på redovisningen.
- W12-W13: Projekt — Individuellt arbete, välj gärna `bank`.
- W14: Munta — På schematider endast ons-tors.
  - Om du är väl förberedd för muntan kan du göra den redan W11–W13 så snart alla labbar t.om. snake är klara.
  - Prata med handledare om din pluggplan.
  - Förbered dig noga med hjälp av https://cs.lth.se/pgk/muntabot

  - Läs om muntan i kompendiet.

- Tentaperiod januari: Valfri tentamen för överbetyg. Anmälan enl. LTH:s normala rutiner.

Se schema: https://cs.lth.se/pgk/schema/

11.2 Övning `context`

Mål

- ☐ Kunna förklara vad en kontextparameter är.
- ☐ Kunna förklara nyttan med kontextparametrar jämfört med en globala variabler och defaultargument vid lösning av konfigurationsproblemet.
- ☐ Kunna använda enkla kontextuella abstraktioner med **`given`** och **`using`**.

Förberedelser

- ☐ Studera begreppen i kapitel 11

11.2.1 Grunduppgifter; förberedelse inför laboration

Uppgift 1. *Kontextparameter.* Deklarera följande funktioner som tar ett heltal som kontextparameter. Skapa även en **`given`**-deklaration som erbjuder det givna heltalsvärdet noll:

```
scala> def f(using i: Int) = i + 1

scala> def g(x: Int)(using y: Int) = x + y
```

a) Anropa funktionerna `f` och `g` med ett explicit givet argument som skiljer sig från det givna heltalsvärdet med hjälp av **`using`** i anropet. Vad händer om du utelämnar **`using`**?

b) Anropa funktionerna `f` och `g` utan att ange **`using`**-argument. Förklara vad som händer.

c) Går det att blanda vanliga parametrar och kontextparametrar i samma parameterlista? Om inte vad händer?

Uppgift 2. *Flera olika givna värden i lokal kontext.* Olika värden beroende på kontext.

```
case class Delta(value: Int)
object Delta:
  given default: Delta = Delta(1)

def inc(x: Int)(using dx: Delta) = x + dx.value

object Context1:
  val a = inc(1)

object Context2:
  given Delta = Delta(42)
  val a = inc(1)
```

a) Vilket värde har `Context1.a`?

b) Vilket värde har `Context2.a`?

c) Förklara vad som händer.

Uppgift 3. *Lösning på konfigurationsproblemet med hjälp av givna värden.* Antag att vi vill kunna konfigurera beteendet hos en funktion för att göra den mer flexibel. Nedan visas tre principiellt olika sätt att göra detta på för en funktion `greet` som skriver ut en hälsning: 1) en globalt åtkomlig variabel, 2) defaultargument, samt 3) kontextuell abstraktion med **`given`** och **`using`**.

```
object GlobalVar:
  case class GreetConfig(greeting: String, receiver: String)
  object GreetConfig:
    val default = GreetConfig(greeting = "Hello", receiver = "World")
    var config = default

  def greetMsg =
    s"${GreetConfig.config.greeting} ${GreetConfig.config.receiver}!"

object DefaultArgs:
  case class GreetConfig(greeting: String, receiver: String)
  object GreetConfig:
    val default = GreetConfig(greeting = "Hello", receiver = "World")

  def greetMsg(config: GreetConfig = GreetConfig.default) =
    s"${config.greeting} ${config.receiver}!"

object GivenVal:
  case class GreetConfig(greeting: String, receiver: String)
  object GreetConfig:
    given default: GreetConfig = GreetConfig("Hello", "World")

  def greetMsg(using g: GreetConfig) = s"${g.greeting} ${g.receiver}"
```

a) Skriv kod som testar de olika varianterna ovan. Visa speciellt hur du kan använda default-konfigurationen och därefter ge en konfiguration som skiljer sig från `default`.

b) Vad är för- och nackdelar med de olika varianterna ovan? Diskutera speciellt vilken/vilka lösningar som medger flera lokala konfigurationer utan att de påverkar varandra.

c) Förklara vad som händer vid anrop av `summon[GivenVal.GreetConfig]`.

d) Vad händer om du försöker framkalla ett givet värde för en typ som inte har något sådant?

e) Måste det givna värdet vara unikt?

11.2.2 Extrauppgifter; träna mer

Uppgift 4. *Kontextparameter och givet värde.* Prova nedan i REPL.

```
scala> def add(x: Int)(using y: Int) = x + y
scala> add(1)(using 2)
scala> add(1)
scala> given ngtNamn = 42
scala> add(1)
```

a) Vad blir felmeddelandet på rad 3 ovan?

b) Varför fungerar det på rad 5 utan fel?

c) Definiera och testa en motsvarande funktion `sub` som kan subtrahera ett givet värde.

11.2.3 Fördjupningsuppgifter; utmaningar

Uppgift 5. *Varians och typgränser.* Koden nedan är en modell av husdjur med följande innebörd: Husdjur kan vara friska eller sjuka och föds i normalfallet friska. Det kan finnas många katter och hundar, vilka alla är olika slags husdjur.

```
trait Pet(var isHealthy: Boolean = true)
class Cat extends Pet()
class Dog extends Pet()
```

a) Förändra koden nedan så att efterföljande REPL-sats *inte* ger kompileringsfel?

```
case class Box[A](x: A)
```

```
scala> val b: Box[Any] = Box[Cat](Cat())
```

b) Prova nedan i REPL och förklara vad som händer.

```
scala> val v: Vector[Pet] = Vector[Cat](Cat())

scala> val s: Set[Pet] = Set[Cat](Cat())

scala> :settings -explain

scala> val s: Set[Pet] = Set[Cat](Cat())
```

Ledtråd: I Scalas standardbibliotek så är ärver `Set[T]` funktionstypen `T => Boolean` som är deklarerad kontravariant i sin inparameter.

c) Det ska finnas veterinärer som kan behandla husdjur och göra dem friska. Varför fungerar inte nedan kod? Är det ett kompileringsfel eller körtidsfel?

```
class Vet[-A]:
  def treat(x: A): Unit = x.isHealthy = true
```

d) Inför en typgräns i veterinärens typparametern som åtgärdar felet.

e) Skriv valfri kod som visar 1) att kompilatorn tillåter kattveterinärer att behandla katter men 2) förhindrar att kattveterinärer får behandla godtyckliga husdjur och att 3) en veterinär som har komptens att behandla godtyckliga husdjur kan behandla både katter och hundar. Förklara varför kompilatorn tillåter/förhindrar detta.

Uppgift 6. *Typklasser och kontextparametrar.* I Scala finns möjligheter till avancerad funktionsprogrammering med s.k. **typklasser** (ä.k. *ad hoc polymorfism*). En typklass definierar generella beteenden som fungerar för godtyckliga befintliga typer utan att implementationen av dessa behöver ändras. Vi nosar i denna uppgift på hur kontextuella abstraktioner kan användas för att skapa typklasser i Scala, illustrerat med hjälp av givna ordningarna vid sortering.

Genom att kombinera koncepten givna värden, generiska klasser och kontextparametrar får man möjligheten till ad hoc polymorfism, exemplifierat med typklassen `CanCompare` nedan, som vi kan få att fungera för befintliga typer *utan* att de behöver ändras. Speciellt så har vi ju inte möjligheten att lägga till metoder på befintliga typer i standardbiblioteket, eftersom det inte är våran egen kod.

a) Vad händer nedan? Vilka rader ger felmeddelande? Varför?

```
scala> trait CanCompare[T]:
         def compare(a: T, b: T): Int

scala> def sort[T](a: T, b: T)(using cc: CanCompare[T]): (T, T) =
         if cc.compare(a, b) > 0 then (b, a) else (a, b)

scala> sort(42, 41)

scala> given intComparator: CanCompare[Int] with
         override def compare(a: Int, b: Int): Int = a - b

scala> sort(42, 41)

scala> sort(42.0, 41.0)
```

b) Definiera ett givet värde som gör så att `sort` fungerar för värden av typen `Double`.

c) Definiera ett givet värde som gör så att `sort` fungerar för värden av typen `String`.
Tips: Du har nytta av de befintliga jämförelseoperatorerna på strängar, men tänk på att `compare` fortfarande måste returnera ett heltal även vid jämförelse av strängar.

Uppgift 7. *Användning av given ordning.* Vi ska nu skapa en funktion `isSorted` som är generellt användbar genom att göra givna ordningsfunktioner tillgängliga för olika typer. Funktionen `def isSorted(xs: Vector[Int]): Boolean = ???` fungerar bara för samlingar av typen `Vector[Int]`.

Om vi i stället använder `def isSorted(xs: Seq[Int]): Boolean = ???` fungerar den för olika samlingar med heltal, även `Vector` och `List`.

a) Testa nedan funktion i REPL med heltalssekvenser av olika typ.

```
def isSorted(xs: Seq[Int]): Boolean = xs == xs.sorted
```

b) Det blir problem med nedan försök att göra `isSorted` generisk. Hur lyder felmeddelandet? Vad saknas enligt felmeddelandet?

```
scala> def isSorted[T](xs: Seq[T]): Boolean = xs == xs.sorted
```

c) Vi vill gärna att `isSorted` ska fungera för godtyckliga typer T som har en ordningsdefinition. Detta kan göras med nedan funktion där den speciella typparametern `[T:Ordering]` betyder att `isSorted` är definierad för alla samlingar där typen `T` har en given ordning `Ordering[T]`. Speciellt gäller detta för alla grundtyperna `Int`, `Double`, `String`, etc., som alla har specifika implementationer av typklassen `Ordering`.

```
def isSorted[T:Ordering](xs: Seq[T]): Boolean = xs == xs.sorted
```

Testa metoden ovan i REPL enligt nedan.

```
scala> isSorted(Vector(1,2,3))
scala> isSorted(List(1,2,3,1))
```

```
scala> isSorted(Vector("A","B","C"))
scala> isSorted(List("A","B","C","A"))
scala> case class Person(firstName: String, familyName: String)
scala> val persons = Vector(Person("Kim", "Finkodare"), Person("Voldemort","Ful
scala> isSorted(persons)
```

Vad ger sista raden för felmeddelande? Varför?

d) *Implicita ordningar.* En typparameter på formen `[T:Ordering]` kallas kontextgräns (eng. *context bound*) och föranleder kompilatorn att automatiskt expandera funktionshuvudet för `isSorted` med en kontextparameter. I stället för att använda `[T:Ordering]` kan vi själva lägga till en kontextparameter som motsvarar kontextgränsen. Då får vi också tillgång till ett namn, här nedan `ord`, på den implicita ordningen och kan använda det namnet i funktionskroppen och anropa metoder som är medlemmar av typklassen `Ordering`. (Namnet på kontextparametern kan också utelämnas, men då får vi istället gå omvägen via inbyggda funktionen `summon[T]` för att be kompilatorn leta upp den givna instansen för den typparameter som ges vid anropet.)

```
def isSorted[T](xs: Seq[T])(using ord: Ordering[T]): Boolean =
  xs.zip(xs.tail).forall(x => ord.lteq(x._1, x._2))
```

Objekt av typen `Ordering` har jämförelsemetoder som t.ex. `lteq` (förk. *less than or equal*) och `gt` (förk. *greater than*).

Det finns givna ordningar för alla grundtyper i standardbiblioteket, alltså t.ex. `Ordering[Int]`, `Ordering[String]`, etc. Testa så att kompilatorn hittar ordningen för samlingar med värden av några grundtyper. Kontrollera även enligt nedan att det fortfarande blir problem för egendefinierade klasser, t.ex. `Person` (detta ska vi råda bot på i uppgift 8).

```
scala> isSorted(Vector(1,2,3))
scala> isSorted(Array(1,2,3,1))
scala> isSorted(Vector("A","B","C"))
scala> isSorted(List("A","B","C","A"))
scala> class Person(firstName: String, familyName: String)
scala> val persons = Vector(Person("Kim", "Finkodare"), Person("Robin","Fulhack
scala> isSorted(persons)
```

e) *Importera implicita ordningsoperatorer från en `Ordering`.* Om man gör import på ett `Ordering`-objekt får man tillgång till implicita konverteringar som gör att jämförelseoperatorerna fungerar. Testa nedan variant av `isSorted` på olika sekvenstyper och verifiera att `<=`, `>`, etc., nu fungerar enligt nedan.

```
def isSorted[T](xs: Seq[T])(given ord: Ordering[T]): Boolean = {
  import ord._
  xs.zip(xs.tail).forall(x => x._1 <= x._2)
}
```

Uppgift 8. *Skapa egen implicit ordning med `Ordering`.*

a) Ett sätt att skapa en egen, specialanpassad ordning för dina egna klasser är att mappa dina objekt till typer som redan har en implicit ordning. Med hjälp av metoden `by` i objektet `scala.math.Ordering` kan man skapa ordningar genom att bifoga en funktion `T => S` där `T` är typen för de objekt du vill ordna och `S` är någon annan typ, t.ex. `String` eller `Int`, där det redan finns en given ordning.

```
scala> case class Team(name: String, rank: Int)
scala> val xs =
         Vector(Team("fnatic", 1499), Team("nip", 1473), Team("lumi", 1601))
scala> xs.sorted  // Hur lyder felmeddelandet? Varför blir det fel?
scala> val teamNameOrdering: Ordering[Team] = Ordering.by(t => t.name)
scala> xs.sorted(using teamNameOrdering)   //explicit ordning
scala> given Ordering[Team] = Ordering.by(t => t.rank)
scala> xs.sorted   // Varför funkar det nu?
```

b) Vill man sortera i omvänd ordning kan man använda `Ordering.fromLessThan` som tar en funktion `(T, T) => Boolean` vilken ska ge **`true`** om första parametern ska komma före, annars **`false`**. Om vi vill sortera efter `rank` i omvänd ordning kan vi göra så här:

```
scala> val highestRankFirst: Ordering[Team] =
         Ordering.fromLessThan((t1, t2) => t1.rank > t2.rank)
scala> xs.sorted(using highestRankFirst)
```

c) Om du har en case-klass med flera fält och vill ha en fördefinierad implicit sorteringsordning samt *även* erbjuda en alternativ sorteringsordning, så kan du placera en default ordningsdefinition i ett kompanjonsobjekt; detta är nämligen ett av de ställen där kompilatorn söker sist efter eventuella implicita värden innan den ger upp att leta.

```
case class Team(name: String, rank: Int)
object Team:
  given highestRankFirst: Ordering[Team] =
    Ordering.fromLessThan((t1, t2) => t1.rank > t2.rank)
  val nameOrdering: Ordering[Team] = Ordering.by(t => t.name)
```

```
scala> val xs =
         Vector(Team("fnatic", 1499), Team("nip", 1473), Team("lumi", 1601))
scala> xs.sorted
scala> xs.sorted(Team.nameOrdering)
```

d) Det går också med kompanjonsobjektet ovan att få jämförelseoperatorer att fungera med din case-klass, genom att importera medlemmarna i lämpligt ordningsobjekt. Verifiera att så är fallet enligt nedan:

```
scala> Team("fnatic",1499) < Team("gurka", 2)  // Vilket fel? Varför?
scala> import Team.highestRankFirst.given
scala> Team("fnatic",1499) < Team("gurka", 2)  // Inget fel? Varför?
```

Uppgift 9. *Specialanpassad ordning genom att ärva från `Ordered`* Om det finns *en* väldefinierad, specifik ordning som man vill ska gälla för sina case-klass-instanser kan man göra den ordnad genom att låta case-klassen mixa in traiten `Ordered` och implementera den abstrakta metoden `compare`. (Detta illustrerar användning av subtypspolymorfism (d.v.s arv) i stället för ad hoc polymorfism med typklasser.)

Bakgrund: En trait som används på detta sätt kallas **gränssnitt** (eng. *interface*), och om man *implementerar* ett gränssnitt så uppfyller man ett ”kontrakt”, som i detta fall innebär att man implementerar det som krävs av ordnade objekt, nämligen att de har en konkret `compare`-metod. Du lär dig mer om gränssnitt i kommande kurser.

a) Implementera case-klassen `Team` så att den är en subtyp till `Ordered` enligt nedan skiss. Högre rankade lag ska komma före lägre rankade lag. Metoden `compare` ska ge ett heltal som är negativt om `this` kommer före `that`, noll om de ordnas lika, annars positivt.

```
case class Team(name: String, rank: Int) extends Ordered[Team]:
  override def compare(that: Team): Int = ???
```

Tips: Du kan anropa metoden `compare` på alla grundtyper, t.ex. `Int`, eftersom de implementerar gränssnittet `Oredered`. Genom att negera uttrycket blir ordningen den omvända.

```
scala> -(2.compare(1))
```

b) Testa att din case-klass nu uppfyller det som krävs för att vara ordnad.

```
scala> Team("fnatic",1499) < Team("gurka", 2)
```

c) Diskutera med handledare eller kursare skillnader och likheter mellan gränssnitt och typklasser, med ledning av denna och föregående uppgifter.

11.3 Laboration: `snake1`

Mål

- ☐ Se mål i förra veckans uppgift. Denna laboration sträcker sig över två veckor. Förra veckans redovisning avsåg arbetsupplägget och pågående utveckling, samt ditt individuella ansvar. Denna vecka ska de färdig lösningarna presenteras av respektive huvudansvarig individ.

Förberedelser

- ☐ Gör övning `context` i avsnitt 11.2

11.3.1 Redovisning av grupplabb

1. Förklara kodens övergripande struktur.
2. Beskriv vilka delar du har bidragit till i koden.
3. Ge en kort förklaring av koncept som du tränat på.
4. Redogör för vad du lärt dig om utmaningarna med systemutveckling i grupp.
5. Redogör den återkoppling du fått från granskningar och hur du arbetat med att förbättra läsbarheten under dina stegvisa utvidgningar av din kod.
6. Redogör för hur och vad du återkopplat när du granskat någon annans kod.

Kapitel 12

Fördjupning, Projekt

Begrepp som ingår i denna veckas studier:

- ☐ välj valfritt fördjupningsområde
- ☐ påbörja projekt

12.1 Projektuppgift

12.1.1 Om din avslutande projektuppgift

Läs noga kompendium Del 1, kapitel -1 avsnitt ”Projektuppgift”!
Några viktiga punkter:

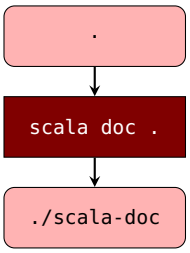
- Mål: skapa ett stort program med många samverkande klasser/moduler.
- Du väljer själv projektuppgift. Börja redan idag att planera ditt arbete!
- Projektet går över **två veckor**. Schemalagda labbar i december är **obligatoriska** för de som har kvar obligatoriska moment.
- I kompendium Del 2, kapitel 13, finns flera förslag att välja bland, men du kan också definiera ett eget projekt som passar just dig.
- Inför redovisningen ska du skapa automatiskt genererad dokumentation utifrån relevanta dokumentationskommentarer för minst hälften av dina publika metoder, enligt instruktioner i Appendix E.
- Redovisning sker i datorsal på schemalagd tid:
  - Förklara hur din kod fungerar.
  - Beskriv framväxten av ditt program.
  - Gå igenom den genererade dokumentationen av din kod.

12.1.2 Projektuppgifter

- `bank`
  - känd domän: skapa bank med transaktionshistorik
  - oföränderlig data tillsammans med tillståndsförändring
- `music`
  - skapa ett enkelt kompositionsverktyg som spelar musik
  - sätta sig in i en domänmodell
- `photo`
  - en enkel variant av photoshop
  - inblick i enkel matrismatematik
- egendefinierat projekt
  - Lagom svårt: ej för enkel uppgift, men ta dig inte vatten över huvudet!
  - Diskutera med en handledare och dokumentera egna uppgiften och dess omfattning och relation till lärandemålen i kursen så att andra handledare och kursansvarig kan förstå varför projektet passar i kursen.
  - Du måste få OK från en handledare innan du startar egendefinierat projekt.

12.1.3 Skapa dokumentation

Scala CLI kan ta dokumentationskommentarerna i källkoden och skapa en webbsajt med dokumentation.



Aktuell katalog med `.scala`-filer

`index.html` och en massa andra filer som ger en hel webbsajt!

Öppna `scala-doc/index.html` i en webbläsare.

12.1.4 Dokumentationskommentarer

För att kod ska bli begriplig för människor är det bra att dokumentera vad den gör. Det finns **tre olika sorters kommentarer**:

```
// Enradskommentarer börjar med dubbla snedstreck
//       men de gäller bara till radslut

/* Flerradskommentarer börjar med
   snedstreck-asterisk
   och slutar med asterisk-snedstreck.  */

/** Dokumentationskommentarer placeras före
 *   t.ex. en funktion och berättar vad den gör
 *   och vad eventuella parametrar används till.
 *   Börjar med snedstreck-asterisk-asterisk.
 *   Varje ny kommentarsrad börjar med asterisk.
 *   Avslutas med asterisk-stjärna.
 */
```

Kommentarer påverkar inte hur maskinen exekverar koden, men hjälper människor att använda koden och verktyg att visa hjälp. Se Appendix E.

12.2 Övning `extra`

Mål

- ☐ Denna veckas övning innehåller valfri fördjupning.
- ☐ Sökning och sortering:
  - Kunna använda inbyggda sökmetoder.
  - Förstå när binärsökning är lämplig och möjlig.
  - Kunna implementera binärsökning.
  - Kunna implementera urvalssortering, både till ny samling och på plats.
- ☐ Trådar och jämlöpande exekvering:
  - Känna till vad en tråd är och kunna förklara begreppet jämlöpande exekvering.
  - Känna till vad metoderna `run` och `start` gör i klassen `Thread`.
  - Kunna skapa och starta en tråd med överskuggad `run`-metod.
  - Kunna skapa ett enkelt program som från två trådar tävlar om att uppdatera en variabel och förklara varför beteendet kan bli oförutsägbart.
  - Kunna använda en `Future` för att köra igång flera parallella beräkningar.
  - Kunna registrera en callback på en `Future` med metoden `onComplete`.

12.2.1 Uppgifter om sökning och sortering

Uppgift 1. *Tidmätning.* I kommande uppgifter kommer du att ha nytta av funktionen `timed` enligt nedan.

```
def timed[T](code: => T): (T, Long) =
  val now = System.nanoTime
  val result = code
  val elapsed = System.nanoTime - now
  println("\ntime: " + (elapsed / 1e6) + " ms")
  (result, elapsed)
```

a) Klistra in `timed` i REPL och testa så att den fungerar, genom att mäta hur lång tid nedan uttryck tar att exekvera.

```
1 scala> val (v, t1) = timed{ (1 to 1000000).toVector.reverse }
2 scala> val (s, t2) = timed{ v.toSet }
3 scala> timed{ v.find(_ == 1) }
4 scala> timed{ s.find(_ == 1) }
5 scala> timed{ s.contains(1) }
```

b) Försök förklara skillnaderna i exekveringstid mellan de olika sätten att söka reda på talet 1 i samlingen. Ungefär hur många gånger behöver man använda `contains` på heltalsmängden `s` för att det ska löna sig att skapa `s` i stället för att linjärsöka i `v` med `find` i ovan exempel?

Uppgift 2. *Sökning med inbyggda sökmetoder.*

a) *Linjärsökning framifrån med `indexOfSlice`.* Studera dokumentationen för Scalas samlingsmetod `indexOfSlice`[1] och skriv 8 olika uttryck i REPL som, både med en sträng och med en vektor med heltal, provar 4 olika fall: (1) finns i början, (2) finns någonstans i mitten, (3) finns i slutet, samt (4) finns ej.

b) *Linjärsökning bakifrån med `lastIndexOfSlice`.* Studera dokumentationen för Scalas samlingsmetod `lastIndexOfSlice`[2] och skriv 8 olika uttryck i REPL som, både med en sträng och med en vektor med heltal, provar 4 olika fall: (1) finns i början, (2) finns någonstans i mitten, (3) finns i slutet, samt (4) finns ej.

c) *Sökning med inbyggd binärsökning.* Om en samling är sorterad kan man utnyttja detta för att göra snabbare sökning. Vid **binärsökning** (eng. *binary search*)[3] börjar man på mitten och kollar vilken halva att söka vidare i; sedan delar man upp denna halva på mitten och kollar vilken fjärdedel att söka vidare i, etc.

I objektet `scala.collection.Searching`[4] finns en metod `search` som, om den importeras, erbjuder binärsökning för alla sorterade sekvenssamlingar. Om samlingen är sorterad ger den ett objekt av case-klassen `Found` som innehåller indexet för platsen där elementet först hittats; alternativt om det som eftersöks ej finns, ges ett objekt av case-klassen `InsertionPoint` som innehåller indexet där elementet borde ha varit placerad om det funnits i samlingen. Observera att om samlingen inte är sorterad är resultatet ”odefinierat”, d.v.s. något returneras men det är *inte* att lita på; man måste alltså först sortera samlingen eller vara helt säker på att den är sorterad.

Undersök hur `search` fungerar genom att förklara vad som händer nedan. Vilken är snabbast av `lin` och `bin` nedan? Använd `timed` från uppgift 1.

```
scala> val udda = (1 to 1000000 by 2).toVector
scala> import scala.collection.Searching._
scala> udda.search(udda.last)
scala> udda.search(udda.last + 1)
scala> udda.reverse.search(udda(0))
scala> def lin(x: Int, xs: Seq[Int]) = xs.indexOf(x)
scala> def bin(x: Int, xs: Seq[Int]) = xs.search(x) match
         case Found(i) => i
         case InsertionPoint(i) => -i
scala> timed{ lin(udda.last, udda) }
scala> timed{ bin(udda.last, udda) }
```

d) Om en samling innehåller n element, hur många jämförelser behövs då vid binärsökning i värsta fall? *Tips:* Läs om algoritmen på Wikipedia[3].

Uppgift 3. *Sök bland LTH:s kurser med linjärsökning.*

a) Via denna URL kan du ladda ner en tab-separerad lista med alla kurser som ges på LTH under innevarande läsår: `http://cs.lth.se/pgk/kurser`
Vilken data finns i filen? Du kan undersöka detta t.ex. med:

```
scala> import scala.io.Source.fromURL
scala> val url = "https://fileadmin.cs.lth.se/pgk/lthkurser201819.txt"
scala> val data = fromURL(url,"UTF-8").getLines.mkString("\n")
```

[1] docs.scala-lang.org/overviews/collections/seqs.html
[2] docs.scala-lang.org/overviews/collections/seqs.html
[3] en.wikipedia.org/wiki/Binary_search_algorithm
[4] http://www.scala-lang.org/api/current/scala/collection/Searching$.html

b) Klistra in objektet `courses` på sidan 146 i REPL.[5] Vad gör koden? Hur många kurser innehåller `courses.lth`?

```
object courses:
  def download(): Vector[Course] =
    val url = "https://fileadmin.cs.lth.se/pgk/lthkurser201819.txt"
    val lines = scala.io.Source.fromURL(url, "UTF-8").getLines.toVector
    lines.drop(1).map(s => Course.fromLine(s, '\t'))

  lazy val lth: Vector[Course] = download()

  case class Course(
    code:    String,
    nameSv:  String,
    nameEn:  String,
    credits: Double,
    level:   String
  )

  object Course:
    def fromLine(line: String, separator: Char): Course =
      import scala.util.Try
      val xs = line.split(separator).toSeq
      def str(i: Int): String = Try(xs(i)).getOrElse("")
      def num(i: Int): Double = Try(xs(i).toDouble).getOrElse(0.0)
      Course(str(0), str(1), str(2), num(3), str(4))
```

Figur 12.1: Kod för att ladda ner data om alla kurser på LTH.

c) *Linjärsökning med find.* Teknologen Oddput Clementina vill gå första bästa datavetenskapskurs som är på G2-nivå. Hjälp Oddput med att söka upp första förekommande kurs genom linjärsökning med samlingsmetoden `find`. Kurskoder vid datavetenskap börjar på EDA eller ETS[6]. *Tips:* Du har nytta av att definiera predikatet `def isCS(s: String): Boolean` som i sin tur lämpligen nyttjar strängmetoden `startsWith`.

d) *Implementera linjärsökning.* Som träning ska du nu implementera en egen linjärsökningsfunktion med signaturen:
`def linearSearch[T](xs: Seq[T])(p: T => Boolean): Int = ???`
Funktionen ska ta en sekvenssamling `xs` och ett predikat `p` som är en funktion som tar ett element och returnerar ett booleskt värde. Typen `Seq` är supertyp till alla sekvenssamlingar, så om vi använder den som parametertyp för parametern `xs` så fungerar funktionen för `Vector`, `Array`, `List`, etc. Genom typparametern `T` blir funktionen generisk och fungerar för godtycklig typ. Funktionen `p` ska ge `true` om parametern är ett eftersökt element. Funktionen `linearSearch` ska returnera index för första hittade elementet i `xs` där `p` gäller. Om det inte finns något element som uppfyller

[5] Du kan ladda ner koden från:
github.com/lunduniversity/introprog/tree/master/compendium/examples/lth-courses/courses.scala

[6] Detta är en förenklad bild av LTH:s kurskodnamnsystem. Några kurser från EIT-institutionen kommer att slinka med, men det bortser vi ifrån i denna uppgift.

predikatet ska -1 returneras. Skriv först pseudokod för funktionen med penna och papper. Du ska använda `while`.

e) Implementera en funktion `def rndCode: String` som genererar slumpmässiga kurskoder som består av 4 bokstäver mellan A och Z följt av 2 siffror mellan 0 och 9. *Tips:* Använd REPL i kombination med en editor för att stegvis skapa och testa hjälpfunktioner som löser lämpliga delproblem.

f) Använd `rndCode` från föregående deluppgift för att fylla en vektor kallad `xs` med en halv miljon slumpmässiga kurskoder. För varje slumpkod i `xs` sök med din funktion `linearSearch` efter index i vektorn `courses.lth` från deluppgift b. Mät totala tiden för de 500000 linjärsökningarna med hjälp av funktionen `timed` från uppgift 1. Hur många av de slumpmässiga kurskoderna hittades bland de verkliga kurskoderna på LTH?

g) Hur kan du implementera `linearSearch` med den inbyggda samlingsmetoden `indexWhere`?

Uppgift 4. *Sök bland LTH:s kurser med binärsökning.* Sökalgoritmen BINSEARCH kan formuleras med nedan pseudokod:

Indata: En växande sorterad sekvens xs med n heltal och ett eftersökt heltal key
Utdata: Ett heltal $i \geq 0$ som anger platsen där x finns, eller ett negativt tal i där $-i$ motsvarar platsen där x ska sättas in i sorterad ordning om x ej finns i samlingen.

```
1  sätt intervallet (low, high) till (0, n − 1)
2  found ← false
3  mid ← −1
4  while low ≤ high and not found do
5  |  mid ← platsen mitt emellan low och high
6  |  if xs(mid) == key then
7  |  |  found ← true
8  |  else
9  |  |  if xs(mid) < key then
10 |  |  |  low ← mid + 1
11 |  |  else
12 |  |  |  high ← mid − 1
13 |  |  end
14 |  end
15 end
16 if found then
17 |  mid
18 else
19 |  −(low + 1)
20 end
```

a) Prova algoritmen ovan med penna och papper på en sorterad sekvens med mindre än 10 heltal. Prova om algoritmen fungerar med ett jämnt antal tal, ett udda antal tal, en sekvens med ett heltal och en tom sekvens. Prova både om talet du letar efter finns och om det inte finns.

b) Implementera binärsökning i en funktion med signaturen

```
def binarySearch(xs: Seq[String], key: String): Int = ???
```

och testa i REPL för olika fall. Vad händer om sekvensen inte är sorterad?

c) Använd `binarySearch` för att leta efter LTH-kurser enligt nedan. Använd `rndCode`, `timed` och `courses` från tidigare uppgifter.

```
def binarySearch(xs: Seq[String], key: String): Int = ???

val lthCodesSorted = courses.lth.map(_.code).sorted
val xs = Vector.fill(500000)(rndCode)
val (_, elapsedBin) =
  timed{xs.map(x => binarySearch(lthCodesSorted, x))}
val (_, elapsedLin) =
  timed{xs.map(x => linearSearch(lthCodesSorted)(_ == x))}
println(elapsedLin / elapsedBin)
```

d) Hur mycket snabbare blev binärsökningen jämfört med linjärsökningen?[7]

Uppgift 5. *Insättningssortering.* Implementera sortering av en heltalssekvens till en sekvens med **insättningssortering** (eng. *insertion sort*) i en funktion med följande signatur:

```
def insertionSort(xs: Seq[Int]): Seq[Int] = ???
```

Lösningsidé: Skapa en ny, tom sekvens som ska bli vårt sorterade resultat. För varje element i den osorterade sekvensen: Sätt in det på rätt plats i den nya sorterade sekvensen.

a) *Pseudokod:* Kör nedan pseudokod med papper och penna t.ex. på sekvensen 5 1 4 3 2 1. Rita minnessituationen efter varje runda i loopen. Här använder vi internt i funktionen föränderliga `ArrayBuffer` som är snabb på insättning och avslutar med `toVector` så att vi lämnar ifrån oss en oföränderlig sekvens.

```
1 result ← en ny, tom ArrayBuffer
2 foreach element e in xs do
3 |   pos ← leta upp rätt position i result
4 |   stoppa in e på plats pos i result
5 end
6 result.toVector
```

b) Implementera `insertionSort`. Använd en `while`-loop för att implementera rad 3 i pseudokoden. Sök upp dokumentationen för metoden `insert` på `ArrayBuffer`. Testa `insert` på `ArrayBuffer` i REPL och verifiera att den kan användas för att stoppa in på slutet på den ”oanvända” positionen som är precis efter sista positionen. Vad händer om man gör `insert` på positionen `size + 2`?

Klistra in din implementation av `insertionSort` i REPL och testa så att allt fungerar:

```
scala> insertionSort(Vector())
res0: Seq[Int] = Vector()

scala> insertionSort(Vector(42))
res1: Seq[Int] = Vector(42)

```

[7] Vid en körning på en i7-4970K med 4.0GHz tog `elapsedLin` cirka 3000 *ms* och `elapsedBin` cirka 60 *ms*. Binärsökning var alltså i detta fall ungefär 50 gånger snabbare än linjärsökning.

```
scala> insertionSort(Vector(1,2,3))
res2: Seq[Int] = Vector(1, 2, 3)

scala> insertionSort(Vector(5,1,4,3,2,1))
res3: Seq[Int] = Vector(1, 1, 2, 3, 4, 5)
```

Uppgift 6. *Sortering på plats.* Implementera sortering på plats (eng. *in-place*) i en `Array[String]` med urvalssortering (eng. *selection sort*)

Lösningsidé: För alla index i: sök $minIndex$ för ”minsta” strängen från plats i till sista plats och byt plats mellan strängarna på plats i och plats $minIndex$. Se även animering här: sv.wikipedia.org/wiki/Urvalssortering

Implementera enligt nedan skiss. *Tips:* Du har nytta av en modifierad variant av lösningen till uppgift **??** i kapitel **??**.

```
def selectionSortInPlace(xs: Array[String]): Unit =
  def indexOfMin(startFrom: Int): Int = ???
  def swapIndex(i1: Int, i2: Int): Unit = ???
  for i <- 0 to xs.size - 1 do swapIndex(i, indexOfMin(i))
```

Uppgift 7. *Undersök om en sekvens är sorterad.* Ett enkelt och lättläst sätt att undersöka om en sekvens är sorterad visas nedan.

```
scala> def isSorted(xs: Vector[Int]): Boolean = xs == xs.sorted
```

a) Om `xs` har 10^6 element, hur många jämförelser kommer i värsta fall att ske med `isSorted` enligt ovan? Metoden `sorted` använder algoritmen Timsort[8]. Sök upp antalet jämförelser i värstafallet på Wikipedia. ✎

Denna lösning är dock relativt långsam för stora samlingar. Man behöver ju inte först sortera för att avgöra om det är sorterat (om man inte ändå hade tänkt sortera av andra skäl), det räcker att kolla att elementen är i växande ordning.

b) Implementera en effektivare variant av `isSorted` som använder en **`while`**-sats och kollar att elementen är i växande ordning. Din algoritm ska sluta söka så fort osorterade element hittats.

c) Vad blir antalet jämförelser i värstafallet med metoden i deluppgift b om du har n element? ✎

d) Man kan kolla om en sekvens är sorterad med det listiga tricket att först zippa sekvensen med sin egen svans och sedan kolla om alla element-par uppfyller sorteringskriteriet, alltså `xs.zip(xs.tail).forall(???)` där `???` byts ut mot lämpligt predikat. Vilken typ har 2-tupeln `xs.zip(xs.tail)` om `xs` är av typen `Vector[Int]`? Implementera `isSorted` med detta listiga trick.

Uppgift 8. *Insättningssortering på plats.* Implementera och testa sortering på plats i en array med heltal med [9].

Implementera och testa funktionen nedan i Scala med följande signatur:

```
def insertionSort(xs: Array[Int]): Unit
```

Placera metoden i ett objekt med lämpligt namn, samt skapa ett huvudprogram med testkod. Kompilera och kör från terminalen. Börja med att skriva sorteringsalgoritmen i pseudokod.

[8] stackoverflow.com/questions/14146990/what-algorithm-is-used-by-the-scala-library-method-vector-sorted

[9] en.wikipedia.org/wiki/Insertion_sort

Uppgift 9. *Sortering till ny sekvens med urvalssortering.* Implementera och testa sortering till ny sekvens med urvalssortering[10] i Scala, enligt nedan skiss. Du har nytta av lösningen till uppgift **??** i kapitel **??**.

```
def selectionSort(xs: Seq[String]): Seq[String] =
  def indexOfMin(xs: Seq[String]): Int = ???
  val unsorted = xs.toBuffer
  val result = scala.collection.mutable.ArrayBuffer.empty[String]
  /*
  så länge unsorted inte är tom
    minPos = indexOfMin(unsorted)
    elem   = unsorted.remove(minPos)
    result.append(elem)
  */
  result.toVector
end selectionSort
```

12.2.2 Uppgifter om trådar och jämlöpande exekvering

Uppgift 10. *Trådar.* Klassen `java.lang.Thread` används för att skapa **trådar** med jämlöpande exekvering (eng. *concurrent execution*). På så sätt kan man få olika koddelar att köra samtidigt.

Klassen `Thread` definierar en tom `run`-metod. Vill man att tråden ska göra något vettigt får man överskugga `run` med det man vill ska göras.

En tråd körs igång med metoden `start` och då anropas automatiskt `run`-metoden och tråden exekverar koden i `run` jämlöpande med övriga trådar. Om man anropar `run` direkt blir det *inte* jämlöpande exekvering.

a) Skapa en tråd som gör något som tar lite tid och kör med `run` resp. `start`.

```
1 def zzz = { print("zzzzzz"); Thread.sleep(5000); println(" VAKEN!")}
2 zzz
3 val t2 = new Thread{ override def run = zzz }
4 t2.run; println("Gomorron!")
5 t2.start; println("Gomorron!")
6 t2.start
```

b) Vad händer om man anropar `start` mer än en gång på samma tråd?

c) Skapa två trådar med överskuggade `run`-metoder och kör igång dem samtidigt enligt nedan. Vilken ordning skrivs hälsningarna ut efter rad 3 resp. rad 4 nedan? Förklara vad som händer.

```
1 val g = new Thread{ override def run = for i <- 1 to 100 do print("Gurka ") }
2 val t = new Thread{ override def run = for i <- 1 to 100 do print("Tomat ") }
3 g.run; t.run
4 g.start; t.start
```

d) Använd `Thread.sleep` enligt nedan. Är beteendet helt förutsägbart (deterministiskt)? Förklara vad som händer. Du kan (om du kör Linux) avbryta REPL med Ctrl+C[11].

[10] en.wikipedia.org/wiki/Selection_sort

[11] stackoverflow.com/questions/6248884/can-i-stop-the-execution-of-an-infinite-loop-in-scala-repl

```
def ibland(block: => Unit) = new Thread {
  override def run = while(true) { block; Thread.sleep(600) }
}.start
ibland(print("zzz ")); ibland(print("snark ")); ibland(println("hej!"))
```

Uppgift 11. *Jämlöpande variabeluppdatering.* Skriv klasserna `Bank` och `Kund` i en editor och klistra sedan in koden i REPL.

```
class Bank:
  private var _saldo = 0;
  def saldo: Int = _saldo
  def sättIn(): Unit = _saldo += 1
  def taUt(): Unit   = _saldo -= 1
end Bank

class Kund(bank: Bank):
  def slösaSpara(): Unit =
    bank.taUt()
    Thread.sleep(1)
    bank.sättIn()
  end slösaSpara
end Kund
```

a) Använd funktionen `ibland` från föregående uppgift och kör nedan rader i REPL. Resultatet av jämlöpande variabeluppdatering blir här heltokigt och leder till mycket upprörda bankkunder och -ägare. Förklara vad som händer.

```
val bank = new Bank
println(bank.saldo)
bank.sättIn()
println(bank.saldo)
bank.taUt()
println(bank.saldo)

val bamse = new Kund(bank)
val skutt = new Kund(bank)

bamse.slösaSpara()
skutt.slösaSpara()
println(bank.saldo)

def ofta(block: => Unit) = new Thread { // varje millisekund
  override def run = while true do { block;  Thread.sleep(1)}
}.start

ofta(bamse.slösaSpara()); ofta(skutt.slösaSpara())

def ibland(block: => Unit) = new Thread {  // varje 600 ms
  override def run = while(true) do { block; Thread.sleep(600) }
}.start

ibland(println(bank.saldo))
```

Uppgift 12. *Trådsäkra `AtomicInteger`.* Det finns stöd i JVM för att åstadkomma uppdateringar som inte kan avbrytas av andra trådar under pågående minnesskrivning. En operation som inte kan avbrytas kallas **atomär** (eng. *atomic*). Studera dokumentationen för `AtomicInteger`[12] och prova nedan kod. Förklara vad som händer.

Använd funktionerna `ofta` och `ibland` från tidigare uppgifter.

```
class SäkerBank:
  import java.util.concurrent.atomic.AtomicInteger
  private var _saldo = new AtomicInteger
  def saldo: Int = _saldo.get
  def sättIn(): Unit = _saldo.incrementAndGet()
  def taUt(): Unit   = _saldo.decrementAndGet()
end SäkerBank

class SäkerKund(bank: SäkerBank):
  def slösaSpara =
    bank.taUt()
    Thread.sleep(1)
    bank.sättIn()
  end slösaSpara
end SäkerKund
```

```
1 val sb = new SäkerBank
2 val farmor = new SäkerKund(sb)
3 val vargen = new SäkerKund(sb)
4
5 ofta(farmor.slösaSpara); ofta(vargen.slösaSpara)
6
7 ibland(println(sb.saldo))
```

Uppgift 13. *Jämlöpande exekvering med `scala.concurrent.Future`.* Att skapa och hålla reda på trådar kan bli ganska omständligt och knepigt att få rätt på. Med hjälp av `scala.concurrent.Future` kan man på ett enklare sätta skapa jämlöpande exekvering.

> *Bakgrund:* Med en `Future` skapas jämlöpande exekvering som ”under huven” använder ett ramverk som heter Akka[13], skrivet i Scala och Java. Akka erbjuder automatisk multitrådning med s.k. trådpooler och möjliggör avancerad parallellprogrammering på en hög abstraktionsnivå, där man själv slipper skapa instanser av klassen `Thread`. I stället kan man helt enkelt placera sin kod inramad med `Future{ "körs parallellt" }` efter att man importerat det som behövs.

a) För att skapa jämlöpande exekvering med `Future` behöver man först göra import enligt nedan; då skapas ett exekveringssammanhang med trådpooler redo för användning. Starta om REPL och studera felmeddelandet efter rad 1 nedan. Importera därefter enligt nedan. Vad har `f` för typ?

```
1 scala> concurrent.Future { Thread.sleep(1000); println("En sekund senare!") }
2 scala> import scala.concurrent._
3 scala> import ExecutionContext.Implicits.global
4 scala> val f = Future { Thread.sleep(1000); println("En sekund senare!") }
```

[12] docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicInteger.html
[13] http://akka.io/

b) Skapa en procedur `printLater` enligt nedan som skriver ut argumentet efter slumpmässig tid. Förklara vad som händer nedan.

```
scala> def printLater(a: Any): Unit =
         Future { Thread.sleep((math.random() * 10000).toInt); print(a + " ") }
scala> (1 to 42).foreach(i => printLater(i)); println("alla är igång!")
```

c) Skapa enligt nedan en `Future` som räknar ut hur många siffror det är i ett väldigt stort tal. Med `onComplete` kan man ange vad som ska göras när den tunga beräkningen är färdig; detta kallas att ”registrera en callback”. Vilken returtyp har `big`? Hur många siffror har det stora talet? Vad har `r` för typ? Justera argumentet till `big` om du inte orkar vänta på resultatet...

```
scala> BigInt(10).pow(100)
scala> BigInt(10).pow(100).toString.size
scala> def big(n: Int) = Future { BigInt(n).pow(n).toString.size }
scala> big(1234567).onComplete{r => println(r + " siffror") }
```

d) Den stora vinsten med `Future` är att man kan köra vidare under tiden, varför anropet av `Future` kallas **icke-blockerande** (eng. *non-blocking*). Det händer ibland att man ändå vill blockera exekveringen i väntan på ett resultat. Man kan då använda objektet `scala.concurrent.Await` och dess metod `result` enligt nedan. Använd `big` från föregående uppgift och gör en blockerande väntan på resultatet enligt nedan. Vad händer? Vad händer om du väntar för kort tid?

```
scala> import scala.concurrent.duration._
scala> Await.result(big(1234567), 20.seconds)
```

Uppgift 14. *Använda `Future` för att göra flera saker samtidigt.* I denna uppgift ska du ladda ner webbsidor parallellt med hjälp av `Future`, så att en nedladdning kan avslutas under tiden en annan dröjer.

a) Koden för en minimal webbsida ser ut som nedan. Du kan beskåda sidan här: `http://fileadmin.cs.lth.se/pgk/mini.html` eller skriva in nedan kod i en fil som heter något som slutar på `.html` och öppna filen i din webbläsare.

```
<!DOCTYPE html>
<html>
<body>
HELLO WORLD!
</body>
</html>
```

b) För att simulera slöa webbservrar kan man ladda ner en sida via sajten `http://deelay.me/`
Ladda ner ovan sida med 2 sekunders fördröjning:
`http://deelay.me/2000/http://fileadmin.cs.lth.se/pgk/mini.html`

c) Man kan ladda ner webbsidor med `scala.io.Source`. Vad händer nedan? Försök, med ledning av hur `delay` beräknas, uppskatta hur lång tid du måste vänta i medeltal, i bästa fall, respektive värsta fall, innan du kan se första webbsidan i vektorn `laddningar` nedan?

```
scala> def ladda(url: String) = scala.io.Source.fromURL(url).getLines.toVector
scala> def slöladda(url: String) =
         val delay = (math.random() * 1000 + 2000).toInt
         val delaySite = s"http://deelay.me/$delay/"
         ladda(delaySite+url)
       end slöladda
scala> ladda("http://fileadmin.cs.lth.se/pgk/mini.html")
scala> def seg = slöladda("http://fileadmin.cs.lth.se/pgk/mini.html")
scala> val laddningar = Vector.fill(10)(seg)
scala> laddningar(0)
```

d) Innan vi kan köra igång en `Future` så måste vi, som visats i uppgift 13 importera den underliggande exekveringsmiljön som är redo att parallelisera ditt program i trådar utan att du själv måste skapa dem. Vad händer nedan?

```
scala> import scala.concurrent._
scala> import ExecutionContext.Implicits.global
scala> val f = Future(seg)
scala> f   // kolla om den är klar annars prova igen senare
scala> f
```

e) Ladda indata utan att blockera (eng. *non-blocking input*). Förklara vad som händer nedan.

```
scala> val nonBlocking = Future(Vector.fill(10)(seg))
scala> nonBlocking   // kolla igen senare om ej klar
scala> nonBlocking
```

f) Ladda indata separat i olika parallella trådar. Förklara vad som händer nedan. Kör uttrycket på rad 3 nedan upprepade gånger i snabb följd efter varandra med pil-upp+Enter i REPL.

```
scala> val para = Vector.fill(10)(Future(seg))
scala> para
scala> para.map(_.isCompleted)
scala> para.map(_.isCompleted) // studera hur de blir färdiga en efter en
scala> para(0)
```

g) Registrera en callback med metoden `onComplete`. Förklara vad som händer nedan.

```
scala> val action = Vector.fill(10)(Future(seg))
scala> action(0).onComplete(xs => println(s"ready:$xs"))
scala> // vänta tills laddning på plats 0 är klar
```

h) Registrera en callback för felhantering i händelse av undantag med metoden `onFailure`. Förklara vad som händer nedan.

```
scala> def lycka  = { Thread.sleep(3000); println(":)") }
scala> def olycka = { Thread.sleep(3000); 42 / 0; lycka }
scala> Future(lycka ).onFailure{ case e => println(s":( $e") }
scala> Future(olycka).onFailure{ case e => println(s":( $e") }
```

Uppgift 15. Räkna ut stora primtal parallellt genom att använda nedan funktioner. Implementera `isPrime` enligt pseudokod från den engelska wikipediasidan om primtalstest[14] med den s.k. ”naiva algoritmen”. Räkna ut 10 st slumpvisa primtal med 16 siffror vardera. Gör beräkningarna parallellt med hjälp av `Future`.

```
def isPrime(n: BigInt): Boolean = ???

def nextPrime(start: BigInt): BigInt =
  var i = start
  while !isPrime(i) do i += 1
  i
end nextPrime

def randomBigInt(nDigits: Int): BigInt =
   def rndChar = ('0' + (math.random() * 10).toInt).toChar
   val str = Array.fill(nDigits)(rndChar).mkString
   BigInt(str)
randomBigInt
```

Uppgift 16. *Svara på teorifrågor.*

a) Vad är en tråd?

b) Hur skapar man en tråd med klassen `Thread`?

c) Hur startar man en tråd?

d) Vilka problem kan man råka ut för om man uppdaterar samma resurs i flera olika trådar?

e) Vad innbär det att kod är *trådsäker*?

f) Nämn några fördelar med att använda Future jämfört med att använda trådar direkt.

Uppgift 17. *Klasser med atomär uppdatering.* Läs om och testa klasserna AtomicBoolean, AtomicDouble och AtomicReference för atomär uppdatering i paketet `java.util.concurrent.atomic`.

Använd några av dessa tillsammans med `scala.concurrent.Future`.

Uppgift 18. *Skapa din egen multitrådade webbserver.*

a) Skriv in[15] nedan kod i en editor och spara i en fil med namn `webserver.scala` och kompilera och kör med `scala-cli run webserver.scala` och beskriv vad som händer när du med din webbläsare surfar till adressen:
`http://localhost:8089/abbasillen`

```
1 package webserver
2
3 import java.net.{ServerSocket, Socket}
4 import java.io.OutputStream
5 import java.util.Scanner
6 import scala.util.Try
7
```

[14] en.wikipedia.org/wiki/Primality_test

[15] Eller ladda ner här: github.com/lunduniversity/introprog/blob/master/compendium/examples/simple-web-server/webserver.scala

```
8  object html:
9    def page(body: String): String = //minimal web page
10     s"""<!DOCTYPE html>
11        |<html>
12        |<head><meta charset="UTF-8"><title>Min Sörver</title></head>
13        |<body>
14        |$body
15        |</body>
16        |</html>
17        """.stripMargin
18
19   def header(length: Int): String = //standardized header of reply
20     s"HTTP/1.0 200 OK\nContent-length: $length\n" +
21      "Content-type: text/html\n\n"
22
23
24 object start:
25   def handleRequest(cmd: String, uri: String, socket: Socket): Unit =
26     val os = socket.getOutputStream
27     val response = html.page("Baklänges: " + uri.reverse)
28     os.write(html.header(response.size).getBytes("UTF-8"))
29     os.write(response.getBytes("UTF-8"))
30     os.close
31     socket.close
32   end handleRequest
33
34   def serverLoop(server: ServerSocket): Unit =
35     println(s"http://localhost:${server.getLocalPort}/hej")
36     while true do
37       Try {
38         var socket = server.accept  // blocks thread until connect
39         val scan = new Scanner(socket.getInputStream, "UTF-8")
40         val (cmd, uri) = (scan.next, scan.next)
41         println(s"Request: $cmd $uri")
42         handleRequest(cmd, uri, socket)
43       }.recover{ case e: Throwable => s"Connection failed: $e" }
44
45   def main(args: Array[String]) =
46     val port = Try{ args(0).toInt }.getOrElse(8089)
47     serverLoop(new ServerSocket(port))
```

b) Du ska nu skapa en webbserver som gör något lite mer intressant. Den ska svara med det 13:e Fibonacci-talet[16] om du surfar till `http://localhost:8089/fib/13`. Spara din webbserver från föregående deluppgift under det nya namnet `fibserver.scala` och använd koden nedan och lägg till och ändra så att din server kan svara med Fibonaccital. Vi börjar med att räkna ut Fibonaccital i funktionen `compute.fib` nedan på ett onödigt processorkrävande sätt med exponentiell tidskomplexitet så att webbservern verkligen får jobba, för att i senare deluppgifter implementera `compute.fib` med linjär tidskomplexitet och därmed undvika onödig planetuppvärmning.

```
// lägg till nedan i webserver.scala från
//    https://github.com/lunduniversity/introprog/blob/master/compendium/examples/simple-web

object compute:
```

[16] https://sv.wikipedia.org/wiki/Fibonaccital

```
  def fib(n: BigInt): BigInt =
    if n < 0 then 0 else
    if n == 1 || n == 2 then 1
    else fib(n - 1) + fib(n -2)
  end fib
end compute

def fibResponse(num: String) =
  num.toIntOption match
    case Some(n) => html.page(s"fib($n) == " + compute.fib(n))
    case None    => html.page(s"FEL: skriv ett heltal, inte $num")

def errorResponse(uri:String) = html.page(s"Error: $uri </br> use /fib/heltal")


// ändra handleRequest i start i webserver.scala till
  def handleRequest(cmd: String, uri: String, socket: Socket): Unit =
    val os = socket.getOutputStream
    val afterSlash = uri.toString.drop(1) // skip initial slash
    println(s"afterSlash:$afterSlash")
    val response: String =
      if afterSlash.startsWith("fib/") then fibResponse(afterSlash.stripPrefix("fib/"))
      else errorResponse(uri)
    os.write(html.header(response.size).getBytes("UTF-8"))
    os.write(response.getBytes("UTF-8"))
    os.close
    socket.close
  end handleRequest
```

Kör i terminalen med `scala-cli run webserver.scala` och beskriv vad som händer i din webbläsare när du surfar till servern.

c) Surfa efter flera stora Fibonacci-tal samtidigt i olika flikar i din browser. Hur märks det att servern bara kör i en enda tråd?

d) Gör din server multitrådad med hjälp av den nya server-loopen nedan.

```
import scala.concurrent._
import ExecutionContext.Implicits.global

  def serverLoop(server: ServerSocket): Unit = {
    println(s"http://localhost:${server.getLocalPort}/hej")
    while (true) {
      Try {
        var socket = server.accept  // blocks thread until connect
        val scan = new Scanner(socket.getInputStream, "UTF-8")
        val (cmd, uri) = (scan.next, scan.next)
        println(s"Request: $cmd $uri")
        Future { handleRequest(cmd, uri, socket) }.onFailure {
          case e => println(s"Reqest failed: $e")
        }
      }.recover{ case e: Throwable => s"Connection failed: $e" }
    }
  }
```

e) Surfa efter flera stora Fibonacci-tal samtidigt i olika flikar i din browser. Hur märks det att servern är multitrådad?

f) Det är onödigt att räkna ut samma Fibonacci-tal flera gånger. Med hjälp av en cache i form av en föränderlig `Map` kan du spara undan redan uträknade värden. Det funkar dock inte med en vanlig `scala.collection.mutable.Map` i vår multitrådade webbserver, eftersom den inte är **trådsäker** (eng. *thread-safe*). Med trådosäkra föränderliga datastrukturer blir det samma besvärliga beteende som i uppgift 11.

Du ska i stället använda `java.util.concurrent.ConcurrentHashMap`. Sök upp dokumentationen för `ConcurrentHashMap` och försök förstå koden nedan. Hur fungerar metoderna `containsKey`, `put` och `get`?

```
object compute {
  import java.util.concurrent.ConcurrentHashMap
  val memcache = new ConcurrentHashMap[BigInt, BigInt]

  def fib(n: BigInt): BigInt =
    if (memcache.containsKey(n)) {
      println("CACHE HIT!!! no need to compute: " + n)
      memcache.get(n)
    } else {
      println("cache miss :( must compute fib:  " + n)
      val f = fastFib(n)
      memcache.put(n, f)
      f
    }

  private def fastFib(n: BigInt): BigInt = {
    if (n < 0) 0 else
    if (n == 1 || n == 2) 1
    else fib(n - 1) + fib(n -2)
  }
}
```

g) Använd ovan `fib`-objekt i en ny version av din webserver. Spara den i en ny kodfil med namnet `fibserver-memcached.scala`. Undersök hur snabbt det går med stora Fibonaccital med den nya varianten. Hur stora tal kan du räkna ut? Kan servern fortsätta efter överflödad stack? Förklara varför.

h) Nu när vi kan få väldigt stora Fibonacci-tal kan det vara användbart att stoppa in radbrytningar på webbsidan. Html-taggen `</br>` ger en radbrytning.

```
  def insertBreak(s: String, n: Int = 80): String = {
    if (s.size < n) s
    else s.take(n) + "</br>" + insertBreak(s.drop(n),n)
  }
```

Använd den rekursiva funktionen ovan för att pilla in radbrytningstaggar på var n:te position i långa strängar. Testa hur det ser ut på webbsidan med ovan funktion när din server svarar med väldigt stora tal.

i) Vi ska nu använda det större heap-minnet i stället för stack-minnet och därmed inte begränsas av stackens max-storlek. Skriv om `fastFib` så att den använder en `while`-sats i stället för ett rekursivt anrop. Denna uppgift är ganska klurig, men om du kör fast kan du snegla i lösningarna i Appendix för inspiration.

Hur stora tal klarar din server nu? Vad händer med servern när minnet tar slut? Hur kan du skydda servern så att den inte kan hänga sig?

12.3 Projektuppgift: bank

12.3.1 Fokus

- ☐ Kunna implementera ett helt program efter given specifikation
- ☐ Kunna sätta samman olika delar från olika moduler
- ☐ Förstå hur Java-klasser kan användas i Scala
- ☐ Förstå och bedöma när immutable/mutable såväl som var/val bör användas i större sammanhang
- ☐ Kunna använda sig av kompanjonsobjekt
- ☐ Kunna läsa och skriva till fil
- ☐ Kunna söka i olika datastrukturer på olika sätt

12.3.2 Bakgrund

I detta projekt ska du skriva ett program som håller reda på bankkonton och kunder i en bank. Programmet ska utöver att hålla reda på bankens nuvarande tillstånd även föra historik över alla tillståndsändringar. Historiken ska vara så pass detaljerad att det nuvarande tillståndet kan återskapas genom att återuppspela alla ändringar som finns lagrade i historiken.

Programmet ska vara helt textbaserat, man ska alltså interagera med programmet via terminalen där en meny skrivs ut och input görs via tangentbordet.

Du ska skriva större delen av programmet själv, utan någon färdig kod. Programmet ska dock följa de specifikationer som ges i uppgiften, såväl som de objektorienterade principer du lärt dig i kursen.

12.3.3 Krav

Kraven för bankapplikationen återfinns här nedan. För att bli godkänd på denna uppgift måste samtliga krav uppfyllas:

- Programmet ska ha följande menyval:
  - 1. Hitta konton för en viss kontoinnehavare med angivet ID.
  - 2. Söka efter kunder på (del av) namn.
  - 3. Sätta in pengar på ett konto.
  - 4. Ta ut pengar på ett konto.
  - 5. Överföra pengar mellan två olika konton.
  - 6. Skapa ett nytt konto.
  - 7. Ta bort ett befintligt konto.
  - 8. Skriva ut bankens alla konton, sorterade i bokstavsordning efter innehavare.
  - 9. Skriva ut historiken över alla ändringar av bankens tillstånd.
  - 10. Återställa banken till tillståndet den hade vid ett givet datum. För enkelhetens skull får du permanent kassera all historik som skapades efter det datum banken återställs till.
  - 11. Avsluta.
- När något av följande sker ska programmet notera det i historiken:

  - Pengar sätts in på ett konto.
  - Pengar tas ut från ett konto.
  - Pengar överförs mellan två konton.
  - Ett konto skapas.
  - Ett konto tas bort.

- Historiken ska sparas både i minnet och i en fil.
- Då programmet startas ska det läsa in historikfilen för att återskapa tillståndet som banken hade tidigare.
- Formatet för historikfilen ska vara detsamma som i denna exempelfil: https://github.com/lunduniversity/introprog/blob/master/workspace/w13_bank_proj/bank_log.txt
- Allt som berör användargränssnittet (såsom utskrifter till terminalen och inläsning från terminalen) ska ske i `BankApplication` eller hjälpklasser till `BankApplication`, inte i någon annan av klasserna som specificeras i uppgiften.
- Alla metoder och attribut ska ha lämplig synlighet, så att interna, förändringsbara delar inte i onödan exponeras.
- Valen av val/var och immutable/mutable måste vara lämpliga.
- Programmet ska fungera som i de bifogade exemplen på körning av programmet.
- Rimlig felhantering ska finnas. Det är alltså önskvärt att programmet inte kraschar då användaren matar in felaktig input, utan istället säger till användaren att input är ogiltig. Du kan dock anta att historikfilen alltid är i rätt format.
- Programdesignen ska följa de specifikationer som är angivna nedan.
- Det räcker med att banken ska kunna hantera heltal, men detta ska göras med klassen `BigInt` för att tillåta stora belopp. Om din bank hanterar decimaltal ska detta ske med `BigDecimal` för att undvika avrundningsfel.
- Klassen `BankAccount` ska generera ett unikt kontonummer för varje konto. Dessa ska återställas om bankens tillstånd återställs till ett tidigare datum, d.v.s. att om en återställning av banken tar bort ett konto så ska dess kontonummer återigen bli tillgängligt.
- Det enda sättet att förändra tillståndet för en `Bank` ska vara (förutom att anropa `returnToState`) att anropa `doEvent` med en `BankEvent` som beskriver tillståndsförändringen. Vid en första anblick kan detta kan verka lite väl bökigt, men när ändringshistoriken ska implementeras kommer det vara till stor hjälp att det finns en `BankEvent` som representerar varje ändring.
- Du ska inför redovisningen generera automatisk dokumentation baserat på dokumentationskommentarer enligt instruktioner i Appendix E. Du ska skriva relevanta dokumentationskommentarer för minst hälften av dina publika metoder. Det är ofta användbart att skriva dokumentationskommentarerna *före* implementationen av metodkroppen.

12.3.4 Design

Nedan följer beskriving av medlemmar som de olika klasserna bankapplikationen måste innehålla. Dessa påbörjade klasser finns i kursens workspace, tillsammans med de färdigskrivna klasserna `HistoryEntry` och `Date` samt `BankEvent` med tillhörande subtyper: https://github.com/lunduniversity/introprog/tree/master/workspace/w13_bank_proj

```
package bank

/**
 * A customer of a bank with provided name and id.
 */
case class Customer(name: String, id: Long):
  override def toString(): String = ???
```

```
package bank

/**
 * A bank account for hold by a specific customer.
 * The account is given a unique account number and initially
 * has a balance of 0 kr.
 */
class BankAccount(val holder: Customer):
  val accountNumber: Int = ???

  /**
   * Deposits the provided amount in this account.
   */
  def deposit(amount: BigInt): Unit = ???

  /**
   * Returns the balance of this account.
   */
  def balance: BigInt = ???

  /**
   * Withdraws the provided amount from this account,
   * if there is enough money in the account. Returns true
   * if the transaction was successful, otherwise false.
   */
  def withdraw(amount: BigInt): Boolean = ???

  override def toString(): String = ???
```

```
package bank

import time.Date

/**
 * A bank with no accounts and no history.
 */
```

```
class Bank():
 /**
   * Returns a list of every bank account in the bank.
   * The returned list is sorted in alphabetical order based
   * on customer name.
   */
  def allAccounts(): Vector[BankAccount] = ???

  /**
   * Returns the account holding the provided account number.
   */
  def findByNumber(accountNbr: Int): Option[BankAccount] = ???

  /**
   * Returns a list of every account belonging to
   * the customer with the provided id.
   */
  def findAccountsForHolder(id: Long): Vector[BankAccount] = ???

  /**
   * Returns a list of all customers whose names match
   * the provided name pattern.
   */
  def findByName(namePattern: String): Vector[Customer] = ???

  /**
   * Executes an event in the bank.
   * Returns a string describing whether the
   * event was successful or failed.
   */
  def doEvent(event: BankEvent): String = ???

  /**
   * Returns a log of all changes to the bank's state.
   */
  def history(): Vector[HistoryEntry] = ???

  /**
   * Resets the bank to the state it had at the provided date.
   * Returns a string describing whether the event was
   * successful or failed.
   */
  def returnToState(returnDate: Date): String = ???
```

12.3.5 Tips

- Använd ett `match`-uttryck för att hantera de olika subtyperna av `BankEvent` när du implementerar `doEvent`.

```
event match {
  case Deposit(account, amount) => ???
  case Withdraw(account, amount) => ???
```

```
  case Transfer(accFrom, accTo, amount) => ???
  case NewAccount(id, name) => ???
  case DeleteAccount(account) => ???
}
```

- För att skriva till fil på ett enkelt sätt kan man t.ex. använda sig av statiska metoder i klassen `Files` som finns tillgänglig i `java.nio.file`. För att undvika portabilitetsproblem kan man då använda sig av ett bestämt `Charset`, t.ex. `UTF_8`, som finns tillgänglig i `java.nio.charset.StandardCharsets.UTF_8`.

- För att läsa ifrån en fil kan du använda `introprog.IO`. Studera speciellt metoden `appensString` och hur ny-rad-tecken hanteras i `appendLines` https://github.com/lunduniversity/introprog-scalalib/blob/master/src/main/scala/introprog/IO.scala

- Var noggrann med att dina tester innehåller fler fall än de som givits i exempel (se 12.3.9), vilka kan behövas för mer omfattande testning och avlusning och efterfrågas på redovisningen.

12.3.6 Obligatoriska uppgifter

Uppgift 1. Implementera klassen `Customer`. Testa så att den fungerar REPL.

Uppgift 2. Implementera klassen `BankAccount`. Testa så att den fungerar i REPL.

Uppgift 3. Skapa ett huvudprogram i singelobjektet `BankApplication`. Gör så att huvudprogrammet skriver ut menyn korrekt och kan ta input från tangentbordet som motsvarar de menyval som ska finnas. Låt val av menyerna ge ett meddelande som berättar för användaren att att de ännu ej är implementerade.

Uppgift 4. Implementera klassen `Bank`.

a) Implementera menyval 6. När användaren väljer att skapa ett nytt konto ska `BankApplication` skapa ett `NewAccount`-objekt som den sedan använder som argument i ett anrop till `doEvent` i `Bank`. Det är i `doEvent` (eller en privat funktion som anropas från `doEvent`) som kontot faktiskt ska skapas.

b) Implementera menyval 8. Kontrollera att både menyval 6 och 8 fungerar rätt.

c) Implementera menyval 9. Varje gång `doEvent` exekveras utan fel ska dess `BankEvent`-argument läggas till i historiken tillsammans med det nuvarande datumet.

d) Implementera alla andra menyval, förutom menyval 10. Testa de nya menyvalen noga efterhand som du implementerar dem, i synnerhet så att ändringshistoriken fungerar korrekt. Gör de utökningar du anser behövs.

Uppgift 5. Implementera säkerhetskopiering av historiken.

a) När något läggs till i historiken ska det också skrivas till en historikfil omedelbart. Banken ska ej behöva avslutas för att utskriften ska hamna på fil, om så vore fallet kan information gå förlorad om banken kraschar. Använd `toLogString`-metoden i `HistoryEntry` för att få utskrifter i rätt format.

b) När programmet startar ska det läsa in alla händelser från historikfilen och återuppspela dem en efter en. På så sätt kan bankens tillstånd återställas, fastän vi

bara har sparat ändringshistoriken och inte själva tillståndet. Använd `fromLogString`-metoden i `HistoryEntry` när du läser in strängar från filen.

Uppgift 6. Implementera menyval 10 genom att först nollställa bankens tillstånd och sedan återuppspela allt i historiken som hände före det givna datumet. Resten av historiken bör tas bort permanent, både i minnet och i historikfilen.

12.3.7 Frivilliga extrauppgifter

Gör först klart projektets obligatoriska delar. Därefter kan du, om du vill, utöka ditt program enligt följande.

Uppgift 7. Implementera ett nytt menyalternativ som skriver ut all kontohistorik för en given person. I historiken ska finnas typ av händelse med tillhörande parametrar, dåvarande saldo vid händelsen, såväl som datumet för händelsen. (Du kan ha nytta av denna funktion när du testar ditt program.)

Uppgift 8. Skriv en eller flera av klasserna `Customer` och `BankAccount` i Java istället och använd dig av dessa i din Scala-kod. (Detta är en nyttig uppgift som förberedelse inför efterkommande fördjupningskurs, som har Java som huvudspråk.)

12.3.8 Exempel på historikfil

I workspace-katalogen för denna projektuppgift medföljer en historikfil. Inläsning och utskrift ska ske med dess format. Varje rad representerar en händelse, och formatet för en rad är: **År Månad Dag Timme Minut BankEventTyp Argument**. De olika sorternas `BankEvent` representeras med följande bokstäver: D för `Deposit`, W för `Withdraw`, T för `Transfer`, N för `NewAccount` och E för `DeleteAccount`.

12.3.9 Exempel på körning av programmet

Nedan visas möjliga exempel på körning av programmet. Data som matas in av användaren är markerad i fetstil. Ditt program måste inte se identiskt ut, men den övergripande strukturen såväl som resultat av körningen ska vara densamma. När det första exemplet börjar förutsätts det att banken inte har några konton.

Listan över val, som är markerad i kursiv stil i det första exemplet, är inte utskriven i senare exempel för att spara plats på pappret. Ditt program ska alltid skriva ut listan över val före användaren ska mata in ett val.

---

1. Hitta konton för en given kund
2. Sök efter kunder på (del av) namn
3. Sätt in pengar
4. Ta ut pengar
5. Överför pengar mellan konton
6. Skapa nytt konto
7. Radera existerande konto
8. Skriv ut alla konton i banken
9. Skriv ut ändringshistoriken
10. Återställ banken till ett tidigare datum
11. Avsluta
Val: **6**
Namn: **Adam Asson**
Id: **6707071234**
Nytt konto skapat med kontonummer:
1000
10:03 14/5-2016

Val: **1**
Id: **6707071234**
Konto 1000 (Adam Asson, id 6707071234) 0 kr
10:04 14/5-2016

- -
Val: **6**
Namn: **Berit Besson**
Id: **8505255678**
Nytt konto skapat med kontonummer: 1001
10:12 14/5-2016

- -
Val: **8**
Konto 1000 (Adam Asson, id 6707071234) 0 kr
Konto 1001 (Berit Besson, id 8505255678) 0 kr
10:13 14/5-2016

- -
Val: **2**
Namn: **adam**
Adam Asson, id 6707071234
10:15 14/5-2016

- -
Val: **6**
Namn: **Berit Besson**
Id: **8505255678**
Nytt konto skapat med kontonummer: 1002
13:56 14/5-2016

- -
Val: **2**
Namn: **erit**
Berit Besson, id 8505255678
14:01 14/5-2016

- -
Val: **3**
Kontonummer: **1000**
Summa: **5000**
Transaktionen lyckades.
14:36 14/5-2016

- -
Val: **5**
Kontonummer att överföra ifrån: **1000**
Kontonummer att överföra till: **1001**
Summa: **1000**
Transaktionen lyckades.
14:37 14/5-2016

- -
Val: **8**
Konto 1000 (Adam Asson, id 6707071234) 4000 kr
Konto 1001 (Berit Besson, id 8505255678) 1000 kr
Konto 1002 (Berit Besson, id 8505255678) 0 kr
14:52 14/5-2016

- -
Val: **7**
Ange konto att radera: **1002**
Transaktionen lyckades.
14:54 14/5-2016

- -
Val: **8**
Konto 1000 (Adam Asson, id 6707071234) 4000 kr
Konto 1001 (Berit Besson, id 8505255678) 1000 kr
14:55 14/5-2016

- -
Val: **9**
10:03 14/5-2016: Skapade ett konto tillhörandes Adam Asson, id 6707071234
10:12 14/5-2016: Skapade ett konto tillhörandes Berit Besson, id 8505255678
13:56 14/5-2016: Skapade ett konto tillhörandes Berit Besson, id 8505255678
14:36 14/5-2016: Satte in 5000 kr i konto 1000
14:37 14/5-2016: Överförde 1000 kr från konto 1000 till konto 1001
14:54 14/5-2016: Raderade konto 1002
14:58 14/5-2016

- -
Val: **10**
Vilket datum vill du återställa banken till?
År: **2016**
Månad: **5**
Datum (dag): **14**
Timme: **10**
Minut: **5**
Banken återställd.
15:00 14/5-2016

- -

Val: **9**
10:03 14/5-2016: Skapade ett konto tillhörandes Adam Asson, id 6707071234
15:00 14/5-2016

- -

Val: **8**
Konto 1000 (Adam Asson, id 6707071234) 0 kr
15:01 14/5-2016

- -

Val: **3**
Kontonummer: **1001**
Summa: **5000**
Transaktionen misslyckades. Inget sådant konto hittades.
15:06 14/5-2016

- -

Val: **4**
Kontonummer: **1000**
Summa: **1000**
Transaktionen misslyckades. Otillräckligt saldo.
15:23 14/5-2016

12.4 Projektuppgift: `music`

Förberedelser

- ☐ Testa så att datorn du ska använda på redovisningen kan spela upp ljud med `javax.sound.midi` genom att köra igång `Main` i den givna koden.
- ☐ Det är bra om du kan ta med hörlurar till datorsalen så att du inte stör andra.
- ☐ Hämta given kod via kursen github-plats eller via hemsidan under Download.

12.4.1 Bakgrund

När man skriver program skapar man ofta modeller av en viss verklig *domän*, som kan vara t.ex. försäkringskassans regelverk eller en fysiksimulering i ett datorspel. För att kunna skapa sådana modeller behöver man ofta skaffa sig *domänkunskap* genom att noga sätta sig in i vad olika koncept i domänen innebär och hur de är relaterade. Med denna kunskap kan du skapa kod som modellerar domänen, utifrån noga valda förenklingar av den komplexa verkligheten. Förmåga att kunna skapa domänmodeller utgör en viktig grund för konsten att utveckla bra programvarusystem, och du kommer lära dig mer om detta i kommande kurser.

I denna laboration ska du skapa ett program baserat på en förenklad modell av domänen *musik*. Du får färdig kod som modellerar hur toner är uppbyggda, samt hur olika stränginstrument fungerar. Med denna domänmodell ska du skapa ditt eget musikprogram som använder *ackord* som är uppbyggda av flera toner som spelas tillsammans.

12.4.2 Domänmodell

Tonhöjd

En **ton** (eng. *note*) som spelas på ett instrument, t.ex. ett piano eller en gitarr, har en **tonhöjd** (eng. *pitch*) som är relaterad till den specifika grundfrekvens som tonens ljud har. I vår modell av musikdomänen tillordnar vi olika distinkta tonhöjder ett unikt heltal. En tonhöjd kan då beskrivas av en `case class Pitch(nbr: Int)` där vi använder `nbr` i intervallet `(0 to 127)`. Heltalet `60` motsvarar en viss ton, som även har namnet `"C5"`, och som ligger ungefär i mitten av tangentbordet på ett piano.

Inom (västerländsk) musik utgår man från 12 olika *tonklasser* (eng. *pitch classes*). Dessa tolv tonklasser är ordnade i en sekvens av så kallade *halva tonsteg* och har följande **tonklassnamn**:

```
val pitchClassNames: Vector[String] =
  Vector("C","C#","D","D#","E","F","F#","G","G#","A","A#","B")
```

Efter tonklassen med namnet `B` återkommer tonklassen med namnet `C`. Symbolen `#` representerar en höjning ett halvt tonsteg. Tonklassen `C#` uttalas *siss* på svenska, och *see sharp* på engelska.[17] Notera att det är ett halvt tonsteg mellan `E` och `F`, samt mellan `B` och `C` (det finns därför varken `E#` eller `B#` i listan med tonklassnamn.[18])

På ett piano motsvaras de vita tangenterna av tonklassnamen `C D E F G A B` och de svarta tangenterna motsvaras av tonklassnamnen `C# D# F# G# A#`.

[17] Man använder även b-förtecknet ♭, som uttalas *flat* på engelska, för sänkning av en ton ett halvt tonsteg, men för enkelhetens skull bortser vi i vår modell från detta sätt att namnge toner.

[18] Varför det är på detta viset kan du läsa mer om på t.ex. Wikipedia, men du kan också nöja dig med att det helt enkelt är så på grund av historiska skäl.

En s.k. **tonklass** är ett positivt heltal i intervallet `0 until 12` som motsvaras av index för tonklassnamnet i `pitchClassNames`. En tonhöjd `Pitch(nbr)` tillhör tonklassen `nbr % 12`.

Med hjälp av heltalsdivision med 12 får man fram tonhöjdens så kallade **oktav**, alltså `nbr / 12`. Ett piano har normalt toner som spänner över 7 eller 8 oktaver. En tonhöjd `Pitch(nbr)` kan även namnges med en kombination av tonklassnamnet och tonens oktav, t.ex. `"C5"`.

Med denna domänbeskrivning kan vi skapa en mer detaljerad modell av konceptet tonhöjd med hjälp av en case-klass och tillhörande kompanjonsobjekt:

```
package music

case class Pitch(nbr: Int):  //Tonhöjd
  assert((0 to 127) contains nbr, s"Error: nbr $nbr outside (0 to 127)")
  def pitchClass: Int        = nbr % 12
  def pitchClassName: String = Pitch.pitchClassNames(pitchClass)
  def name: String           = s"$pitchClassName$octave"
  def octave: Int            = nbr / 12
  def +(offset: Int): Pitch  = Pitch(nbr + offset)
  override def toString      = s"Pitch($name)"

object Pitch:
  val defaultOctave = 5  // mittenoktaven på ett pianos tangentbord

  val pitchClassNames: Vector[String] =
    Vector("C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B")

  val pitchClassIndex: Map[String, Int] = pitchClassNames.zipWithIndex.toMap

  def fromString(s: String): Option[Pitch] = scala.util.Try {
    val (pitchClassName, octaveName) = s.partition(c => !c.isDigit)
    val octave = if octaveName.nonEmpty then octaveName.toInt else 5
    Pitch(pitchClassIndex(pitchClassName) + octave  * 12)
  }.toOption

  def apply(s: String): Pitch =
    fromString(s).getOrElse(throw new IllegalArgumentException)
```

Kompanjonsobjektet har två fabriksmetoder som kan skapa `Pitch`-objekt från en strängrepresentation av en tonhöjd.

- Metoden `fromString` omvandlar en sträng till en `Option[Pitch]`.
- Metoden `apply` kastar ett undantag om det inte går att omvandla en sträng till ett `Pitch`-objekt.

Uppgift 1. Vilka två uttryck i `Try`-blocket kan ge undantagen `NumberFormatException` respektive `NoSuchElementException`? Undersök liknande uttryck i REPL som ger dessa undantag. Hur kan fabriksmetoden `fromString` skrivas om så att den använder `toIntOption` i stället för `toInt` på strängen och `get` i stället för `apply` på nyckelvärdetabellen och utan att använda `scala.util.Try`? Vad finns det för nackdelar med att gå omvägen via `scala.util.Try` i stället för metoder som direkt ger `Option`?

Uppgift 2. Undersök klassen `Pitch` i REPL.

```
> sbt
sbt> console
Welcome to Scala 2.13.3 (OpenJDK 64-Bit Server VM, Java 11.0.8).
Type in expressions for evaluation. Or try :help.

scala> import music._
import music._

scala> Pitch("C#5").nbr
res0: Int = 61

scala> Pitch("C") + 1
res1: music.Pitch = Pitch("C#5")
```

✎ a) Ge ett exempel på argument till `Pitch.apply` som gör att undantag kastas.

✎ b) Ge ett exempel på argument till `Pitch.fromString` som ger `None`.

✎ c) Ge ett exempel på argument till `Pitch.+` som gör att undantag kastas.

Uppgift 3. Ändra i implementationen av `fromString` så att du i stället för `.toOption` gör en mönstermatchning med `match` på `Try`-resultaten `Success` och `Failure` i varsin case-klausul på formen `case Success(x) =>` och `case Failure(e: ???) => ???` och returnera lämpligt värde. Ta endast hand om de två förväntade undantagstyperna som du identifierade i uppgift 1. Gör så att alla övriga eventuella undantag kastas genom denna klausul: `case Failure(e) => throw e`

a) Testa så att din lösning fungerar i både normalfall och vid felaktigt tonhöjdsnamn.

✎ b) Undersök vad som händer om du kommenterar bort olika case-klausuler. När ger kompilatorn varning? Varför?

✎ c) Finns det någon fördel resp. nackdel med att bara fånga vissa undantag?

Ackord

Ett ackord består av flera toner som spelas tillsammans. Man kan spela ett ackord på ett stränginstrument genom att slå an en mängd toner samtidigt eller en sekvens av toner i snabb följd. Man väljer att kalla en av tonerna i ackordet (oftast den lägsta/första tonen) för **grundton** (eng. *root*).

Ett **intervall** är en tons relativa tonhöjdsavstånd från grundtonen. Ackord har olika namn beroende på vilka intervall som ingår i ackordet. Det finns väldigt många olika ackordnamn, men här begränsar vi oss för enkelhetens skull till fyra olika typer av ackord: [19]

- dur-ackord, betecknas t.ex. `"C"`,
- moll-ackord, betecknas t.ex. `"Cm"`
- sju-ackord, betecknas t.ex. `"C7"`
- maj-sju-ackord som betecknas t.ex. `"Cmaj7"`.

[19] Om du vill veta mer om ackordnamn läs här: https://en.wikipedia.org/wiki/Chord_(music)

I case-klassen `Chord` nedan finns en metod `name` som definerar vilka intervall som ingår i de olika ackordtyperna ovan, utom maj-sju-ackord. Den krångliga modulo-12-omräkningen innan matchningen gör så att intervall i olika oktaver behandlas lika, även för negativa intervall.

```
package music

case class Chord(ps: Vector[Pitch]):
  assert(ps.nonEmpty, "Chord pitch sequence is empty")

  val pitchClasses: Vector[Int]   = ps.map(_.pitchClass).toVector

  def apply(i: Int): Pitch = ps(i)

  def intervals(root: Pitch = ps(0)): Vector[Int] = ps.map(_.nbr - root.nbr)

  def relativePitchClasses(root: Pitch = ps(0)): Vector[Int] =
    intervals(root).map(i => (i%12 + 12) % 12).distinct.sorted

  def name(root: Pitch = ps(0)): String = relativePitchClasses(root) match
    case Vector(0, 4, 7)     => root.pitchClassName
    case Vector(0, 3, 7)     => root.pitchClassName + "m"
    case Vector(0, 4, 7, 10) => root.pitchClassName + "7"
    case _ => root.pitchClassName + intervals(root).mkString("[",",","]")

  override def toString = ps.map(_.name).mkString("Chord(",",",")")

object Chord:
  def apply(xs: String*): Chord = Chord(xs.map(Pitch.apply).toVector)

  def random(pitchNumbers: Seq[Int] = (60 to 72), n: Int = 3): Chord =
    val shuffled = scala.util.Random.shuffle(pitchNumbers).toVector
    Chord(shuffled.take(n).map(Pitch.apply))
```

Uppgift 4.

a) Maj-sju-ackord har samma intervall som sju-ackord, förutom att det fjärde intervallet ska vara `11` halva tonsteg från grundtonen i stället för `10`. Lägg till en case-klausul i `Chord.name` så att maj-sju-ackord ges namn som slutar med ändelsen `"maj7"`.

b) Testa din kod och kontrollera så att ackordet `Chord("D4","F#4","A4","C#5")` får namnet `"Dmaj7"`

```
scala> Chord("D4","F#4","A4","C#5").name()
res2: String = "Dmaj7"
```

c) Vilka fyra toner har ett `Cmaj7`-ackord med grundtonen `"C5"`?

Stränginstrument

Ett stränginstrument, t.ex. ett piano eller en gitarr, kännetecknas av att det kan spela ackord genom att flera strängar kan sättas i svängning så att många toner spelas

tillsammans. I vår modell fångar vi denna egenskap med en trait `StringInstrument` som har en metod `toChordOpt` som ger något ackord om minst en sträng spelas.

Gitarr och ukulele är exempel på stränginstrument som har en greppbräda (eng. *fret board*). Man spelar på ett stränginstrument med greppbräda (eng. *fretted instrument*) genom att trycka strängar mot greppbrädan med en hand, samtidigt som man knäpper på strängarna med den andra handen. Olika instanser av dessa instrument kan skilja sig åt vad gäller antalet strängar och hur dessa strängar är stämda. En normal gitarr har 6 strängar, medan en normal ukulele bara har 4 strängar. Dessa egenskaper modelleras i koden nedan.

Varje sträng har en stämskruv med vilken kan man ändra strängens spänning, strängens s.k. **stämning** (eng. *tuning*). Om man knäpper på alla lösa strängarna på en gitarr med standardstämning spelas tonerna E3, A3, D4, G4, B4, E5, räknat från den tjockaste till den tunnaste strängen.

```
package music

trait StringInstrument { def toChordOpt: Option[Chord] }

case class Piano(isKeyDown: Set[Int]) extends StringInstrument:
  def toChordOpt: Option[Chord] =
    if isKeyDown.nonEmpty then
      Some(Chord(isKeyDown.toVector.sorted.map(Pitch.apply)))
    else None

trait FrettedInstrument extends StringInstrument:
  def nbrOfStrings: Int
  def tuning: Vector[Pitch]
  def grip:   Vector[Int]
  def toChordOpt: Option[Chord] =
    val notes =
      for i <- grip.indices if grip(i) >= 0
      yield tuning(i) + grip(i)
    if notes.nonEmpty then Some(Chord(notes.toVector)) else None

case class Guitar(pos: (Int,Int,Int,Int,Int,Int)) extends FrettedInstrument:
  val grip = Vector(pos._1, pos._2, pos._3, pos._4, pos._5, pos._6)
  val nbrOfStrings = 6
  val tuning =
    "E3 A3 D4 G4 B4 E5".split(' ').map(Pitch.apply).toVector

case class Ukulele(pos: (Int,Int,Int,Int)) extends FrettedInstrument:
  val grip = Vector(pos._1, pos._2, pos._3, pos._4)
  val nbrOfStrings = 4
  val tuning =
    "A5 D5 F#5 B5".split(' ').map(Pitch.apply).toVector
```

Om man trycker på greppbrädans olika positioner får man olika toner, beroende på vilken position man trycker på. Positionerna på greppbrädan räknas från ett och uppåt. Position `0` motsvarar **lös sträng**, alltså att man slår an en sträng utan att trycka på greppbrädan över denna sträng. En negativ position, tex. `-1`, anger att en sträng inte spelas alls; många gitarrackord spelas genom att bara en delmängd av strängarna slås an. Ett exempel på ett gitarrackord visas i figur 12.2.

Figur 12.2: Ett C-dur-ackord på en gitarr motsvarande `Guitar(3,3,2,0,1,0)`.

Uppgift 5. Studera modellen av stränginstrument ovan och använd REPL för att svara på dessa frågor:

a) Vad är namnet på detta pianoackord om vi väljer att lägsta tonen i ackordet är grundton: `Piano(Set(60, 64, 67, 70))`

b) Vad heter tonerna som ingår i ackordet `Guitar(3,3,2,0,1,0)`.

c) Vad heter detta ackord om vi väljer ett A som grundton: `Ukulele(0,2,1,2)`

Elektroniska instrument

Ett elektroniskt instrument syntetiserar ljud med hjälp av analog och/eller digital elektronik, och kallas därför **synthesizer**, ofta förkortat *synt* (eng. *synth*).

De flesta moderna PC-operativsystem inkluderar mjukvaruimplementerade syntar som följer den så kallade MIDI-standarden. Java-paketet `javax.sound.midi` innehåller klasser som kan få en sådan MIDI-synt att spela musik.

MIDI-standarden baseras på en modell av ett pianotangentbord där olika toner kan vara ”på” eller ”av” beroende på om en tangent är nedtryckt eller ej. Dessa toners höjd är modellerade på samma sätt som i vår klass `Pitch`, där alltså tonhöjden `60` motsvarar tonen `"C5"`, etc. En tangent kan tryckas ner olika hårt, vilket representeras av ett heltalsvärde i `Range(0,128)` kallat `velocity`. Ett högt värde ger en stark ton, medan ett litet värde motsvarar en svag (tyst) ton.

En synt som följer MIDI-standarden kan spela upp ljud via 16 olika så kallade **kanaler** (eng. *channel*), numrerade `(0 until 16)`, där varje kanal kan ställas in så att den spelar ett ljud som t.ex. liknar ett visst verkligt instrument, så som piano eller gitarr.

I kursens workspace i paketet `music` finns en `Synth`-modul som förenklar användningen av Java-paketet `javax.sound.midi`. I modulen `Synth` finns metoden `playBlocking` som kan spela flera toner under en viss tid med hjälp av synten på ditt ljudkort. Exekveringen av ditt program blockeras tills tonerna spelats klart, därav *”blocking”* i namnet.

Metoden `playBlocking` har följande parametrar, default-argument och returtyp:[20]

```
def playBlocking(
  noteNumbers: Seq[Int] = Seq(60), // en sekvens av tonhöjder
  velocity: Int         = 60,      // hur hårt anslag i Range(0, 128)
  duration: Long        = 300,     // hur länge i millisekunder
  spread:   Long        = 50,      // millisekunder mellan tonerna
  after:    Long        = 0,       // millisekunder innan första tonen
  channel:  Int         = 0        // MIDI-kanal som spelar tonerna
): Unit
```

Uppgift 6. Anropa `playBlocking()` i REPL och undersök om din dator kan spela tonen `"C5"`. Använd gärna lurar så att du inte stör dina labbkamrater. Prova vad som händer när du ger olika argument till `playBlocking`.

Uppgift 7. Gör klart modulen `ChordPlayer` enligt nedan så att metoden `play` kan spela ett ackord. Case-klassen `Strike` representerar ett ackordanslag.

```
package music

object ChordPlayer:

  case class Strike(
    velocity: Int         = 50,   // hur hårt anslag i Range(0, 128)
    duration: Long        = 500,  // hur länge i millisekunder
    spread:   Long        = 50,   // millisekunder mellan tonerna
    after:    Long        = 0     // millisekunder innan första tonen
  )

  def play(chord: Chord, strike: Strike = Strike(), channel: Int = 0): Unit =
    strike match
      case Strike(v, d, s, a) => ???
```

Uppgift 8. Implementera ett singelobjekt med namnet `Test` med en `main`-metod som med hjälp av din `play`-metod från föregående uppgift spelar några olika ackord.

Uppgift 9. Gör en terminalapp som kan spela ackord. I kursens workspace i `w13_music_proj` finns en påbörjad terminalapp som du kan bygga vidare på. Den har redan en `Main.main`-metod som startar en loop där användaren kan ge kommando (eng. *Command Line Interface, CLI*). Kommandot `?` ger hjälp och kommandot `:q` avslutar.

```
*** Welcome to music!
music> ?
?         print help
:q        quit this app
!         play chord TODO
music> !
```

[20] Om du är nyfiken kan du studera implementationen av `Synth`-modulen här: https://github.com/lunduniversity/introprog/tree/master/workspace/w13_music_proj Koden blir lättare att förstå om du samtidigt läser api-dokumentationen av paketet `javax.sound.midi` och även lära dig mer om MIDI-standarden med hjälp av t.ex. wikipedia.

```
play chord TODO
music> :q
Goodbye music!
```

Det finns, som syns ovan, också ett påbörjat kommando `!` som är tänkt att spela ett ackord, men som än så länge bara skriver ut ett TODO-meddelande. Gör så att användaren med `!` kan spela ackord från olika instrument enligt nedan:

```
music> ! p 60 64 67
Play Piano(Set(60, 64, 67)) Chord(C5,E5,G5)
music> ! g 0 2 2 0 0 0
Play Guitar((0,2,2,0,0,0)) Chord(E3,B3,E4,G4,B4,E5)
```

Uppgift 10. Skapa ett kommando som låter användare definierar egna namn på kommandon som sedan enkelt kan köras med hjälp av det definierade namnet. Vid definition med tidigare existerande namn så ska den gamla definitionen ersättas

```
music> def Em = ! g 0 2 2 0 0 0
defined Em = ! g 0 2 2 0 0 0
music> Em
Play Guitar((0,2,2,0,0,0)) Chord(E3,B3,E4,G4,B4,E5)
```

Det ska fungera att göra nästlade def-kommando, alltså att kroppen för en def innehåller namnet på en annan def. Testa att definiera en hel låt som i sin tur består av definierade ackord.

Uppgift 11. Du ska inför redovisningen generera automatisk dokumentation baserat på dokumentationskommentarer enligt instruktioner i Appendix E. Du ska skriva relevanta dokumentationskommentarer för minst hälften av dina publika metoder. Det är ofta användbart att skriva dokumentationskommentarerna *före* implementationen av metodkroppen.

12.4.3 Valfria uppgifter

Uppgift 12. Gör så att definitioner sparas mellan körningar.

Uppgift 13. Implementera fler valfria kommandon. Du kan t.ex.

- skapa kommando som ritar en grepptabell för gitarrackord eller fingersättning på piano med `introprog.PixelWindow`.
- skapa kommando för att göra drumbeats, t.ex. `drum mygrove x x o x` för 2 hihat + basatrumma + 1 hihat och `start mygrove` och `stop mygrove` för uppspelning av beat i bakgrunden med `playConcurrently`. Se i `Synth.scala` för vilka instrument som ger trumljud med ledning av deras namn `music.Synth.instruments.map(_.getName)`, t.ex `Standard Kit` eller `Synth Drum`. Använd kanal 10 för att spela upp trumljuden.

Uppgift 14. Använd öppen-källkodsprokjektet `jline` i stället för `scala.io.StdIn.readLine` för att automatiskt få pil-upp-historik, Ctrl+A Ctrl+K, TAB-completion, etc. Se exempel på användning av `jline` här: https://github.com/bjornregnell/termut

12.5 Projektuppgift: `photo`

12.5.1 Bakgrund

Detta projekt innebär att du ska implementera en egen bildbehandlingsapplikation, en mycket förenklad variant av *Phtotoshop* eller *Gimp*.

En digital bild består av ett rutnät, en s.k. matris (eng. *matrix*), av pixlar, var och en med en viss färg. Om man har många små pixlar bredvid varandra i ett rutnät, så flyter de samman för ögat och betraktaren upplever en bild.

Bilder kan manipuleras genom applicering av olika s.k. *filter*, som förändrar bildens pixlar på ett intressanta sätt. Du ska, utifrån given matematisk teori, implementera olika filter med hjälp av speciella matrisoperationer.

Det finns olika system för hur man färgsätter pixlar. T.ex. så används CMYK-systemet (cyan, magenta, gul, svart) vid blandning av färg som ska tryckas på papper eller annat material. På en dator, däremot, används vanligtvis RGB-systemet, som har de tre grundfärgerna röd, grön och blå. Mättnaden av varje grundfärg anges av ett heltal som vi i fortsättningen förutsätter ligger i intervallet [0, 255]. 0 anger ”ingen färg” och 255 anger ”maximal färg”. Man kan därmed representera 256 × 256 × 256 = 16 777 216 olika färgnyanser. Man kan också representera gråskalor; det gör man med färger som har samma värde på alla tre grundfärgerna: (0, 0, 0) är helt svart, (255, 255, 255) är helt vitt.

I detta projekt kommer vi skapa matriser av heltal för att beräkna intressanta egenskaper hos en bild, till exempel intensiteten för varje pixel. För att spara plats vid bearbetning av stora bilder så använder vi, heltalsmatriser med typen `Short`, som använder 16 bitar i minnet, i stället för `Int`, som använder 32 bitar i minnet.

12.5.2 Förberedelser

I detta projekt har du nytta av följande delar av `introprog-scalalib` och `java.awt`:

- `introprog.Image` för bildhantering.
- `introprog.PixelWindow` och `introprog.Dialog` för användarinteraktion.
- `introprog.IO` för filhantering.
- `java.awt.Color` för hantering av pixelfärger.

Läs noga dokumentationen för klasserna i introprog här och gör egna experiment i REPL så du förstår hur de kan användas: `https://cs.lth.se/pgk/api/`
Läs om klassen `java.awt.Color` här:
`https://docs.oracle.com/en/java/javase/11/docs/api/java.desktop/java/awt/Color.html` Hämta och studera noga den kod som är given för detta projekt här:
`https://github.com/lunduniversity/introprog/tree/master/workspace/`

12.5.3 Matris med värden av typen `Short`

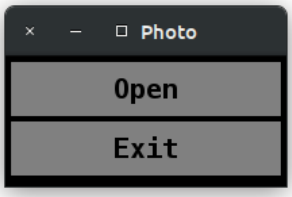
Uppgift 1. **`Matrix`**. I den givna kodfilen `Matrix.scala` finns hjälp-funktioner för att skapa och uppdatera matriser med värden av typen `Short`, för att spara minne vid stora bilder.

Gör klart saknade implementationer och testa noga i REPL så att allt fungerar som det ska innan du går vidare. *Tips:* Du har nytta av `Array.tabulate`.

```
scala> import photo.*

scala> val m = Matrix(3,3)(1,2,3,4,5,6,7,8,9) // en 3x3-matiris med Short-värden
val m: photo.Matrix = Array(Array(1, 4, 7), Array(2, 5, 8), Array(3, 6, 9))

scala> m(0,1)
val res0: Short = 4

scala> m(1,0) = 42

scala> m
val res1: photo.Matrix = Array(Array(1, 4, 7), Array(42, 5, 8), Array(3, 6, 9))

scala> m.row(0)
val res2: Array[Short] = Array(1, 42, 3)
```

12.5.4 Användargränssnitt

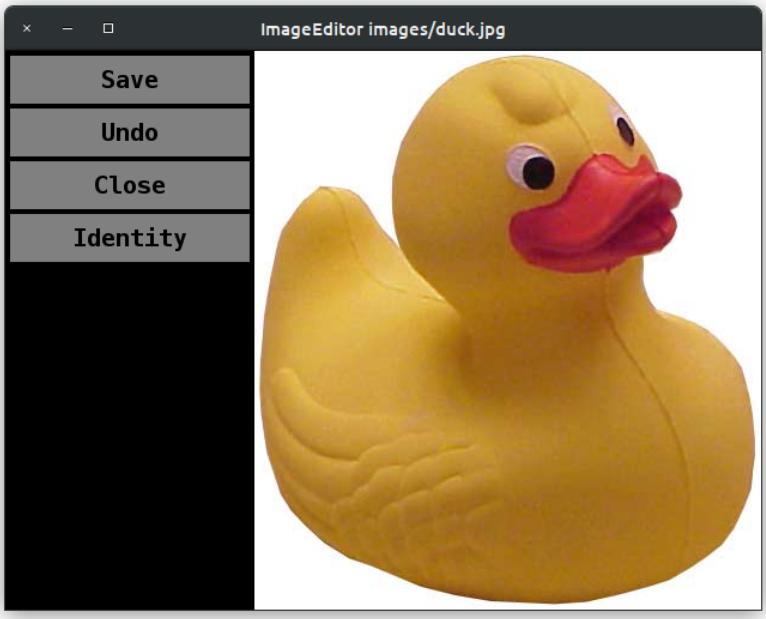
När appen startar så visas ett fönster enligt fig. 12.3, implementerat av givna koden i `Main.scala`. Med hjälp av givna `Button.scala` skapas en kolumn med knappar som är klickbara. Studera koden i `Main.scala` och `Button.scala` noga så att du förstår vad som händer. Ännu öppnas inget ImageEditor-fönster, men det ingår i näst uppgift.



Figur 12.3: Photo-applikationens startfönster.

Uppgift 2. **`ImageEditor`**. Följande krav ska implementeras:

- Du ska skapa en kodfil `ImageEditor.scala` som innehåller en klass med samma namn som implementerar ett bildredigeringsfönster med en kolumn med knappar till vänster och en bild inläst från fil till höger, så som visas i fig. 12.4.
- Vid tryck på Exit-knappen ska en varningsfråga ”Ok to Exit without save?” ges med `introprog.Dialog.isOK` och användaren ska kunna ångra avslut. Om avslut ändå väljs så ska detta ske med `System.exit(0)` så att alla ev. aktiva fönster och tillhörande trådar avbryts direkt.
- Vid tryck på Open-knappen ska en fil väljas med hjälp av `introprog.Dialog.file`. Om det i aktuell katalog finns en underkatalog vid namn `images` så ska filbläddringen börja där, annars i aktuell katalog.
- Efter OK på filöppningen ska en bild öppnas i ett bildredigeringsfönster enligt fig. 12.4 med knapparna Save, Undo, Close, plus en knapp för varje filter, till vänster om bilden. Fönstrets höjd och bredd ska avpassas så att hela bilden och alla knappar får plats.



Figur 12.4: Bildredigeringsfönstret, innan fler filter (utöver identitetsfiltret) implementerats. Varje filter som implementeras ska ha en motsvarande knapp.

- Huvudfönstret och alla bildredigeringsfönster ska fungera parallellt. Detta ska du åstadkomma genom att händelseloopen i `ImageEditor`-klassen körs som argument till metoden `runInParallell` enligt nedan:

```
def runInParallell(block: => Unit) =
  new Thread{ override def run(): Unit = block }.start

def startEventLoop(): Unit = runInParallell:
  // initialisering och händelseloop här
```

- Det ska gå att göra Undo i flera steg och återställa alla bilder före applicering av filter i tur och ordning. *Tips:* Inför i attributet `var history: Vector[Image]` som från början innehåller den ursprungliga bilden.

- Fönstrets titel ska innehålla namnet `ImageEditor` och de två sista delarna av den sökväg (eng. *path*) som öppnats, enligt exempel i fig. 12.4, eftersom en fullständig sökväg t.ex. `/home/userxyz/workspace/photo/images/duck.jpg` riskerar att inte få plats i fönstrets titelbalk.

- Vid Save ska fråga om filnamn ställas med `introprog.Dialog.file` och kontroller göras om filen redan finns eller ej, och om den finns ska en fråga med `introprog.Dialog.isOK` ställas om den ska skrivas över eller ej.

- Alla implementerade filter ska ha en knapp som applicerar filtret och sparar resultatet i historiken så att filtret kan ångras med Undo. Om filtret har argument så ska en informativ dialog öppnas där användaren kan ange argument via `introprog.Dialog.input`. Filterknapparna ska vara sorterade i bokstavsordning efter filtrets namn, se fig. 12.5.

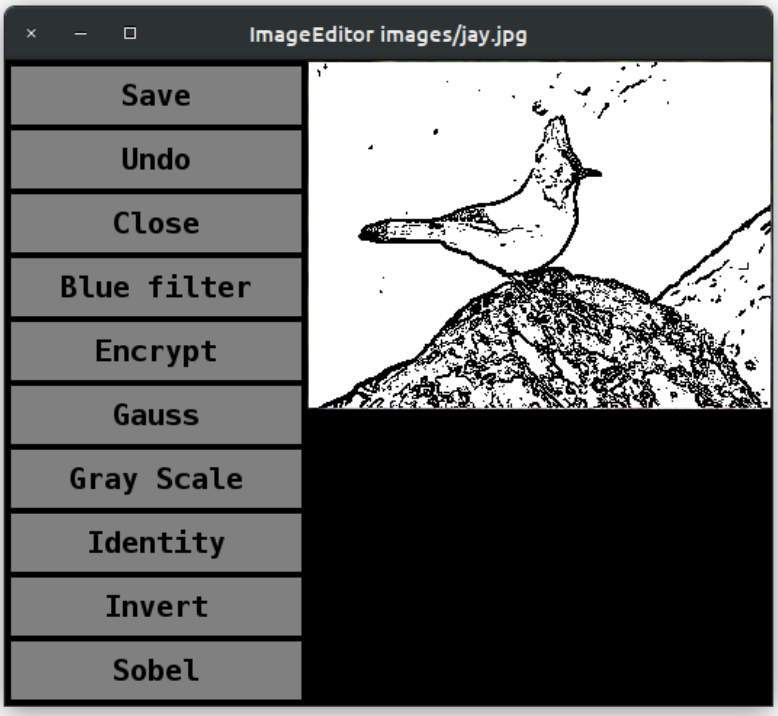
12.5.5 Filter

Du ska bygga vidare på givna koden i `Filter.scala` som visas nedan. Du ska implementera och testa ett antal olika filter som ändrar bilder på intressanta sätt med hjälp av olika matrisalgoritmer.

```
package photo

import introprog.Image

trait Filter:
  def name: String

  def argDescriptions: Seq[String] = Seq()

  def nbrOfArgs = argDescriptions.length

  def apply(im: Image, args: Double*): Image

object Filter:
  val byIndex: Vector[Filter] = Vector(Identity)

  val byName: Map[String, Filter] = byIndex.map(f => f.name -> f).toMap

  def intensity(im: Image): Matrix = ???

  def convolve(p: Matrix, x: Int, y: Int, kernel: Matrix, weight: Int): Short = ???

  object Identity extends Filter:
    val name = "Identity"
    def apply(im: Image, args: Double*): Image =
      val result = Image.ofDim(im.width, im.height)
      result.updated((x, y) => im(x, y))
```

`Filter.scala` innehåller en bastyp för alla filter med ett antal medlemmar som alla filter ska implementera, enligt nedan krav. Det finns ett färdigimplementerat filter, `Identity`, som kan användas för testsyften; detta filter gör inget annat än kopierar alla bildpunkter till en ny bild och har således ingen editerande effekt.

- Metoden `apply` ska returnera en ny bilden där filtret applicerats, utan att förändra inparameter-bilden.
- Ett filter ska kunna ha noll eller flera argument av typen `Double` som kan påverka vad som händer när filtret appliceras. Varje sådant argument ska i tur och ordning ha en kort, instruktiv beskrivning i sekvensen `argDescription`.
- Metoden `intensity` ska beräkna en s.k. *intensitetsmatris* och behövs vid implementeringen av gråskale-, Gauss- och Sobel-filtren. Hur en intensitetsmatris beräknas beskrivs nedan.
- Metoden `convolve` ska göra en s.k. faltning (medelvärdesbildning i matriser) och behövs vid implementering av Gauss- och Sobel-filtren. Hur en faltning görs beskrivs nedan.
- Alla implementerade filter ska finnas i sekvensen `byIndex`, som används i tabellen `byName`. Dessa behövs för att skapa alla filterknappar och applicera respektive filter.



Figur 12.5: Bildredigeringsfönstret med alla filter implementerade. Ett kontruförstärkande så kallat Sobel-filter är applicerat med tröskelvärde 150.

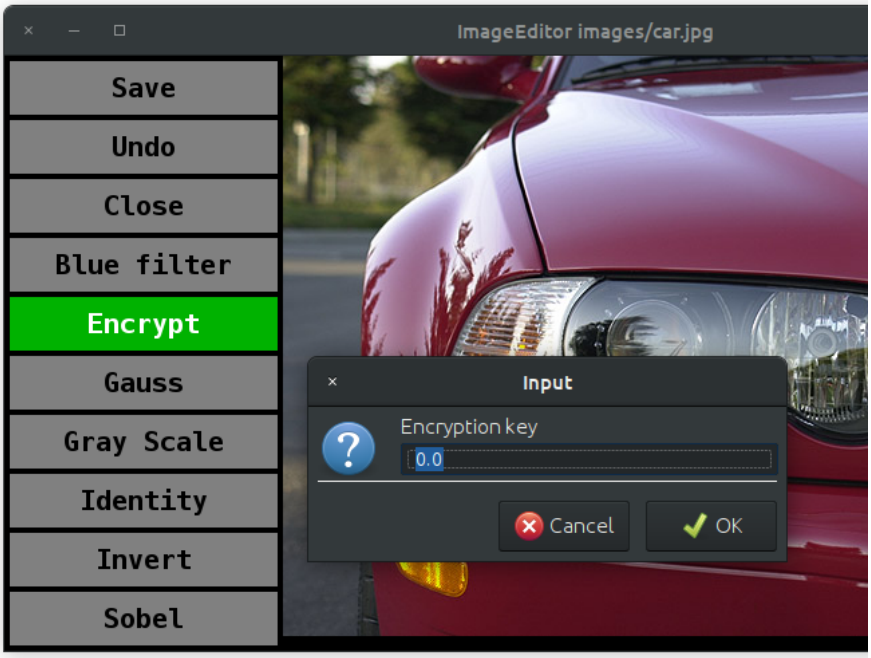
Du ska implementera och testa alla filter i uppgifterna nedan, ett i taget. Uppgifterna är ordnade i stigande svårighetsgrad.

Uppgift 3. Blåfilter. Skapa ett filter i ett singelobjekt med namn `Blue` som vid applicering ger en blå version av bilden, där varje pixel bara innehåller den blå komponenten. *Tips:* Du har nytta av metoden `getBlue` i klassen `java.awt.Color`.

Uppgift 4. Negativ. Skapa ett filter `Invert` som inverterar en bild, dvs skapar en ”negativ” kopia av bilden. Ljusa färger ska alltså bli mörka och mörka färger ska bli ljusa. Fundera över vad som kan menas med en inverterad eller negativ kopia. *Tips:* Även de nya RGB-värdena ska vara i heltal i intervallet 0 – 255. De nya RGB-värdena beräknas *inte* med något divisionsuttryck över de gamla värdena (då skulle de nya värdena bli decimaltal och inte heltal i intervallet 0 – 255).

Uppgift 5. Gråskalefilter. Skapa ett filter `GrayScale` som gör om bilden till en gråskalebild. Implementera först `intensity`-metoden i `trait Filter` genom att bilda medelvärdet av alla tre RGB-komponenterna. Använd sedan intensiteten för varje pixel för att bestämma gråskalenivån. Om intensiteten i en pixel till exempel är 105 så ska den nya gråskale-pixeln var ett `Color`-objekt med RGB-värdena (105, 105, 105).

Uppgift 6. Kryptering. Skapa ett filter `XOrCrypto` som krypterar bilden med xor-operatorn ^. Denna operator gör binär xor mellan bitarna i ett heltal. Exempelvis ger 8 ^ 127 värdet 119. Om man gör xor igen med 127, alltså 119 ^ 127, får man tillbaka värdet 8. Varje pixel krypteras genom att använda xor-operatorn med ursprungsvärdena för rött, grönt och blått tillsammans med slumpmässiga heltalsvärden som genereras



Figur 12.6: Krypteringsfilter före applicering, under pågående inmatning av nyckel.

ur en ny instans av `scala.util.Random`. Tre nya slumptal ska dras för varje pixels RGB-komponent ur samma Random-instans. Låt användaren ge ett argument som du använder som slumptalsfrö vid skapande av `Random`-instansen. På så sätt kan du återskapa bilden genom att applicera krypteringsfiltret igen, med samma argument, på den numera krypterade bilden.

Om filtrets `argDescriptions`-sekvens är icke-tom så ska `ImageEditor` fråga efter varje argument i tur och ordning och visa varje beskrivning i dialogrutan. Användarens indata görs om till ett decimaltal av typen `Double` före att argumenten används i metoden `apply`. Bestäm själv hur du vill hantera defaultvärden och felhantering om användaren anger en sträng som inte går att göra om till en `Double`. *Tips:* Du har nytta av `toDoubleOption` och `getOrElse`.

```
object XOrCrypto extends Filter:
  val name = "Encrypt"
  override val argDescriptions = Seq("Encryption key")

  def apply(im: Image, args: Double*): Image = ???
```

```
scala> Seq(8, 127, 8 ^ 127).map(_.toBinaryString)
val res0: Seq[String] = List(1000, 1111111, 1110111)

scala> 8 ^ 127
val res1: Int = 119

scala> 119 ^ 127
val res2: Int = 8
```

Uppgift 7. Gaussfilter. Ett Gaussfilter gör bilden lite mindre skarp. Gaussfiltrering är ett exempel på så kallad *faltningsfiltrering*. Faltning (eng. *convolution*) är en slags lokal medelvärdesbildning. Nya pixlar skapas genom att kombinera varje pixel med dess omgivande pixlar enligt en speciell matrisalgoritm.

För att åstadkomma detta utnyttjar man en så kallad *faltningskärna* K som är en liten kvadratisk heltalsmatris. Man placerar K över varje element i intensitetsmatrisen och multiplicerar varje element i K med motsvarande element i intensitetsmatrisen. Man summerar produkterna och dividerar summan med summan av elementen i K för att få det nya värdet på intensiteten i punkten (alltså ett slags medelvärde). Divisionen görs för att den nya intensiteten ska hamna i rätt intervall (0 – 255). Exempel:

$$intensity = \begin{pmatrix} 5 & 4 & 2 & 8 & \dots \\ 4 & 3 & 4 & 9 & \dots \\ 9 & 8 & 7 & 7 & \dots \\ 8 & 6 & 6 & 5 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \qquad K = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Här är summan av elementen i K $1+1+4+1+1 = 8$. För att räkna ut det nya värdet på intensiteten i punkten `(1, 1)` med det nuvarande värdet är 3, beräknar man följande:

$$newintensity = \frac{0\cdot 5+1\cdot 4+0\cdot 2+1\cdot 4+4\cdot 3+1\cdot 4+0\cdot 9+1\cdot 8+0\cdot 7}{8} = \frac{32}{8} = 4$$

Man fortsätter med att flytta K ett steg åt höger och beräknar på motsvarande sätt ett nytt värde för elementet med index `(1)(2)` (där det nuvarande värdet är 4 och det nya värdet blir 5). Därefter gör man på samma sätt för alla element utom för ”ramen” dvs elementen i matrisens ytterkanter.

Implementera och testa noga först metoden
`convolve(p: Matrix, x: Int, y: Int, kernel: Matrix, weight: Int): Short`
i **`trait`** `Filter` som alltså ska ge den normerade produktsumman av `kernel` och punkterna i närheten av `(x,y)` i matrisen `p` normerat med `weight`. `Tips:` Du har nytta av metoderna `round` och `toShort`.

```
scala> import photo.*

scala> val p = Matrix(4,4)(5,4,2,8,4,3,4,9,9,8,7,7,8,6,6,5)
val p: photo.Matrix = Array(Array(5, 4, 9, 8), Array(4, 3, 8, 6), Array(2, 4, 7

scala> val K = Matrix(3,3)(0,1,0,1,4,1,0,1,0)
val K: photo.Matrix = Array(Array(0, 1, 0), Array(1, 4, 1), Array(0, 1, 0))

scala> Filter.convolve(p, 1, 1, K, K.flatten.sum)
val res0: Short = 4
```

Skapa därefter ett filter `Gauss` som gör en faltning med hjälp av `convolve` för varje färgkomponent separat. Gör på följande sätt:

1. Bilda tre `short`-matriser och lagra pixlarnas red-, green- och blue-komponenter i matriserna.

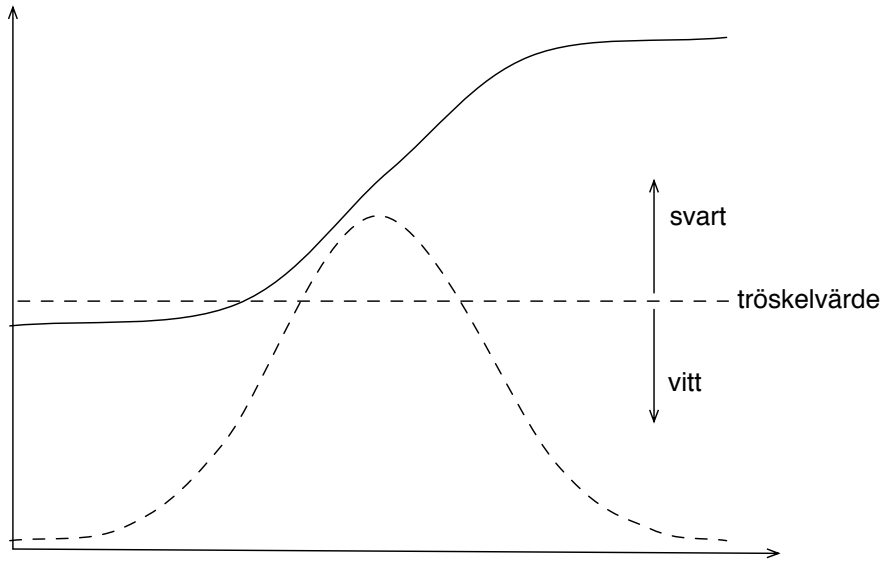
2. Utför faltningen av de tre komponenterna för varje element och uppdatera `result` med de uträknade värdena.

3. Elementen i ramen behandlas inte, men i `result` måste också dessa element få värden. Enklast är att flytta över dessa element oförändrade från `im` till `result`. (Man kan också sätta dem till `Color.BLACK`, men då kommer den filtrerade bilden att se något mindre ut.)

Använd `kernel` K enligt ovan och låt `weight` vara summan av alla element i K.

Det kan vara intressant att prova med andra värden än 4 i mitten av faltningsmatrisen. Med värdet 0 får man en större utjämning eftersom man då inte alls tar hänsyn till den aktuella pixelns värde. Låt användaren mata in argument för mittvärdet, mellan 0 och 50, och beskriv detta i `argDescriptions`. [21]

Uppgift 8. Sobelfilter. Sobelfiltrering är, precis som Gaussfiltrering, en typ av faltningsfiltrering. Med Sobelfiltrering får man dock motsatt effekt i jämförelse med Gaussfiltrering, dvs man förstärker konturer i en bild. I princip deriverar man bilden i x- och y-led och sammanställer resultatet.



Figur 12.7: En funktion (heldragen linje) och dess derivata (streckad linje).

I figur 12.7 visas en funktion f (heldragen linje) och funktionens derivata f' (streckad linje). Vi ser att där funktionen gör ett ”hopp” så får derivatan ett stort värde. Om funktionen representerar intensiteten hos pixlarna längs en linje i x-led eller y-led så motsvarar ”hoppen” en kontur i bilden. Om man sedan bestämmer sig för att pixlar där derivatans värde överstiger ett visst tröskelvärde ska vara svarta och andra pixlar vita så får man en bild med starka konturer.

Nu är ju intensiteten hos pixlarna inte en kontinuerlig funktion som man kan derivera enligt vanliga matematiska regler. Men man kan approximera derivatan, till exempel med följande formel:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

[21] Det kan ibland vara svårt att se någon skillnad mellan den Gauss-filtrerade bilden och originalbilden. Om man vill ha en riktigt suddig bild så måste man använda en större matris som faltningskärna. Prova gärna detta som extrauppgift.

Om man låter h gå mot noll så får man definitionen av derivatan. Efter ytterligare teoretiska utredningar så kan man visa att det går att uttrycka derivering i en matris med hjälp av faltning enligt följande:

1. Beräkna intensitetsmatrisen med metoden `intensity`.

2. För varje punkt i intensitetsmatrisen gör två faltningar med dessa kärnor:

$$SobelX = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad SobelY = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

   Använd metoden `convolve` med vikten 1. Koefficienterna i matrisen $SobelX$ uttrycker derivering i x-led, medan $SobelY$ uttrycker derivering i y-led. För att förklara varför koefficienterna ibland är 1, ibland 2, ibland positiva och ibland negativa, måste man studera den bakomliggande teorin noggrant, men det gör vi inte här.

3. Om resultaten av faltningen i en punkt betecknas med `sx` och `sy` så får man en indikator på närvaron av en kontur med `math.abs(sx) + math.abs(sy)`. Absolutbelopp behöver man eftersom man har negativa koefficienter i faltningsmatriserna.

4. Sätt pixeln till svart om indikatorn är större än tröskelvärdet, till vit annars. Låt tröskelvärdet bestämmas av ett argument som användaren kan ange.

Skapa ett filter `Sobel` som implementerar konturförstärkning med ovan algoritm. Se exempel i fig. 12.5. Du ska låta användaren ge tröskelvärdet med argumentbeskrivningen `"Threshold (0.0 - 255.0)"`.

Uppgift 9. Du ska inför redovisningen generera automatisk dokumentation baserat på dokumentationskommentarer enligt instruktioner i Appendix E. Du ska skriva relevanta dokumentationskommentarer för minst hälften av dina publika metoder. Det är ofta användbart att skriva dokumentationskommentarerna *före* implementationen av metodkroppen.

12.5.6 Frivilliga extrauppgifter

Uppgift 10. Kortkommando. Gör så att det blir möjligt att applicera filter med hjälp av tangenttryck. Utvidga `trait Filter` så att alla filter kan ha kortkommando. Skriv på knappen vad kortkommandot är så att användaren kan upptäcka det.

Uppgift 11. Kontrastfilter. Om man applicerar kontrastfiltrering på en färgbild så kommer bilden att konverteras till en gråskalebild. (Man kan naturligtvis förbättra kontrasten i en färgbild och få en färgbild som resultat. Då behandlar man de tre färgkanalerna var för sig.) Många bilder lider av alltför låg kontrast. Det beror på att bilden inte utnyttjar hela det tillgängliga området 0–255 för intensiteten. Man får en bild med bättre kontrast om man ”töjer ut” intervallet enligt följande formel (linjär interpolation):

```
val newIntensity = 255 * (intensity - 45) / (225 - 45)
```

Som synes kommer en punkt med intensiteten 45 att få den nya intensiteten 0 och en punkt med intensiteten 225 att få den nya intensiteten 255. Mellanliggande punkter sprids ut jämnt över intervallet `[0, 255]`. För punkter med en intensitet mindre än 45 sätter man den nya intensiteten till 0, för punkter med en intensitet större än 225 sätter man den nya intensiteten till 255. Vi kallar intervallet där de flesta pixlarna finns för `[lowCut, highCut]`. De punkter som har intensitet mindre än `lowCut` sätter man till 0, de som har intensitet större än `highCut` sätter man till 255. För de övriga punkterna interpolerar man med formeln ovan (45 ersätts med `lowCut`, 225 med `highCut`).

Det återstår nu att hitta lämpliga värden på `lowCut` och `highCut`. Detta är inte något som kan göras enkelt, eftersom värdena beror på intensitetsfördelningen hos bildpunkterna. Man börjar därför med att först beräkna bildens intensitetshistogram, dvs hur många punkter i bilden som har intensiteten 0, hur många som har intensiteten 1, . . . , till och med 255.

I de flesta bildbehandlingsprogram kan man sedan titta på histogrammet och interaktivt bestämma värdena på `lowCut` och `highCut`. Så ska vi dock inte göra här. I stället bestämmer vi oss för ett procenttal `cutOff`, som användaren kan ange som argument från terminalen, och som beräknar `lowCut` så att `cutOff` procent av punkterna i bilden har en intensitet som är mindre än `lowCut` och `highCut` så att `cutOff` procent av punkterna har en intensitet som är större än `highCut`.

Exempel: antag att en bild innehåller 100 000 pixlar och att `cutOff` är 1.5. Beräkna bildens intensitetshistogram genom registrering av varje intensitet i en heltals-array
`val histogram = Array.ofDIm[Int](256)`
och beräkna `lowCut` så att
`histogram(0) + ... + histogram(lowCut)` = 0.015 * 100000
så nära det går att komma, det blir troligen inte exakt likhet. Beräkna `highCut` på liknande sätt.

Sammanfattning av algoritmen:

1. Beräkna intensitetsmatrisen.
2. Beräkna bildens intensitetshistogram.
3. Argument från användaren användas som `cutOff`.
4. Beräkna `lowCut` och `highCut` enligt exempel ovan.
5. Beräkna den nya intensiteten för varje pixel enligt interpolationsformeln och lagra de nya pixlarna i `result`.

Skapa ett filter `Contrast` som implementerar algoritmen. I katalogen *images* kan bilden *moon.jpg* vara lämplig att testa, eftersom den har låg kontrast. Anmärkning: om `cutOff` sätts = 0 så får man samma resultat av denna filtrering som man får av `GrayScale`. Detta kan man se genom att studera interpolationsformeln.

Uppgift 12. Eget filter. Skapa ett eget valfritt filter. Till exempel så kan du skapa ett filter som tar fem argument, där de två första värdena representerar ett intensitetsintervall och de tre sista värdena representerar röd-, grön- och blå-komponenterna till en pixel som ska ersättas med denna färg då intensiteten ligger utanför det givna intervallet.

Uppgift 13. Egna interaktiva verktyg. Skapa valfria interaktiva redigeringsverktyg med mus- och tangentinput. Börja med ett markeringsverktyg som gör så att en rektangelformad del av bilden kan markeras med hjälp av musen. Gör det möjligt att applicera filter på den markerade delen av bilden. Du kan också göra så att argument till t.ex. Gauss-filtret kan ställas in med ett skjutreglage som du ritar under knappen och som kan regleras med mus eller piltangenter.

Kapitel 13

Repetition

Begrepp som ingår i denna veckas studier:

- ☐ träna på extentor
- ☐ redovisa projekt
- ☐ träna inför muntligt prov

13.1 Tips

13.1.1 På begäran 2024

Grumligt

1. När är det bra/dåligt att använda anonyma funktioner? w03
2. Klasser och kompanjonsobjekt: vad passar bäst var? w05
3. Hur göra felhantering med `Option` och `Try`? w06
4. Skillnaden mellan sats & uttryck, tex **`if`**, **`for`**? w01

Nyfiken-på

1. Flertrådad programmering
2. Fönsterhantering i introprog under huven
3. Generiska typgränser **`<: >:`**

13.1.2 Repetition: Tumregler/tips vid val av abstraktion

Extensionsmetod, singelobjekt, case-klass, klass, trait, eller enum?

- Om du vill lägga till en metod på befintlig typ utan behov av nya attribut etc., använd **`extension`**.
- Använd **`object`** om du behöver samla metoder (och variabler) i en modul som bara finns i en upplaga. Du får lokal namnrymd och punktnotation på köpet.
- Behöver du modellera **oföränderlig data**, använd en **`case class`** eller **`enum`**.
- Om du vill ha uppräknade värden som du vill kunna iterera över och matcha på i förseglad struktur, med värden i egen namnrymd, använd **`enum`**.
- Med **`case class`** och **`enum`** får du även innehållslikhet och en massa annat godis på köpet!
- Behöver du **förändringsbart tillstånd** (eng. *mutable state*) använd en vanlig **`class`**. Det normala är att det föränderliga tillståndet (de attribut som är föränderliga) är **`private`** eller **`protected`** och att all uppdatering och avläsning av tillståndet sker indirekt genom metoder (getters/setters/...).
- Behöver du en abstrakt bastyp använd en **`trait`**, speciellt om du vill ha möjlighet till inmixning. Om du vill förhindra inmixning eller underlätta användning från Java, använd **`abstract class`**.

13.1.3 Repetition: Tips om val av samling

Det är ofta enklare med oföränderliga samlingar med oföränderliga element och skapa nya samlingar vid förändring. Men för vissa algoritmer blir det enklare eller effektivare om du ändrar på plats i förändringsbar samling.

- Behöver du hantera värden i sekvens?
  - Om du klarar dig utan förändring av innehållet efter konstruktion: **`val`**-referens till `Vector`

  - Om du behöver ändra innehåll men **inte** antal element:
    `val`-referens till `Array`
  - Om du behöver ändra innehåll **och** antal element:
    `var`-referens till `Vector` och t.ex. metoden `patch`, eller
    `val`-referens till `ArrayBuffer` och t.ex. metoden `insert`
- Behöver du hantera värden `x` som ska vara unika?
  - Oföränderlig: `Set`
  - Förändringsbar: `val`-referens till `scala.collection.mutable.Set`
- Behöver du leta upp värden `x:Int` utifrån en nyckel av t.ex. String?
  - Oföränderlig: `Map[String, Int]`
  - Förändringsbar: `val`-referens till `scala.collection.mutable.Map[String, Int]`

13.1.4 Före tentan:

1. Repetera övningar och labbar i kompendiet.
2. Läs igenom föreläsningsanteckningar.
3. Studera **snabbref** **mycket noga** så att du vet vad som är givet och var det står, så att du kan hitta det du behöver snabbt.
4. Skapa och **memorera** en personlig **checklista** med programmeringsfel du brukar göra, som även inkluderar småfel, så som glömda parenteser och semikolon, och annat som en kompilator/IDE normalt hittar.
5. Tänk igenom hur du ska disponera dina 5 timmar på tentan.
6. Gör minst en extenta som om det vore **skarpt läge**:
   (a) Avsätt 5 ostörda timmar (stäng av telefon, dator etc).
   (b) Inga hjälpmedel. Bara snabbref.
   (c) Förbered dryck och tilltugg.

13.1.5 På tentan:

1. Läs igenom **hela** tentan först.
   Varför? Förstå helheten. Delarna hänger ihop.
2. Notera och begrunda specifika begrepp och definitioner.
   Varför? Begreppen är avgörande för förståelsen av uppgiften.
3. Notera förenklingar, antaganden och specialfall.
   Varför? Uppgiften blir mkt enklare om du inte behöver hantera dessa.
4. **Fråga** tentamensansvarig om du inte förstår uppgiften – speciellt om det finns misstänkta felaktigheter eller förmodat oavsiktliga oklarheter.
   Varför? Det är inte lätt att konstruera en ”perfekt” tenta.
   Du får fråga vad du vill, men det är inte säkert du får svar...
5. Läs specifikationskommentarerna och metodsignaturerna i alla givna klassspecifikationer **mycket noga**.
   Varför? Det är ett vanligt misstag att förbise de ledtrådar som ges där.
6. Återskapa din memorerade personliga checklista för vanliga fel som du brukar göra och avsätt tid till att gå igenom den på tentan. Varje fix plockar poäng!

7. Lämna in ett försök även om du vet att lösningen inte är fullständig. Det gäller att plocka så många poäng det går. En ofullständig lösning kan ändå ge poäng.
8. Om du har svårigheter kan det bli kamp mot klockan. Försök hålla huvudet kallt och prioritera utifrån var du kan plocka flest poäng. Ge inte upp! Ta en kort äta-dricka-paus för att få mer energi!

13.2 Övning examprep

Uppgift 1. *Gör klart ditt projekt.*

Uppgift 2. *Gör en extenta.*

Uppgift 3. *Förbered din projektredovisning.*

Uppgift 4. *Skapa dokumentation för ditt projekt.* Läs mer i Appendix E om hur du skapar dokumentation.

Uppgift 5. *Repetera övningar och laborationer.*

Kapitel 14

MUNTLIGT PROV

På schemalagd tid senast sista läsveckan i december ska du avlägga ett obligatoriskt muntligt prov för handledare. Du måste vara godkänt på alla laborationer för att få göra det muntliga provet. Syftet med provet är att kontrollera att du har godkänd förståelse för de begrepp som ingår i kursen. Du rekommenderas att förbereda dig noga inför provet, t.ex. genom att gå igenom grundläggande begrepp för varje kursmodul och repetera grundövningar och laborationer.

Provet sker som ett stickprov ur kursens innehåll. Du kommer att få några slumpvis valda frågor där du ombeds förklara några av de begrepp som ingår i kursen. Du får även uppdrag att skriva kod som liknar kursens övningar och förklara hur koden fungerar. Du kan träna på typiska frågor här: https://cs.lth.se/pgk/muntabot/

Om det visar sig oklart huruvida du uppnått godkänd förståelse kan du behöva komplettera ditt muntliga prov. Kontakta kursansvarig för information om omprov.

Del III

Appendix

Appendix A

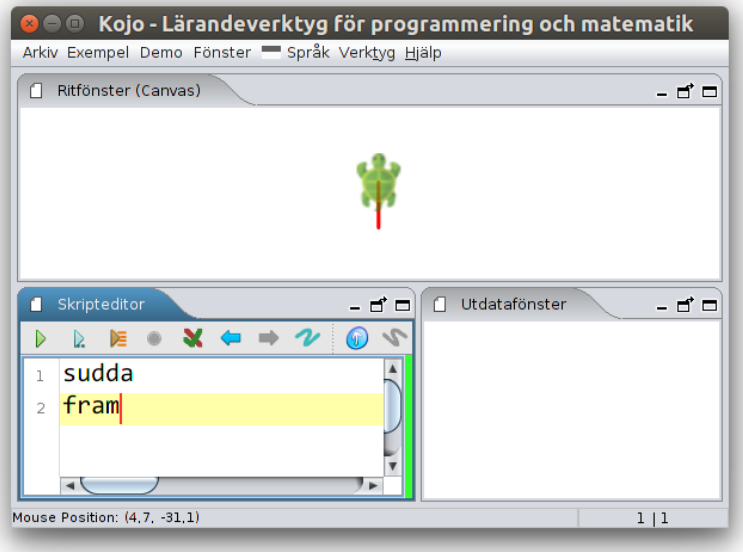
Kojo

A.1 Vad är Kojo?

Kojo[1] är en integrerad utvecklingsmiljö för Scala som är speciellt anpassad för programmeringsundervisning i grundskolan. Kojo används i LTH:s Science Center Vattenhallen för utbildning av grundskolelärare i programmering och vid skolbesök och annan besöksverksamhet, i vilken lärare och studenter vid LTH arbetar som handledare.

Kojo är öppen källkod och utvecklingsgemenskapen leds av Lalit Pant från Indien. I Kojo finns även lättillgängliga bibliotek som gör tröskeln lägre att programmera rörlig grafik och enkla spel.

Under kursens första laboration använder vi grafikbiblioteket i Kojo för att illustrera grundläggande begrepp, så som sekvens, alternativ, repetition och abstraktion.



Figur A.1: Den nybörjarvänliga utvecklingsmiljön Kojo för Scala på svenska.

[1] en.wikipedia.org/wiki/Kojo_(programming_language)

A.2 Använda grafikbiblioteket i Kojo

Kojo bygger på den beprövade pedagogiska idén med sköldpaddsgrafik (eng. *turtle graphics*)[2], där du skriver program som styr en sköldpadda med en penna under magen. När sköldpaddan rör sig bildas ett streck av valfri färg på skärmen. Beroende på hur du bestämmer att sköldpaddan ska röra sig och vilken färg du bestämmer att pennan ska ha, kan du skapa olika intressanta bilder och samtidigt lära dig om programmeringens grunder.

Under kursens första laboration ska du använda grafikbiblioteket i Kojo tillsammans med editorn VS `code` och `scala-cli` i terminalen (se appendix B och C). Ladda ner filen `kojolib.scala` från `https://fileadmin.cs.lth.se/kojolib.scala` och spara i en ny katalog med hjälp av din webbläsare, eller via dessa kommandon (notera att det är stora bokstaven `O` och inte en nolla i optionen `-sLO`):

```
> mkdir w01-kojo
> cd w01-kojo
> curl -sLO https://fileadmin.cs.lth.se/kojolib.scala
```

Nu kan du starta Scala REPL och rita med Kojo så här:

```
> scala-cli repl .
Welcome to Scala 3.1.2 (17.0.2, Java OpenJDK 64-Bit Server VM).
Type in expressions for evaluation. Or try :help.

scala> fram; höger; fram; vänster
```

Du kan starta VS `code` i aktuellt bibliotek så här:

```
> code .
```

Skriv nedan progam i VS `code` och spara det i samma katalog som den tidigare nedladdade filen, under ett nytt valfritt filnamn, t.ex. `rita.scala`:

```
@main def rita = { fram; höger; fram; vänster }
```

Kör ditt fristående program med:

```
> scala-cli run .
```

Du ska nu få upp ett fönster som heter Kojo Canvas med en sköldpadda som ritat två streck. När du stänger fönstret så avslutas programmet. Prova fler sköldpaddsfunktioner enligt tabell A.1.

I stället för att ladda ned filen `kojolib.scala` så kan du placera dess innehåll på lämpligt ställe i ditt program enligt nedan. Observera att raden som börjar med `//> using dep` ska vara en enda lång rad utan radbrytningar.

```
//> using scala "3"
//> using dep "net.kogics:kojo-lib:0.2.0,url=https://github.com/lunduniversity/
    introprog/releases/download/kojo-lib-0.2.0/kojo-lib-0.2.0.jar"

export net.kogics.kojo.Swedish.*, padda.*, CanvasAPI.*, TurtleAPI.*
export java.awt.Color
```

Scala-koden för den svenska paddans api finns här:
github.com/litan/kojo-lib/blob/main/src/main/scala/net/kogics/kojo/i18n/Swedish.scala

[2] `https://en.wikipedia.org/wiki/Turtle_graphics`

A.3 Kojo Desktop

Kojo finns som fristående skrivbordsapplikation, kallad Kojo Desktop. Kojo Desktop innehåller en egen editor med syntaxfärgning för Scala, men fungerar ännu så länge bara för Scala 2. En av de synligaste skillnaderna mellan Scala 2 och Scala 3 är att klammerparenteser vid flerradiga funktioner är nödvändiga i Scala 2, medan Scala 3 har valfria klammerparenteser. Så om du använder Kojo Desktop behöver du komma ihåg att omgärda sekvenser av rader som hör ihop med { och }.

Kojo Desktop är förinstallerad på LTH:s datorer och körs igång med terminalkommandot `kojo` eller via applikationsmenyn. För instruktioner om hur du installerar Kojo Desktop på din egen dator se här: lth.se/programmera/installera

När du startar Kojo första gången, välj ”Svenska” i språkmenyn och starta om Kojo. Därefter fungerar grafikfunktionerna på svenska enligt tabell A.1 på sidan 201. När du startat om Kojo inställt på svenska ser programmet ut ungefär som i figur A.1 på sidan 199.

Det finns ett antal användbara kortkommando som du hittar i menyerna i Kojo Desktop. Undersök speciellt Ctrl+Alt+Mellanslag som ger autokomplettering baserat på det du börjat skriva.

A.4 Kojo i Webbläsaren

En begränsad variant av Kojo finns tillgänglig för programmering direkt i din webbläsare här: `http://kojo.lu.se/`

När du trycker på play-knappen så kompileras din kod på en server till Javascript via ScalaJS och därefter körs Javascript-koden i din webbläsare. Kojo på webben är också ännu så länge begränsad till Scala 2 och kräver att du omgärdar sekvenser av rader som hör ihop med { och }.

A.5 Mer om Kojo

I detta dokument finns en enkel introduktion till Kojo:
”Introduction to Kojo” `http://www.kogics.net/kojo-ebooks#intro`
I tabell A.1, som fortsätter på efterföljande sidor, finns ett urval av kommando i Kojo på svenska och engelska.

Tabell A.1: Ett urval av funktioner i Kojo. Se även lth.se/programmera

Svenska/Engelska	*Vad händer?*
`sudda` `clear()`	Ritfönstret suddas
`fram` `forward()`	Paddan går framåt 25 steg.
`fram(100)` `forward(100)`	Paddan går framåt 100 steg.
`höger` `right(90)`	Paddan vrider sig 90 grader åt höger.
`höger(45)` `right(45)`	Paddan vrider sig 45 grader åt höger.

`vänster` `left(90)`	Paddan vrider sig 90 grader åt vänster.
`vänster(45)` `left(45)`	Paddan vrider sig 45 grader åt vänster.
`hoppa` `hop()`	Paddan hoppar 25 steg utan att rita.
`hoppa(100)` `hop(100)`	Paddan hoppar 100 steg utan att rita.
`hoppaTill(100, 200)` `jumpTo(100, 200)`	Paddan hoppar till läget (100, 200) utan att rita.
`gåTill(100, 200)` `moveTo(100, 200)`	Paddan vrider sig och går till läget (100, 200).
`hem` `home()`	Paddan går tillbaka till utgångsläget (0, 0).
`öster` `setHeading(0)`	Paddan vrider sig så att nosen pekar åt höger.
`väster` `setHeading(180)`	Paddan vrider sig så att nosen pekar åt vänster.
`norr` `setHeading(90)`	Paddan vrider sig så att nosen pekar uppåt.
`söder` `setHeading(-90)`	Paddan vrider sig så att nosen pekar neråt.
`mot(100,200)` `towards(100, 200)`	Paddan vrider sig så att nosen pekar mot läget (100, 200)
`sättVinkel(90)` `setHeading(90)`	Paddan vrider nosen till vinkeln 90 grader.
`vinkel` `heading`	Ger vinkelvärdet dit paddans nos pekar.
`sakta(5000)` `setAnimationDelay(5000)`	Gör så att paddan ritar jättesakta.
`println("hej")`	Skriver texten `hej`.
`textstorlek(100)` `setPenFontSize(100)`	Paddan skriver med jättestor text nästa gång du gör skriv.
`båge(100, 90)` `arc(100, 90)`	Paddan ritar en båge med radie 100 och vinkel 90.
`cirkel(100)` `circle(radie)`	Paddan ritar en cirkel med radie 100.
`synlig` `visible()`	Paddan blir synlig.
`osynlig` `invisible()`	Paddan blir osynlig.
`läge.x` `position.x`	Ger paddans x-läge
`läge.y` `position.y`	Ger paddans y-läge
`pennaNer` `penDown()`	Sätter ner paddans penna så att den ritar när den går.

`pennaUpp` `penUp()`	Lyfter upp paddans penna så att den INTE ritar när den går.
`pennanÄrNere`	Kollar om pennan är nere eller inte.
`färg(rosa)` `setPenColor(pink)`	Sätter pennans färg till rosa.
`fyll(lila)` `setFillColor(purple)`	Sätter ifyllnadsfärgen till lila.
`fyll(genomskinlig)` `setFillColor(noColor)`	Gör så att paddan inte fyller i något när den ritar.
`bredd(20)` `setPenThickness(20)`	Gör så att pennan får bredden 20.
`sparaStil` `saveStyle()`	Sparar pennans färg, bredd och fyllfärg.
`laddaStil` `restoreStyle()`	Laddar tidigare sparad färg, bredd och fyllfärg.
`sparaLägeRiktning` `savePosHe()`	Sparar pennans läge och riktning
`laddaLägeRiktning` `restorePosHe()`	Laddar tidigare sparad riktning och läge
`siktePå` `beamsOn()`	Sätter på siktet.
`sikteAv` `beamsOff()`	Stänger av siktet.
`bakgrund(svart)` `setBackground(black)`	Bakgrundsfärgen blir svart.
`bakgrund2(grön,gul)` `setBackgroundV(green, yellow)`	Bakgrund med övergång från grönt till gult.
`upprepa(4){fram; höger}` `repeat(4){forward; right}`	Paddan går fram och svänger höger 4 gånger.
`avrunda(3.99, 2)`	Avrundar 3.99 till två decimaler, alltså 4.0
`slumptal(100)`	Ger ett slumptal mellan 0 och 99.
`slumptalMedDecimaler(100)`	Ger ett slumptal mellan 0 och 99.99999999
`systemtid`	Ger nuvarande systemklocka i sekunder.
`räknaTill(5000)`	Kollar hur lång tid det tar för din dator att räkna till 5000.

Appendix B

Terminalfönster

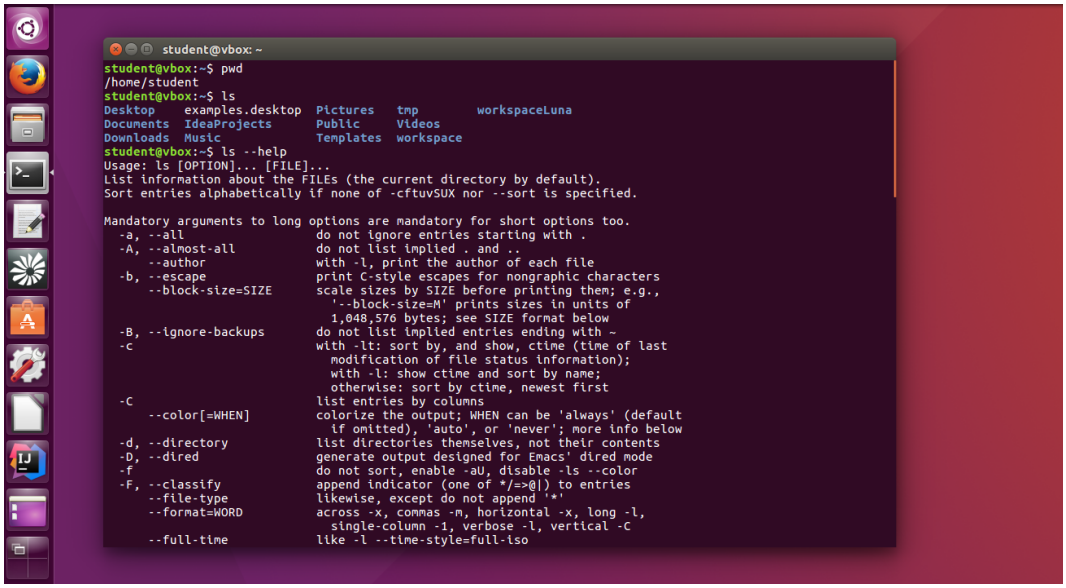
B.1 Vad är ett terminalfönster?

I ett terminalfönster kan man skriva kommandon som kör program och hanterar filer. När man programmerar använder man ofta terminalkommandon för att kompilera och exekvera sina program.

Terminal i Linux

I Ubuntu trycker du lättast **Ctrl+Alt+T** eller sök efter ”terminal” i app-menyn. Då öppnas ett fönster med en blinkande markör som visar att det är redo att ta emot dina textkommando. Ett exempel på kommando är `ls` som skriver ut en lista med filer i den aktuella katalogen, så som visas i fig. B.1.

Det som visas i ett terminalfönster sköts av ett **kommandoskal** (eng. *command shell*), som är redo att ta emot kommando efter en prompt som slutar med ett $-tecken. När du skriver ett kommando och trycker Enter anropar kommandoskalet en kommandotolk som tolkar och utför dina kommandon. Om ett kommando inte kan tolkas, skrivs ett felmeddelande.



Figur B.1: Terminalfönster i Ubuntu öppnas med Ctrl+Alt+T.

Det finns många användbara kortkommando, varav de viktigaste visas i tabell B.1. Det är bra om du lär dig dessa kortkommandon utantill så att ditt arbete i terminalen går snabbt och smidigt.

pil upp/ner	bläddra i kommandohistoriken
Tab	”auto-complete”, fyll i resten baserat på vad du skrivit hittills
Tab Tab	två tryck på Tab listar flera alternativ, om så finnes
Ctrl+A	”ahead”, flytta markören till början av raden
Ctrl+E	”end”, flytta markören till slutet av raden
Ctrl+K	”kill”, ta bort tecken från markören till radens slut
Ctrl+U	”undo”, ta bort tecken från markören till början av raden
Ctrl+Y	”yank”, sätt in det som senast togs bort
Ctrl+Z	”zleep”, stoppa pågående process, skriv sedan `bg` för bakgrundskörning
Ctrl+L	rensa terminalfönstret
Ctrl+D	avsluta kommandoskalet

Tabell B.1: Viktiga kortkommandon i Linux terminalfönster.

Ctrl+C orsakar normalt ett avbrott av pågående process, men om du vill att Ctrl+C ska vara ”Copy” som vanligt för att kopiera markerad text, kan du ställa om detta med terminalförnstrets meny ”Edit → Keyboard Shortcuts”, eller liknande.

PowerShell, Cmd och Linux i Microsoft Windows

Det finns flera olika sätt att köra terminalkommando i Windows:

- **Powershell**. I Microsoft Windows finns kommandotolken *Powershell* med speciell kommandosyntax. Den är inte Linux-baserad men det finns alias definierade för några vanliga Linux-kommandon, inkluderat `ls`, `cd` och `pwd`. Du startar Powershell t.ex. genom att trycka på Windows-knappen och skriva `powershell`. Du kan också, medan du bläddrar bland filer, klicka på filnamnsraden överst i filbläddraren och skriva `powershell` och tryck Enter; då startas Powershell i aktuellt katalog.

- **Cmd**. Det finns även i Windows den ursprungliga, gamla kommandotolken *Cmd* med helt andra kommandon. Till exempel skriver man i Cmd kommandot `dir` i stället för `ls` för att lista filer.

- **WSL**. I både Windows 10 och 11 kan du även köra Ubuntu-terminalen med hjälp av Windows Linux Subsystem (WSL), vilket rekommenderas, speciellt om du inte har möjlighet att göra s.k. dual boot[1].
  - Se vidare här om hur du kan installera WSL under Windows, (WSL2 rekommenderas före WSL1 om din maskin klarar det): `https://docs.microsoft.com/en-us/windows/wsl/install`
  - Det finns även ett smidigt tillägg till VS Code som heter Remote-WSL som gör att du kan editera filer i Windows som finns i WSL, se vidare här: `https://code.visualstudio.com/docs/remote/wsl-tutorial`

[1] Läs mer om dual boot här och be gärna någon om hjälp som gjort det förr: https://www.linuxtechi.com/dual-boot-ubuntu-22-04-and-windows-11/

- **Windows Terminal**. Den nya Microsoft-appen *Windows Terminal* rekommenderas oavsett om du använder Powershell, Cmd eller WSL. Läs mer här om hur du installerar Windows Terminal:
  `https://docs.microsoft.com/en-us/windows/terminal/`

Terminal i Apple macOS/OS X

Apple OS X och macOS är Unix-baserade operativsystem. De flesta vanliga terminalkommandon som fungerar i Linux fungerar också under Apple OS X och macOS. Du startar ett terminalfönster i Apples operativsystem genom att klicka på förstoringsglaset uppe till höger, skriva `terminal`, och trycka Enter.

B.2 Vad är en path/sökväg?

När du skriver ett kommando i terminalen, eller kör vilket program som helst på din dator, behöver operativsystemet identifera i vilken fil programmets maskinkod ligger innan programmet kan köras.

Lokaliseringen av filer sker med hjälp av en **sökväg** (eng. *path*), som anger en position i filsystemet. Ofta betraktas filsystemet som ett upp-och-ned-vänt träd, och kallas därför även ”filträdet”. Den ”översta” positionen kallas ”rot” (eng. *root*) och betecknas med ett enkelt snedstreck `/`. Kataloger som ligger i kataloger utgör förgreningar i trädet. En sökväg pekar ut vägar genom trädet som behövs för att nå ”löven”, som utgörs av själva filerna.

Du kan se var ett program ligger i Linux med hjälp av kommandot `which` enligt nedan.[2] Listan med kataloger i sökvägen avskiljs med snedstreck.

```
$ which java
/usr/lib/jvm/oracle_jdk8/bin/java
$ which ls
/bin/ls
```

En sökväg kan vara **absolut** eller **relativ**. En absolut sökväg utgår från roten och visar hela vägen från rot till destination, t.ex. `/usr/bin/firefox`, medan en relativ sökväg utgår från aktuellt katalog (där du ”står”) och börjar *inte* med ett snedstreck.

Alla operativsystem håller reda på en mängd olika sökvägar för att kunna hitta speciella filer i filträdet. Dessa sökvägar lagras i s.k. **miljövariabler** (eng. *environment variables*). Det finns en *speciell* miljövariabel som heter kort och gott **PATH**, i vilken alla sökvägar till de program finns, som ska vara tillgängliga för din användaridentitet direkt för exekvering genom sina filnamn, *utan* att man behöver ange absoluta sökvägar.

Du kan i Linux se vad som ligger i din PATH med kommandot `echo $PATH` medan man i Windows Powershell skriver `$env:Path` där det bara är första bokstaven som ska vara en versal. I Linux separeras katalogerna i sökvägen med kolon, medan Windows använder semikolon.

Ibland kan du behöva uppdatera din PATH för att program som du installerat och ska bli allmänt tillgängliga. Detta görs på lite olika sätt i olika operativsystem, för Linux se t.ex. här: stackoverflow.com/questions/14637979/how-to-permanently-set-path-on-linux

[2] Skriv `gcm ls` i Windows Powershell för motsvarighet till `which ls`
Eller skriv `New-Alias which get-command` för tillgång till kommandot `which` i Powershell.
stackoverflow.com/questions/63805/equivalent-of-nix-which-command-in-powershell

När man anger sökvägar finns några tecken med speciell betydelse:

`~`	”tilde”, din hemkatalog
`/`	”slash”, snedstreck anger filträdets rot om det finns i början av sökvägen, men utgör katalogsavskiljare inuti sökvägen
`.`	en punkt anger aktuell katalog, där du ”står”
`..`	två punkter anger ett steg ”upp” i filträdet
`"`	omgärda en sökväg med citationstecken, först och sist, om den innehåller annat än engelska bokstäver, t.ex. blanktecken
`\`	*backslash+blanktecken* används för att beteckna mellanslag i sökvägar som *inte* omgärdas av citationstecken

B.3 Några viktiga terminalkommando

I tabell B.2 finns en lista med några viktiga terminalkommando som är bra att lära sig utantill.

En introduktion till LTH:s datorer med exempel på hur du använder vanliga Linuxkommandon finns i denna skrift `http://www.ddg.lth.se/perf/unix/` som används i introduktionsveckan för nybörjare på datateknikprogrammet vid LTH.

På sajten `http://ss64.com/` finns en mer omfattande lista med användbara terminalkommando och tillhörande förklaringarför för Linux (Bash), Windows (Powershell, Cmd) och Apple OS X (Bash).

`ls`	lista filer i aktuell katalog (alltså där du ”står”)
`ls` *p*	lista filer i katalogen *p*
`ls -A`	lista alla filer i aktuell katalog, även gömda
`man ls`	manual för kommandot `ls`; testa även `man` för andra kommandon!
`cd` *p*	”change directory”, ändra aktuell katalog till *p*
`pwd`	”print working directory”, skriv ut sökväg för aktuell katalog
`cp` *p1 p2*	”copy”, kopiera filen med path *p1* till en ny fil kallad *p2*
`mv` *p1 p2*	”move”, byt namn på filen *p1* till *p2*
`rm` *p*	”remove”, ta bort filen *p*
`rm -r` *p*	”remove recursive”, ta bort katalogen *p* med allt innehåll; var försiktig!
`mkdir` *p*	”make dir”, skapa ett en katalog *p*
`cat` *p1 p2*	”concatenate”, skriv ut hela innehållet i en eller flera filer *p1 p2 etc.*
`less` *p*	skriv ut innehållet i filen *p*, en skärm i taget
`wget` *url*	ladda ner *url*, t.ex. `wget http://cs.lth.se/pgk/ws -o ws.zip`
`unzip` *p*	packa upp *p*, t.ex. `unzip ws.zip`

Tabell B.2: Några viktiga terminalkommando i Linux. Med *p*, *p1*, *p2*, etc. avses en absolut eller relativ sökväg (eng. *path*), se avsnitt B.2.

Appendix C

Editera, kompilera och exekvera

C.1 Vad är en editor?

En editor används för att redigera programkod. Det finns många olika editorer att välja på. Erfarna utvecklare lägger ofta mycket energi på att lära sig att använda favoriteditorns kortkommandon och specialfunktioner, eftersom detta påverkar stort hur snabbt kodredigeringen kan göras.

En bra editor har **syntaxfärgning** för språket du använder, så att olika delar av koden visas i olika färger. Då går det mycket lättare att läsa och hitta i koden.

Nedan listas några viktiga funktioner som man använder många gånger dagligen när man kodar:

- **Navigera**. Det finns flera olika sätt att flytta markören och bläddra genom koden. Alla editorer erbjuder sökmöjligheter, och de flesta editorer har även mer avancerade sökfunktioner där kodmönster kan identifieras och multipla sökträffar markeras över flera kodfiler.

- **Markera**. Att markera kod kan göras på många sätt: med piltangenter+Shift, med olika speciella menyalternativ, med mus + dubbelklick eller trippelklick, etc. I vissa editorer finns även möjlighet att ha multipla markörer så att flera rader kan editeras samtidigt.

- **Kopiera**. Genom Copy-Paste slipper du du skriva samma sak många gånger. Kortkommandona Ctrl+C för Copy och Ctrl+V för Paste sitter i fingrarna efter ett tag. Man ska dock vara medveten om att det lätt blir fel när man kopierar en stor del som sedan ska ändras lite; många Copy-Paste-buggar kommer av att man inte är tillräckligt noggrann och ofta är det bättre att skriva från grunden i stället för att kopiera så att du hinner tänka efter medan du skriver.

- **Klipp ut**. Genom Ctrl+X för Cut och Ctrl+V för Paste, kan du lätt flytta kod. Att skriva kod är en stegvis process där man gör många förändringar under resans gång för att förbättra och vidareutveckla koden. Att flytta på kod för att skapa en bättre struktur är mycket vanligt.

- **Formatering**. Med indragningar, radbrytningar och nästlade block i flera nivåer får koden struktur. Många editorer kan hjälpa till med detta och har speciella kortkommandon för att ändra indragningsnivå inåt eller utåt.

- **Parentesmatchning**. Olika former av parenteser, `( { [ ) } ]` , behöver matchas för att koden ska fungera; annars går kompilatorn ofta helt vilse och

konstiga felmeddelanden kan peka på helt fel plats i koden. En bra kodeditor kan hjälpa dig att markera vilka parentespar som hör ihop så att du undviker att spendera för mycket tid med att leta efter en parentes som saknas eller står i vägen.

C.1.1 Välj editor

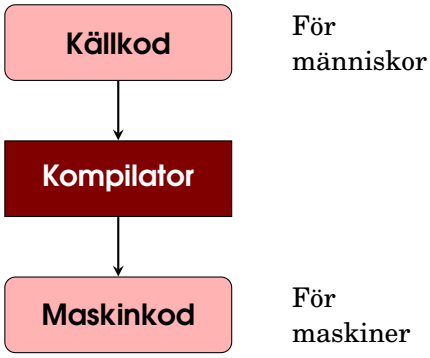
I tabell C.1 visas en lista med några populära editorer. Det är en stor fördel om din favoriteditor finns på flera plattformar så att du har nytta av dina förvärvade färdigheter när du behöver växla mellan olika operativsystem.

I denna kurs rekommenderas Visual Studio **`code`**, eftersom den är öppen, gratis och finns för Linux, Windows och Mac, och har bra stöd för Scala och Java. Men om du redan är van vid någon annan av editorerna i tabell C.1 så fungerar de också bra.

En integrerad utvecklingsmiljö (eng. *integrated development environment, IDE*), se appendix H, erbjuder många avancerade funktioner som hjälper dig att koda effektivt när du väl lärt dig handgreppen. VS `code` har numera flera IDE–funktioner, och gränsen mellan en renodlad editor och en IDE, så som IntelliJ och Eclipse, är inte längre lika tydlig som förr.

C.2 Vad är en kompilator?

En **kompilator** (eng. *compiler*) är ett program som läser programtext och översätter den till exekverbar maskinkod, så som visas i figur C.1. Programtexten som kompileras kallas källkod och utgörs av text som följer reglerna för ett programmeringsspråk, till exempel Scala eller Java.



Figur C.1: En kompilator översätter från källkod till maskinkod.

Vissa kompilatorer genererar kod som kan köras av en processor direkt, medan andra kompilatorer genererar ett mellanformat som tolkas under exekveringen. Det senare är fallet med Java och Scala, vilket möjliggör att programmet kan kompileras en gång för alla plattformar och sedan kan programmet köras på all de processorer till vilka det finns en s.k. virtuell maskin för Java (eng. *Java Virtual Machine, JVM*). Den kod som genereras av en kompilator för JVM kallas **bytekod**.

Om kompileringen inte lyckas skriver kompilatorn ut ett felmeddelande och ingen maskinkod genereras. Det är inte lätt att bygga en kompilator som ger bra felmeddelanden i alla lägen, men felmeddelandet ger oftast goda ledtrådar till felorsaken efter att man lärt sig tolka det programmeringsspråksspecifika vokabulär som kompilatorn använder.

Tabell C.1: Några populära editorer. I kursen rekommenderas VS Code.

Editor	*Beskrivning*

VS Code	Öppen, fri och gratis. Finns för Linux, Windows, & Mac. Är förinstallerad på LTH:s Linux-datorer och startas med kommandot `code`. Öppenkällkodsprojektet startades av Microsoft och har en aktiv gemenskap med många utvecklare och många användbara tillägg (eng. *extensions*). Sök efter tillägget `scalameta.metals` och installera så får du syntaxfärgning och många andra IDE-funktioner för Scala.
	`https://code.visualstudio.com/`
	`https://scalameta.org/metals/docs/editors/vscode/#installation`

Gedit	Öppen, fri och gratis. Lätt att lära men inte så avancerad. Är förinstallerad på LTH:s Linux-datorer och startas med kommandot `gedit`. `https://wiki.gnome.org/Apps/Gedit`
Nano	Öppen, fri och gratis. En simpel editor för enkla småjobb i terminalen. Är förinstallerad på de flesta Linux-system på planeten Jorden. Startas med kommandot `nano`. `https://www.nano-editor.org/`

Notepad++	Öppen, fri och gratis. Utvecklad speciellt för Windows men finns även för Linux.
	`https://notepad-plus-plus.org/`
	`https://snapcraft.io/notepad-plus-plus`

Vim	Öppen, fri och gratis. Hög inlärningströskel. Finns för Linux, Windows, & Mac. Är förinstallerad på LTH:s Linux-datorer och startas med kommandot `vim`. Med Scala Metals (se länk nedan) får du IDE-liknande funktioner. Du avslutar vim genom att trycka Escape och sedan skriva :q och trycka Enter.
	`http://www.vim.org/`
	`https://scalameta.org/metals/docs/editors/vim.html`

Emacs	Öppen, fri och gratis. Hög inlärningströskel. Finns för Linux, Windows, & Mac. Är förinstallerad på LTH:s Linux-datorer och startas med kommandot `emacs`. Med Scala Metals (se länk nedan) får du IDE-liknande funktioner.
	`http://www.gnu.org/software/emacs/`
	`https://scalameta.org/metals/docs/editors/emacs.html`

Sublime Text	Sluten källkod. Gratis att prova på, men programmet föreslår då och då att du köper en licens. Finns för Linux, Windows, & Mac. Med Scala Metals (se länk nedan) får du IDE-funktioner.
	`http://www.sublimetext.com/3`
	`https://scalameta.org/metals/docs/editors/sublime.html`

Även om programmet kompilerar utan felmeddelande och genererar exekverbar maskinkod, är det vanligt att programmet ändå inte fungerar som det är tänkt. Ibland är det mycket svårt att lista ut vad problemet beror på och man kan behöva göra omfattande undersökningar av vad som händer under körningen, genom att t.ex. skriva ut olika variablers värden eller på annat sätt ändra koden och se vad som händer. Denna process kallas felsökning eller avlusning (eng. *debugging*), och är en väsentlig del av all systemutveckling. Läs mer om debugging i Appendix D.

En uttömmande testning av ett större program, som kör programmets *alla* möjliga exekveringsvägar, är i praktiken omöjlig att genomföra inom rimlig tid, då antalet kombinationsmöjligheter växer mycket snabbt med storleken på programmet. Därför är kompilatorn ett mycket viktigt hjälpmedel. Med hjälp av den analys och de kontroller som görs av kompilatorn kan många buggar, som annars vore mycket svåra att hitta, undvikas och åtgärdas i kompileringsfasen, redan *innan* man exekverar programmet.

C.3 Java JDK

Scala, Java och flera andra språk använder Java-plattformen som exekveringsmiljö. Om man inte bara vill köra program som andra har utvecklat, utan även utveckla egna program som fungerar i denna miljö, behöver man installera Java Develpment Kit (JDK). Detta utvecklingspaket innehåller flera delar, bland annat:

- Kompilatorn `javac` kompilerar Java-program till bytekod som lagras i klassfiler med filnamnsändelsen `.class`.
- Exekveringsmiljön Java Runtime Enviroment (JRE) med kommandot `java` som drar igång den virtuella javamaskinen (Java Virtual Machine) som kan ladda och exekvera bytekod lagrade i klassfiler.
- Programmet `jar` som packar ihop många sammanhörande klassfiler till en enda jar-fil som lätt kan distribueras via nätet och sedan köras med `java`-kommandot på alla maskiner med JRE.
- Programmet `javap` som läser klassfiler och skriver ut vad de innehåller i ett format som kan läsas av människor (ett sådant program kallas disassembler).
- I JDK ingår också en mycket stor mängd färdiga programbibliotek med stöd för nätverkskommunikation, filhantering, grafik, kryptering och en massa annat som behövs när man bygger moderna system.

Du kan läsa mer om Java och dess historik här:
https://en.wikipedia.org/wiki/Java_(programming_language)

C.3.1 Kontrollera om du har JDK installerat

Öppna ett terminalfönster (se appendix B) och skriv (observera det avslutande c:et i `javac`):

```
javac -version
```

Då ska något som liknar följande skrivas ut, där `x` och `y` är siffror:
`javac 21.x.y`
Om utskriften säger att `javac` saknas, installera JDK enl. nedan.

Vi använder alltså JDK 21 i kursen. Det går också bra att använda de äldre versionerna JDK 8 och JDK 11, men JDK 9 eller 10 fungerar inte med alla verktyg vi använder och senare versioner än 21 kan också ge problem. Läs mer under ”Verktyg” på kurshemsidan.

C.3.2 Installera JDK

Det finns flera JDK-distributioner att välja mellan, varav OpenJDK och Oracle JDK är två exempel. Vi använder OpenJDK i kursen, som kan installeras via `https://adoptium.net/temurin/releases/?version=21`.

Om du installerar alla Scala-verktyg med hjälp av Coursier enligt instruktioner på kurshemsidan under ”Verktyg”, `http://cs.lth.se/pgk/verktyg` så kommer JDK att installeras automatiskt (om du inte redan har JDK).

C.4 Scala

Scala använder Java Virtual Machine (JVM) som exekveringsmiljö, men går även att köra i browsern med hjälp av ScalaJS-kompilatorn som kompilerar från Scala till JavaScript. I denna kurs använder vi i Scala på JVM. I en Scala-installation ingår bl.a. kompilatorn `scalac` och även ett interaktivt kommandoskal kallat Scala REPL (se nedan C.4.2) där du kan testa din Scala-kod rad för rad och se vad som händer direkt.

Den officiella hemsidan för Scala finns här: `http://www.scala-lang.org/`

Du hittar mer om Scalas historik och annan bakgrundsinformation här: en.wikipedia.org/wiki/Scala_(programming_language)

C.4.1 Installera Scala

Scala finns förinstallerat på LTH:s datorer. På kurshemsidan under ”Verktyg” finns detaljerade instruktioner om hur du installerar Scala på din egen dator: `http://cs.lth.se/pgk/verktyg`

C.4.2 Scala Read-Evaluate-Print-Loop (REPL)

För många språk, t.ex. Scala och Python, finns det ett interaktivt program ämnat för terminalen som gör det möjligt att exekvera enstaka programrader och direkt se effekten. Ett sådant program kallas *Read-Evaluate-Print-Loop* (REPL), eftersom det läser och tolkar en rad i taget. Resultatet av evalueringen av din kod skrivs ut i terminalen och därefter är kommandoskalet redo för nästa kodrad.

Kursens övningar bygger till stor del på att du använder Scala REPL för att undersöka principer och begrepp som ingår i kursen genom dina egna kodexperiment. Även när du på labbarna utvecklar större program med en editor och en IDE, är det bra att ha Scala REPL till hands. Då kan du klistra in delar av programmet du håller på att utveckla i Scala REPL och stegvis utveckla delprogram, som till slut fungerar så som du vill.

I Scala REPL får du se typinformation för variabler och metoder, vilket är till stor hjälp när man försöker lista ut vad en kodrad innebär. Genom att öva upp din förmåga att dra nytta av Scala REPL, kommer din produktivitet öka.

Du startar Scala REPL med kommandot `scala` och skriver Scala-kod efter prompten `scala>` och kompilering+exekvering sker när du trycker Enter.

```
> scala
```

```
Welcome to Scala 3.1.2 (17.0.2, Java OpenJDK 64-Bit Server VM).
Type in expressions for evaluation. Or try :help.

scala> 41 + 1
val res0: Int = 42
```

Varje evaluerat värde sparas i en ny variabel, här `res0`.

Om du skriver en ofullständig rad fortsätter editeringen på nästa rad. Du kan navigera mellan raderna med pil-upp- och pil-ner-tangenterna. När du avslutar med en rad som gör din kod fullständig så kompileras och exekveras alla raderna. Du kan avbryta flerradsediteringen i förtid genom skriva ett semikolon `;` och sen trycka Enter. Vill du fortsätta editeringen med en ny rad och förhindra för tidig evaluering så tryck Esc+Enter. Escape-tangenten finns överst till vänster på tangentbordet. Se exempel nedan:

```
scala> def fleraRader = 42  // Esc+Enter ger ny rad
     |   + "ny rad".length  // fortsättningsrad, avsluta med Enter
```

Beroende på vilket operativsystem du kör så kan även andra tangentkombinationer fungera för att starta ny rad i REPL; prova t.ex. Linux: Left Alt+Enter, Windows: Left Alt + Shift + Enter, VS Code Terminal i Windows Left Alt + Enter, VS Code Terminal i MacOS: Option + Enter.

Många av de kortkommandon som fungerar i terminalens kommandoskal, fungerar också i Scala REPL. Gå gärna igenom listan i tabell B.1 på sidan 206, och testa vad som händer i Scala REPL. Om du tränar upp din fingerfärdighet med dessa kortkommandon, går ditt arbete i Scala REPL väsentligt snabbare.

Med kommandot `:help` får du se en lista med specialkommandon för Scala REPL:

```
The REPL has several commands available:

:help                    print this summary
:load <path>             interpret lines in a file
:quit                    exit the interpreter
:type <expression>       evaluate the type of the given expression
:doc <expression>        print the documentation for the given expression
:imports                 show import history
:reset [options]         reset the repl to its initial state, forgetting all session en
:settings <options>      update compiler options, if possible
```

Du kan också starta Scala REPL med hjälpa av kommandot `scala-cli repl .` med ett blanktecken och en punkt på slutet. Punkten gör att alla `.scala`-filer som finns i aktuell katalog kompileras av Scala CLI och görs tillgänglig för användning i REPL.

C.4.3 Kompilera och kör med Scala Command Line Interface

Det finns sedan 2022 ett nytt smidigt kommandoradsgränssnitt (eng. *command line interface*) för att kompilera, exekvera och paketera Scala-program som kallas *Scala CLI*. Om du installerar Scala-verktygen enligt instruktioner på kurshemsidan under ”Verktyg”, `http://cs.lth.se/pgk/verktyg` så medföljer Scala CLI.

Här finns några användbara kommandon:

- Första gången du kör en nyinstallerad Scala CLI-installation så kör detta kommando så att du får tillgång till smidiga kompletteringar med TAB-tangenten: `scala-cli install completions`

- Med hjälp av detta kommando kan du förbereda VS Code för samverkan med Scala CLI (notera blanktecken och avslutande punkt):
  `scala-cli setup-ide .`
  Kör ovan kommando innan du startar VS Code första gången med `code .` i aktuell katalog, eller avsluta VS Code och kör ovan kommando och starta VS Code igen med `code .` i aktuell katalog.

- Scala CLI kan köra igång REPL i aktuell katalog med dina Scala- och Java-program automatiskt kompilerade och tillgängliggjorda i REPL med hjälp av nedan kommando. Med optionen `-S` anger du vilken version av Scala du vill köra:
  `scala-cli repl . -S 3`

- I stället för att ange Scala-version med optionen `-S` på kommandoraden kan du inuti ditt program, på första raden, skriva denna ”magiska” kommentar:
  `//> using scala "3.1.2"`
  Då kommer Scala CLI att automatiskt välja (och vid behov ladda ned) önskad version av Scala-kompilatorn (notera > efter `//`):

- Kompilera alla Scala- och Java-program i aktuell katalog och se eventuella felmeddelanden. Med hjälp av `--watch` (kan förkortas till `-w`) så kompileras alla filer automatiskt om så fort ändringar sparas i VS Code (kortkommando Ctrl+S):
  `scala-cli compile . --watch`

- Kör Scala- och Java-program i aktuell katalog med start av den topp-nivå-`def` som är märkt `@main` (om det finns flera får du en frågan om vilken `@main def` du vill köra).:
  `scala-cli run .`

- Skapa en exekverbar fil:
  `scala-cli package .`

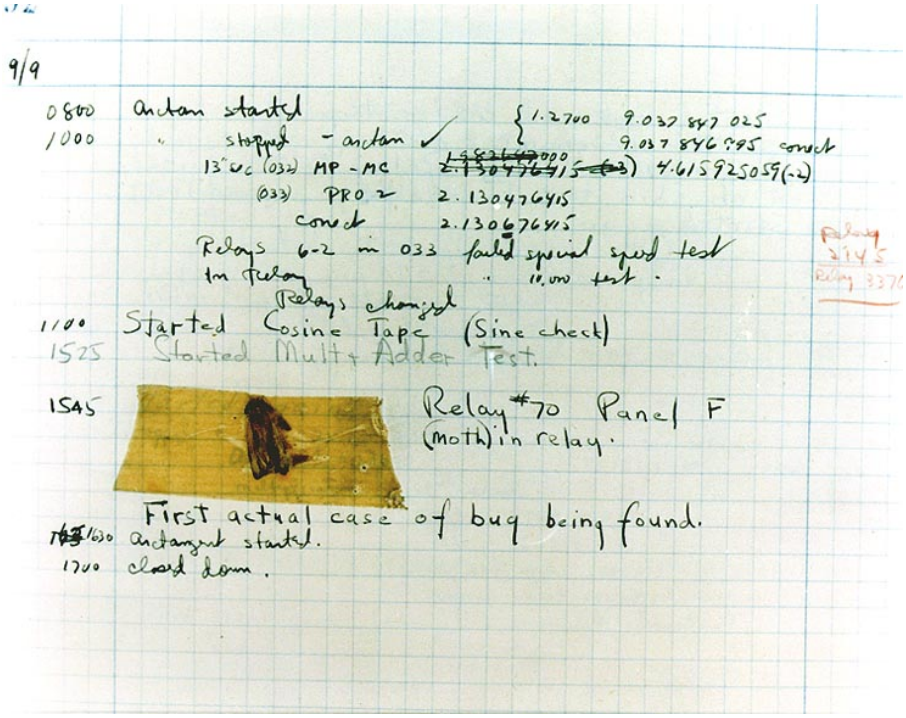
- Skapa en kopia av ditt projekt med katalogstruktur och filer anpassade för byggverktyget `sbt` (se Appendix F):
  `scala-cli export . --sbt --output ../nameofnewprojdir`
  Ändra katalognamnet `nameofnewprojdir` till valfritt nytt namn på en katalog som inte existerar. Notera de dubbla punkterna som gör att nya katalogen hamnar på samma nivå som ditt nuvarande projekt, och *inte* i din aktuella katalog (för att undvika att dubbletter av dina scala-filer ger kompileringsfel).

- Om du skriver `scala-cli help` så får du se vad du mer kan göra.

Läs mer om Scala CLI i Appendix F.2 och här:
https://scala-cli.virtuslab.org/

Appendix D

Fixa buggar



Figur D.1: Den första dokumenterade buggen hittades 9 september 1947 i en Mark II Aiken Relay Calculator av Grace Hopper.[1]

D.1 Vad är en bugg?

En bugg, även kallad lus (eng. *bug*), är en felaktighet som kan göra så att ett program inte beter sig som det är tänkt, och kan innebära oönskad utdata, att programmet kraschar, eller till och med ond bråd död.[2]

Ursprunget till ordets användning i programmeringssammanhang är något oklar, men kan härledas till engelskans *bug* som betyder insekt eller småkryp. Man brukar

[1] commons.wikimedia.org/w/index.php?curid=165211 Courtesy of the Naval Surface Warfare Center, Dahlgren, VA., 1988. - U.S. Naval Historical Center Online Library Photograph NH 96566-KN, Public Domain.

[2] www.theguardian.com/technology/2016/jul/01/tesla-driver-killed-autopilot-self-driving-car-harry-potter

berätta att vid en felsökning av ett program som körde i en tidig dator byggd med elektromekaniska reläer, uppdagades en död nattfjäril ihjälklämd mellan drivankaret och spolen i ett relä, som orsakade att programmet inte kunde exekveras korrekt.

D.1.1 Olika sorters fel

När man ska lära sig mer om fel i programvarubaserade system, och hur de kan åtgärdas, är det viktigt att noga skilja på **misstag** (eng. *mistake*), **felorsak** (eng. *fault*) och **felyttring** (eng. *failure*). Med ”misstag” menar vi här ett fel som begås av människor (utvecklare, systemadministratörer, operatörer, användare, etc.) medan de skapar och använder ett programvarusystem.

Det kan bli fel i olika delar av processen:

- **Kravfel** uppstår medan man tänker ut vad systemet ska göra och då misstar sig angående användarnas behov och önskemål.
- **Designfel** uppkommer när man utformar systemets struktur på ett dåligt sätt.
- **Implementeringsfel** begås när man programmerar och skriver felaktiga kodrader.
- **Testfel** förekommer vid provkörning av systemet då testkoden är felaktig och därför ger falskt alarm om ”fel”, trots att beteendet egentligen är korrekt.
- **Operatörsfel** sker när systemet lämnas över till de, som ska installera och köra systemet i skarp produktion, och där systemdriften (eng. *operations, ”ops”*) sköts på ett sätt som får problematiska konsekvenser.
- **Användarfel** händer då användarna ger felaktig indata, eventuellt i strid med riktlinjerna för hur systemet ska användas, som systemet inte klarar att hantera korrekt, varpå mer eller mindre allvarliga felbeteenden hos systemet följer.

I olika delar av utvecklingsprocessen kan alltså misstag begås som, antingen omedelbart, eller någon gång i framtiden, kan orsaka fel. Men det är inte säkert att ett fel någonsin kommer att märkas. Kanske kommer de felaktiga kodraderna, som *skulle* kunna orsaka ett fel, aldrig att exekveras. Eller så kommer ingen användare att någonsin vilja använda systemet så som stipuleras av (onödiga) krav. Det är alltså först när fel *yttrar* sig vid exekvering som misstag märks.

Fel kan också kategoriseras utifrån *hur* de upptäcks i utvecklingsprocessen. Man brukar skilja på fel upptäckta vid granskning, kompileringsfel och exekveringsfel, som diskuteras nedan:

- Fel upptäckta vid **granskning**. Ett effektivt sätt att upptäcka fel är att människor noga läser igenom sin egen, och andras kod och försöker leta efter möjliga problem och brister. Man blir ofta ”hemmablind” när det gäller ens egen kod. Därför kan någon annans, oberoende granskning med ”nya, friska” ögon vara mycket fruktbar. I samband med kodgranskning kan man med fördel försöka bedöma huruvida koden är lätt att läsa, lätt att ändra i eller om koden har andra viktiga kvaliteter som har betydelse för den framtida utvecklingen av koden. Ofta hittar man vid granskning även enkla programmeringsmisstag, så som felaktiga villkor och loop-räknare som inte räknas upp på rätt sätt etc.
- **Kompileringsfel** uppkommer under kompilering och upptäcks tack vare kontroller som sker av kompilatorn.

Vid kompileringsfel får man också ofta av kompilatorn reda på *var* i koden det är fel och *varför* det är fel, så att sökandet efter felorsaken och åtgärdandet av misstaget underlättas. Men ibland är felmeddelandet från kompilatorn missvisande och pekar på helt fel ställe i koden, så det gäller att inte alltid lita blint på det kompilatorn skriver. Dessutom är felmeddelanden från kompilatorn ofta uttryckta i termer av språkets syntaktiska och semantiska regler och det tar tid att lära sig tolka kompilatorers felmeddelanden. Att skapa kompilatorer som ger bra felmeddelande är ett svårt problem som studeras inom den datavetenskapliga disciplinen *kompilatorteknik*, vilken du kan lära mer om i kurser på avancerad nivå.

Olika programmeringsspråk erbjuder olika stora möjligheter att göra kontroller vid kompileringstid. En kompilator för ett språk med ett avancerat typsystem, som till exempel Scala, ger förhållandevis stora möjligheter att identifiera fel redan under kompileringen, medan man med ett språk med ett svagare typsystem, till exempel Javascript, får förlita sig på prestandahämmande kontroller som kompilatorn genererar i maskinkoden eller som du själv väljer att lägga in i källkoden för säkerhets skull.

- **Exekveringsfel**, även kallat körtidsfel (eng. *runtime error*), sker medan programmet körs. Det kan krävas viss, specifik indata under specifika exekveringsomständigheter (en viss processor, en viss minnesstorlek, en viss nätverkskapacitet etc.) för att ett exekveringsfel ska yttra sig. När ett exekveringsfel väl yttrar sig, kan olika saker hända:

  - **Exekveringen ger oönskat resultat.** Det är inte säkert att ett exekveringsfel avbryter exekveringen; det är vanligt att felet ”bara” resulterar i inkorrekt utdata eller på annat sätt ger dålig kvalitet. För att upptäcka detta innan systemet sätts i drift, är det allmän praxis att man skriver noga uttänkta **testfall** och analyserar **testresultat** från exekveringen av testfallen i detalj genom att undersöka utdata i jämförelse med önskat resultat eller med vad som anses vara en tillräckligt hög kvalitetsnivå.
  - **Exekveringen hänger sig** (eng. *hang*). Ibland yttrar sig fel genom att inget alls ser ut att hända under exekveringen, vilket kan beror på t.ex.:
    * en **oändlig loop**, som aldrig blir färdig,
    * att det går **väldigt långsamt** eftersom bearbetningen av indata tar orimligt lång tid,
    * att programmet **väntar på indata** som aldrig kommer,
    * att olika jämlöpande delar av programmet väntar på varandra så att ett **dödläge** (eng. *deadlock*) uppstår.

    När exekveringen hänger sig och man inte orkar vänta längre på att något ska hända, är det bara att brutalt avbryta exekveringen genom något lämpligt kommando som erbjuds i din körmiljö.[3] I värsta fall får man stänga av strömmen.
  - **Exekveringen kraschar** (eng. *crash*). Ibland blir det ett plötsligt tvärstopp och exekveringen avbryts med ett körtidsfelmeddelande. Detta kan bero på t.ex.:
    * att **minnet är slut**, antingen är det parameterminnet för funktionsanrop (eng. *stack memory*) som tagit slut eller så är minnet för allokering

[3] `kill -9 <pid>`, Ctrl+C, Ctrl+Shift+C, Ctrl+Z eller något annat beroende på körmiljö.

av objekt som skapas under programmets gång (eng. *heap memory*) fullt,

* misstaget att försöka referera en **null-referens** som inte refererar till något objekt, utan har värdet `null`, vilket resulterar i *null pointer exceeption*,
* att ett s.k. **undantag** har ”kastats” (eng. *throw exception*) genom att den som skrivit programmet medvetet kodat så att ett oönskat feltillstånd *ska* orsaka en krasch, om inte undantaget ”fångas” (eng. *catch*) och hanteras av omgivande kod.

När systemet kraschar får man en lista med den aktuella kedjan av funktionsanrop i en **stackspårning** (eng. *stack trace*). Man kan också begära en utskrift av hela innehållet i minnet vid kraschen (eng. *memory dump*), men en sådan kan vara svår att tolka.

När systemet ger oönskade resultat, hänger sig eller kraschar, får man försöka återskapa exekveringsfelet i en omkörning och, med hjälp av instrumentering eller en debugger, försöka lista ut vad som händer precis *innan* exekveringsfelet uppstår, se avsnitt D.5.

I kursen *Programvarutestning* (eng. *Software Testing*) lär du dig mer om systematiska metoder för att testa system så att fel kan förebyggas, identifieras och åtgärdas.

Bugg eller feature?

När ett (eventuellt) fel upptäcks, kan det vara på sin plats att först ställa sig några grundläggande frågor:

- Är detta verkligen ett ”fel” eller är det egentligen ett avsett beteende? Det är inte alltid självklart om det är en bugg eller en medvetet skapad systemegenskap/funktion (eng. *feature*).
- Är det kanske testfallet som har felaktig testkod, medan koden som testas egentligen fungerar alldeles utmärkt? Sådan problem kan vara speciellt svåra att lösa, då man ofta letar på fel ställe efter orsaken.
- Om buggen rör någon kvalitetsegenskap hos systemet kan man fråga sig: Var går egentligen gränsen för ”fel”? Är detta bra nog givet vad det kostar att förbättra kvaliteten? Kvalitetskrav berör egenskaper hos ett program som kan uttryckas på en glidande skala, där något kan vara mer eller mindre *bra* eller *dåligt* ur olika synvinklar. Sådana krav leder ofta till viktiga men svåra avvägningsbeslut under design och implementation. Dessutom kan testresultat bli svårbedömda och det kan finnas olika åsikter om huruvida ett eventuellt fel är en bugg eller inte.

  Här är några exempel på kvalitetskrav:

  – **Prestandakrav** (eng. *performance requirements*) avser hur snabbt och effektivt programmet ska arbeta under olika omständigheter.
  – **Kapacitetskrav** (eng. *capacity requirements*) avser hur mycket data systemet ska klara av under olika omständigheter.
  – **Användbarhetskrav**[4] (eng. *usability requirements*) avser krav på hur lättanvänt systemet ska vara för en given användarkategori.

[4] sv.wikipedia.org/wiki/Användbarhet

I kursen *Kravhantering* (eng. *Sofware Requirements Engineering*) lär du dig mer om att identifiera, specificera och följa upp kvalitetskrav.

Felärendehanteringsverktyg

Det är allmän praxis i industriell systemutveckling att använda sig av ett felärendehanteringsverktyg (eng. *issue tracker*) så att samarbetande utvecklare får stöd i att hålla reda på alla uppkomna fel och problem (eng. *issue*). Många av de populära kodlagringsplatserna som finns på nätet, så som GitLab, GitHub och BitBucket (se avsnitt G.3), erbjuder felärendehanteringsfunktioner. Dessa kan till exempel vara:

- hantering och sammanställning av alla olika ärendetillstånd, så att man kan se vilka issues som är i tillstånden *Open* eller *Closed*,
- tillordning av ärende till specifika personer som ska åtgärda problemet,
- gradering av ärende i olika allvarlighetsgrader,
- meddelandegenerering till inblandade personer när ett ärende kommenteras eller ändrar tillstånd.

D.2 Att förebygga fel

Även om det nästan är oundvikligt att låta buggar slinka in i koden allteftersom den blir mer och mer komplex, är det ändå viktigt att lägga stor möda vid att försöka undvika att så sker. Det är ofta mycket bättre investerad tid att jobba med buggförebyggande åtgärder medan du skapar koden, än att jaga buggar som skulle kunna ha undvikts med allmän noggrannhet och stramare disciplin i kodningen. Nedan sammanfattas några åtgärder som kan hjälpa till att minska mängden fel.

- **Skapa begriplig kod**. Grunden för att undvika buggar är anstränga sig att skriva begriplig kod som är lätt att läsa. Detta är en ständig kamp; kodens komplexitet växer för varje tillägg och med jämna mellanrum behövs omstruktureringar (eng. *refactoring*) för att bibehålla en god struktur som underlättar begripligheten och gör utvidgningar lättare.
- **Tänk ut bra namn**. En viktig pusselbit för att skapa begriplig kod är att tänka ut bra namn. Detta kan vara förvånansvärt svårt och kan kräva mycket diskussioner och tankemöda. Om du inser att ett namn är illa valt är det förmodligen värt jobbet att omstrukturera koden och införa ett bättre namn, speciellt om andra ännu inte vant sig alltför mycket vid begreppet.
- **Kontrollera parametrar och variabler**. Ofta känner man till vilka villkor som måste gälla för olika variablers värden. Till exempel vet man ofta att en viss funktionsparameter av heltalstyp inte får vara negativ. Då kan man säkerställa detta genom att lägga in kontroller av att villkoret är uppfyllt. Vid villkor som gäller parametrar, brukar man i Scala anropa `require`, till exempel: `require(x >= 0, "x must be positive")`. Det finns också en metod `assert` som fungerar på samma vis[5]; medan `require` används för att kontrollerar parametrar, brukar `assert` användas för att kontrollera generella villkor som ska gälla, till exempel `assert(x + y > n, "overflow")`. Fördelen med att lägga

[5] stackoverflow.com/questions/26140757/what-to-choose-between-require-and-assert-in-scala

in kontroller av villkor är att villkorsbrott upptäcks direkt och felsökningen blir lättare.

- **Kontrollera typer**. Med *typannoteringar* får du hjälp av kompilatorn att kontrollera dina hypoteser om vilka typer olika värden har. I Scala kan du nästan var du vill i ett uttryck lägga till ett kolon och en typ för att begära att kompilatorn kontrollerar typen. Till exempel kan du skriva `(xs + f(42)) : Set[Int]` för att säkerställa att uttrycket `xs + f(42)` verkligen ger en mängd med heltal. Även om du sällan i Scala behöver ange typer explicit, tack vare kompilatorns typinferens, bidrar det till läsbarheten och skapar säkrare kod om du på lämpliga ställen ändå anger de typer som du förväntar dig, speciellt vid i komplicerade uttryck eller långa kedjor av metodanrop, och när metoders returtyper inte är uppenbara. Dessutom kan kompilatorn ibland undvika att gå vilse i speciellt svåra typhärledningar, om du hjälper den på traven med explicita typannoteringar.

- **Hantera saknade värden**. Det är mycket vanligt att man måste hantera situationer där ett värde saknas, inte kan beräknas, eller inte finns tillgängligt av andra orsaker. Man kan hålla reda på att ett värde saknas genom att representera detta med speciella värden, t.ex. `-1` eller `null`. Men den strategin leder mycket lätt till buggar, då man lätt glömmer att på andra ställen i koden kontrollera dessa speciella värden. Med sådana speciella värden får man heller ingen hjälp av kompilatorn att upptäcka att man missat att ta hand om dem. Om man istället hanterar eventuellt saknade värden med `Option` (se kapitel 10), så får man hjälp vid kompileringstid och slipper exekveringsfel och besvärlig felsökning. Det blir dessutom väldigt tydligt för alla som läser din kod, inklusive du själv, att ett värde kan saknas.

- **Hantera undantag**. När undantag uppstår, t.ex. att en fil inte kan läsas eller det blir division med noll, avbryts exekveringen och programmets användare kan inte använda programmet längre, vilket i värsta fall kan få ödesdigra konsekvenser. Därför vill man hantera undantagssituationer på ett sådant sätt att programmet blir robust och inte kraschar. Detta kan man med fördel göra genom att kapsla in undantaget i ett värde av typen `Try`, se kapitel 10. I likhet med `Option` för saknade värden, blir det tydligt i koden att ett värde av typen `Try` kan innebära ett lyckat resultat (`Success`), eller så fallerar beräkningen (`Failure`) med en inkapslad, förhindrad krasch.

- **Granska kod**. Det är allmän praxis i industriell programvaruutveckling att göra kodgranskningar, vid vilka en grupp människor noga studerar någon annans kod och ger kommentarer och identifierar potentiella problem. Ofta har man en checklista att utgå ifrån medan man läser koden, som innehåller punkter man vill kontrollera speciellt, t.ex. begriplighet, namngivning, kontroller av parametrar, hantering av saknade värden och undantag, etc. Många organisationer har en överenskommen kodningsstandard med riktlinjer för kodens utseende och stil som alla ska följa om inte speciella skäl finns. Att sådana stilriktlinjer följs kan kontrolleras genom granskningar. Det finns också verktygsstöd för att göra sådana kontroller. Ett exempel på kodningsriktlinjer för Scala finns på den officiella dokumentationssajten[6].

- **Testa kod**. Det är allmän praxis i industriell programvaruutveckling att genomföra tester på flera olika nivåer. Man kombinerar ofta **enhetstest** (eng. *unit test*)

[6] https://docs.scala-lang.org/style/

av enskilda delar av koden, med **funktionstest** (eng. *feature test*) för att se så att indata i en tänkt användningssituation ger önskat resultat, och **systemtest** (eng. *system test*) för att se att alla delar fungerar tillsammans under realistiska omständigheter.

- **Lär av användarnas upplevelser**. När koden sätts i produktion finns möjlighet att lära sig genom återkoppling från användare. Hur systemet används och hur användarna upplever det att använda systemet är mycket viktig information när man ska besluta om hu koden bäst ska utveckla vidare. Från användarna kan man få reda på både okända buggar och få briljanta idéer till nya värdefulla funktioner. En mjukvaruutvecklande organisations innovationsförmåga beror i stor utsträckning på dess förmåga att kontinuerligt leverera kod som får allt fler funktioner som användarna gillar, utan att för många irriterande eller ödesdigra buggar.

D.3 Vad är debugging?

När en felyttring identifierats, t.ex. genom testning eller slutanvändare rapporterar om problem, vidtar sökandet efter den bakomliggande felorsaken, så att vi förstår *varför* det blev fel och sedan kan *åtgärda* misstaget. Denna process kallas **avlusning** (eng. *debugging*).

D.3.1 Hur hittas felorsaken?

Första steget i avlusningsprocessen är att hitta den bakomliggande felorsaken. Detta kan vara mycket svårt, speciellt om systemet är stort och komplicerat.

När du stirrar dig blind på koden utan att hitta felorsaken, kan det bero på att du har en felaktig hypotes om vad koden egentligen gör. Du är övertygad om att en viss sak händer, men *egentligen* är det *inte* det du *tror* händer som *verkligen* händer. Exempelvis kanske du antar att en räknare räknas upp i en loop, men i själva verket saknas uppräkningen. Om du oreflekterat accepterar ditt felaktiga antagande, är det stor risk att du letar på fel ställe i koden.

Följande åtgärder är ofta lämpliga när man jagar buggar:

- **Återskapa buggen med ett minimalt testfall**. När du upptäckt en felyttring är det viktigt att kunna återskapa felet, så att koden som körs precis *innan* buggen uppstår kan felsökas. Allra bäst är det om du kan skapa ett **minimalt testfall** där precis den minimala indata och de enskilda förutsättningar nedtecknas, som ska gälla för att buggen ska uppstå. Beskrivningen av det minimala testfallet är första pusselbiten i det detektivarbete som vidtar under felsökningen.

- **Formulera och verifiera hypoteser om buggen**. En grundläggande princip vid felsökning är att uttryckligen formulera hypoteser som du har om vad som sker i systemet medan buggen uppstår och sedan *verifiera* att de verkligen stämmer, genom olika undersökningar av det exekverande systemet. Du ska alltså tydligt beskriva hur du tror att koden fungerar och sedan med olika former av instrumentering, t.ex. genom utskrifter i terminalen av variablers värden, kontrollera att så verkligen är fallet. Detta kan göras med instrumentering enligt nedan.

- **Instrumentering med utskrifter, ”print-debugging”**.

  För att verifiera din hypotes om vad som leder fram till buggen, behöver du kontrollera vad som händer. Det kan du göra genom att på väl valda ställen ligga in `println`-utskrifter i koden där värden på intressanta variabler skrivs ut. Det kan behövas lite klurighet för att hitta precis rätt utskrifter; om man skriver ut allt som händer i alla loopar drunknar man i all information, men skriver man ut för lite förbiser man kanske den falsifierade hypotesen och får ingen hjälp att knäcka bugg-gåtan.

  Du kan även använda en avlusare (eng. *debugger*), som normalt ingår i en integrerad utvecklingsmiljö, för att instrumentera din kod. Se vidare i avsnitt D.5 om hur du använder avlusarna i Eclipse och IntelliJ IDEA.

D.4 Åtgärda fel

Ofta är det det svåraste att *hitta* buggen, medan själva buggrättningen visar sig trivial. Har du, till exempel, väl hittat den saknade uppräkningen av din loop-variabel är det uppenbart vad du ska göra.

Men ibland är det riktigt knepigt att åtgärda felet. Nedan sammanfattas några av de situationer som kan uppkomma, som gör att felrättningen blir extra svår.

- Kanske är själva algoritmen i grunden feltänkt och en helt ny algoritm behöver konstrueras. Att skapa nya algoritmer från grunden kan visa sig mycket svårt i en del fall. I fortsättningskurser får du lära dig mer om algoritmkonstruktionens ädla konst.

- Kanske algoritmen fungerar för olika normalfall, medan ovanliga undantagsfall inte hanteras korrekt. Att på ett bra sätt hantera alla upptänkliga fall kan visa sig väldigt knepigt. Tyvärr är det ofta undantagsfall i kombination med buggar som öppnar för säkerhetsluckor redo att utnyttjas av elaka hackare för att krascha systemet eller smitta ner det med virus.

- Kanske är problemet i sig väldigt svårt att lösa på ett korrekt sätt. Algoritmen kan vara riktigt knepig med många villkor, loopar och nästlade datastrukturer. Blir det fel i en sådan algoritm kan det ta lång tid att få ändringar att fungera och alla villkor, loopar och nästlade datastrukturer att passa ihop igen efter felrättningen.

- Medan man rättar en bug kan man råka att, av misstag, skapa nya buggar. Risken för detta är speciellt stor om koden är komplex. Ibland låter man till och med bli att åtgärda ett fel om systemet ändå fungerar hjälpligt i andra avseenden och risken är för stor att nya buggar skapas. Då behöver systemet strukturerats om så att det blir lättare att ändra i.

- Kanske växer exekveringstiden exponentiellt med datamängden. Det kan då i praktiken vara omöjligt att skriva ett program som i alla lägen blir färdigt inom rimlig tid. Då får man försöka tänka ut kluriga genvägar till suboptimala lösningar som ändå duger, vilket ibland kräver mycket avancerad programmeringsteknik.

Det finns ingen allenarådande snabbfix att ta till när man stöter på svåra fel. Att bli en produktiv systemutvecklare, som framgångsrikt reder ut allehanda buggar, handlar

i stor utsträckning om att kombinera en bred allmänbildning inom datavetenskap med ett livslångt lärande, där varje bugg du hittar och åtgärdar ger dig nya kunskaper och erfarenheter inför framtiden. Även om din bugg är irriterande, försök se den som en ny chans till ökad lärdom!

D.5 Använda en debugger

Med en professionell integrerad utvecklingsmiljö kommer ofta en avancerad debugger, som kan hjälpa dig att följa exekveringen och se vad som händer medan koden kör. Normalt ingår dessa funktioner i en debugger:

- **Sätta brytpunkter**. Med hjälp av debuggern kan du sätta så kallade *brytpunkter* på speciella ställen i koden. Detta görs ofta genom att du markerar en kodrad i marginalen varpå en brytpunktssymbol placeras där. När exekveringen når brytpunkten avbryts exekveringen och du kan stega dig vidare eller inspektera variablers värden vid brytpunkten.

- **Stegad exekvering**. När du nått en brytpunkt kan du med hjälp av debuggern stega dig fram genom koden rad för rad och se vad som händer. Om du kommer till ett funktionsanrop kan du välja att stega in i koden som implementerar funktionen eller bara köra funktionen i ett svep och stega till nästa rad som kommer efter funktionsanropet. Det kan kräva många omkörningar från en viss brytpunkt, innan man hittar vilka funktioner som inte verkar relevanta alls för buggen och bara kan stegas över, eller vilka funktioner som utgör gåtans lösning och som du vill stega in i och undersöka närmare. Stegar man djupt ner i funktionsanropskedjan, går man lätt vilse och får börja om. (Det går inte att stega bakåt...)

- **Inspektera variabler**. Medan du stegar dig fram i koden kan du inspektera variablers värden. I ett område på skärmen presenterar debuggern både enkla värden så som heltal och strängar, men även datastrukturer, så som vektorer och listor, kan inspekteras genom att debuggern låter dig bläddra bland arrayer och objektreferenser. Ett program kan ha väldigt många variabler och djupa strukturer av objekt som refererar till nya objekt. Det är ofta ett knepigt detektivarbete att försöka lista ut hur olika variabelvärden relaterar till orsaken bakom buggen som du letar efter. Du behöver ofta växla mellan att läsa koden, stega dig fram, sätta nya brytpunkter och inspektera variabler för att förstå vad som händer.

I Kojo Desktop (se appendix A) finns lättanvända debug-funktioner. Man kan till exempel följa stegen i exekveringen med hjälp av den brandgula play-knappen ”Kör och spåra programmet” (kortkommando: Alt+Enter). Då öppnas ett nytt fönster som visar exekveringsstegen. Man kan klicka på ett steg och få information om parametrar vid funktionsanrop etc.

Du kan läsa mer om hur man använder en avancerad debugger i en professionell integrerad utvecklingsmiljö i appendix H.

Appendix E

Dokumentation

Dokumentation hjälper andra att använda din kod, men underlättar även för dig själv när du vid ett senare tillfälle ska erinra dig hur den fungerar och hur du ska använda och bygga vidare på din kod. Modern systemutveckling baseras ofta på öppen källkod och färdiga api (eng. *application programming interface*), där kvaliteten på dokumentationen är avgörande för hur lätt det är att komma igång med att använda koden.

Nedan listas exempel på olika typer av dokumentation[1]:

- **Kravdokumentation** beskriver det övergripande målet med mjukvaran, samt funktionella krav och kvalitetskrav som uppfylls av systemet.
- **Designdokumentation** beskriver arkitekturen, hur koden är organiserad i moduler, och den interna systemstrukturen t.ex. i form av klasser, objekt och deras relation.
- **Slutanvändardokumentation** kan t.ex. vara manualer för användning av systemet och installationsanvisningar.
- **Teknisk dokumentation** kan t.ex. vara api-dokumentation som beskriver vilka funktioner som ingår i ett programbibliotek. Sådan dokumentation genereras ofta med hjälp av ett **dokumentationsverktyg** (se avsnitt E.1). Andra typer av teknisk dokumentation är instruktioner om hur man bygger koden med eventuellt tillhörande byggverktygskonfigurationsfiler; ofta beskrivs byggförfarandet steg för steg i en textfil med namnet `README`. (Läs mer om byggverktyg i appendix F.)

Det är en stor utmaning att hålla dokumentationen uppdaterad allteftersom koden utvecklas. Även om man får hjälp att generera en navigerbar sajt av ett dokumentationsverktyg, måste själva *innehållet* i de manuellt författade dokumentationskommentarerna vara i överensstämmelse med den aktuella versionen av koden. Uppdateras koden, måste man alltså vara noga med att uppdatera dokumentationskommentarerna, annars uppstår stor förvirring.

Detta problem är så pass allvarligt att man ska tänka sig noga för hur man kan formulera dokumentationskommentarerna på ett framtidssäkert sätt, och hur omfattande de ska vara i förhållande till den framtida arbetsinsatsen med att hålla dem uppdaterade. Desto mer omfattande kommentarer desto mer jobb att hålla dem uppdaterade.

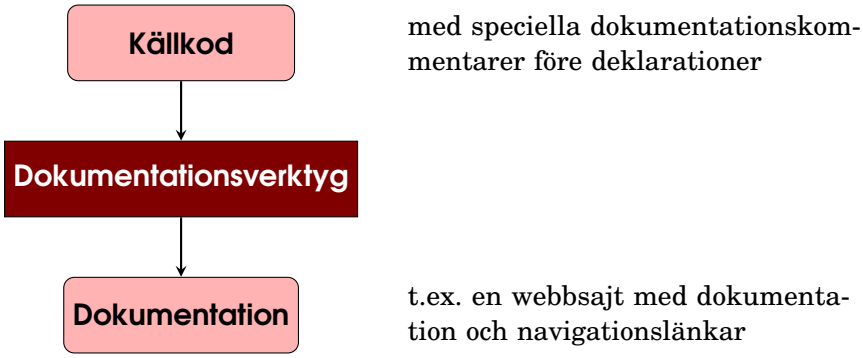
[1] en.wikipedia.org/wiki/Software_documentation

Det är i praktiken svårt att uppnå en optimal balans mellan bra och många kommentarer som *hjälper* användaren, och å andra sidan svårunderhållna och föråldrade kommentarer som *stjälper* användare.

E.1 Vad gör ett dokumentationsverktyg?

Ett dokumentationsverktyg genererar teknisk dokumentation av koden baserat på speciella **dokumentationskommentarer** som skrivs i koden omedelbart före deklarationer av det som ska dokumenteras. Dessa dokumentationskommentarer skrivs enligt en speciell syntax som dokumentationsverktyget kan tolka.

Utdata från ett dokumentationsverktyg utgörs typiskt av en webbsajt med ändamålsenlig formatering och navigationslänkar, se figur E.1.



Figur E.1: Ett dokumentationsverktyg läser koden och dokumentationskommentarer och genererar dokumentation, t.ex. i form av en webbsajt.

E.2 scaladoc

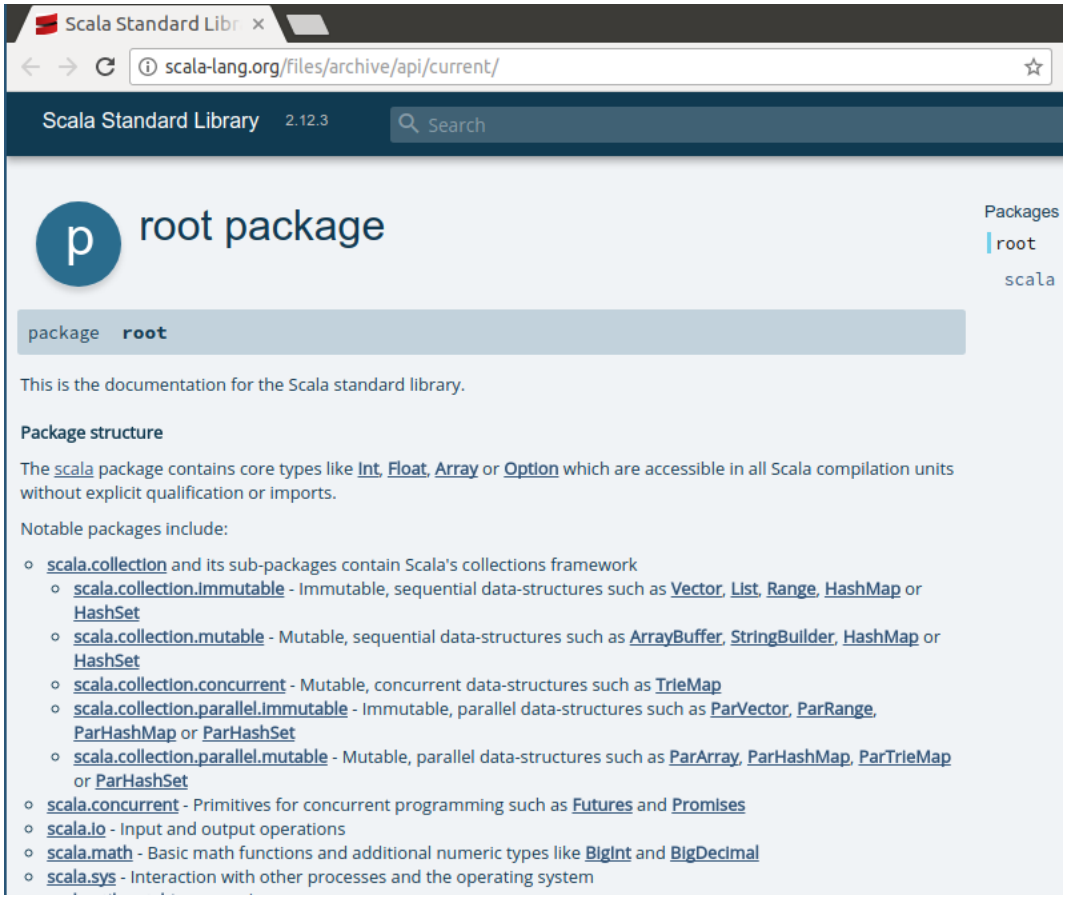
Med Scala-installationen följer dokumentationsverktyget `scaladoc`, som genererar en webbsajt med ändamålsenlig layout och specialfunktioner för att söka, filtrera och navigera i dokumentationen.

Dokumentationen av stora bibliotek kan bli omfattande och det krävs träning i att använda dokumentationssajter för att få maximal nytta av dem. I efterföljande avsnitt beskrivs först hur du använder dokumentation som är genererad med `scaladoc`. Därefter visas hur du själv kan generera dokumentation för din egen kod.

E.2.1 Använda dokumentation från scaladoc

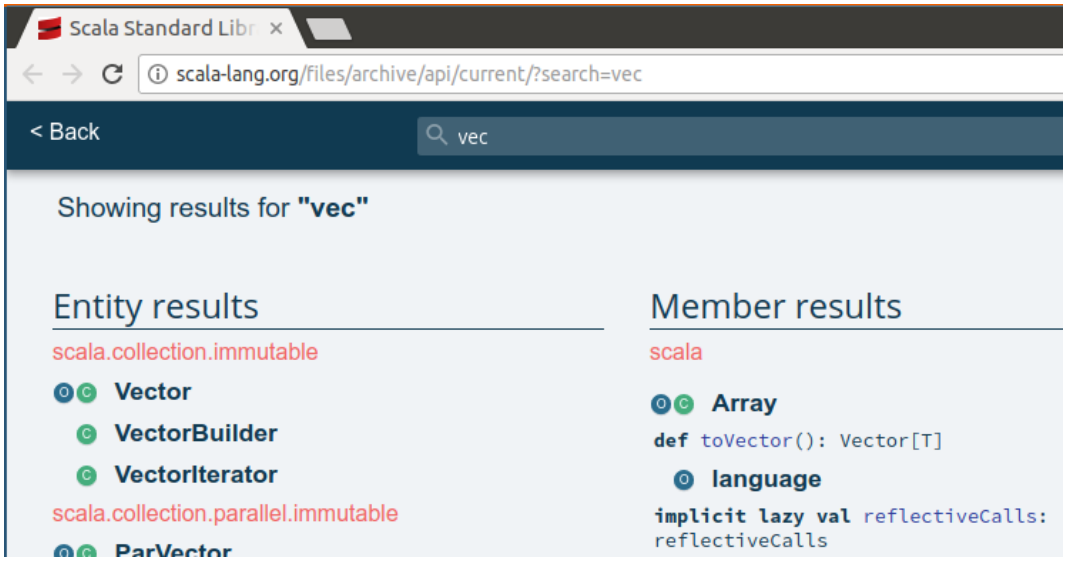
Dokumentationen av Scalas standardbiliotek är genererad med `scaladoc` och att navigera i denna ger bra träning i hur man använder avancerad api-dokumentation. Du hittar dokumentationen för Scalas standardbibliotek här:
`http://scala-lang.org/api/current`

När du surfar dit möts du av dokumentationen för *root package*, som ger en översikt av olika paket i standardbiblioteket. I sökrutan uppe till vänster kan du skriva början på namnet på klasser, traits, eller objekt som du letar efter, så som visas i figure E.2.



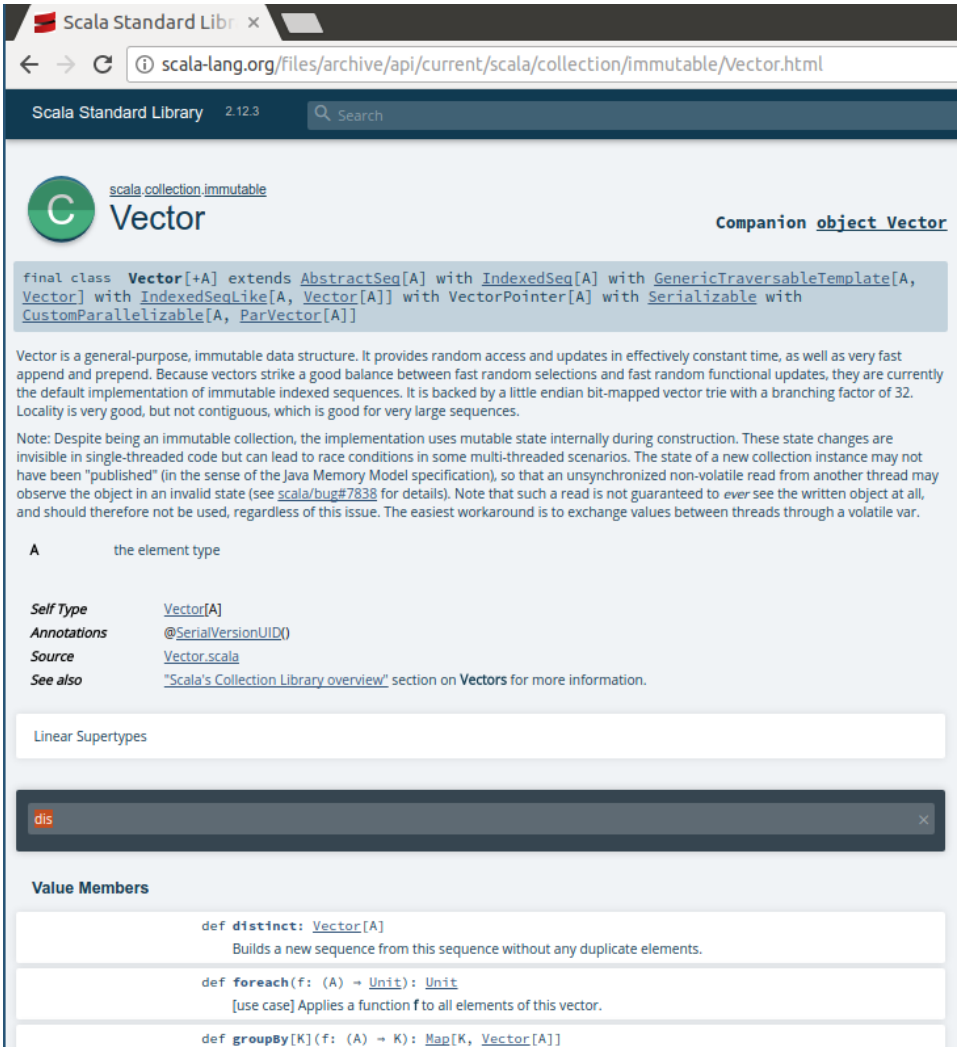
Figur E.2: Förstasidan med dokumentationen av standardbiblioteket i Scala.

Genom att skriva text i sökrutan får du en filtrerad lista på allt som har ett namn som börjar på det söker efter. I figur E.3 visas en sökning efter `vec`.



Figur E.3: Sökning efter innehåll som börjar på `vec`.

Om du klickar på `Vector` får du se dokumentationen i figur E.4



Figur E.4: En del av dokumentationen av klassen `Vector`

Genom att skriva text i den nedre, mörkgrå sökrutan kan du filtrera listan med klassmedlemmar. Om klickar på länken **`object`** `Vector`, eller på den runda, gröna ikonen med ett stort C, får du se kompanjonsobjektets medlemmer. Du kan få fram en filtreringsruta med fler möjligheter genom att klicka på expanderingspilen i den mörkgrå sökrutan så som visas i figur E.5.

Figur E.5: Expanderad sökruta med extra filtreringsmöjligheter.

E.2.2 Skriva dokumentationskommentarer

Verktyget `scaladoc` läser kommentarer som börjar med `/**` och slutar med `*/` och associeras till efterföljande deklaration. Notera de dubbla asteriskerna. Alla rader som följer efter `/**` ska, enligt konventionen för Scalas dokumentationskommentarer, börja med en asterisk `*` med indrag med flera blanksteg så att den hamnar under *andra* asterisken i öppningskommentaren, som nedan:

```
/** Först kommer en sammanfattning på en enda rad.
  *
  * Sedan kommer eventuellt en mer detaljerad beskrivning,
  * som kan vara flera rader lång.
  */
```

Dokumentationskommentaren slutar med `*/` rakt under asterisk-kolumnen.

- Med `@constructor` i början på en rad ges en speciell kommentar om konstruktorer.
- Med `@param` i början på en rad ges en speciell kommentar angående parametrar.
- Med `@return` i början av en rad ges en speciell kommentar angående vad som returneras vid metodanrop.
- Länkar skrivs inom dubbla hakparenteser, enligt exempel nedan.

```
/** A person who uses our application.
 *
 *  @constructor create a new person with a name and age.
 *  @param name the person's name
 *  @param age the person's age in years
 */
class Person(name: String, age: Int)

/** Factory for [[mypackage.Person]] instances. */
object Person:
  /** Creates a person with a given name and age.
   *
   *  @param name their name
   *  @param age the age of the person to create
   */
  def apply(name: String, age: Int) = ???

  /** Creates a person with a given name and birthdate
   *
   *  @param name their name
   *  @param birthDate the person's birthdate
   *  @return a new Person instance with the age determined by the
   *          birthdate and current date.
   */
  def apply(name: String, birthDate: java.util.Date) = ???
```

Läs mer här om hur du skriver dokumentationskommentarer:
https://docs.scala-lang.org/scala3/guides/scaladoc/

Läs mer om dokumentation här:
https://docs.scala-lang.org/scala3/scaladoc.html
https://scala-cli.virtuslab.org/docs/commands/doc

E.2.3 Generera dokumentation

Du genererar dokumentation enklast med hjälp av körverktyget `scala-cli`. I terminalen skriv

```
scala-cli doc . -o api
```

När `scala-cli` är färdig med att generera dokumentationen så meddelas vilken katalog som dokumentationen ligger i. För att länkarna inom dokumentationen ska fungera krävs antingen att du kör en lokal webbserver i katalogen eller att du använder ett program för att konvertera länkarna till lokala sådana.

Använda en lokal webbserver

Om du har Python 3 installerat på din dator har du en inkluderad webbserver. Du startar denna i terminalen med `python -m http.server` när du står i dokumentationens katalog. För att öppna dokumentationen besöker du sedan `http://localhost:8000` i din webbläsare.

Använda ett program för att konvertera länkar

Om du inte har Python installerat kan du köra ett Scala-program som byter ut alla länkar till lokala sådana, vilket gör att de går att öppna direkt. Detta kan laddas ner från
https://github.com/dixine55/Scaladoc-Local-Version-Patcher/blob/main/scaladocPatch.scala.
Placera programmet i dokumentationsmappen och kör det med `scala-cli run .` när du är i dokumentationens mapp. Du öppnar sedan dokumentationen genom filen `index.html` i din webbläsare.

Mer att läsa om att generera dokumentation finns här:
https://scala-cli.virtuslab.org/docs/commands/doc

Appendix F

Byggverktyg

F.1 Vad gör ett byggverktyg?

Ett **byggverktyg** (eng. *build tool*) används för att

- ladda ner,
- kompilera,
- testköra,
- paketera och
- distribuera

programvara. Ett stort utvecklingsprojekt kan innehålla många hundra kodfiler och under utvecklingens gång vill man kontinuerligt testköra systemet för att kontrollera att allt fortfarande fungerar; även den kod som inte ändrats, men som kanske ändå påverkas av ändringen. Ett byggverktyg används för att *automatisera* denna process.

Ett viktigt begrepp i byggsammanhang är **beroende** (eng. *dependency*). Om koden X behöver annan kod Y för att fungera, sägs kod X ha ett beroende till kod Y.

I konfigurationsfiler, som är skrivna i ett format som byggverktyget kan läsa, specificeras de beroenden som finns mellan olika koddelar. Byggverktyget analyserar dessa beroenden och, baserat på ändringstidsmarkeringar för kodfilerna, avgör byggverktyget vilken delmängd av kodfilerna som behöver **omkompileras** efter en ändring. Detta snabbar upp kompileringen avsevärt jämfört med en total omkompilering från grunden, som för ett stort projekt kan ta många minuter eller till och med timmar. Efter omkompilering av det som ändrats, kan byggverktyget instrueras att köra igenom testprogram och rapportera om testernas utfall, men även ladda upp körbara programpaket till t.ex. en webbserver.

En vanlig typ av beroende är färdiga programbibliotek som utnyttjas av systemet under utveckling, vilket i praktiken ofta innebär att en sökväg till den kompilerade koden för programbiblioteket behöver göras tillgänglig. I JVM-sammanhang innebär detta att sökvägen till alla nödvändiga jar-filer behöver finnas på sökvägslistan kallad **classpath**.

Många byggverktyg kan utföra så kallad **beroendeupplösning** (eng. *dependency resolution*), vilket innebär att nätverket av beroenden analyseras och rätt uppsättning programpaket görs tillgänglig under bygget. Detta kan även innebära att programpaket som är tillgängliga via nätet automatiskt laddas ned inför bygget, t.ex. via lagringsplatser för öppen källkod.

Även om man bara har ett litet kodprojekt med några få kodfiler, är det ändå smidigt att använda ett byggverktyg. Man kan nämligen göra så att byggverktyget är aktivt i bakgrunden och, så fort man sparar en ändring av koden, gör omkompilering och rapporterar eventuella kompileringsfel.

Det är klokt att kompilera om ofta, helst vid varje liten ändring, och rätta eventuella fel *innan* nya ändringar görs, eftersom det är mycket lättare att klura ut ett enskilt problem efter en mindre ändring, än att åtgärda en massa svåra följdfel, som beror på en sekvens av omfattande ändringar, där misstaget begicks någon gång långt tidigare.

En integrerad utvecklingsmiljö, så som VS Code eller IntelliJ IDEA, bygger om koden kontinuerligt och kan ofta konfigureras att kommunicera med flera olika byggverktyg. Exempelvis kan du med VS Code välja om du vill att Scala CLI eller `sbt` ska användas för att bygga ditt projekt.

Det finns många olika byggverktyg. Några allmänt kända byggverktyg listas nedan så att du ska känna igen vilket byggverktyg som används i öppen-källkods-projekt som du stöter på, t.ex. på GitHub.

- `Scala CLI`. Verktyget Scala CLI (Command Line Interface) är öppen källkod utvecklad av VirtusLab[1] för att kompilera och köra Scala- och Java-program och innehåller också grundläggande byggverktygsfunktioner, så som att köra testfall, paketera jar-filer och skapa dokumentation. Kommandot `scala-cli` övertog år 2023 rollen som det officiella `scala`-kommandot. Detta är det enklaste och rekommenderade sättet att bygga system med Scala-kod. Grundläggande användning av Scala CLI beskrivs i Appendix C.4.3, medan en mer utförlig beskrivning återfinns nedan i avsnitt F.2.

- `sbt`. Även kallad *Scala Build Tool*. Används för att bygga Java- och Scala-program i samexistens, men även för att automatisera en mängd andra saker. `sbt` är utvecklat i Scala och konfigurationsfilerna, som heter `build.sbt`, innehåller Scala-kod som styr byggprocessen. `sbt` är avancerat och klarar bygga system som består av många projekt (eng. *multi-project build*). `sbt` är det i särklass vanligaste byggverktyget för Scala och många öppen-källkodsprojekt använder `sbt`. Läs mer om `sbt` i avsnitt F.3 nedan.

  Efter kritik om att `sbt` är komplicerat så har flera alternativa byggverktyg för Scala utvecklats, däribland `bleep`[2] och `mill`[3].

- Apache Maven, `mvn` är också skriven i Java och är en efterföljare till `ant`. Maven används av många Java-utvecklare. Konfigurationsfilerna heter `pom.xml` och innehåller en s.k. projektobjektmodell specificerad i XML enligt speciella regler.

- `gradle` bygger vidare på idéerna från `ant` och `maven` och är skrivet i Java och Groovy. Konfigurationsfilerna skrivs i Groovy och heter `build.gradle`.

- Apache `ant`. Detta byggverktyg är utvecklat i Java som ett alternativ till `make` och används fortfarande i många Java-projekt, även om Maven och Gradle är vanligare numera. Konfigurationsfilerna heter `build.xml` och skrivs i det standardiserade språket XML enligt speciella regler.

[1] https://scala-cli.virtuslab.org/
[2] https://bleep.build/docs/
[3] https://mill-build.com/mill/Intro_to_Mill.html

- make. Detta anrika byggverktyg har varit med ända sedan 1970-talet och används fortfarande för att bygga många system under Linux, och är populärt vid utveckling med programspråken C och C++. En konfigurationsfil för make heter Makefile och har en egen, speciell syntax.

F.2 Scala Command Line Interface scala-cli

Utvecklingen av Scala CLI[4] påbörjades 2022 och planeras under 2024 bli det officiella bygg- och körverktyget. Det kommer då medfölja den officiella installationen av Scala via https://www.scala-lang.org. Scala CLI kan även installeras separat från https://scala-cli.virtuslab.org och köras med kommandot scala-cli, men någon gång under 2024 när Scala 3.5 släpps så blir kommandot scala liktydigt med scala-cli.[5]

Efter nyinstallation av Scala CLI kan du ange följande kommando för att, en gång för alla, få tillgång till kompletteringar av optioner med Tab-tangenten i terminalen:

```
scala-cli install-completions
```

Innan du börjar skriva källkod i en ny katalog i VS Code kan du göra VS Code redo för att använda Scala CLI som byggverktyg i aktuell katalog med följande kommando (vänta med att öppna VS Code till efter att du kört kommandot):

```
mkdir minNyaKatalog
cd minNyaKatalog
scala-cli setup-ide .
code .
```

I senare versionerna av VS Code med Metals och Scala 3.4 behövs ej setup-ide då Metals kör scala-cli som default.

F.2.1 Exempel på användning av Scala CLI

Nedan beskrivs de viktigaste Scala-CLI-kommandona för att stegvis bygga upp din kod med många små steg och experimenterande med dellösningar.

- Med kommandot scala-cli compile . -w i ett eget terminalfönster bredvid din editor startar du Scala CLI i så kallad *watch mode*. Då bevakas alla filändringar och omkompilering sker direkt när någon ny ändring sparats och du kan se eventuella kompileringsfel direkt. Åtgärda helst ett kompileringsfel innan du bygger vidare på din kod, då följdfel kan vara svåra att lösa speciellt om de är många. Dela upp stora kodändringar i små steg och försök att så fort som möjligt få den senaste ändringen att kompilera felfritt.

- Med kommandot scala-cli repl . i ett annat terminalfönster startar du REPL med dina klasser i aktuella katalogen automatiskt tillgängliga på classpath (därav punkten efter blanksteg efter scala repl) och du kan anropa dina metoder, efter ev. **import** när du gör experiment inför nästa steg.

[4] CLI är en förkortning för *command line interface*. Läs mer om Scala CLI här: https://scala-cli.virtuslab.org/docs/overview

[5] I skrivande stund har gamla scala-kommandot ännu inte uppgraderats, men det förväntas ske i början av hösten. När så väl sker kan scala-cli ersättas med scala om du uppdaterar till Scala 3.5.0 eller senare.

På så sätt kan du skapa och testa små funktioner och få dem att att fungera innan inför dem i ditt program och sätter samman dessa med redan skapade funktioner. Det är ofta lättare att felsöka och bygga upp ett större program om du har många små funktioner som samverkar, snarare än få väldigt stora funktioner. Och det är oftast lättare att testköra nya lösningsidéer i REPL innan du skapar ”färdig” kod i ditt program.

- Med `scala-cli run .` sker kompilering och körning av `main`-metoden i aktuella katalogen. Du kan ange en annan katalog genom att skicka med sökvägen som argument till kommandot. Du kan också ange optionen `-w` för *watch mode* vid körning. Då kommer ditt program att köras om vid varje ändring. Watch mode vid körning är användbart om programmet ger resultat utan att vänta på input, men inte så smidigt om varje körning kräver att användaren skriver indata eller om fönster måste stängas.

- Om det finns flera `main`-metoder i aktuella katalogen, går det att specificera vilken av dessa som ska exekveras med optionen `--main-class`, exempelvis `scala-cli run .  --main-class hello`

- Argument till ditt huvudprogam anges efter dubbla minustecken `--` så här: `scala-cli run . -- arg1 arg2 arg3`

F.2.2 Grundläggande byggfunktioner i Scala CLI

De grundläggande funktionerna sammanfattas nedan (se även Appendix C.4.3):

`scala-cli repl`	Starta Scala REPL. Det går även bra med enbart `scala-cli`
`scala-cli repl hello.scala`	Starta Scala REPL med kompilerade koden i `hello.scala` på classpath.
`scala-cli repl .`	Starta repl med kodfiler i aktuell katalog tillgängliga på classpath.
`scala-cli compile hello.scala`	Kompilera koden i filen `hello.scala`
`scala-cli compile .`	Kompilera alla kodfiler i aktuell katalog.
`scala-cli run hello.scala`	Kompilera koden i filen `hello.scala` och kör igång eventuellt huvudprogram om kompileringen gick bra.
`scala-cli run .`	Kompilera och kör alla kodfiler i aktuell katalog.
`scala-cli run . --list-main-class`	Lista alla huvudprogram.
`scala-cli run . -M mypkg.myMain`	Kör ett specifikt huvudprogram. Förk. `-M` kan även skrivas `--main-class`
`scala-cli package .`	Paketera all kompilerade kodfiler i en körbar fil.

F.2.3 Använda optioner för att styra Scala CLI

Det finns en mängd olika optioner som du kan lägga till för att styra vad Scala CLI ska göra. Se exempel nedan och förklaringar i efterföljande tabell:

```
scala-cli run . -S 3.3 -O -unchecked --dep se.lth.cs::introprog::1.3.1 -w
```

Här förklaras några vanliga optioner som kan användas vid både kompilering och exekvering:

`--scala 3.3`	Använd version 3.3 av Scala. Optionen `--scala` kan förkortas med `-S`
`--watch`	Upprepa kommando vid sparad ändring. Optionen `--watch` förkortas med `-w`
`--jar introprog.jar`	Lägg till en jar-fil på classpath.
`--dep se.lth.cs::introprog::1.3.1`	Lägg till ett beroende på classpath.
`--scalac-option -unchecked`	Lägg till en kompilator-option som ger extra varningar vid osäker kod. Optionen `--scalac-option` kan förkortas med `-O`

Fördelen med att explicit ange en viss Scala-version är att byggprocessen blir *upprepningsbar* även på en annan dator som kanske råkar har en annan Scala-version installerad. Det går att ”spika fast” Scala-versionen till en ännu mer precis version, t.ex. `3.2.2`. Om inte den versionen av kompilatorn finns installerad på datorn så kommer Scala CLI att automatiskt ladda ner och använda den explicit efterfrågade versionen under byggandet.

Det går också att be om den absolut mest rykande färska kompilatorversionen om man vill använda det allra senaste i Scala-språkets utveckling med `3.nigthly`. Speciellt kräver s.k. experimentella funktioner att du använder `nightly`-versionen[6].

F.2.4 Generera dokumentation med Scala CLI

Scala CLI kan också skapa dokumentation baserat på dokumentationskommentarer (se vidare Appendix E), enligt nedan. Med optionen `--ouput` kan du ange destinationskatalog och med `--force` så skrivs ev. gammal dokumentation över.

```
scala-cli doc . --output apidoc --force
```

F.2.5 Paketering av exekverbar fil med Scala CLI

Scala CLI kan paketera din kod i en exekverbar fil så här:

```
scala-cli package . --force --standalone --output myapp
```

Här förklaras några vanliga optioner som kan användas vid paketering:

[6] https://stackoverflow.com/questions/40622878/how-do-i-use-a-nightly-build-of-scala

`--output`	Ange namn på utfilen med paketerad kod. Optionen `--output` kan förkortas med `-o`
`--force`	Skriv över utfilen om den redan finns. Optionen `--force` kan förkortas med `-f`
`--standalone`	Skapa en självständig, exekverbar jar-fil med din kod och dess beroenden.
`--library`	Skapa en jar-fil med din kod för användning av andra program.
`--assembly`	Skapa en fet jar-fil med din kod och alla dess beroenden för användning av andra program.

Några av optioner, t.ex. `--assembly` kräver s.k. power-läge. Om du inte slagit på power-läge får du en varning som berättar hur det görs: du kan antingen slå på power-läget tillfälligt med att även inkludera den inledande optionen `--power`, eller slå på det permanent med en inställning så här:

```
scala-cli config power true
```

F.2.6 Optioner som användningsdirektiv i ”magiska” kommentarer

I stället för att använda optioner i terminalen så kan du ge dessa som s.k. användningsdirektiv (eng. *using directives*) i ”magiska” kommentarer som börjar med `//> using` i början av valfri kod-fil.

Om du har flera kodfiler i samma katalog brukar man skapa en speciell fil som vanligtvis kallas `project.scala` och i den samla alla användningsdirektiv som styr byggandet. Här visas ett exempel hur det kan se ut:

```
//> using scala 3.3
//> using option -unchecked -deprecation
//> using option -Wunused:all -Wvalue-discard -Wsafe-init
//> using dep se.lth.cs::introprog::1.3.1
```

De kompilatoroptioner som föreslås för att få extra varningar ovan har följande betydelser:

`-unchecked`	Extra varningar vid flera fall av osäker kod.
`-deprecation`	Förklaring vid användning av utgående funktioner.
`-Wunused:all`	Varning om deklarationer ej används.
`-Wvalue-discard`	Varning vid förlorat värde.
`-Wsafe-init`	Varna vid risk för ej initialiserade värden.

Du hittar mer information om Scala CLI här: `https://scala-cli.virtuslab.org/`

F.3 Scala Build Tool `sbt`

Byggverktyget `sbt` är skrivet i Scala och är det mest använda byggverktyget bland Scala-utvecklare. Med `sbt` kan du skriva byggkonfigurationsfiler i Scala och även styra byggprocessen via ett interaktivt kommandoskal i terminalfönstret. Med inkrementell (stegvis) kompilering och parallellkörning av byggprocessens olika delar, kan den snabbas upp avsevärt.

F.3.1 Installera sbt

`sbt` finns förinstallerat på LTH:s datorer och körs igång med kommandot `sbt` i terminalen.

Om du vill installera `sbt` på din egen dator, säkerställ först att du har `java` på din dator med terminalkommandot `java -version`. Om `java` saknas, följ instruktionerna i avsnitt C.3.2 på sidan 213. Följ sedan instruktionerna här för att installera `sbt`: `http://www.scala-sbt.org/download.html`

- **Linux**. Om du surfar till ovan sida från en Linux-dator syns några terminalkommando som du använder för att installera `sbt` i terminalen.
- **Windows**. Om du surfar till ovan sida från en Windows-dator visas en länk till en `.msi`-fil. Ladda ner och dubbelklicka på den. Innan du kör igång med sbt i en Windowsterminal är det bra att skriva `chcp 65001` för att särskilda tecken (t.ex. ÅÄÖ) ska fungera som de ska.
- **macOS**. Följ instruktionerna under rubriken *Manual Installation*.

När du kör sbt första gången kommer ytterligare filer att laddas ner och installeras och delar av denna process kan ta lång tid. Ha tålamod och avbryt inte körningen, även om inget speciellt ser ut att hända på ett bra tag.

F.3.2 Använda sbt

`sbt` är konstruerat för att klara mycket stora projekt, men det är enkelt att använda `sbt` även om du bara har ett litet projekt med någon enstaka kodfil. Med `sbt` installerat, är det bara att köra igång `sbt`och skriva `run` enligt nedan

```
> sbt
sbt> run
```

i terminalen i den katalog där dina kodfiler ligger. `sbt` letar då upp och kompilerar alla de `.scala`-filer som ligger i katalogen och, om det bara finns ett objekt med main-metod, kör `sbt` igång denna main-metod direkt, förutsatt att kompileringen kan avlutas utan fel. Även `.java`-filer kompileras automatiskt om de ligger i samma katalog.

Om du enbart skriver `sbt` körs det interaktiva kommandoskalet igång, där du kan köra kommando så som `compile` och `run`. Om du skriver ett ~ före kommandot `run`, enligt nedan kommer `sbt` vara aktivt i bakgrunden medan du editerar och så fort du sparar en ändring kommer omkompilering av ändrade kodfiler ske, varefter main-metoden exekveras om kompileringen lyckades.

```
> sbt
[info] Set current project to hello (in build file:/home/bjornr/hello/)
> ~run
[info] Running hello
Hello, World!
[success] Total time: 0 s, completed Aug 9, 2016 9:50:16 PM
1. Waiting for source changes... (press enter to interrupt)
[info] Compiling 1 Scala source to /home/bjornr/hello/target/scala-2.10/classes
[info] Running hello
Hello again, World!
[success] Total time: 1 s, completed Aug 9, 2016 9:50:45 PM
2. Waiting for source changes... (press enter to interrupt)
```

I ovan körning gör sbt en omkompilering, efter att en ändring av utskriftssträngen sparats.

```
// in file hello.scala

@main def run = println("Hello again, World!") // add 'again'; Ctrl+S
```

Katalogstruktur

Om man har kod i underkataloger förutsätter sbt att du följer en viss, specifik katalogstruktur. Denna katalogstruktur används även av andra byggverktyg, så som Maven, och fungerar även i många utvecklingsmiljöer så som Eclipse och IntelliJ.

Det blir också mindre rörigt och lättare för alla att hitta i projektets kataloger om dina kodfiler placeras i en given struktur som är allmänt accepterad. Placera därför gärna dina kodfiler i underkataloger enligt strukturen som visas i figur F.1.

```
src/
  main/
    resources/
       <files to include in main jar here>
    scala/
       <main Scala sources>
    java/
       <main Java sources>
  test/
    resources
       <files to include in test jar here>
    scala/
       <test Scala sources>
    java/
       <test Java sources>
```

Figur F.1: Katalogstrukturen i ett sbt-projekt. Bara de kataloger som har något innehåll behöver finnas.

Lägg enligt denna struktur dina .scala-filer i underkatalogen src/main/scala/ och dina .java-filer i underkatalogen src/main/java/. Om du lägger kod i någon av katalogerna src/test/scala/ respektive src/test/java/ kommer denna kod köras när du skriver sbt-kommandot test. Om du lägger filer i underkatalogen src/main/resources/ kommer dessa att paketeras med i jar-filen som skapas när du kör sbt-kommandot package.

Om du använder t.ex. `package x.y.z;` i din Java-kod, måste även strukturer på underkataloger matcha och kodfilen alltså ligga i src/main/java/x/y/z/.

I Scala är det egentligen inte nödvändigt att koden ligger i samma katalog som de kompilerade .class-filerna, men det kan vara bra att följa paketstrukturen även för Scala-källkoden; speciellt om du senare vill kunna köra din kod med Eclipse, som kräver denna överensstämmelse mellan paket och källkodskataloger, inte bara för Java, utan även för Scala.

Konfigurera dina byggen i filen `build.sbt`

Om du vill göra inställningar och även hjälpa andra att kunna återskapa dina byggen, så skapa en konfigurationsfil med namnet `build.sbt` och placera den i projektets baskatalog. Figur F.2 visar en byggkonfigurationsfil som specificerar vilken version av Scala-kompilatorn du använder, så att andra ska kunna bygga din kod under samma förutsättningar som du.

```
scalaVersion := "3.2.2"
```

Figur F.2: Exempel på konfigurationsfil för `sbt`. Filen ska ha namnet `build.sbt` och vara placerad i projektets baskatalog.

Här är ett exempel på en mer omfattande `build.sbt`:

```
scalaVersion   := "3.2.2"
scalacOptions  := Seq("-unchecked", "-deprecation") //mer info vid kompilering

fork           := true   // kör i en egen JVM, bra om ljud och grafik används
connectInput   := true   // koppla indata till rätt JVM vid fork
outputStrategy := Some(StdoutOutput)  // koppla utdata till rätt JVM vid fork

ThisBuild / useSuperShell := false // stänger av rör(l)ig progressinformation
```

Du kan läsa mer om alla möjligheter med `sbt` och hur man skapar mer avancerade byggkonfigurationsfiler här: `http://www.scala-sbt.org`

Fixera versionen för sbt i `project/build.properties`

Om du skapar en katalog `project` (om den inte redan finns) kan du i en fil med namnet `build.properties` fixera versionen av sbt genom att låta filen ha detta innehåll (notera punkten och avsaknaden av citationstecken):

```
sbt.version=1.8.2
```

På så sätt riskerar du inga inkonsekvenser mellan en gammal `build.sbt` vid framtida uppdatering av sbt, ovan inställning garanterar att ditt bygge alltid kommer att byggas med denna version av sbt, och andra kan bygga din kod under samma förutsättningar som du.

Lägga till kursbiblioteket `introprog` som ett beroende

Med följande text i `build.sbt` får du automatisk nedladdning och tillgång till kursens Scala-bibliotek `introprog` med bl.a. klassen `PixelWindow` för grafiska fönster:

```
scalaVersion := "3.2.2"
libraryDependencies += "se.lth.cs" %% "introprog" % "1.3.1"
```

Ändra ev. versionsnummer till senaste versionen. Notera de dubbla procent-tecknen före biblioteksnamnet, som används för Scala-bibliotek som kors-publicerats för olika versioner av Scala, t.ex. 3, 2.12 och 2.13, vilket gör att rätt biblioteksversion för rätt kompilatorversion laddas ned.

Du kan läsa mer om `introprog` här:

- Kod: https://github.com/lunduniversity/introprog-scalalib
- Dokumentation: http://cs.lth.se/pgk/api

Lägga till andra beroenden

I filen `build.sbt` kan man lägga till många beroenden till flera olika kodbibliotek. Det finns på nätplatsen *Maven Central* en mycket omfattande koddatabas, som är sökbar här http://search.maven.org, med en massa användbara öppenkällkodsprojekt. Du kan be `sbt` att ladda ner den färdigkompilerade koden till vilket som helst av projekten på *Maven Central* och automatiskt lägga till jar-filen till `classpath` så att koden blir tillgänglig direkt i ditt program.

Till exempel kan du lägga till Java-biblioteket `jline` som gör det möjligt att göra terminalinläsning från tangentbordet med många bra finesser, t.ex. kommandohistorik med pil-upp, bara genom att lägga till denna rad i din `build.sbt` och den specifika version du önskar (notera enkla procent-tecken för Java-bibliotek):

```
libraryDependencies += "org.jline" % "jline" % "3.20.0"
```

Du kan läsa mer om `jline` här: https://jline.github.io/

Appendix G

Versionshantering och kodlagring

G.1 Vad är versionshantering?

Versionshantering[1] (eng. *version control* eller *revision control*) av mjukvara innebär att hålla koll på olika versioner av koden i ett utvecklingsprojekt allteftersom koden ändras. Versionshantering är en deldisciplin inom **konfigurationshantering** (eng. *software configuration management*) som inbegriper allt i processen för att identifiera, besluta, genomföra och följa upp ändringar.

En viktig del av versionshantering är att *lagra* olika versioner av koden allt eftersom den utvecklas, så att tidigare versioner kan *återskapas* vid behov. Ett bra verktygsstöd och en väldefinierad arbetsprocess för versionshanteringen, som alla i utvecklingsprojektet följer, möjliggör att flera utvecklare kan *arbeta parallellt* med att sammanfoga (eng. *merge*) varandras tillägg och ändringar i den gemensamma kodbasen utan att det blir kaos och förvirring.

God versionshantering är helt avgörande för utvecklarnas produktivitet, speciellt för stora projekt med många utvecklare som jobbar parallellt mot en omfattande kodbas med många olika interna och externa komponenter. Men även ett litet projekt med en enda utvecklare kan ha god nytta av ett versionshanteringsverktyg och ett disciplinerat förfarande för att namnge versioner, t.ex. för att kunna återskapa tidigare versioner av projektets olika kodfiler när en ändring visar sig mindre lyckad.

Det finns flera olika modeller för hur kodlagringen sker:

- **lokal**; alla utvecklare jobbar i samma, lokala filsystem där alla olika versioner lagras.
- **centraliserad**; ett repositorium (förk. repo), alltså en databas med koden, finns centralt på en server som alla jobbar mot med hjälp av en versionshanteringsklient.
- **distribuerad**; alla utvecklare har sitt eget lokala repo och varje utvecklare initierar enskilt delning av ändringar mellan olika repo.

[1] en.wikipedia.org/wiki/Version_control

G.2 Versionshanteringsverktyget Git

Det finns många olika versionshanteringsverktyg[2] som använder olika modeller för kodlagring; lokal, centraliserad, distribuerad eller kombinationer därav. På senare tid har verktyget **Git**[3] fått en stark ställning, speciellt i öppenkällkodsvärlden. Git utvecklades ursprungligen av Linus Torvalds för att versionshantera Linuxkärnan, men har växt till ett omfattande öppenkällkodsprojekt med stor spridning och många användare och bidragsgivare.

Git är skapad för **distribuerad** versionshantering där var och en kan jobba snabbt och smidigt i sitt eget lokala repo, utan att behöva vänta på att en klient ska synkronisera koden med ett centralt repo på en server över nätverket. Ändringar delas mellan repo på begäran av enskilda utvecklare.

Varje ny version av koden lagras som en avgränsad mängd ändringar sedan förra versionen, en s.k. **commit**[4], och hanteras internt av Git i en lokal databas i katalogen `.git` som ligger överst i din projektkatalog. Genom olika kommandon i terminalen, eller via en klient med ett grafiskt användargränssnitt, kan din kod överföras till och från den lokala koddatabasen, alternativt delas med andra repon via nätet.

Det finns en välskriven bok kallad *”Pro Git”* som förklarar Git på djupet och är tillgänglig fritt här: `https://git-scm.com/book/en/v2`. Läs kapitel 1 och 2 så får du en bra grund att stå på.

Dessa termer är bra att kunna utantill innan du kör igång med Git:

- **repo** (*substantiv*: ett repositorium, *eng. a repository*) En koddatabas med ändringshistorik.
- **commit** (*substantiv*: en inlämning, *verb*: att lämna in). En avgränsad mängd nya ändringar lämnas in i det lokala repot. Repots ändringshistorik utgörs av sekvensen av alla inlämningar.
- **push** (*substantiv*: en leverans, *verb*: att leverera, att trycka upp). En eller flera inlämningar trycks upp till ett annat repo.
- **pull** (*substantiv*: en hämtning, *verb*: att hämta, att dra ner). En eller flera inlämningar dras ner från ett annat repo.
- **merge** (*substantiv*: en sammanfogning, *verb*: att sammanfoga). En eller flera inlämningar slås samman till en ny inlämning.
- **merge conflict** (*substantiv*: en sammanfogningskonflikt, *eng. a merge conflict*) Problem vid sammanfogning; ändringar kan inte enkelt sammanfogas på ett entydigt sätt.
- **pull request** (förk. PR, *substantiv*: en hämtningsbegäran, *verb*: att begära en hämtning). Utvecklare A ber en annan utvecklare B att hämta en eller flera inlämningar från A:s repo och sammanfoga med B:s repo.

G.2.1 Installera git

Git finns förinstallerat på LTH:s Linuxdatorer. Du kan kolla om Git redan finns på din maskin genom att skriva `git help` i terminalen.

[2] https://en.wikipedia.org/wiki/List_of_version_control_software
[3] https://en.wikipedia.org/wiki/Git_(software)
[4] På svenska kan t.ex. ”inlämning” användas, men låneordet commit är redan etablerat.

Det finns bra instruktioner om hur du installerar Git på din egen maskin här: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

VS Code har speciellt stöd för git och du kan inne ifrån VS Code göra t.ex. add, commit, push och pull via editorns grafiska gränssnitt. Läs mer här: https://code.visualstudio.com/docs/editor/versioncontrol

Det finns även många andra grafiska användargränssnitt till git, t.ex. GitHub Deskop (Windows/Mac) eller GitKraken (Linux/Windows/Mac). Se fler exempel här: https://git-scm.com/downloads/guis

G.2.2 Anpassa Git

Innan du börjar använda git, konfigurera ditt namn och din email med nedan terminalkommando, där du anger ditt namn i stället för `Förnamn Efternamn` och din mejladress i stället för `mejladr@plats.se`. Namnet och mejladressen kommer lagras i varje commit som du gör så att det går att se vem som har gjort en given ändring.

```
> git config --global user.name "Förnamn Efternamn"
> git config --global user.email mejladr@plats.se
```

Läs mer om hur du gör andra inställningar här, t.ex. hur du anger vilken editor som git startar när du ska skriva commit-beskrivningar:
https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup

G.2.3 Använda git

Nedan listas några vanliga terminalkommandon i Git.

- Skapa ett repo i en katalog:

```
> cd myproject
> git init
```

- Se vilka filer som ändrats och ännu ej lämnats in:

```
> git status
> git status -s
```

- Se vilka ändringar som gjorts i filer som ännu ej lämnats in:

```
> git diff
```

- Se vilka inlämningar som finns i ändringshistoriken:

```
> git log
> git log --oneline -5
```

- Lägg till filer som ska ingå i nästa inlämning och gör sedan inlämningen; ge inlämningen en bra beskrivning som förklarar vad inlämningen omfattar:

```
> git add *.scala
> git commit -m 'initial project version'
```

- Ångra alla tillägg inför inlämning (ändringarna finns kvar och kan läggas till igen om du vill):

```
> git reset
```

- Du kan skippa de senaste, ännu ej commitade, ändringar i filen `filename`, och göra ”*undo*”, med kommandot `git checkout` på filen enligt nedan. Gör bara detta om du är helt säker på att du vill ångra dina senaste ändringar.
  VARNING! Dina senaste ändringar i filen förloras för alltid; kan ej ångras!

```
> git checkout filename
```

- Man vill förhindra versionshantering av vissa filer, t.ex. binärkodsfiler så som `.class`-filer och andra genererade filer. Detta gör du genom att skapa en fil med namnet `.gitignore` och lägga in filändelser enligt nedan syntax, där `**/` avser alla kataloger och underkataloger och `*` kan vara vilken början på ett filnamn som helst. Symbolen # föregår en kommentarsrad.

```
# this is my .gitignore

# Java / Scala
**/*.class

# Sbt
**/target
```

G.3 Kodlagringsplatser på nätet

Många utvecklare använder kodlagringsplatser på nätet (”i molnet”) (eng. *code hosting*) för att underlätta samarbete kring kod och för att dela med sig av öppen källkod. Det finns många olika kodlagringsplatser som kan användas gratis under vissa förutsättningar eller mot betalning med tillhörande extratjänster.

> **OBS!** Du får *inte* lagra dina lösningar på kursens laborationer i ett öppet repo. Om du vill använda en kodlagringsplats måste du säkerställa att dina lösningar förblir i ett stängt repo utan att någon annan kan komma åt det.

Nedan beskrivs några vanliga nätplatser för öppen och sluten kodlagring, som alla är Git-baserade:

- **GitHub**, `https://github.com`, är en av de mest populära kodlagringsplatserna för öppen källkod, men har även blivit en populär plats för jobbsökande utvecklare att visa upp sina kodarbetsprover för framtida arbetsgivare. GitHub är gratis att använda för dig som privatperson. Många företag betalar GitHub för att lagra sin stängda kod med tilläggstjänster för att testa, bygga och driftsätta kod etc. Koden som styr själva kodlagringsplatsen GitHub är stängd, till skillnad från GitLab. GitHub köptes 2018 av Microsoft för 65 miljarder kronor.

- **GitLab**, `https://gitlab.com`, erbjuder gratis kodlagring för öppen källkod, men det är även gratis för privatpersoner och gemenskapsprojekt att ha stängda repo. Företag kan betala för stängd kodlagring med extratjänster för att testa, bygga och driftsätta kod etc. GitLab är i sig ett öppenkällkodsprojekt och koden som styr kodlagringsplatsen är öppen och fri. Detta innebär att du själv kan ladda ner koden och starta en kodlagringsplats. LTH har en GitLab-baserad kodlagringsplats här: `https://git.cs.lth.se`

- **BitBucket**, `https://bitbucket.org`, är en populär kodlagringsplats både för öppen och stängd källkod och drivs av det australiensiska företaget Atlassian. Det är gratis för privatpersoner och små team att ha både öppna och slutna repon, men bara om det är få bidragsgivare. Kostnader tillkommer om antalet bidragsgivare kommer över en viss nivå. Universitetsanställda och studenter kan få mer gynnsamma villkor efter ansökan. Atlassian erbjuder en hel verktygssvit för att hantera buggar och samarbeta över nätet. BitBucket stödjer, förutom Git, även andra versionshanteringsverktyg.

Använda kodlagringsplatser

Om du inte redan gjort det är det bra om du registrerar ditt användarnamn, förslagsvis `fornamnefternamn` som ett ord utan svenska tecken med små bokstäver, på någon eller alla av ovan sajter, dels för att paxa ditt namn och dels för börja samarbeta med utvecklare världen över. Det är bra att välja *ett* användarnamn för *alla* kodlagringsplatser på nätet, speciellt om du jobbar med öppen källkod där ditt namn kommer associeras med alla de kodbidrag du gör under ditt yrkesliv.

Om du inte vet vilken sajt du ska välja, börja med `https://github.com`. Om du vill att även kodlagringssajten ska drivas av öppen källkod, testa `https://gitlab.com`.

Med en Git-baserad kodlagringsplats får du möjlighet att synka ditt lokala repo mot en server på nätet med hjälp av `git`-kommandon i terminalen eller via en Git-klient med grafiskt användargränssnitt, se avsnitt G.2.1.

Innan du börjar använda en kodlagringsplats är det bra att sätta sig in i begreppen nedan.

- **clone** (*substantiv*: en klon är kopia av ett (nätlagrat) repo, *verb*: att klona, att skapa en kopia). Genom att klona ett repo som ligger på en nätlagringsplats kan du bygga, undersöka och vidareutveckla koden lokalt på din dator. Om du har rättigheter att lämna in kod till det centrala originalet kan du pusha dina commits direkt via terminalkommando eller Git-klient.
- **fork** (*substantiv*: en förgrening av ett helt repo, *verb*: att förgrena ett repo, att ”forka”). Genom att förgrena ett repo skapar du en kopia, normalt även den nätlagrad på en kodlagringsplats, som du kan utveckla separat från originalet. Det blir då möjligt för dig att lämna in ändringar och trycka upp dem, även om du inte har rättigheter att leverera (”pusha”) till originalet. Gör en ändringsbegäran (Pull Request, PR) om du vill bidra med dina ändringar, så kan ägaren av originalet sedan välja att sammanfoga (”merga”) dina ändringar med originalet. Många nätlagringsplatser, så som GitHub, har en speciell knapp som du trycker på för att enkelt skapa en fork av ett repo under din användare.
- **upstream** (*preposition*: uppströms, *substantiv*: uppströmsrepo) Ett uppströmsrepo utgör original till ett förgrenat repo (en ”fork”).
  - Här beskrivs hur du länkar en förgrening uppströms: `https://help.github.com/articles/configuring-a-remote-for-a-fork/`
  - Här beskrivs hur du synkar en förgrening uppströms: `https://help.github.com/articles/syncing-a-fork/`

Om du vill bidra till ett öppenkällkodsprojekt, börja med att forka repot på kodlagringsplatsen och sedan klona repot till din lokala dator. Därefter kan du commita ändringar och pusha till din fork och slutligen göra en pull request från din fork till upstream. Läs om hur ett bidrag kan gå till i avsnitt **??**.

Här följer några användbara kommandon:

- Skapa en lokal kopia av ett fjärran (eng. *remote*) repo; här visas hur du klonar kursens repo från GitHub:

```
$ git clone --depth 1 https://github.com/lunduniversity/introprog
```

- Dra ner nya inlämningar från ett fjärran repo:

```
$ git pull
```

- Trycka upp nya lokala inlämning till ett fjärran repo:

```
$ git push
```

Appendix H

Integrerad utvecklingsmiljö

H.1 Vad är en integrerad utvecklingsmiljö?

En integrerad utvecklingsmiljö (eng. *integrated development environment, IDE*) samlar ett flertal verktyg för att skapa, köra och testa program, inklusive en avancerad **editor** och speciella felsökningsverktyg. Det finns flera utvecklingsmiljöer att välja mellan, som kan användas för både Scala och Java.

En IDE ger stöd för **kodkomplettering** (eng. *code completion*) där tillgängliga metoder visas i en lista och resten av ett namn kan fyllas i automatiskt efter att du skrivit de första bokstäverna i namnet. En IDE kan hjälpa dig med formatering och även skapa skelettkod utifrån **kodmallar** (eng. *code templates*). Med **felindikering** (eng. *error highlighting*) får du understrykning av vissa fel direkt i koden och ibland kan du även få hjälp med förslag på åtgärder för att rätta till enkla fel. Funktioner för **avlusning** (eng. *debugging*) hjälper dig att felsöka medan du kör din kod. Med funktioner för **omstrukturering** (eng. *refactoring*) av kod får du hjälp av editorn i samarbete med kompilatorn att göra omfattande strukturförändringar i många kodfiler samtidigt, t.ex. namnbyten med hänsyn taget till språkets synlighetsregler.

Alla dessa avancerade funktioner kan öka produktiviteten avsevärt, men samtidigt tar de tid att lära sig och en IDE kan kräva mycket datorkraft och viss väntetid jämfört med en vanlig, fristående editor. I början kan all funktionalitet upplevas som överväldigande och det kan vara svårt att hitta i alla menyer och inställningar. Det är därför många som föredrar en fristående, snabbstartad kodeditor före en fullfjädrad, tungrodd IDE, speciellt om det rör ett mindre program. Å andra sidan kan en IDE med kodkomplettering vara till stor hjälp när man ska lära sig ett nytt api och experimentera med en okänd kodmassa. Med tiden har hanliga editorer, så som VS Code, fått allt fler funktioner som tidigiare bara fanns i en fullfjädrad IDE[1], och den praktiska skillnaden allt mindre mellan en ”vanlig” editor och en IDE blir allt mindre.

I kursen använder vi flera utvecklingsmiljöer. På första labben använder vi Kojo (se appendix A) som är en IDE speciellt anpassad på nybörjare. Sedan använder vi en editor t.ex. VS Code, gärna i kombination med byggverktyget `sbt`. Du kan under andra halvan av kursen välja att gå över från VS Code till att använda en (annan) IDE, men det går utmärkt att fortsätta med VS Code som numera har en bra debugger för Scala genom tillägget Metals. Om du vill använda en IDE i stället för VS Code så rekommenderas IntelliJ IDEA med Scala-plugin. Om du redan lärt dig Eclipse på djupet och verkligen vill fortsätta med denna IDE, välj då ScalaIDE – dock har denna IDE inte hängt med i den tekniska utvecklingen och ligger kvar på Scala 2.12.

[1] Se t.ex. LSP: https://en.wikipedia.org/wiki/Language_Server_Protocol

H.2 Visual Studio Code med tillägget Scala Metals

Visual Studio Code[2], förkortat VS Code eller bara `code`, är en gratis utvecklingsmiljö som är mestadels öppen källkod[3]. Projektet startades och leds av Microsoft och har en aktiv gemenskap med många utvecklare och många användbara tillägg (eng. *extensions*).

VS Code kallas ofta för ”bara” en editor, men har genom åren utvecklats till en fullfjädrad IDE med bl.a. inbyggd debugger och stöd för många olika språk via ett omfattande bibliotek av tillägg.

- Läs mer om hur man använder VS Code här:
  `https://code.visualstudio.com/docs`
- Läs mer om hur du använder Scala i VS Code här:
  `https://scalameta.org/metals/docs/editors/vscode`

Det finns många användbara kortkommandon som gör dig snabbare och snabbare när du kodar, allteftersom du lär dig nya kortkommandon. Ett bra tips är att du lär dig minst ett nytt kortkommando om dagen och efter ett tag kan du riktigt många. Här finns en sammanfattning av de viktigaste kortkommandona för VS Code för Linux:
`https://code.visualstudio.com/shortcuts/keyboard-shortcuts-linux.pdf`
Byt ut `linux` mot `windows` eller `macos` i adressen ovan för motsvarande plattform.

H.2.1 Installera VS Code och Metals

VS Code är förinstallerad på LTH:s datorer, men du behöver själv installera Scala-tillägget **Metals** första gången du kör igång VS Code på LTH:s datorer. Läs om installation av Metals här:
`https://marketplace.visualstudio.com/items?itemName=scalameta.metals`

Läs mer om hur du installerar VS Code på din egen dator här:
`https://code.visualstudio.com`

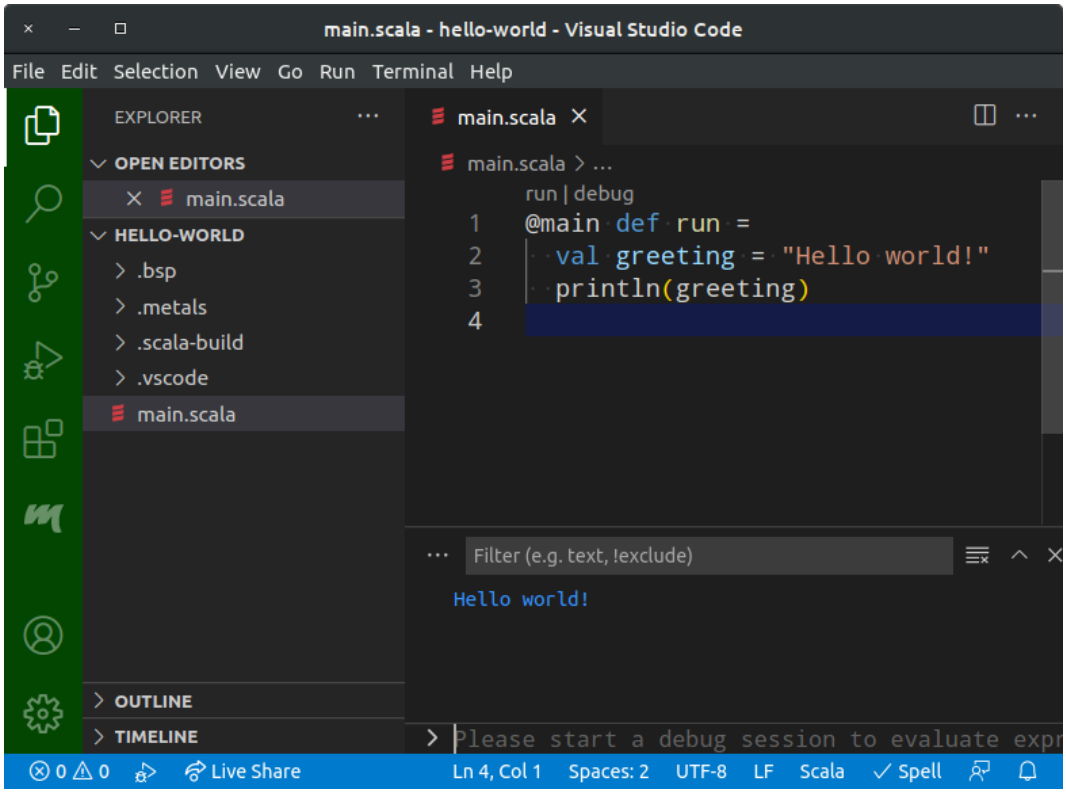
Mer information om installation av verktyg finns på kursens hemsida:
`https://cs.lth.se/pgk/verktyg`

H.2.2 Köra program i VS Code

Det finns olika sätt att köra igång huvudprogrammet i ett projekt i VS Code:

1. Använd `scala-cli run .` i ett separat terminalfönster. Läs mer om `scala-cli` i Appendix C.4.3.
2. Kör igång `sbt` i ett separat terminalfönster och kör kommandot `run` inifrån `sbt`. Detta kräver att du har en giltig `build.sbt`, se Appendix F.
3. Köra igång program inifrån VS Code. Detta kräver att du öppnat katalogen med din kod med File-menyns ”Open Folder”, eller genom att du startar VS Code med `code .` överst i din projektkatalog (du ser att detta är gjort om nedre meddelandefältet är blått i stället för lila). Du behöver *innan* du startar VS Code en första gång köra `scala-cli setup-ide .` (se Appendix C.4.3), eller skapa en giltig `build.sbt`-fil (se Appendix F) som du importerar i VS Code när frågan dyker upp i nedre högra hörnet.

[2] en.wikipedia.org/wiki/Visual_Studio_Code
[3] Varianten VS Codium `https://vscodium.com/` är helt fri från stängd källkod och telemetri.



Figur H.1: Kör program genom att klicka på run ovanför huvudprogrammet.

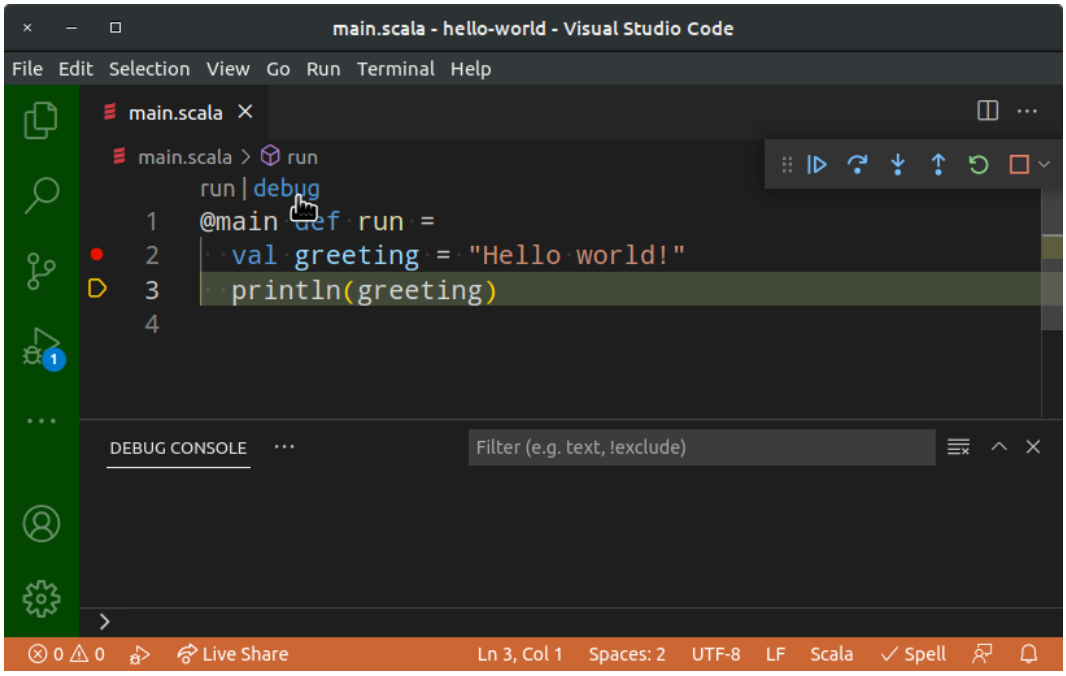
4. Kombinera Scala CLI eller `sbt` och VS Code. Kör detta kommando i ett separat terminalfönster: `scala-cli compile . -w` där `-w` betyder *watch* och gör så att ändringar bevakas. Om du istället använder `sbt` kör `sbt ~compile` i ett separat terminalfönster (notera tilde-tecknet som gör att ändringar bevakas). Vid ändringsbevakning kommer kompileringsfel visas där varje gång du sparar en ändring i VS Code med Ctrl+S. När alla kompileringsfel är åtgärdade och du är redo att testköra så klickar du på run.

Du ser att VS Code är beredd att köra igång ditt program genom att det (efter ett tag) kommer upp en extra rad ovanför ditt huvudprogram med texten run | debug och då kan du klicka på run för att köra ditt program. Utdata från körningen visas i en flik under koden. Observera att det kan ta lite tid för VS Code att förbereda allt som behövs för att kunna köra ditt program. Håll koll på om VS Code håller på med dessa förberedelser i det blåa meddelandefältet längst ned till höger. När allt är klart efter att du startat VS Code står det ”Index complete!” bredvid en raketsymbol i meddelandefältet.

Om något krånglar och du inte får fram run | debug ovanför din `@main`-funktion, trots du har startat VS code enligt ovan, så prova att under Metals-fliken (ikonen med det stiliserade M:et i det gröna verktygsfältet) klicka på någon av ”Restart build server” eller ”Import build” (den senare tar längre tid men börjar om helt) och vänta tills det står ”Index Complete!” i det blå meddelandefältet och då ska run | debug synas ovanför din `@main`-funktion.

H.2.3 Använda debuggern i VS Code

Innan du börjar använda debuggern, läs först om allmän felhantering i Appendix D.

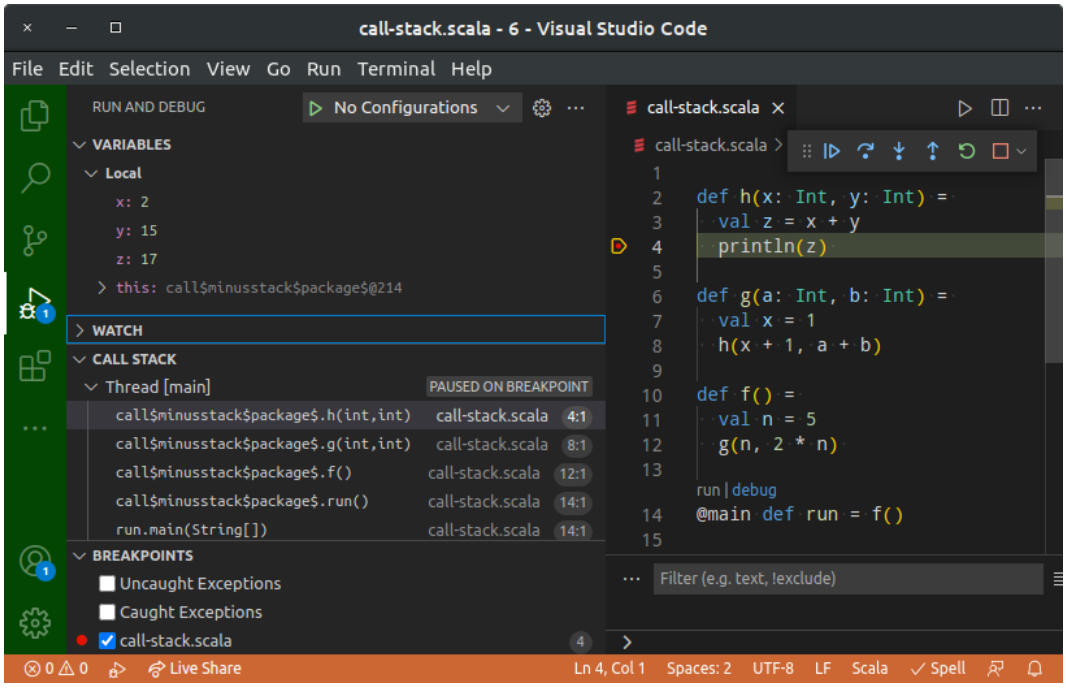


Figur H.2: Debuggern i VS Code. Nederkanten är orange när debuggern kör.

Du kan aktivera debuggern i VS Code för dina Scala-program genom att klicka på ”debug” ovanför din `main`-metod, förutsatt att du har tillägget Metals installerad i VS Code. Du behöver även köra `scala-cli setup-ide .` en första gång (se Appendix C.4.3), eller ha en giltig `build.sbt`-fil (se Appendix F) som du importerar i VS Code när frågan dyker upp i nedre högra hörnet.

Figur H.2 på sidan 252 visar hur det kan se ut när debuggern i VS Code är aktiverad. När debuggern är igång får det nedersta meddelandefältet en orange färg (istället för blå). Till vänster om radnummerkolumnen kan du klicka för att aktivera och avaktivera brytpunkter. Aktiverade brytpunkter visas som en röd prick i marginalen till vänster. Den ihåliga gula pilen i marginalen pekar på den rad som kommer att exekveras härnäst. Notera panelen med olika knappar i överkanten av editorfönstret. Med dessa knappar kan du styra exekveringen enligt följande (lär dig gärna kortkommandona så blir du snabbare):

- **Fortsätt**. Den blåa play-knappen kör vidare till nästa brytpunkt eller tills programmet är klart om brytpunkt ej påträffas. Kortkommando ”Continue”: F5.
- **Stega över**.Den blåa böjda framåtpilen kör en rad i taget *utan* att hoppa in i funktioner. Kortkommando ”Step Over”: F10.
- **Stega in**. Den blåa nedåtpilen kör vidare en rad i taget och hoppar in i funktioner om raden innehåller funktionsanrop. Kortkommando ”Step Into”: F11.
- **Stega ut**. Den blåa uppåtpilen kör klar innevarande funktion. Kortkommando ”Step Out”: Shift+F11.
- **Kör igen**. Den gröna återstartsikonen kör om ditt program. Kortkommando ”Restart”: Ctrl+Shift+F5.
- **Avbryt**. Den röda stoppknappen avbryter denna debuggingsession. Kom ihåg att avbryta innan du startar en ny debuggingsession, annars kan det lätt bli



Figur H.3: Anropsstack och variabler i VS Code.

förvirrande med många samtidigt pågående körningar. Kortkommando ”Stop”: Shift+F5.

Figur H.3 på sidan 253 visar hur VS Code presenterar anropsstacken och värdet på de variabler som syns där exekveringen befinner sig för tillfället. Du får fram detta genom att klicka på ikonen med en lus och en playknapp i det vertikala, gröna verktygsfältet längst till vänster. I en blå ring står en etta om du har startat en debuggingsession. Om det står en tvåa eller mer så har du flera sessioner igång och då kan det vara klokt att avsluta alla utom en, så att inte förvirring uppstår om vilken session som är den aktuella.

Mycket av konsten i debugging handlar om att undersöka variablers värde under exekveringen för att ta reda på om din hypotes om vad som händer under exekvering verkligen stämmer, eller om något egentligen inte fungerar så som du antar. Detta kan du med fördel göra genom att placera brytpunkter på relevanta ställen. Även vid användning av en debugger kan du ha stor nytta av att göra `println` av intressanta uttryck för att i detalj undersöka vad som egentligen händer. Läs mer om debugging i Appendix D.

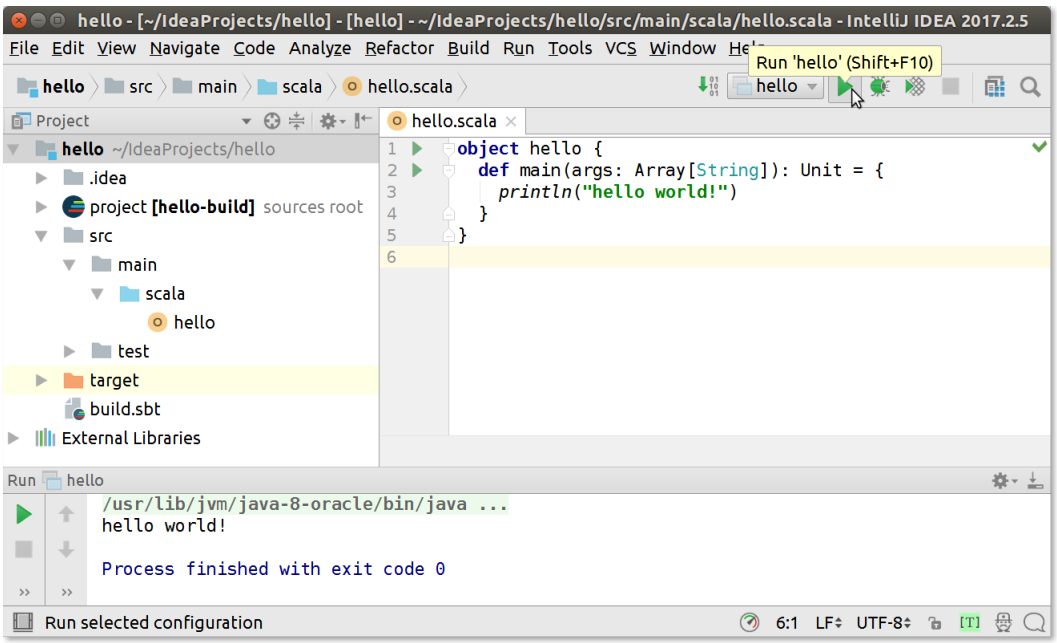
H.3 JetBrains IntelliJ IDEA med Scala-plugin

IntelliJ IDEA[4] är en professionell IDE som stödjer många olika programmeringsspråk. IntelliJ är skriven i Java och utvecklas av det tjeckiska företaget JetBrains.

IntelliJ IDEA finns i två varianter: en gratis gemenskapsvariant med öppenkällkodslicens (eng. *Community edition*), samt en betalvariant med sluten källkod och support-tjänster.

IntelliJ IDEA är en omfattande och avancerad programmeringsmiljö med många funktioner och inställningar. Det finns även en omfattande uppsättning insticksmodu-

[4] en.wikipedia.org/wiki/IntelliJ_IDEA



Figur H.4: Den integrerade utvecklingsmiljön Intellij IDEA.

ler och tilläggsprogram som underlättar utveckling av t.ex. mobilappar, webbprogram, databaser och mycket annat.

Till IntelliJ IDEA finns en insticksmodul (eng. *plug-in*) som stöd för Scala med tillhörande standardbibliotek och byggverktyget `sbt`, med mera. Scala-insticksmodulen kan inkluderas genom att välja Scala i en av de dialoger som visas vid första körningen, enligt instruktioner nedan.

I detta avsnitt ges länkar till installation samt tips om hur du kommer igång med att använda IntelliJ IDEA med Scala. Det går ganska snabbt att lära sig grunderna, men det kräven en viss ansträngning att lära sig de mer avancerade funktionerna. Det finns omfattande resurser på nätet som hjälper dig vidare.

Google tillkännagav 2013 att företaget övergår från Eclipse till IntelliJ som den officiellt understödda utvecklingsmiljön för Android och 2014 lanserades utvecklingsmiljön Android Studio[5] som bygger vidare på IntelliJ.

H.3.1 Installera IntelliJ IDEA

IntelliJ med Scala-plug-in är förinstallerat på LTH:s datorer och startas med kommandot `idea` i ett terminalfönster.

Du kan installera IntelliJ på din egen dator genom att följa instruktionerna för ditt operativsystem (Windows/macOS/Linux) här:
`https://www.jetbrains.com/help/idea/run-for-the-first-time.html`

Du behöver Scala-plugin som du kan välja under installationen av IntelliJ, men det går också att installera plugin för Scala i efterhand, se vidare här:
`https://www.jetbrains.com/help/idea/discover-intellij-idea-for-scala.html`

[5] en.wikipedia.org/wiki/Android_Studio

Appendix J

Introduktion till Java

J.1 Teori

J.1.1 Övning `scalajava` och labb `javatext`

Valfria uppgifter som ger dig en **flygande start** i efterföljande kurs:

- Övning `scalajava`:
  - Översätta från Java till Scala och från Scala till Java
  - Undersöka autoboxning (eng. *autoboxing*)
  - Använda `import scala.jdk.CollectionConverters.*`
- Laboration `javatext`:
  - Gör ett textspel för terminalen huvudsakligen i Java men vissa delar i Scala, enligt krav, tips och inspiration i labb-instruktionerna.
- OBS! Scala CLI och sbt kan blanda `.scala` och `.java` i samma projekt.

```
> scala run .
```

Ovan kommando kommer också kompilera `.java`-filer.
Observera Javas filnamnsregler: klass == filnamn, paket == katalog
Bra plats att lägga `.java`-filer: `src/main/java/`
Bra plats att lägga `.scala`-filer: `src/main/scala/`

J.1.2 ”Hello world!” i Java.

Ett minimalt huvudprogram i Java:

```java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

J.1.3 Testa Java i `jshell`

- Java har en motsvarighet till Scalas REPL: kommandot `jshell`

```
> jshell
|  Welcome to JShell -- Version 11.0.11
|  For an introduction type: /help intro

jshell> /help intro
|
|                                  intro
|                                  =====
|
|  The jshell tool allows you to execute Java code, getting immediate results.
|  You can enter a Java definition (variable, method, class, etc), like:  int x = 8
|  or a Java expression, like:  x + x
```

```
|  or a Java statement or import.
|  These little chunks of Java code are called 'snippets'.
|
|  There are also the jshell tool commands that allow you to understand and
|  control what you are doing, like:  /list
|
|  For a list of commands: /help

jshell>
```

J.1.4 Grundläggande likheter och skillnader Java–Scala

Några likheter:

- Kompilerar till bytekod som kör på JVM på många olika plattformar.
- Statisk typning: ger snabb maskinkod, kompilatorn kan ge stöd vid förändring av kod (s.k. refactoring) och hittar många buggar redan vid kompilering.

Liknande men viss skillnad:

Java

- **Objektorientering**, men inte ”äkta” (eng. *pure*) eftersom inte alla värden är objekt
- Primitiva typer är inte objekt; representeras effektivt, normalt **utan boxning**
- Visst stöd för **funktionsprogrammering**

Scala

- **Äkta objektorienterat** eftersom alla värden är objekt, även funktioner
- `AnyVal`-instanser är äkta objekt men representeras ändå effektivt, normalt **utan boxning**
- Omfattande stöd för **funktionsprogrammering**

J.1.5 Huvudprogram i Scala och Java

Scala 2

```
object Main {
  def main(args: Array[String]): Unit = {
    println("Hello!")
  }
}
```

Java

```
public class JMain {
  public static void main(String[] args){
    System.out.println("Hello!");
  }
}
```

Scala 3

```
@main
def sumFirst(n: Int, xs: Int*): Unit = println(xs.take(n).sum)
```

J.1.6 Loopa genom argumenten i ett Java-huvudprogram

```
> code HelloJavaArgs.java
```

```
public class HelloJavaArgs {
    public static void main(String[] args) {
    int i = 0;
    while (i < args.length) {
      System.out.println(args[i]);
      i = i + 1;
    }
  }
}
```

Kompilera och kör:

```
> javac HelloJavaArgs.java
> java HelloJavaArgs hej gurka tomat
hej
gurka
tomat
```

J.1.7 HIGHSCORE implementerad i Java

```
import java.util.Scanner;

public class HighScore {
    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        System.out.println("Hur många poäng fick du?");
        int points =  scan.nextInt();
        System.out.println("Vad var highscore före senaste spelet?");
        int highscore = scan.nextInt();
        if (points > highscore) {
            System.out.println("GRATTIS!");
        } else {
            System.out.println("Försök igen!");
        }
    }
}
```

J.1.8 Några saker som finns i Scala men inte i Java

- `case`-klasser
- Lokala funktioner

- Metoder som operatorer
- Infix operatornotation
- Defaultargument
- Namngivna argument
- Engångsinitialisering: `val`
- Fördröjd initialisering: `lazy val`
- Enhetlig access för `def`, `val`, `var`
- Egna setters med `def namn_=`
- Namnanrop, fördröjd evaluering
- Matchning, mönster och garder
- Klassparametrar, primärkonstruktor
- Singelobjekt: `object`
- Kompanjonsobjekt
- Inmixning: `trait`
- `for-yield`-uttryck
- Block är uttryck; slipper `return`
- Tomma värdet () av typen `Unit`
- `Option`, `Some`, `None` (Java har `Optional` som ger en del, men inte allt...)
- `Try`, `Success`, `Failure`
- Samlingarna i Scalas standardbibliotek, speciellt de **oföränderliga** samlingarna `Vector`, `Map`, `Set`, `List`, etc.
- Innehållslikhet med == för oföränderliga strukturer, inkl. `< <= > >=` på strängar
- **Enhetlig** användning av samlingar **inkl. Array** (förutom innehållslikhet för Array)
- Kontextuella abstraktioner `given using`
- Mer precis synlighet `private[mypack]`
- Namnändring vid `import`
- Flexibel filstruktur och filnamngivning
- Flexibel nästling av klasser, objekt, traits
- Typ-alias och abstrakta typer med `type`
- Extensionsmetoder `extension`
- ...

J.1.9 Några saker som finns i Java men inte i Scala

+ Snabbare kompilering
+ Mognare verktygsstöd
- Variabeldeklaration utan initialisering
- Förändringsbara parametrar
- C-liknande prefix- och postfix-inkrementering och -dekrementering: `i++ ++i i-- --i`
- C-liknande `for`-sats
- Semikolon krävs efter alla satser
- `return` krävs i alla metoder som har returvärde
- Nyckelordet `void`
- Parenteser efter alla metoder
- Specialsyntax för indexering av array `[]` ej som i andra samlingar

- Hoppa ut ur loop med `break`
  docs.oracle.com/javase/tutorial/
- `switch` ”faller igenom” om du glömmer skriva `break`
- Kontrollerade undantag (eng. *checked exceptions*) och `throws`
- ...

J.1.10 Begränsningar med funktionsprogrammering i Java

Av alla dessa funktionsprogrammeringskoncept i Scala...

- **överlagring**
- **anonyma funktioner**
- **mönstermatchning**
- funktioner etc. på toppnivå
- utelämna tom parameterlista (enhetlig access)
- defaultargument
- namngivna argument
- lokala funktioner
- funktioner som äkta värden
- klammerparentes vid ensam parameter
- multipla parameterlistor
- egendefinierade kontrollstrukturer
- namnanrop (fördröjd evaluering)
- stegade funktioner (”Curry-funktioner”)
- fångad variablelrymd i funktionsobjekt (”closure”)
- ad hoc polymorfism (”typklasser”)
- kontextuella abstraktioner

...kan man i Java endast göra: **överlagring** (”overloading”), **anonyma funktioner** (”lambda”), **mönstermatchning** (de senare två har starka begränsningar)

Läs mer om Java här:
`https://en.wikipedia.org/wiki/Java_version_history` `https://en.wikipedia.org/wiki/Anonymous_function#Java_Limitations`

J.1.11 Grundtyper i Scala och primitiva typer Java

Grundtyp i **Scala**	Antal bitar	Omfång minsta/största värde	**primitiv typ** i **Java**
`Byte`	8	$-2^7 \ldots 2^7 - 1$	`byte`
`Short`	16	$-2^{15} \ldots 2^{15} - 1$	`short`
`Char`	16	$0 \ldots 2^{16} - 1$	`char`
`Int`	32	$-2^{31} \ldots 2^{31} - 1$	`int`
`Long`	64	$-2^{63} \ldots 2^{63} - 1$	`long`
`Float`	32	$\pm\ 3.4028235 \cdot 10^{38}$	`float`
`Double`	64	$\pm\ 1.7976931348623157 \cdot 10^{308}$	`double`

J.1.12 Javas switch-sats

De flesta C-liknande språk (men inte Scala) har en **`switch`**-sats som man kan använda istället för (vissa) nästlade if-else-satser:

```
import java.util.Scanner;

public class Switch {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv grönsak:");
        String g = scan.next();
        switch (g) {
        case "gurka":
            System.out.println("gott!");
            break;
        case "tomat":
            System.out.println("gott!");
            break;
        case "broccoli":
            System.out.println("ganska gott...");
            break;
        default:
            System.out.println("mindre gott...");
            break;
        }
    }
}
```

`switch` från Java 21 har begränsad mönstermatchning https://docs.oracle.com/en/java/javase/21/language/pattern-matching.html

---

J.1.13 Javas switch-sats utan break

Saknad **break**-sats ”faller igenom” till efterföljande gren:

```
import java.util.Scanner;

public class SwitchNoBreak {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv grönsak:");
        String g = scan.next();
        switch (g) {
        case "gurka":
        case "tomat":
            System.out.println("gott!");
            break;
        case "broccoli":
            System.out.println("ganska gott...");
            break;
        default:
            System.out.println(" mindre gott...");
            break;
        }
    }
}
```

En glömd **break** kan ge svårhittad bugg...

J.1.14 Javas switch-sats med glömd break

```
import java.util.Scanner;

public class SwitchForgotBreak {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv grönsak:");
        String g = scan.next();
        switch (g) {
        case "gurka":
            System.out.println("gott!");
        case "tomat":
            System.out.println("mycket gott!");
            break;
        case "broccoli":
            System.out.println("ganska gott...");
            break;
        default:
            System.out.println("mindre gott...");
            break;
        }
    }
```

```
}
```

```
> java SwitchForgotBreak
Skriv grönsak:
gurka
gott!
mycket gott!
```

J.1.15 Syntax för variabeldeklaration i Scala och Java

Exempel på variabeldeklarationer i

Scala

```
var i1: Int = 0
var i2 = 0
var i3 = (i2: Int) + 0
var p1: Point = new Point(0, 0)
var p2 = new Point(0, 0)
var (x, y) = (0, 0)
val a = 0
final val Constant = 42
```

- i2 härledd typ; går ej i Java 8,9 men finns med `var` i Java 10
- i3 typ varhelst i uttryck; går ej i Java
- (x, y) mönster i init; går ej i Java
- `val` ger ”engångsinit”; ingen exakt motsvarighet i Java men `final` kan ofta användas i stället

Java

```
int i1 = 0;
var i2 = 0; // från Java 10
int i4;
Point p1 = new Point(0, 0);
var p2 = new Point(0, 0);
final int CONSTANT = 42;
```

- i4 ej explicit init; går ej i Scala

J.1.16 For-sats i Scala och Java

Scala

```
val s = "Abbasillen"

// Loopa över index framlänges:

for i <- 0 until s.length do
  println(s(i))


// Loopa över index baklänges:

for i <- s.length-1 to 0 by -1 do
  println(s(i))
```

I Scala är `s.indices` att föredra!

Java

```
String s = "Abbasillen";

// Loopa över index framlänges:

for (int i = 0; i < s.length(); i++) {
    System.out.println(s.charAt(i));
}

// Loopa över index baklänges:

for (int i = s.length()-1; i >= 0; i--) {
    System.out.println(s.charAt(i));
}
```

J.1.17 For-sats i Scala med indices

Scala

```scala
val s = "Abbasillen"

// Loopa över index framlänges:

for i <- s.indices do
  println(s(i))


// Loopa över index baklänges:

for i <- s.indices.reverse do
  println(s(i))
```

Java

```java
String s = "Abbasillen";

// Loopa över index framlänges:

for (int i = 0; i < s.length(); i++) {
    System.out.println(s.charAt(i));
}

// Loopa över index baklänges:

for (int i = s.length()-1; i >= 0; i--) {
    System.out.println(s.charAt(i));
}
```

J.1.18 For-satser och arrayer i Java

En for-sats i Java har följande struktur:

```java
for (initialisering; slutvillkor; inkrementering) {
    sats1;
    sats2;
    ...
}
```

En primitiv heltals-array deklareras så här i Java:

```java
int[] xs = new int[42];  // rymmer 42 st heltal, init 0:or
int[] ys = {10, 42, -1}; // initera med 3 st heltal
```

Exempel på for-sats: fyll en array med 1:or

```java
for (int i = 0; i < xs.length; i++){
  xs[i] = 1;    // indexera med [i]
}
```

J.1.19 Implementation av SEQ-COPY i Java med `for`-sats

```java
public class SeqCopyForJava {

    public static int[] arrayCopy(int[] xs){
        int[] result = new int[xs.length];
        for (int i = 0; i < xs.length; i++){
            result[i] = xs[i];
        }
        return result;
    }

    public static String test(){
        int[] xs = {1, 2, 3, 4, 42};
        int[] ys = arrayCopy(xs);
        for (int i = 0; i < xs.length; i++){
            if (xs[i] != ys[i]) {
                return "FAILED!";
            }
        }
        return "OK!";
    }

    public static void main(String[] args) {
        System.out.println(test());
    }
}
```

Lite syntax och semantik för Java:

- En Java-klass med enbart statiska medlemmar motsvarar ett singelobjekt i Scala.
- Typen kommer **före** namnet.
- Man **måste** skriva `return`.
- Man **måste** ha semikolon efter varje sats.
- Metodnamn **måste** följas av parenteser; om inga parametrar finns används `()`
- En array i Java är inget vanligt objekt, men har ett ”attribut” `length` som ger antal element.
- **Övning**: skriv om med Javas `while`-sats i stället.

J.1.20 Element för element med speciell for-each-sats i Java

Scala

```scala
val s = "Abbasillen"

// Loopa över alla tecken:

for ch <- s do println(ch)
```

Java

```java
String s = "Abbasillen";

// Loopa över alla tecken:

for (char ch: s.toCharArray()) {
  System.out.println(ch);
}
```

`s.foreach(println)` går ej i Java men från Java 8 finns metoden `chars` som ger en `IntStream` och då kan man:
`str.chars().forEachOrdered(i -> System.out.println((char) i));`

J.1.21 Typisk utformning av Java-klass

Typisk ”anatomi” hos en Java-klass:

```
public class Klassnamn {
    attribut, normalt privata
    konstruktorer, normalt publika
    metoder: publika getters, och vid förändringsbara objekt även setters
    metoder: privata abstraktioner för internt bruk
    metoder: publika abstraktioner tänkta att användas av klientkoden
}
```

J.1.22 Statiska medlemmar i Java

- Man kan **inte** deklarera explicita singelobjekt i Java och det finns inget nyckelord `object`.
- I stället kan man deklarera **statiska medlemmar** i en klass med Java-nyckelordet `static`.
- Exempel på hur vi ska göra detta inuti en klassen `JPerson`:

```
public static final int ADULT_AGE = 18;
```

- Effekten blir den samma som ett singelobjekt i Scala:
  - Alla statiska medlemmar i en Java-klass allokeras automatiskt och hamnar i en egen singulär ”klassinstans” som existerar oberoende av de dynamiska instanserna.
  - De statiska medlemmarna accessas med punktnotation genom klassnamnet, **utan** `new`:

```
System.out.println(JPerson.ADULT_AGE);
```

J.1.23 Exempel: oföränderlig klass i Scala och Java

Scala:

```
class Person(val name: String, val age: Int):
  def isAdult = age >= Person.AdultAge

object Person:
  val AdultAge = 18
```

Java:

```
public class JPerson {
    private String name;
    private int age;
    public static final int ADULT_AGE = 18;

    public JPerson(String name, int age) {
      this.name = name;
      this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
```

```
    public boolean isAdult() {
        return age >= ADULT_AGE;
    }
}
```

Lär dig detta mönster för en typisk Java-klass utantill så du snabbt får grejerna på plats!

Övning:
Gör `Person` + `JPerson` **förändringsbara** så att namnet och åldern går att uppdatera och följande krav uppfylls:

- namnet ska ges vid konstruktion,
- åldern ska initieras till 0 vid konstr.,
- åldern ska aldrig kunna bli negativ.

J.1.24 Exempel: Scala-klassen Complex

```
class Complex(val re: Double, val im: Double):
  def r  = math.hypot(re, im)
  def fi = math.atan2(im, re)
  def +(other: Complex) = new Complex(re + other.re, im + other.im)
  override def toString = s"$re + $im${Complex.imSymbol}"

object Complex:
  var imSymbol = 'i'
```

Scala: https://github.com/lunduniversity/introprog/blob/master/compendium/examples/complex7.scala
Java: https://github.com/lunduniversity/introprog/blob/master/compendium/examples/JComplex.scala

---

J.1.25 Exempel: Motsvarande Java-klassen JComplex

```
1  public class JComplex { // man kan ej deklarera klassparametrar i Java
2      private double re;  // initialiseras i konstruktorn nedan
3      private double im;  // initialiseras i konstruktorn nedan
4      public static char imSymbol = 'i'; // publikt förändringsbart attribut (ovanligt i Java)
5
6      public JComplex(double real, double imag){ // konstruktor, körs vid new
7          re = real;
8          im = imag;
9      }
10
11 // en "getter" som ger attributvärdet, hindrar förändring av re
12     public double getRe(){
```

```
13          return re;
14      }
15
16  // ej bruklig formattering i Java, så metoder blir minst 3 rader
17      public double getIm(){ return im; }
18
19      public double getR(){
20          return Math.hypot(re, im);       // Math med stort M i Java
21      }
22
23      public double getFi(){
24          return Math.atan2(re, im);
25      }
26
27  // Javametodnamn får ej ha operatortecken t.ex. +, därav namnet add
28      public JComplex add(JComplex other){
29          return new JComplex(re + other.getRe(), im + other.getIm());
30      }
31
32      @Override public String toString(){
33          return re + " + " + im + imSymbol;
34      }
35  }
```

J.1.26 Exempel: Använda JComplex i Scala-kod

```
1  $ javac JComplex.java
2  $ scala
3  Welcome to Scala 2.12.9 (OpenJDK 64-Bit Server VM, Java 1.8.0_222).
4  Type in expressions for evaluation. Or try :help.
5
6  scala> val jc1 = JComplex(3, 4)
7  jc1: JComplex = 3.0 + 4.0i
8
9  scala> val polarForm = (jc1.getR, jc1.getFi)
10 polarForm: (Double, Double) = (5.0,0.6435011087932844)
11
12 scala> val jc2 = JComplex(1, 2)
13 jc2: JComplex = 1.0 + 2.0i
14
15 scala> jc1 add jc2
16 res0: JComplex = 4.0 + 6.0i
```

J.1.27 Exempel: Använda JComplex i Java-kod

```
public class JComplexTest {
```

```
    public static void main(String[] args){
        JComplex jc1 = new JComplex(3,4);
        String polar = "(" + jc1.getR() + ", " + jc1.getFi() + ")";
        System.out.println("Polär form: " + polar);
        JComplex jc2 = new JComplex(1,2);
        System.out.println(jc1.add(jc2));
    }
}
```

- I Java måste man skriva **new**.
- I Java måste man skriva **tomma parentes-par** efter metodnamnet vid **anrop av parameterlösa metoder**.
- **Tupler finns inte** i Java, så det går inte på ett enkelt sätt att skapa par av värden som i Scala; ovan görs polär form till en sträng för utskrift.
- **Operatornotation för metoder finns inte** i Java, så man måste i Java använda punktnotation och skriva: `jc1.add(jc2)`

J.1.28 Exempel: Förändringsbar klass i Scala och Java

Scala:

```
class MutablePerson(var name: String):
  private var _age = 0

  def age: Int = _age

  def age_=(a: Int): Unit =
    if (a >= 0) _age = a else _age = 0
      // eller hellre kasta undantag?

  def isAdult: Boolean =
    age >= MutablePerson.AdultAge

object MutablePerson:
  val AdultAge = 18
```

Java:

```
public class JMutablePerson {
    private String name;
    private int age = 0;
    public static final int ADULT_AGE = 18;

    public JMutablePerson(String name) {
      this.name = name;
    }
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        if (age >= 0) {
          this.age = age;
        } else {
          this.age = 0;
        }
    }

    public boolean isAdult() {
        return age >= ADULT_AGE;
    }
}
```

J.1.29 Scalas ”case-klass-godis” finns inte i Java

En oföränderlig datatyp implementeras i **Scala** helst som en `case`-klass:

```
case class Person(name: String, age: Int):
  def isAdult = age >= Person.AdultAge

object Person:
  val AdultAge = 18
```

En oföränderlig datatyp i **Java** med **motsvarande** funktionalitet kräver egen implementation av dessa metoder:

- en getter för varje attribut
- `equals`
- `hashCode` (förklaras i forts.kurs)
- `apply`
  (men man kallar nog den `create` el. likn.; namnet måste ju skrivas)
- `toString`
- `copy`
  (men det finns ju inte namngivna parametrar och default-argument så denna blir osmidig)
- `unapply`
  (men det finns ju inte mönstermatchning så denna struntar man nog i)

J.1.30 Repetition: Den primitiva typen Array i JVM

- Primitiva arrayer (`Array` i Scala, `[]` i Java) har **fördelar**:[1]
  - Det är den snabbaste indexerbara datastrukturen i JVM: att läsa och uppdatera ett element på en viss plats är mycket effektivt om man vet platsens index.
  - Fungerar lika bra med både primitiva värden och objektreferenser
- ... men också **nackdelar**:
  - Man måste bestämma sig för antalet element som ska allokeras när man gör `new`.
  - Man kan ta i lite extra när man allokerar om man behöver plats för fler senare, men då måste man hålla reda på hur många platser man använder och veta var nästa lediga plats finns.
  - Det är krångligt att stoppa in (eng. *insert*) och ta bort (eng. *delete*) element.
  - Vill man ha fler platser måste man allokera en helt ny, större array och kopiera över alla befintliga element.

J.1.31 Syntax för Array i Scala och Java

[1] stackoverflow.com/questions/2843928/benefits-of-arrays

Scala

```
var xs = Array(42, 43, 44)




val n = xs.length

var strings = new Array[String](42)
// eller      Array.ofDim[String](42)
// eller      Array.fill(42)(null: String)

strings(0) = "first"

strings(1) = "second"
```

Java

```
int[] xs = new int[]{42, 43, 44};

// samma som ovan, men kortare:
int[] xs2 = {42, 43, 44};

int n = xs.length;  // EJ length()

String[] strings = new String[42];



strings[0] = "first";

strings[1] = "second";
```

J.1.32 Exempel: Polygon med primitiv array i Java

```
1  public class Polygon {
2      private Point[] vertices; // array med hörnpunkter
3      private int n;            // antalet hörnpunkter
4
5      /** Skapar en polygon */
6      public Polygon() {
7          vertices = new Point[1];
8          n = 0;
9      }
10
11     ...
```

J.1.33 Polygon med primitiv array i Java: stoppa in sist och vid behov skapa mer plats

Implementera:

```
private void extend()                // dubbla storleken
public void addVertex(int x, int y)  // lägg till hörnpunkt
```

```
1      private void extend(){
2          Point[] oldVertices = vertices;
3          vertices = new Point[2 * vertices.length]; // skapa dubbel plats
4          for (int i = 0; i < oldVertices.length; i++) {  // kopiera
5              vertices[i] = oldVertices[i];
6          }
7      }
8
```

```
9     public void addVertex(int x, int y) {
10        if (n == vertices.length) extend();
11        vertices[n] = new Point(x, y);
12        n++;
13    }
```

J.1.34 Polygon med primitiv array i Java: stoppa in mitt i på angiven plats

Implementera:

```
/** Sätt in hörnpunkt på plats pos */
public void insertVertex(int pos, int x, int y)
```

```
1     public void insertVertex(int pos, int x, int y) {
2         if (n == vertices.length) extend();   // utöka vid behov
3         for (int i = n; i > pos; i--) {       // flytta element bakifrån
4             vertices[i] = vertices[i - 1];
5         }
6         vertices[pos] = new Point(x, y);
7         n++;
8     }
```

J.1.35 Scanna filer och strängar med `java.util.Scanner`

- I Scala kan man läsa från fil så här (se quickref sid 3 längst ner):

```
val names = scala.io.Source.fromFile("src/names.txt").getLines.toVector
```

- Klassen `java.util.Scanner` kan också läsa från fil (se Java Snabbref sid 4):

```
def readFromFile(fileName: String): Vector[String] = {
  val file = new java.io.File(fileName)
  val scan = new java.util.Scanner(file)
  val buffer = scala.collection.mutable.ArrayBuffer.empty[String]
  while (scan.hasNext) {
    buffer += scan.next
  }
  scan.close
  buffer.toVector
}
```

- Med `new java.util.Scanner(System.in)` kan man även scanna tangentbordet.
- Med `new java.util.Scanner("hej 42")` kan man även scanna en sträng.
- Scanna `Int` och `Double` med metoderna `nextInt` och `nextDouble`.
  Se doc: docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

---

J.1.36 Exempel: Scanner

```
scala> val scan = new java.util.Scanner("hej 42 42.0   42 slut")

scala> scan.hasNext
res0: Boolean = true

scala> scan.hasNextInt
res1: Boolean = false

scala> scan.next
res2: String = hej

scala> scan.hasNextInt
res3: Boolean = true

scala> scan.nextInt
res4: Int = 42

scala> while (scan.hasNext) println(scan.next)
42.0
42
slut
```

J.1.37 Använda Java-samlingar i Scala med `CollectionConverters`

Med hjälp av **`import`** `scala.jdk.CollectionConverters.*`
får du smidig **interoperabilitet** med Java och dess standardbibliotek,
speciellt metoderna **`asJava`** och **`asScala`**:

```
scala> import scala.jdk.CollectionConverters.*

scala> Vector(1,2,3).asJava
res0: java.util.List[Int] = [1, 2, 3]

scala> val xs = new java.util.ArrayList[String]()
xs: java.util.ArrayList[String] = []

scala> xs.add("hej")
res1: Boolean = true

scala> xs.asScala
res2: scala.collection.mutable.Buffer[String] = Buffer(hej)
```

Läs mer här:
https://docs.scala-lang.org/overviews/collections-2.13/conversions-between-java-and-scala-collections.html

---

J.1.38 Generiska samlingar i Java

- Från och med version 5 av Java (2004) så introducerades **generics** vilket möjliggör skapandet av klasser som kan erbjuda generell behandling av olika typer

av objekt.
- Generiska klasser i Java känns igen med syntaxen `Klassnamn<Typ>`, till exempel `ArrayList<Point>`
- Fördjupning: docs.oracle.com/javase/tutorial/extra/generics/intro.html, mer om detta i fördjupningskursen.

J.1.39 Om ArrayList i Java

`java.util.ArrayList` liknar `scala.collection.mutable.ArrayBuffer` som båda har dessa fördelar:

- Lagrar sina element internt i snabbindexerade primitiva arrayer.
- Fungerar för alla typer av objekt.
- Utökar samlingens storlek av sig själv vid behov.

Det finns dock vissa nackdelar med `ArrayList` i Java
(som inte gäller för `ArrayBuffer` i Scala):

- Fungerar **inte** rakt av med primitiva typer `int`, `double`, `char`, ...
(men det finns sätt komma runt detta, tack vare s.k. wrapper-klasser och auto-boxning; mer om detta snart)
- Namnet `ArrayList` är inte helt lyckat, eftersom ordet ”lista” normalt används för länkade snarare än array-liknande strukturer.

J.1.40 Polygon med ArrayList i Java

Klassen `Polygon`, nu med ett attribut av typen `ArrayList<Point>`:

```
public class Polygon {
    private ArrayList<Point> vertices; // lista med hörnpunkter

    /** Skapar en polygon */
    public Polygon() {
        vertices = new ArrayList<Point>();
    }

    ...
```

Det behövs inget attribut `n` eftersom vi inte själva behöver hålla reda på antalet allokerade platser: allokering, insättning, och utökning av antalet platser sköts helt automatiskt av `ArrayList`-klassen vid behov.

J.1.41 Några viktiga operationer på ArrayList<E>

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/ArrayList.html

```
/** Tar reda på elementet på plats pos */
E get(int pos);

/** Lägger in objektet obj sist */
void add(E obj);

/** Lägger in obj på plats pos; efterföljande flyttas */
void add(int pos, E obj);

/** Tar bort elementet på plats pos och returnerar det */
E remove(int pos);

/** Tar reda på antalet element i listan */
int size();
```

Lär dig vad som finns om ArrayList i snabbreferensen för Java
Överkurs för den nyfikne: kolla implementation av ArrayList här:
https://hg.openjdk.org/jdk8/jdk8/jdk/file/tip/src/share/classes/java/util/ArrayList.java#l106

J.1.42 Övning ArrayList: new och add

Skriv Java-kod som skapar en lista med element av typen `Point` och lägger in tre punkter i listan med koordinaterna:
(50, 50), (50,10) och (30, 40).

Lösning:

```
ArrayList<Point> vertices = new ArrayList<Point>();
vertices.add(new Point(50, 50));
vertices.add(new Point(50, 10));
vertices.add(new Point(30, 40));
```

J.1.43 For-each-sats i Java:

- Antag att vi vill gå igenom alla element i en lista.

  ```
  ArrayList<String> words = new ArrayList<String>();
  ```

- Det finns två olika typer av `for`-satser i Java som kan göra detta:
  - Vanlig `for`-sats:

    ```
    for (int i = 0; i < words.size(); i++) {
        System.out.println(i + ": " + words.get(i));
    }
    ```

– Så kallad **for-each-sats** med denna syntax:
`for (Elementtyp element: samling) { ... }`
Exempel:

```
for (String s: words) {
    System.out.println(s);
}
```

Men vi får ingen indexvariabel då...

J.1.44 Polygon med ArrayList: metoderna blir enklare

```
public void addVertex(int x, int y) {
    vertices.add(new Point(x, y));
}

public void move(int dx, int dy) {
    for (Point p: vertices){
        p.move(dx, dy);
    }
}

public void insertVertex(int pos, int x, int y) {
    vertices.add(pos, new Point(x, y));
}

public void removeVertex(int pos) {
    vertices.remove(pos);
}
```

Se hela lösningen här: compendium/examples/scalajava/list/Polygon.java

---

J.1.45 Polygon med ArrayList: iterera över alla hörnpunkter i draw med indexering

```
public void draw(SimpleWindow w) {
    if (vertices.size() == 0) {
        return;
    }
    Point start = vertices.get(0);
    w.moveTo(start.getX(), start.getY());
    for (int i = 1; i < vertices.size(); i++) {
        w.lineTo(vertices.get(i).getX(),
                 vertices.get(i).getY());
    }
    w.lineTo(start.getX(), start.getY());
```

```
    }
```

Övning: Skriv om med for-each-sats.

J.1.46 Polygon med ArrayList: iterera över alla hörnpunkter i draw med foreach-sats

```
    public void draw(SimpleWindow w) {
        if (vertices.size() == 0) {
            return;
        }
        Point start = vertices.get(0);
        w.moveTo(start.getX(), start.getY());
        for (Point p: vertices){
            w.lineTo(p.getX(), p.getY());
        }
        w.lineTo(start.getX(), start.getY());
    }
```

Se hela lösningen här: compendium/examples/scalajava/list/Polygon.java

---

J.1.47 Övning ArrayList: implementera metoden hasVertex

Skriv kod som implementerar denna metod i klassen `Polygon`:

```
/** Undersöker om polygonen har någon hörnpunkt med koordinaterna x, y. */
public boolean hasVertex(int x, int y) {
    ???
}
```

J.1.48 Lösning ArrayList: implementera metoden hasVertex

```
    public boolean hasVertex(int x, int y) {
        for (Point p: vertices) {
            if (p.getX() == x && p.getY() == y) {
                return true;
            }
        }
        return false;
    }
```

J.1.49 For-each-sats med array

For-each-sats fungerar även med primitiv array:

```
String[] stringArray = {"hej", "på", "dej"};
for (String s: stringArray) {
    System.out.println(s);
}
```

J.1.50 Generiska klasser (t.ex. ArrayList) med primitiva typer

Detta går tyvärr **INTE** i Java:
~~`ArrayList<int> list = new ArrayList<int>();`~~

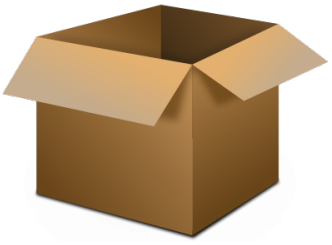
- Hur gör man om man vill ha heltalselement (eller andra primitiva värden) i en generisk samling?
- Javas lösning på problemet består av två delar:
  - Klasser som packar in primitiva typer, (eng. *wrapper classes*)
  - Speciella regler för implicita konverteringar, s.k. ”auto-boxing” (eng. *Boxing / Unboxing conversions*)

Ofta fungerar det fint, men det finns fallgropar.
(Om du är nyfiken på alla intrikata detaljer, se Java tutorial och Javaspecifikationen.)

J.1.51 Wrapper-klassen `Integer`

En skiss av klassen `Integer`
(ligger i paketet `java.lang` och importeras därmed implicit):

```
public class Integer {
    private int value;

    public static final MIN_VALUE = -2147483648;
    public static final MAX_VALUE = 2147483647;

    public Integer(int value) { // deprecated; will be private in future
        this.value = value;
    }

    public static valueOf(int value) {
      return new Integer(value)
    }

    public int intValue() {
        return value;
    }
    ...
}
```



Javadoc för klasen `Integer` finns här:

http://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html

---

J.1.52 Wrapper-klasser i `java.lang`

Primitiv typ	Inpackad typ
boolean	Boolean
byte	Byte
short	Short
char	Character
int	Integer
long	Long
float	Float
double	Double

J.1.53 Övning: primitiva versus inpackade typer

- Deklarera en variabel med namnet `gurka` av den primitiva heltalstypen och initiera den till värdet 42.
- Deklarera en referensvariabel med namnet `tomat` av den inpackade (”wrappade”) heltalstypen och initiera den till värdet 43.
- Rita hur det ser ut i minnet.

J.1.54 Exempel: Lista med heltal utan autoboxning

```
import java.util.ArrayList;
import java.util.Scanner;

public class TestIntegerList {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<Integer>();
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv heltal med blank emellan. Avsluta med <CTRL+D>");
        while (scan.hasNextInt()) {
            int nbr = scan.nextInt();
            Integer obj = Integer.valueOf(nbr);
            list.add(obj);
        }
        System.out.println("Dina heltal i omvänd ordning:");
        for (int i = list.size() - 1; i >= 0; i--) {
            Integer obj = list.get(i);
            int nbr = obj.intValue();
            System.out.println(nbr);
        }
    }
}
```

Koden finns här: compendium/examples/scalajava/TestIntegerList.java

J.1.55 Specialregler för wrapper-klasser

- Om ett `int`-värde förekommer där det behövs ett `Integer`-objekt, så lägger kompilatorn **automatiskt** ut kod som skapar ett `Integer`-objekt som packar in värdet.
- Om ett `Integer`-objekt förekommer där det behövs ett `int`-värde, lägger kompilatorn **automatiskt** ut kod som anropar metoden `intValue()`.

Samma gäller mellan alla primitiva typer och dess wrapper-klasser:

```
boolean  ⇔  Boolean
   byte  ⇔  Byte
  short  ⇔  Short
   char  ⇔  Character
    int  ⇔  Integer
   long  ⇔  Long
  float  ⇔  Float
 double  ⇔  Double
```

J.1.56 Exempel: Lista med heltal och autoboxning

```
import java.util.ArrayList;
import java.util.Scanner;

public class TestIntegerListAutoboxing {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<Integer>();
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv heltal med blank emellan. Avsluta med <CTRL+D>");
        while (scan.hasNextInt()) {
            int nbr = scan.nextInt();
            list.add(nbr);   // motsvarar: list.add(Integer.valueOf(nbr));
        }
        System.out.println("Dina heltal i omvänd ordning:");
        for (int i = list.size() - 1; i >= 0; i--) {
            int nbr = list.get(i);  // motsvarar: int nbr = list.get(i).intValue();
            System.out.println(nbr);
        }
    }
}
```

Koden finns här: scalajava/generics/TestIntegerListAutoboxing.java

J.1.57 Fallgropar vid autoboxning

- Jämförelser med == och `!=`
  compendium/examples/scalajava/generics/TestPitfall1.java

- Kompilatorn hittar inte förväxlad parameterordning, t.ex. `add(pos, item)` i fel ordning: ~~`add(item, pos)`~~
  compendium/examples/scalajava/generics/TestPitfall2.java

J.1.58 Referenslikhet eller innehållslikhet i Scala och Java

Det finns två **principiellt olika** sorters **likhet**:

- **Referenslikhet** (eng. *reference equality*): två referenser anses lika om de refererar till **samma instans** i minnet.
- **Innehållslikhet**, ä.k. strukturlikhet (eng. *structural equality*): två referenser anses lika om de refererar till objekt med **samma innehåll**.
- I Scala finns flera metoder som testar likhet:
  - metoden `eq` testar **referenslikhet** och `r1.eq(r2)` ger `true` om `r1` och `r2` refererar till **samma** instans.
  - metoden `ne` testar referens**o**likhet och `r1.ne(r2)` ger `true` om `r1` och `r2` refererar till **olika** instanser.
  - metoden == som anropar metoden `equals` som default testar referenslikhet men som **kan överskuggas** om man **själv vill bestämma** om det ska vara referenslikhet eller strukturlikhet.
- Scalas **standardbibliotek** och **grundtyperna** `Int`, `String` etc. testar **innehållslikhet** genom metoden ==
- I **Java** är det **annorlunda**: symbolen == är ingen metod i Java utan **specialsyntax** som vid instansjämförelse alltid testar **referenslikhet**, medan metoden `equals` kan överskuggas med valfri likhetstest.

J.1.59 Fallgrop med samlingar: metoden contains kräver implementation av equals

Antag att vi vill implementera `hasVertex()` i klassen `Polygon` genom att använda metoden `contains` på en lista. Hur gör vi då?

```
public boolean hasVertex(int x, int y) {
    return vertices.contains(new Point(x, y)); // FUNKAR INTE om ...
    // ... inte Point har en equals som kollar innehållslikhet
}
```

Vi behöver implementera metoden `equals(Object obj)` i klassen `Point` som kollar innehållslikhet och ersätter den `equals` som finns i `Object` som kollar referenslikhet, eftersom metoden `contains` i klassen `ArrayList` anropar `equals` när den letar igenom listan efter lika objekt.
Se exempel här: compendium/examples/scalajava/generics/TestPitfall3.java

Det krävs ofta även att man även ersätter `hashCode`, mer om det i fortsättningskursen.

J.1.60 Fullständigt recept för `equals`

För den nyfikne inför fortsättningskursen efter jul:

Läs om fallgropar för att implementera equals i **Java** här:
www.artima.com/lejava/articles/equality.html

Läs receptet för att implementera equals i **Scala** här:
www.artima.com/pins1ed/object-equality.html#28.4

J.1.61 Villkorsuttryck i Java

Det går att använda villkorsuttryck i Java, men med syntax från språket C:

Scala

```
var r = math.random()
var answer = if r > 0.5 then 42 else 0
```

Java

```
double r = Math.random();
int answer = (r > 0.5) ? 42 : 0;
```

J.1.62 Typtest och typkonvertering

Scala

```
var x = "hej"

var isString = x.isInstanceOf[String]

var y = 42

var z = y.asInstanceOf[Double]
```

Java

```
String x = "hej";

boolean isString = x instanceof String;

int y = 42;

double z = (double) y;
```

Detta görs ju i Scala bäst med `match` och typmönster!

J.1.63 Regler för överskuggning i Java

http://docs.oracle.com/javase/tutorial/java/IandI/override.html

---

J.1.64 Fånga undantag i Scala och Java

Typisk skillnad mellan Scala och Java:
konstruktioner som är **uttryck** i Scala är ofta **satser** i Java.

Scala

```
val a = try 2 / 0 catch
  case e: ArithmeticException => 0


val b = try 4 / 2 catch
  case e: ArithmeticException => 0
```

Java

```
int a;
try {
    a = 2 / 0;
} catch (ArithmeticException e) {
    a = 0;
}
```

```
int b;
try {
    b = 4 / 2;
} catch (ArithmeticException e) {
    b = 0;
}
```

J.1.65 Gränssnittet `List` i Java

- I Java finns inte **`trait`** och inmixning.
- I stället finns **`interface`** som liknar **`trait`** men är mer begränsad vad gäller vilka medlemmar som får finnas.
- Man kan bara göra **`extends`** på exakt en annan klass, men man kan i Java göra **`implements`** på flera **`interface`**.
- Exempel:

  ```
  public class ArrayList<E> extends AbstractList<E>
      implements List<E>, RandomAccess, Cloneable, java.io.Serializable
  ```

- Att implementera ett gränssnitt innebär att uppfylla ett kontrakt som utlovar att vissa speciella metoder finns tillgängliga.
- Gränssninttet `List` uppfylls av en av dess implementationer `ArrayList`

  på liknande sätt i Scala där gränssnittet `Seq` uppfylls av `Vector` etc.
  `List<String> xs = new ArrayList<String>();`
- Liknande exempel från övningen Hangman:
  `Set<Character> found = new HashSet<Character>();`
- En Scala-trait med enbart abstrakta medlemmar kompileras till ett Java-interface i JVM bytekode.
- Mer om gränssnitt i Java i fördjupningskursen.

J.1.66 Det går inte att skapa generisk Array i Java

- I Java kan man **inte** skapa en primitiv array av godtycklig typ enligt generisk typparameter: ~~`T[] xs = new T[42]`~~
- Man måste istället skapa en array av den mest generella referenstypen:
  `Object[] xs = new Object[42]`
  och sedan typtesta och typkonvertera under körtid; se t.ex. implementationen av `ArrayList` på rad 119: http://developer.classpath.org/doc/java/util/ArrayList-source.html

- Detta går faktiskt att göra i Scala med hjälp av `reflect.ClassTag` så här:

```
scala> def fyll[T](n: Int, x: T): Array[T] = Array.fill(n)(x)
```

```
-- Error:
1 |def fyll[T](n: Int, x: T): Array[T] = Array.fill(n)(x)
  |                                                   ^
  |  No ClassTag available for T

scala> def fyll[T: reflect.ClassTag](n: Int, x: T): Array[T] = Array.fill(n)(x)

scala> fyll(42, "hej")
res2: Array[String] = Array(hej, hej, hej, hej, hej, hej, hej, hej, hej, hej, hej
```

J.1.67 Jämföra strängar i Java

- I Java kan man **inte** jämföra strängar med operatorerna <, <=, >, och >=
- Dessutom ger operatorerna == och `!=` *inte* innehålls(o)likhet utan **referens(o)likhet** :(
- För innehållslikhet måste du använda metoderna `equals` och `compareTo`.
- `s1.compareTo(s2)` ger ett heltal som är:
  - `0` om s1 och s2 har samma innehåll
  - **negativt** om s1 < s2 i lexikografisk mening, alltså s1 ska sorteras **före**
  - **positivt** om s1 > s2 i lexikografisk mening, alltså s1 ska sorteras **efter**
- Undersök följande:

```
1 scala> new String("hej") eq new String("hej") // motsvarar == i Java
2 scala> "hej".equals("hej")                    // samma som == i Scala
3 scala> "hej".compareTo("hej")
4 scala> "hej".compareTo("HEJ")         // alla stora är 'före' alla små
5 scala> "HEJ".compareTo("hej")
```

- Ibland ger `"hej" == "hej"` förvånande nog värdet **`true`** i Java. Detta beror på en minnesoptimering som kallas stränginternalisering (eng. *string interning*) som ofta sker, men inte alltid. Det är därför en svårhittad bug!

docs.oracle.com/javase/8/docs/api/java/lang/String.html#compareTo-java.lang.String-

---

J.1.68 Jämföra strängar i Java: exempel

Vad skriver detta Java-program ut?

```
public class StringEqTest {
    public static void main(String[] args){
        boolean eqTest1 =
          (new String("hej")) == (new String("hej"));
        boolean eqTest2 =
          (new String("hej")).equals(new String("hej"));
        int eqTest3 =
          (new String("hej")).compareTo(new String("hej"));
        System.out.println(eqTest1);
        System.out.println(eqTest2);
```

```
        System.out.println(eqTest3);
    }
}
```

```
$ javac StringEqTest.java
$ java StringEqTest
false
true
0
```

J.2 Övning `java`

Mål

- ☐ Kunna förklara och beskriva viktiga skillnader mellan Scala och Java.
- ☐ Kunna översätta enkla algoritmer, klasser och singeltonobjekt från Scala till Java och vice versa.
- ☐ Känna till vad en case-klass innehåller i termer av en Javaklass.
- ☐ Kunna använda Javatyperna `List`, `ArrayList`, `Set`, `HashSet` och översätta till deras Scalamotsvarigheter med `CollectionConverters`.
- ☐ Kunna förklara hur autoboxning fungerar i Java, samt beskriva fördelar och fallgropar.

Förberedelser

- ☐ Studera teori i början av detta Appendix.

J.2.1 Grunduppgifter; förberedelse inför laboration

Uppgift 1. *Översätta metoder från Java till Scala.* I denna uppgift ska du översätta en Java-klass som används som en modul[2] och bara innehåller statiska metoder och inget förändringsbart tillstånd som kan ändras utifrån. (I nästa uppgift ska du sedan översätta klasser med förändringsbara tillstånd.)

Vi börjar med att göra översättningen från Java till Scala rad för rad och du ska behålla så mycket som möjligt av syntax och semantik så att Scala-koden blir så Java-lik som möjligt. I efterföljande deluppgift ska du sedan omforma översättningen så att Scala-koden blir mer idiomatisk[3].

a) Studera klassen `Hangman` nedan. Du ska översätta den från Java till Scala enlig de riktlinjer och tips som följer efter koden. Läs igenom alla riktlinjer och tips innan du börjar.

```
1  import java.net.URL;
2  import java.util.ArrayList;
3  import java.util.Set;
4  import java.util.HashSet;
5  import java.util.Scanner;
6
7  public class Hangman {
8      private static String[] hangman = new String[]{
9          " ======  ",
10         " |/   |  ",
11         " |    O  ",
12         " |   -|- ",
13         " |   / \\ ",
14         " |       ",
15         " |       ",
16         " ==========================   RIP  :("};
17
18     private static String renderHangman(int n){
```

[2] en.wikipedia.org/wiki/Modular_programming
[3] sv.wikipedia.org/wiki/Idiom_%28programmering%29

```
19          StringBuilder result = new StringBuilder();
20          for (int i = 0; i < n; i++){
21              result.append(hangman[i]);
22              if (i < n - 1) {
23                  result.append("\n");
24              }
25          }
26          return result.toString();
27      }
28
29      private static String hideSecret(String secret,
30                                       Set<Character> found){
31          String result = "";
32          for (int i = 0; i < secret.length(); i++) {
33              if (found.contains(secret.charAt(i))) {
34                  result += secret.charAt(i);
35              } else {
36                  result += '_';
37              }
38          }
39          return result;
40      }
41
42      private static boolean foundAll(String secret,
43                                      Set<Character> found){
44          boolean foundMissing = false;
45          int i = 0;
46          while (i < secret.length() && !foundMissing) {
47              foundMissing = !found.contains(secret.charAt(i));
48              i++;
49          }
50          return !foundMissing;
51      }
52
53      private static char makeGuess(){
54          Scanner scan = new Scanner(System.in);
55          String guess = "";
56          do {
57             System.out.println("Gissa ett tecken: ");
58             guess = scan.next();
59          } while (guess.length() != 1);
60          return Character.toLowerCase(guess.charAt(0));
61      }
62
63      public static String download(String address, String coding){
64          String result = "lackalänga";
65          try {
66              URL url = new URL(address);
67              ArrayList<String> words = new ArrayList<String>();
68              Scanner scan = new Scanner(url.openStream(), coding);
```

```
            while (scan.hasNext()) {
                words.add(scan.next());
            }
            int rnd = (int) (Math.random() * words.size());
            result = words.get(rnd);
        } catch (Exception e) {
            System.out.println("Error: " + e);
        }
        return result;
    }

    public static void play(String secret){
        Set<Character> found = new HashSet<Character>();
        int bad = 0;
        boolean won = false;
        while (bad < hangman.length && !won){
            System.out.print("\nFelgissningar: " + bad + "\t");
            System.out.println(hideSecret(secret, found));
            char guess = makeGuess();
            if (secret.indexOf(guess) >= 0) {
                found.add(guess);
            } else {
              bad++;
              System.out.println(renderHangman(bad));
            }
            won = foundAll(secret, found);
        }
        if (won) {
            System.out.println("BRA! :)");
        } else {
            System.out.println("Hängd! :(");
        }
        System.out.println("Rätt svar: " + secret);
        System.out.println("Antal felgissningar: " + bad);
    }

    public static void main(String[] args){
        if (args.length == 0) {
            String runeberg =
                "http://runeberg.org/words/ord.ortsnamn.posten";
            play(download(runeberg, "ISO-8859-1"));
        } else {
            int rnd = (int) (Math.random() * args.length);
            play(args[rnd]);
        }
    }
}
```

Riktlinjer och tips för översättningen:

1. Skriv Scala-koden med en texteditor i en fil som heter `hangman1.scala` och

kompilera med `scalac hangman1.scala` i terminalen; använd alltså *inte* en IDE, så som Eclipse eller IntelliJ, utan en ”vanlig” texteditor, t.ex. VS `code`.

2. Översätt i denna första deluppgift rad för rad så likt den ursprungliga Java-kodens utseende (syntax) som möjligt, med så få ändringar som möjligt. Du ska alltså ha kvar dessa Scalaovanligheter, även om det inte alls blir som man brukar skriva i Scala:
   (a) långa indrag,
   (b) onödiga semikolon,
   (c) onödiga `()`,
   (d) onödiga `{}`,
   (e) onödiga `System.out`, och
   (f) onödiga **`return`**.
3. Försök också i denna deluppgift göra så att betydelsen (semantiken) så långt som möjligt motsvarar den i Java, t.ex. genom att använda **`var`** överallt, även där man i Scala normalt använder **`val`**.
4. En Javaklass med bara statiska medlemmar motsvarar ett singeltonobjekt i Scala, alltså en **`object`**-deklaration innehållande ”vanliga” medlemmar.
5. För att tydliggöra att du använder Javas `Set` och `HashSet` i din Scala-kod, använd följande import-satser i `hangman1.scala`, som därmed döper om dina importerade namn och gör så att de inte krockar med Scalas inbyggda `Set`. Denna form av import går inte att göra i Java.

   ```
   import java.util.{Set => JSet};
   import java.util.{HashSet => JHashSet};
   ```

6. Javas `i++` fungerar inte i Scala; man får istället skriva `i += 1` eller mindre vanliga `i = i + 1`.
7. Typparametrar i Java skrivs inom <> medan Scalas syntax för typparametrar använder `[]`.
8. Till skillnad från Java så har Scalas metoddeklarationer ett tilldelningstecken = efter returtypen, före kroppen.
9. Du kan ladda ner Java-koden till `Hangman`-klassen nedan från kursens repo[4]. I samma bibliotek ligger även lösningarna till översättningen i Scala, men kolla *inte* på dessa förrän du gjort klart översättningarna och fått dem att kompilera och köra felfritt! Tanken är att du ska träna på att läsa felmeddelande från kompilatorn och åtgärda dem i en upprepad kompilera-testa-rätta-cykel.

b) Skapa en ny fil `hangman2.scala` som till att börja med innehåller en kopia av din direkt-översatta Java-kod från föregående deluppgift. Omforma koden så att den blir mer som man brukar skriva i Scala, alltså mer Scala-idiomatisk. Försök förenkla och förkorta så mycket du kan utan att göra avkall på läsbarheten.

Tips och riktlinjer:

1. Kalla Scala-objektet för `hangman`. När man använder ett Scalaobjekt som en modul (alltså en samling funktioner i en gemensam, avgränsad namnrymd) har man gärna liten begynnelsebokstav, i likhet med konventionen för paketnamn. Ett paket är ju också en slags modul och med en namngivningskonvention som är gemensam kan man senare, utan att behöva ändra koden som använder modulen, ändra från ett singelobjekt till ett paket och vice versa om man så önskar.

[4] github.com/lunduniversity/introprog/blob/master/compendium/examples/scalajava/Hangman.java

2. Gör alla metoder publikt tillgängliga och låt även strängvektorn `hangman` vara publikt tillgänglig. Deklarera `hangman` som en **`val`** och konstruera den med `Vector`. Eftersom `Vector` är oföränderlig och man inte kan ärva från singelobjekt och `hangman` är deklarerad med **`val`** finns inga speciella risker med att göra den konstanta vektorn publik om vi inte har något emot att annan kod kan läsa (och eventuellt göra sig beroende av) vår hänggubbetext.
3. I metoden `renderHangman`, använd `take` och `mkString`.
4. I metoden `hideSecret`, använd `map` i stället för en **`for`**-sats.
5. Det går att ersätta metoden `foundAll` med det kärnfulla uttrycket `(secret forall found)` där `secret` är en sträng och `found` är en mängd av tecken (undersök gärna i REPL hur detta fungerar). Skippa därför den metoden helt och använd det kortare uttrycket direkt.
6. I metoden `makeGuess`, i stället för `Scanner`, använd `scala.io.StdIn.readLine`.
7. Om du vill träna på att använda rekursion i stället för imperativa loopar: Gör metoden `makeGuess` rekursiv i stället för att använda **`do-while`**.
8. I metoden `download`, i stället för `java.net.URL` och `java.util.ArrayList`, använd `scala.io.Source.fromURL(address, coding).getLines.toVector` och gör en lokal import av `scala.io.Source.fromURL` överst i det block där den används. Det går inte att ha lokala **`import`**-satser i Java.
9. Låt metoden `download` returnera en `Option[String]` som i fallet att nedladdningen misslyckas returnerar `None`.
10. I metoden `download`, i stället för **`try-catch`** använd `scala.util.Try` och dess smidiga metod `toOption`.
11. Om du vill träna på att använda rekursion i stället för imperativa loopar: Använd, i stället för **`while`**-satsen i metoden `play`, en lokal rekursiv funktion med denna signatur:

    ```
    def loop(found: Set[Char], bad: Int): (Int, Boolean)
    ```

    Funktionen `loop` returnerar en 2-tupel med antalet felgissningar och **`true`** om man hittat alla bokstäver eller **`false`** om man blev hängd.

Uppgift 2. *Översätta mellan klasser i Scala och klasser i Java.* Klassen `Point` nedan är en modell av en punkt som kan sparas på begäran i en lista. Listan är privat för kompanjonsobjektet och kan skrivas ut med en metod `showSaved`. I koden används en `ArrayBuffer`, men i framtiden vill man, vid behov, kunna ändra från `ArrayBuffer` till en annan sekvenssamlingsimplementation, t.ex. `ListBuffer`, som uppfyller egenskaperna hos supertypen `Buffer`, men har andra prestandaegenskaper för olika operationer. Därför är attributet `saved` i kompanjonsobjektet deklarerat med den mer generella typen.

```
1  class Point(val x: Int, val y: Int, save: Boolean = false):
2    import Point.*
3
4    if save then saved.prepend(this)
5
6    def this() = this(0, 0)
7
8    def distanceTo(that: Point) = distanceBetween(this, that)
9
10   override def toString = s"Point($x, $y)"
11
```

```
12 object Point:
13   import scala.collection.mutable.{ArrayBuffer, Buffer}
14
15   private val saved: Buffer[Point] = ArrayBuffer.empty
16
17   def distanceBetween(p1: Point, p2: Point) =
18     math.hypot(p1.x - p2.x, p1.y - p2.y)
19
20   def showSaved: Unit =
21     println(saved.mkString("Saved: ", ", ", "\n"))
```

a) Översätt klassen `Point` ovan från Scala till Java. Vi ska i nästa deluppgift kompilera både Scala-programmet ovan och ditt motsvarande Java-program i terminalen och testa i REPL att klasserna har motsvarande funktionalitet.

Tips och riktlinjer:

1. För att namnen inte ska krocka i våra kommande tester, kalla Javatypen för `JPoint`.
2. I stället för Scalas `ArrayBuffer` och `Buffer`, använd Javas `ArrayList` och `List` som båda ligger i paketet `java.util`.
3. Undersök dokumentationen för `java.util.List` för att hitta en motsvarighet till `prepend` för att lägga till i början av listan.
4. I stället för default-argumentet i Scalas primärkonstruktor, använd en extra Java-konstruktor.
5. Det finns inga singelobjekt och inga kompanjonsobjekt i Java; istället kan man använda statiska klassmedlemmar. Placera kompanjonsobjektets medlemmars motsvarigheter *inuti* Java-klassen och gör dem till **`static`**-medlemmar.
6. Kod i klasskroppen i Scalaklassen, så som if-satsen på rad 4, placeras i lämplig konstruktor i Javaklassen.
7. Utskrifter med `print` och `println` behöver i Java föregås av `System.out`.
8. Det finns inget nyckelord **`override`** i Java, men en s.k. annotering som ger samma kompilatorhjälp. Den skrivs med ett snabel-a och stor begynnelsebokstav, så här: `@Override` före metoddeklarationen.
9. I Java används konventionen att börja getter-metoder med ordet `get`, t.ex. `getX()`.
10. Det finns ingen motsvarighet till `mkString` för `List` så du behöver själv gå igenom listan och hämta elementreferenser för utskrift med en **`for`**-loop. Notera att efter sista elementet ska radbrytning göras i utskriften och att inget komma ska skrivas ut efter sista elementet.
11. I Java behövs en ny **`import`**-deklaration om man vill importera ännu en typ från samma paket. Man kan även i Java använda asterisk $*$, (motsvarande _ i Scala), för att importera allt i ett paket, men då får man med alla möjliga namn och det vill man kanske inte.
12. Metoder i Java slutar med `()` om de saknar parametrar.
13. Alla satser i Java slutar med lättglömda semikolon. (Efter att man i skrivit mycket Javakod och växlar till Scalakod är det svårt att vänja sig av med att skriva semikolon...)

b) Starta REPL i samma bibliotek som du kompilerat kodfilerna. Testa så att klasserna `Point` och `JPoint` beter sig på samma vis enligt nedan. Skriv även testkod i REPL för att avläsa de attributvärden som har getters och undersök att allt funkar som det ska.

```
> scalac Point.scala
> javac JPoint.java
> scala
scala> val (p, jp) = (new Point, new JPoint)
scala> p.distanceTo(new Point(3, 4))
scala> Point.showSaved
scala> jp.distanceTo(new JPoint(3, 4))
scala> JPoint.showSaved
scala> for (i <- 1 to 10) { new Point(i, i, true) }
scala> Point.showSaved
scala> for (i <- 1 to 10) { new JPoint(i, i, true) }
scala> JPoint.showSaved
```

c) Översätt nedan Javaklass `JPerson` till en **`case class`** `Person` i Scala med motsvarande funktionalitet.

```
1  public class JPerson {
2      private final String name;
3      private final int age;
4
5      public JPerson(final String name, final int age){
6          this.name = name;
7          this.age = age;
8      }
9
10     public JPerson(final String name){
11         this(name, 0);
12     }
13
14     public String getName() {
15         return name;
16     }
17
18     public int getAge() {
19         return age;
20     }
21
22     public boolean canEqual(Object other) {
23         return (other instanceof JPerson);
24     }
25
26     @Override public boolean equals(Object other){
27         boolean result = false;
28         if (other instanceof JPerson) {
29             JPerson that = (JPerson) other;
30             result = that.canEqual(this) &&
31               this.getName() == that.getName() &&
32               this.getAge()  == that.getAge();
33         }
34         return result;
35     }
```

```
36
37      @Override public int hashCode() {
38        return name.hashCode() * 41 + age;
39      }
40
41      @Override public String toString() {
42        return "JPerson(" + name + ", " + age + ")";
43      }
44  }
```

✎ d) Undersök i REPL vilken funktionalitet i Scala-case-klassen `Person` som *inte* är implementerad i Java-klassen `JPerson` ovan. Skriv upp namnen på några av case-klassens extra metoder samt deras signatur genom att för en `Person`-instans, och för kompanjonsobjektet `Person`, trycka på TAB-tangenten. Prova några av de extra metoderna i REPL och förklara vad de gör.

```
1  scala> val p = Person("Björn", 49)
2  scala> p.      // tryck TAB en gång
3  scala> Person. // tryck TAB en gång
4  scala> p.copy  // tryck TAB en gång
5  scala> p.copy()
6  scala> p.copy(age = p.age + 1)
7  scala> Person.unapply(p)
```

Uppgift 3. *Oföränderlig Java-klass.* Översätt nedan Scala-klass till Java-klassen `JPoint3D`. Alla attribut ska vara privata (varför?). Översätt defaultargumentet till en alternativ konstruktor. Kalla getters för t.ex. `getX()`. Kör `javac` och testa i REPL.

```
class Point3D(val x: Int, val y: Int, val z: Int = 0)
```

Uppgift 4. *Förändringsbar Java-klass.*
Översätt nedan Scala-klass till Java-klassen `JMutablePoint3D`. Alla attribut ska vara privata (varför?). Översätt defaultargumentet till en alternativ konstruktor. Kalla setters för t.ex. `setX`. Kör `javac` och testa i REPL.

```
class MutablePoint3D(var x: Int, var y: Int, var z: Int = 0)
```

Uppgift 5. *Jämföra strängar i Java.* I Java kan man **inte** jämföra strängar med operatorerna <, <=, >, och >=. Dessutom ger operatorerna == och `!=` inte innehålls(o)likhet utan referens(o)likhet. Istället får man använda metoderna `equals` och `compareTo`, vilka också fungerar i Scala eftersom strängar i Scala och Java är av samma typ, nämligen `java.lang.String`.

a) Vad ger följande uttryck för värde?

```
1  scala> "hej".getClass.getTypeName
2  scala> "hej".equals("hej")
3  scala> "hej".compareTo("hej")
```

b) Studera dokumentationen för metoden `compareTo` i `java.lang.String`[5] och skriv minst 3 olika uttryck i Scala REPL som testar hur metoden fungerar i olika fall.

[5] docs.oracle.com/javase/8/docs/api/java/lang/String.html#compareTo-java.lang.String-

c) Studera dokumentationen `compareToIgnoreCase` [6] och skriv minst 3 olika strängutryck i Scala REPL som testar hur metoden fungerar i olika fall.

d) Vad skriver följande Java-program ut?

```
public class StringEqTest {
    public static void main(String[] args){
        boolean eqTest1 =
          (new String("hej")) == (new String("hej"));
        boolean eqTest2 =
          (new String("hej")).equals(new String("hej"));
        int eqTest3 =
          (new String("hej")).compareTo(new String("hej"));
        System.out.println(eqTest1);
        System.out.println(eqTest2);
        System.out.println(eqTest3);
    }
}
```

Uppgift 6. *Linjärsökning i Java.* Denna uppgift bygger vidare på uppgift 13 i kapitel 8. Du ska göra en variant på linjärsökning som innebär att leta upp första yatzy-raden i en matris där varje rad innehåller utfallet av 5 tärningskast.

a) Du ska lägga till metoderna `isYatzy` och `findFirstYatzyRow` i klassen `ArrayMatrix` i uppgift 13 i kapitel 8 enligt nedan skiss. Vi börjar med metoden `isYatzy` i denna deluppgift (nästa deluppgift handlar om `findFirstYatzyRow`). OBS! Det finns en bugg i `isYatzy` – rätta buggen och testa så att den fungerar.

```
    public static boolean isYatzy(int[] dice){ /* has one bug! */
        int col = 1;
        boolean allSimilar = true;
        while (col < dice.length && allSimilar) {
          allSimilar = dice[0] == dice[col];
        }
        return allSimilar;
    }

    /** Finds first yatzy row in m; returns -1 if not found */
    public static int findFirstYatzyRow(int[][] m){
        int row = 0;
        int result = -1;
        while (???) {
             /* linear search  */
        }
        return result;
    }
```

b) Implementera `findFirstYatzyRow`. Skapa först pseudo-kod för linjärsökningsalgoritmen innan du skriver implementationen i Java. Testa ditt program genom att lägga till följande rader i huvudprogrammet. Metoden `fillRnd` ingår i uppgift 13 i kapitel 8.

[6] docs.oracle.com/javase/8/docs/api/java/lang/String.html#compareToIgnoreCase-java.lang.String-

```
        int[][] yss = new int[2500][5];
        fillRnd(yss, 6);
        int i = findFirstYatzyRow(yss);
        System.out.println("First Yatzy Index: " + i);
```

Uppgift 7. *Jämförelsestöd i Java.* Java har motsvarigheter till Scalas `Ordering` och `Ordered`, som heter `java.util.Comparator` och `java.lang.Comparable`. I själva verket så är Scalas `Ordering` en subtyp till Javas `Comparator`, medan Scalas `Ordered` är en subtyp till Javas `Comparable`.

- Javas `Comparator` och Scalas `Ordering` används för att skapa fristående ordningar som kan jämföra *två olika* objekt. I Scala kan dessa göras implicit tillgängliga. I Javas samlingsbibliotek skickas instanser av `Comparator` med som explicita argument.
- Javas `Comparable` och Scalas `Ordered` används som supertyp för klasser som vill kunna jämföra ”sig själv” med andra objekt och har *en* naturlig ordningsdefinition.

a) Sök upp dokumentationen för `java.util.Comparator`. Vilken abstrakt metod måste implementeras och vad gör den?

b) I paketet `java.util.Arrays` finns en metod `sort` som tar en `Array[T]` och en `Comparable[T]`. Testa att använda dessa i REPL enligt nedan skiss. Starta om REPL så att ev. tidigare implicita ordningar för `Team` inte finns kvar.

```
scala> import java.util.Comparator
scala> val teamComparator = new Comparator[Team]{
         def compare(o1: Team, o2: Team) = ???
       }
scala> val xs =
         Array(Team("fnatic", 1499), Team("nip", 1473), Team("lumi", 1601))
scala> java.util.Arrays.sort(xs.toArray, teamComparator)
scala> xs
```

c) I Scala finns en behändig metod `Ordering.comparatorToOrdering` som skapar en implicit tillgänglig ordning om man har en `java.util.Comparator`. Testa detta enligt nedan i REPL, med deklarationerna från föregående deluppgift.

```
scala> implicit val teamOrd = Ordering.comparatorToOrdering(teamComparator)
scala> xs.sorted
```

d) Sök upp dokumentationen för `java.lang.Comparable`. Vilken abstrakt metod måste implementeras och vad gör den?

e) Gör så att klassen `Point` är `Comparable` och att punkter närmare origo sorteras före punkter som är längre ifrån origo enligt nedan skiss. I Scala är typer som är `Comparable` implicit även `Ordered`, varför sorteringen nedan funkar. Verfiera detta i REPL när du klurat ut hur implementera `compareTo`.

```
case class Point(x: Int, y: Int) extends Comparable[Point] {
  def distanceFromOrigin: Double = ???
  def compareTo(that: Point): Int = ???
}
```

```
scala> val xs = Seq(Point(10,10), Point(2,1), Point(5,3), Point(0,0))
scala> xs.sorted
```

Uppgift 8. *`java.util.Arrays.binarySearch`* I klassen `java.util.Arrays`[7] finns en statisk metod `binarySearch` som kan användas enligt nedan.

```
scala> val xs = Array(5,1,3,42,-1)
scala> java.util.Arrays.sort(xs)
scala> xs
scala> java.util.Arrays.binarySearch(xs, 42)
scala> java.util.Arrays.binarySearch(xs, 43)
```

Skriv ett valfritt Javaprogram som testar `java.util.Arrays.binarySearch`. Använd en array av typen `int[]` med några heltal som först sorteras med `java.util.Arrays.sort`. Skriv ut det som returneras från `java.util.Arrays.binarySearch` i olika fall genom att asöka efter tal som finns först, mitt i, sist och tal som saknas. *Tips:* Man kan deklarera en array, allokera den och fylla den med värden så här i Java:
`int[] xs = new int[]{5, 1, 3, 42, -1};`

Uppgift 9. *Auto(un)boxing.* I JVM måste typparametern för generiska klasser vara av referenstyp. I Scala löser kompilatorn detta åt oss så att vi ändå kan ha t.ex. `Int` som argument till en typparameter i Scala, medan man i Java *inte* direkt kan ha den primitiva typen `int` som typparameter till t.ex. `ArrayList`.

I Java och i den underliggande plattformen JVM används s.k. wrapper-klasser för att lösa detta, t.ex. genom wrapper-klassen `Integer` som boxar den primitiva typen `int`. Java-kompilatorn har stöd för att automatiskt packa in värden av primitiv typ i sådana wrapper-klasser för att skapa referenstyper och kan även automatiskt packa upp dem.

a) Studera hur Scala-kompilatorn låter oss arbeta med en `Cell[Int]` även om det underliggande JVM:ens körtidstyp (eng. *runtime type*) är en wrapper-klass. Man kan se JVM-körtidstypen med metoderna `getClass` och `getTypeName` enligt nedan.

```
scala> class Cell[T](var value: T){
         val typeName: String = value.getClass.getTypeName
         override def toString = "Cell[" + typeName + "](" + value + ")"
       }
scala> val c = new Cell[Int](42)
scala> c.value.getClass.getTypeName
```

b) Vad är körtidstypen för `c.value` ovan? Förklara hur det kan komma sig trots att vi deklarerade med typargumentet `Int`?

c) Studera dokumentationen för `java.lang.Integer`[8] och testa i REPL några av *klassmetoderna* (de som är `static` och därmed kan anropas med punktnotation direkt på klassens namn utan `new`) och några av *instansmetoderna* (de som inte är `static`).

```
scala> Integer.  //tryck TAB
scala> Integer.
scala> Integer.toBinaryString(42)
scala> Integer.valueOf(42)
scala> val i = new Integer(42)
```

[7] docs.oracle.com/javase/8/docs/api/java/util/Arrays.html
[8] docs.oracle.com/javase/8/docs/api/java/lang/Integer.html

```
scala> i.  // tryck TAB
scala> i.toString
scala> i.compareTo  // tryck TAB 2 gånger
scala> i.compareTo(Integer.valueOf(42))
scala> i.compareTo(42)  // varför fungerar detta?
```

✎ d) Enligt dokumentationen[9] tar instansmetoden `compareTo` i klassen `Integer` en `Integer` som parameter. Hur kan det då komma sig att sista raden ovan fungerar med en `Int`?

e) Studera nedan Java-program och beskriv vad som kommer att skrivas ut *innan* du kompilerar och testkör.

```
1  import java.util.ArrayList;
2
3  public class Autoboxing {
4      public static void main(String[] args) {
5          ArrayList<Integer> xs = new ArrayList<Integer>();
6          for (int i = 0; i < 42; i++) {
7              xs.add(new Integer(i));
8          }
9          for (Integer x: xs) {
10             int y = x.intValue() * 10;
11             System.out.print(y + " ");
12         }
13         int pos = xs.size();
14         xs.add(pos, new Integer(0));
15         System.out.println("\n\n[0]: " + xs.get(0).intValue());
16         System.out.println("[" + pos + "]: " + xs.get(pos));
17         if (xs.get(0) == xs.get(pos)) {
18             System.out.println("EQUAL");
19         } else {
20             System.out.println("NOT EQUAL");
21         }
22     }
23 }
```

f) Ändra i programmet ovan så att autoboxing och autounboxing utnyttjas på alla ställen där så är möjligt. Utnyttja även att `toString`-metoden på `Integer` ger samma stränrepresentation som `int` vid utskrift. Fixa också så att du undviker *fallgropen* att i Java jämföra med referenslikhet i stället för att använda `equals`. Testa så att allt fungerar som det borde efter dina ändringar.

✎ g) Antag att du råkar skriva `xs.add(0, pos)` på rad 14 i ditt program från föregående uppgift. Förklara hur autoboxingen stjälper dig i en *fallgrop* då.

✎ h) Med ledning av de båda tidigare deluppgifterna: sammanfatta de två nämnda fallgropar med autoboxing i Java i två generella punkter, så att du har nytta av att memorera dem inför din framtida Javakodning.

Uppgift 10. *CollectionConverters.* Med `import scala.jdk.CollectionConverters._` får man i sina Scalaprogram tillgång till de smidiga metoderna `asJava` och `asScala`

[9] docs.oracle.com/javase/8/docs/api/java/lang/Integer.html#compareTo-java.lang.Integer-

som översätter mellan motsvarande samlingar i resp språks standardbibliotek. Kör nedan i REPL och gör efterföljande deluppgifter.

```
scala> val sv = Vector(1,2,3)
scala> val ss = Set('a','b','c')
scala> val sm = Map("gurka" -> 42, "tomat" -> 0)
scala> val ja = new java.util.ArrayList[Int]
scala> ja.add(42)
scala> val js = new java.util.HashSet[Char]
scala> js.add('a')
scala> import scala.jdk.CollectionConverters._
```

a) Till vilka typer konverteras Scalasamlingarna `Vector[Int]`, `Set[Char]` och `Map[String, Int]` om du anropar metoden `asJava` på dessa?

b) Till vilka typer konverteras Javasamlingarna `ArrayList[Int]` och `HashSet[Char]` om du anropar metoden `asScala` på dessa? Blir det föränderliga eller oföränderliga motsvarigheter?

c) Vad får resultatet för typ om du kör `toSet` på en samling av typen `mutable.Set`?

d) Undersök hur du kan efter att du gjort `sm.asJava.asScala` anropa ytterligare en metod för att få tillbaka en oföränderlig `immutable.Map`.

e) Läs mer i dokumentationen om CollectionConverters[10] och prova några fler konverteringar.

Uppgift 11. *Hur fungerar en* ***`switch`****-sats i Java (och flera andra språk)?* Det händer ofta att man vill testa om ett värde är ett av många olika alternativ. Då kan man använda en sekvens av många **`if-else`**, ett för varje alternativ. Men det finns ett annat sätt i Java och många andra språk: man kan använda **`switch`** som kollar flera alternativ i en och samma sats, se t.ex. en.wikipedia.org/wiki/Switch_statement.

a) Skriv in nedan kod i en kodeditor. Spara med namnet `Switch.java` och kompilera filen med kommandot `javac Switch.java`. Kör den med `java Switch` och ange din favoritgrönsak som argument till programmet. Vad händer? Förklara hur **`switch`**-satsen fungerar.

```
public class Switch {
    public static void main(String[] args) {
        String favorite = "selleri";
        if (args.length > 0) {
            favorite = args[0];
        }
        System.out.println("Din favoritgrönsak: " + favorite);
        char firstChar = Character.toLowerCase(favorite.charAt(0));
        System.out.print("Jag tycker att ");
        switch (firstChar) {
        case 'g':
            System.out.println("gurka är gott!");
            break;
        case 't':
            System.out.println("tomat är gott!");
            break;
        case 'b':
            System.out.println("broccoli är gott!");
            break;
```

[10] docs.scala-lang.org/overviews/collections-2.13/conversions-between-java-and-scala-collections.html

```
        default:
            System.out.println(favorite + " är mindre gott...");
            break;
        }
    }
}
```

b) Vad händer om du tar bort **break**-satsen på rad 16?

Uppgift 12. *Fånga undantantag i Java med en* ***try-catch****-sats.* Det finns som vi såg i förra uppgiften inbyggt stöd i JVM för att hantera när program avbryts på oväntade sätt, t.ex. på grund av division med noll eller ej förväntade indata från användaren. Spara koden nedan[11] i en fil med namnet `TryCatch.java` och kompilera med `javac TryCatch.java` i terminalen.

```
// TryCatch.java

public class TryCatch {
    public static void main(String[] args) {
        int input;
        int output;
        if (args[0].equals("safe")) {
            try {
                input = Integer.parseInt(args[1]);
                System.out.println("Skyddad division!");
                output = 42 / input;
            } catch (Exception e) {
                System.out.println("Undantag fångat: " + e);
                System.out.println("Dividerar ändå med säker default!");
                input = 1;
                output = 42 / input;
            }
        } else {
            input = Integer.parseInt(args[0]);
            System.out.println("Oskyddad division!");
            output = 42 / input;
        }
        System.out.println("42 / " + input + " == " + output);
    }
}
```

a) Förklara vad som händer när du kör programmet med olika indata:

```
> java TryCatch 42
> java TryCatch 0
> java TryCatch safe 42
> java TryCatch safe 0
> java TryCatch
```

b) Vad händer om du ”glömmer bort” raden 15 och därmed missar att initialisera input? Hur lyder felmeddelandet? Är det ett körtidsfel eller kompileringsfel?

[11] https://github.com/lunduniversity/introprog/blob/master/compendium/examples/TryCatch.java

Uppgift 13. *Matriser med array i Java.* Om man redan vid allokering vet hur många element en matris ska ha, använder man i Java gärna en array av arrayer. En heltalsmatris (en array av array av heltal) skrivs i Java med dubbla hakparentespar `int[][]` direkt efter typen. Vid allokering använder man nyckelordet `new` och antalet element i respektive dimension anges inom hakparenteserna; t.ex. så ger `new int[42][21]` en matris med 42 rader och 21 kolumner, vilket motsvarar att man i Scala skriver `Array.ofDim[Int](42,21)`[12]. Alla element får defaultvärdet för typen, här `0` för heltal.

a) Skriv nedan program i en editor och spara koden i filen `JavaArrayTest.java` och kompilera med `javac JavaArrayTest.java` och kör i terminalen med `java JavaArrayTest` och undersök utskriften. Förklara vad som händer. Notera några skillnader i hur matriser används i Scala och Java.

```
public class JavaArrayTest {

    public static void showMatrix(int[][] m){
        System.out.println("\n--- showMatrix ---");
        for (int row = 0; row < m.length; row++){
            for (int col = 0; col < m[row].length; col++) {
                System.out.print("[" + row + "]");
                System.out.print("[" + col + "] = ");
                System.out.print(m[row][col] + "; ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        System.out.println("Hello JavaArrayTest!");
        int[][] xss = new int[10][5];
        showMatrix(xss);
    }
}
```

b) Implementera nedan metod `fillRnd` inuti klassen `JavaArrayTest`. Skriv kod som fyller matrisen `m` med slumptal mellan `1` och `n`.

```
    public static void fillRnd(int[][] m, int n){
        /* ??? */
    }
```

Tips: med detta uttryck skapas ett slumptal mellan 1 och 42 i Java:
`(int) (Math.random() * 42 + 1);`
där typkonverteringen `(int)` ger samma effekt som ett anrop av metoden `toInt` i Scala; alltså att dubbelprecisionsflyttal omvandlas till heltal genom avkortning av alla decimaler.

Ändra huvudprogrammet så det anropar `fillRnd(xss, 6)`. Programmet ska ge en utskrift som liknar följande:

```
1 Hello JavaArrayTest!
```

[12] Ett annat sätt att skriva detta i Scala där initialvärdet framgår explicit: `Array.fill(42,21)(0)`

```
--- showMatrix ---
[0][0] = 6; [0][1] = 2; [0][2] = 6; [0][3] = 3; [0][4] = 5;
[1][0] = 2; [1][1] = 4; [1][2] = 6; [1][3] = 1; [1][4] = 1;
[2][0] = 5; [2][1] = 4; [2][2] = 4; [2][3] = 1; [2][4] = 5;
[3][0] = 4; [3][1] = 6; [3][2] = 6; [3][3] = 1; [3][4] = 3;
[4][0] = 4; [4][1] = 6; [4][2] = 2; [4][3] = 3; [4][4] = 2;
[5][0] = 2; [5][1] = 4; [5][2] = 5; [5][3] = 5; [5][4] = 3;
[6][0] = 6; [6][1] = 5; [6][2] = 2; [6][3] = 4; [6][4] = 3;
[7][0] = 1; [7][1] = 6; [7][2] = 1; [7][3] = 6; [7][4] = 2;
[8][0] = 1; [8][1] = 1; [8][2] = 5; [8][3] = 3; [8][4] = 2;
[9][0] = 1; [9][1] = 1; [9][2] = 1; [9][3] = 5; [9][4] = 4;
```

Uppgift 14. *Översätta från Java till Scala.* Översätt nedan kod från Java till Scala. Skriv koden i en fil som heter `showInt.scala` och kalla Scala-objektet med `main`-metoden för `showInt`. Läs tipsen som följer efter koden innan du börjar.

```java
import java.util.Scanner;

public class JShowInt {
    private static Scanner scan = new Scanner(System.in);

    public static void show(Object obj) {
        System.out.println(obj);
    }

    public static void show(Object obj, String msg) {
        System.out.println(msg + obj);
    }

    public static String repeatChar(char ch, int n) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < n; i++) {
            sb.append(ch);
        }
        return sb.toString();
    }

    public static String readLine(String prompt) {
        System.out.print(prompt);
        return scan.nextLine();
    }

    public static void showInt(int i) {
        int leading = Integer.numberOfLeadingZeros(i);
        String binaryString =
            repeatChar('0', leading) + Integer.toBinaryString(i);
        show(i,                         "Heltal: ");
        show((char) i,                  "Tecken: ");
        show(binaryString,              "Binärt: ");
        show(Integer.toHexString(i),    "Hex   : ");
        show(Integer.toOctalString(i), "Oktalt: ");
```

```
36      }
37
38      public static void loop() {
39          boolean hasExploded = false;
40          while (!hasExploded) {
41              try {
42                  String s = readLine("Heltal annars pang: ");
43                  showInt(Integer.parseInt(s));
44              } catch (Throwable e){
45                  show(e);
46                  hasExploded = true;
47              }
48          }
49          show("PANG!");
50      }
51
52      public static void main(String[] args){
53          if (args.length == 0) {
54              loop();
55          } else {
56              for (String arg: args) {
57                  showInt(Integer.parseInt(arg));
58                  System.out.println();
59              }
60          }
61      }
62  }
```

Tips:

- En Javaklass med bara statiska medlemmar motsvaras av ett singeltonobjekt i Scala, alltså en **`object`**-deklaration. Scala har därför inte nyckelordet **`static`**.
- Typen `Object` i Java motsvaras av Scalas `Any`.
- Du kan använda Scalas möjlighet med default-argument (som saknas i Java) för att bara definiera en enda `show`-metod med en tom sträng som default `msg`-argument.
- I Scala har objekt av typen `Char` en metod **`def`** `*(n: Int): String` som skapar en sträng med tecknet repeterat `n` gånger. Men du kan ju välja att ändå implementera metoden `repeatChar` med `StringBuilder` som nedan om du vill träna på att översätta en **`for`**-loop från Java till Scala.
- I stället för `Scanner.nextLine` kan du använda `scala.io.StdIn.readLine` som tar en prompt som parameter, men du kan också använda `Scanner` i Scala om du vill träna på det.
- I Java *måste* man använda nyckelordet **`return`** om metoden inte är en **`void`**-metod, medan man i Scala faktiskt *får* använda **`return`** även om man brukar undvika det och i stället utnyttja att satser i Scala också är uttryck.

Kompilera din Scala-kod och kör i terminalen och testa så att allt funkar. Vill du även kompilera Java-koden så finns den i kursens repo i filen `compendium/examples/scalajava/JShowInt.java`

Uppgift 15. *Innehållslikhet och referenslikhet i Java.* Studera och prova denna

fallgrop med innehållslikhet: TestPitfall3.java

✎ **Uppgift 16.** *Implementera innehållslikhet i Java.* Studera fallgropar för hur man skriver en `equals`-metod i Java här: www.artima.com/lejava/articles/equality.html och jämför med det fullständiga receptet för hur man skriver en välfungerande `equals` och `hashcode` i Scala här: www.artima.com/pins1ed/object-equality.html

a) Vilka skillnader och likheter finns vid överskuggning av equals i Java respektive Scala, som ska ge en fungerande innehållstest för en hierarki med bastyper och subtyper?

b) Vilka fallgropar är gemensamma för Java och Scala?

Uppgift 17. *Array och `for`-sats i Java.* Ladda ner programet nedan från kursens GitHub-repo: `compendium/examples/DiceReg.java`

a) Kompilera med `javac DiceReg.java` och kör med `java DiceReg 10000 42` och förklara vad som händer.

```
// DiceReg.java
import java.util.Random;

public class DiceReg {
    public static void main(String[] args) {
        int[] diceReg = new int[6];
        int n = 100;
        Random rnd = new Random();
        if (args.length > 0) {
            n = Integer.parseInt(args[0]);
        }
        System.out.print("Rolling the dice " + n + " times");
        if (args.length > 1) {
            int seed = Integer.parseInt(args[1]);
            rnd.setSeed(seed);
            System.out.print(" with seed " + seed);
        }
        System.out.println(".");
        for (int i = 0; i < n; i++) {
            int pips = rnd.nextInt(6);
            diceReg[pips]++;
        }
        for (int i = 1; i <= 6; i++) {
            System.out.println("Number of " + i + "'s: " +
                diceReg[i-1]);
        }
    }
}
```

b) Beskriv skillnaderna mellan Scala och Java, vad gäller syntaxen för array och `for`-sats. Beskriv några andra skillnader mellan språken som syns i programmet ovan.

c) Ändra i programmet ovan så att loop-variabeln `i` skrivs ut i varje runda i varje `for`-sats. Kompilera om och kör.

d) Skriv om programmet ovan genom att abstrahera huvudprogrammets delar till de statiska metoderna `parseArguments`, `registerPips` och `printReg` enligt nedan skelett. Spara programmet i filen `DiceReg2.java` och kompilera med `javac DiceReg2.java` i terminalen.

```
// DiceReg2.java
import java.util.Random;

public class DiceReg2 {
    public static int[] diceReg = new int[6];
    private static Random rnd = new Random();

    public static int parseArguments(String[] args) {
        // ???
        return n;
    }

    public static void registerPips(int n){
        // ???
    }

    public static void printReg() {
        // ???
    }

    public static void main(String[] args) {
        int n = parseArguments(args);
        registerPips(n);
        printReg();
    }
}
```

e) Starta Scala REPL i samma katalog som filen `DiceReg2.class` ligger i och kör nedan 7 rader i REPL och förklara vad som händer:

```
scala> DiceReg2.main(Array("1000","42"))
scala> DiceReg2.diceReg
scala> DiceReg2.registerPips(1000)
scala> DiceReg2.printReg
scala> DiceReg2.registerPips(1000)
scala> DiceReg2.printReg
scala> DiceReg2.rnd
```

Uppgift 18. *Läsa in sekvens av tal med `Scanner` i Java.* Läs i Java-delen av snabbreferensen om `java.util.Scanner`. Med `new Scanner(System.in)` skapas ett objekt som kan läsa in tal från teckensträngar som användaren skriver i terminalfönstret, så som visas i Java-programmet nedan:

```
// DiceScanBuggy.java
import java.util.Random;
import java.util.Scanner;
```

```
public class DiceScanBuggy {
    public static int[] diceReg = new int[6];
    public static Scanner scan = new Scanner(System.in);

    public static void registerPips(){
        System.out.println("Enter pips separated by blanks.");
        System.out.println("End with -1 and <Enter>.");
        boolean isPips = true;
        while (isPips && scan.hasNextInt()) {
            int pips = scan.nextInt();
            if (pips >= 1 && pips <=6 ) {
              diceReg[pips]++;
            } else {
              isPips = false;
            }
        }
    }

    public static void printReg() {
        for (int i = 0; i < 6; i++) {
            System.out.println("Number of " + i + "'s: " +
                diceReg[i-1]);
        }
    }

    public static void main(String[] args) {
        registerPips();
        printReg();
    }
}
```

Ladda ner programmet `compendium/examples/DiceScanBuggy.java` och kompilera och kör med indatasekvensen `1 2 3 4 -1` och notera hur registreringen sker.

a) Sök upp och läs JDK8-dokumentationen av `java.util.Scanner`. Vad gör `hasNextInt()` och `nextInt()`?

b) Programmet fungerar inte som det ska. Du behöver korrigera 3 saker för att programmet ska göra rätt. Rätta buggarna och spara det rättade programmet som `DiceScan.java`. Kompilera och testa det rättade programmet.

J.3 Laboration: `java`

Mål

- ☐ Förstå skillnaden mellan primitiva typer och objekt i Java.
- ☐ Kunna förklara hur autoboxing fungerar i Java.
- ☐ Kunna förklara vad statiska metoder och attribut i Java innebär.
- ☐ Kunna använda `ArrayList` och arrayer i Java.
- ☐ Kunna använda Java-klassen `Scanner`.
- ☐ Kunna skapa en for-sats i Java.
- ☐ Känna till hur man kan förenkla användandet av Java och Scala i samma program med hjälp av `scala.jdk.CollectionConverters`.

Förberedelser

- ☐ Gör övningarna tidigare i detta Appendix.
- ☐ Studera given kod här: github.com/lunduniversity/introprog/tree/master/workspace/javatext/

J.3.1 Krav

Du ska skapa ett textspel för terminalen som är (lagom) intressant/roligt att spela och sparar poäng per spelomgång för olika spelare. Till din hjälp har du den färdiga filen `Main.java` (som går bra att förändra om det behövs) samt de två kodskeletten `Game.java` och `UserInterface.scala`. Ditt textspel ska köras i terminalen och uppfylla följande krav och riktlinjer:

1. När ditt program kör ska man ska kunna starta flera spelomgångar efter varandra utan att behöva avsluta programmet.
2. För varje spelomgång ska programmet komma ihåg spelarens namn[13] med tillhörande resultat.
3. Efter begäran ska programmet kunna visa en topplista med bästa poäng, både för alla spelare och för ett specifikt spelarnamn.
4. Koden för själva spelet ska vara skriven i Java, men Scala ska användas för att implementera funktionerna i singelobjektet `UserInterface`.
5. I Scala-koden ska du för träningens skull använda Java-klassen `java.util.Scanner` när du läser in data från terminalen.
6. Koden i singelobjektet `UserInterface` ska använda omvandlingsmetoden `asScala` efter `import scala.jdk.CollectionConverters._` för att omvandla argument av typen `java.util.ArrayList` till `scala.collection.mutable.Buffer`[14].
7. Ditt spel ska i Java-kod använda minst en av datastrukturerna `ArrayList`, `HashSet`, `HashMap` ur paketet `java.util`, samt minst en array. (Den givna koden i `Main.java` räknas inte till detta krav.)
8. Du ska spela någon annans halvfärdiga spel och, efter att du studerat koden, ge återkoppling på kodens läsbarhet.

[13] eller spelar*nas* namn om det är ett spel för två eller flera personer
[14] Notera att `asJava` på `Buffer` ger en Java-samling av typen `List`.

9. Du ska låta någon annan spela ditt halvfärdiga spel och visa din kod och fråga om återkoppling på läsbarheten. Du ska anteckna den återkoppling du får.

10. Du ska inför redovisningen förbereda följande:

    (a) en kort genomgång av spelets idé,

    (b) en kort förklaring av kodens struktur och de olika Java-klassernas ansvar,

    (c) en kort redogörelse för den återkoppling du fått på din kods läsbarhet och hur du arbetat med att förbättra läsbarheten under dina stegvisa utvidgningar av din kod,

    (d) en lista med koncept som du tränat på när du skapat ditt textspel.

J.3.2 Frivilliga extrauppgifter

1. Spara resultat i en fil efter varje spelomgång, och läs in resultat från filen antingen när programmet startas eller när användaren vill se poänglistan, så att det går att se spelresultat från tidigare körningar av programmet. Den kod du behöver lägga till för att åstadkomma detta kan vara skriven antingen i Java eller Scala. Tänk på att du kan behöva göra ändringar även i `Main`-klassen.

2. Mät speltiden för varje spelomgång och spara tiden tillsammans med poängresultatet för respektive spelare.

J.3.3 Inspiration och tips

1. Utgå från Hangman i veckans övning eller,

2. Yatzy från tidigare övningar, eller

3. skapa ett kortspel inspirerat av `shuffle`-labben, eller

4. inspireras av listan med sällskapsspel på wikipedia:
   sv.wikipedia.org/wiki/Kategori:Sällskapsspel

5. eller hitta på ett eget textspel.

6. Börja med en starkt förenklad variant som du sedan bygger vidare på.

7. Kompilera och testa efter varje ändring, så att du hela tiden har ett fungerande program.

8. Dela upp din spelkod i flera metoder, och även flera klasser om det är lämpligt.

9. Det finns mycket information på nätet om hur man skriver Java-kod och använder JDK, t.ex. på `https://stackoverflow.com/`

10. Träna på att använda JDK-dokumentationen här:
    `https://docs.oracle.com/javase/8/docs/api/`

Del IV

Lösningar

Appendix L

Lösningar till övningarna

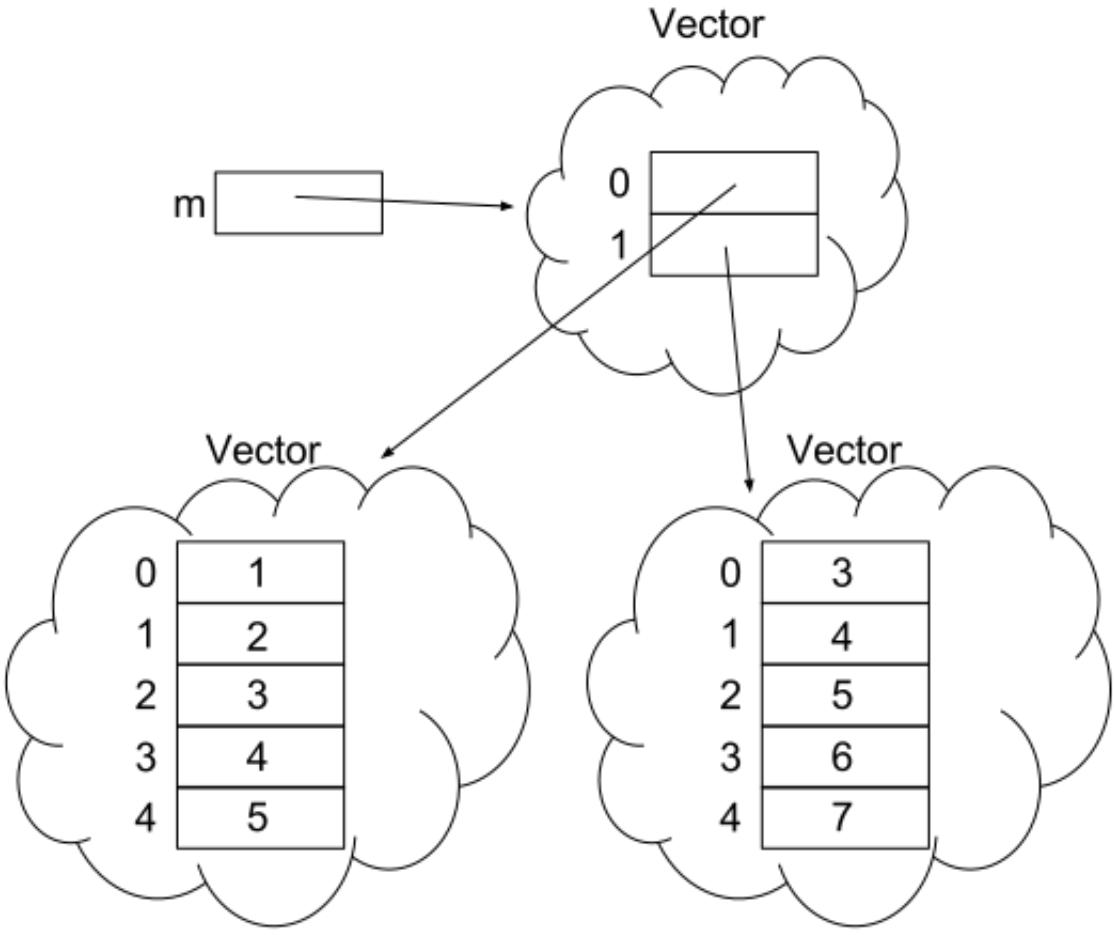
L.8 Lösning `matrices`

L.8.1 Grunduppgifter; förberedelse inför laboration

Lösn. uppg. 1. *Para ihop begrepp med beskrivning.*

matris	1	⇝	A	indexerbar datastruktur i två dimensioner
radvektor	2	⇝	F	matris av dimension $1 \times m$ med m horisontella värden
kolumnvektor	3	⇝	G	matris av dimension $m \times 1$ med m vertikala värden
kolonn	4	⇝	C	annat ord för kolumn
generisk	5	⇝	B	har abstrakt typparameter, typen är generell
typargument	6	⇝	D	konkret typ, binds till typparameter vid kompilering
typhärledning	7	⇝	E	kompilatorn beräknar typ ur sammanhanget

Lösn. uppg. 2. *Skapa matriser med hjälp av nästlade samlingar.*



a)
Typ: `Vector[Vector[Int]]`
Värde: `Vector(Vector(1, 2, 3, 4, 5), Vector(3, 4, 5, 6, 7))`
Dimensioner: 2×5
Inom matematiken sker indexering enligt konvention med 1 som lägsta index. I scala

är lägsta index 0, man använder s.k. 0-indexering. [1]

b)

```
scala> val m = Vector((1 to 5).toVector, (3 to 7).toVector)
m: Vector[Vector[Int]] = Vector(Vector(1, 2, 3, 4, 5), Vector(3, 4, 5, 6, 7))

scala> m.apply(0).apply(1)
res4: Int = 2

scala> m(1)
res5: Vector[Int] = Vector(3, 4, 5, 6, 7)

scala> m(1)(4)
res6: Int = 7
```

c)

m2: `Vector[Vector[Int]]`
m3: `Vector[Vector[Int | Double]]`
m4: `Vector[Vector[Int | Double | String]]`
m5: `Vector[Vector[Int]]`

d) m5, 42×2

Lösn. uppg. 3. *Skapa och iterera över matriser.*

a)

```
def throwDie: Int = (math.random() * 6).toInt + 1
```

Eller:

```
def throwDie: Int = scala.util.Random.nextInt(6) + 1
```

b) Matrisdimension i matematisk notation: 1000×5, vilket motsvarar en matris med 1000 rader och 5 kolumner.

c)

```
ds1: IndexedSeq[IndexedSeq[Int]]
ds2: IndexedSeq[IndexedSeq[Int]]
ds3: IndexedSeq[Vector[Int]]
ds4: IndexedSeq[Vector[Int]]
ds5: Vector[Vector[Int]]
ds6: Vector[Vector[Int]]
```

`IndexedSeq` och `Vector` ovan finns i paketet `scala.collection.immutable`

d)

```
def roll(n: Int) = Vector.fill(n)(throwDie).sorted
```

e)

```
def isYatzy(xs: Vector[Int]): Boolean = xs.forall(_ == xs(0))
```

[1] Detta är inte fallet i alla programmeringsspråk, vilket du kan läsa mer om på https://en.wikipedia.org/wiki/Array_data_type#Index_origin

f)

```
def diceMatrix(m: Int, n: Int): Vector[Vector[Int]] =
  Vector.fill(m)(roll(n))
```

g)

```
def diceMatrixToString(xss: Vector[Vector[Int]]): String =
  xss.map(_.mkString(" ")).mkString("\n")
```

h)

```
def filterYatzy(xss: Vector[Vector[Int]]): Vector[Vector[Int]] =
  xss.filter(isYatzy)
```

i)

```
def yatzyPips(xss: Vector[Vector[Int]]): Vector[Int] =
  filterYatzy(xss).map(_.head)
```

Lösn. uppg. 4. *En oföränderlig, generisk matris-klass till veckans laboration* `life`.

a) Typen på `m` blir `Matrix`.

b) Typen på `e` blir `String`.

c) Man behöver ändra på 3 ställen från `String` till `Int`.

d) Generisk matris `Matrix[T]` för element av godtycklig typ `T`:

```
case class Matrix[T](data: Vector[Vector[T]]):
  def apply(row: Int, col: Int): T = data(row)(col)

object Matrix:
  def fill[T](dim: (Int, Int))(value: T): Matrix[T] =
    Matrix[T](Vector.fill(dim._1, dim._2)(value))
```

e) Tack vare kompilatorns typinferens så får `bm` typen `Matrix[Boolean]`.

f) Typen på `be` blir `Boolean`.

g) h) i) j) k) är alla implementerade i koden nedan:

```
case class Matrix[T](data: Vector[Vector[T]]):
  require(data.forall(row => row.length == data(0).length))

  val dim: (Int, Int) = (data.length, data(0).length)

  def apply(row: Int, col: Int): T = data(row)(col)

  def updated(row: Int, col: Int)(value: T): Matrix[T] =
    Matrix(data.updated(row, data(row).updated(col, value)))

  def foreachIndex(f: (Int, Int) => Unit): Unit =
    for r <- data.indices; c <- data(r).indices do f(r, c)

  override def toString =
    s"""Matrix of dim $dim:\n${ data.map(_.mkString(" ")).mkString("\n") }"""
```

```
object Matrix:
  def fill[T](dim: (Int, Int))(value: T): Matrix[T] =
    Matrix[T](Vector.fill(dim._1, dim._2)(value))
```

L.8.2 Extrauppgifter; träna mer

Lösn. uppg. 5. *Imperativa matrisalgoritmer.*

a)

```
def isYatzy(xs: Vector[Int]): Boolean =
  var foundDiff = false
  var i = 0
  while (i < xs.size && !foundDiff) do
    foundDiff = xs(i) != xs(0)
    i += 1
  end while
  !foundDiff
```

b) Funktionen går igenom varje matrisrad, där den i sin tur går igenom varje element på raden och lägger till i `StringBuilder`-objektet. Om det inte är det sista elementet på raden läggs även ett blanktecken till, annars läggs ett nyradstecken till. Undantaget är sista raden, där inget nyradstecken läggs till. Slutligen konverteras `StringBuilder`-objektet till en `String` som returneras.

Är `xss` tom blir `xss.indices` en tom `Range` och den yttre `for`-loopen hoppas över och en tom sträng returneras. Är alla rader tomma hoppas i stället de inre `for`-looparna över, med samma resultat.

Fördel: `StringBuilder` är snabbare vid tillägg på slutet vid stora strängar (men här kommer det inte märkas eftersom strängen är så liten).

Nackdel: StringBuilder-koden uppfattas av många som svårare att läsa.

c)

```
def filterYatzy(xss: Vector[Vector[Int]]): Vector[Vector[Int]] =
  var result: Vector[Vector[Int]] = Vector()
  for i <- xss.indices if isYatzy(xss(i)) do result = result :+ xss(i)
  result
```

d) Varje looprunda ger en vektor `xss(i)` om filtervillkoret är uppfyllt och resultatet av `for`-uttrycket blir en vektor med vektorer som är yatzyslag.

Lösn. uppg. 6. *Strängtabell med kolumnrubriker.*

a)

```
case class Table(
  data: Vector[Vector[String]],
  headings: Vector[String],
  sep: Char
):

  val dim: (Int, Int) = (data.size, headings.size)

  def apply(r: Int, c: Int): String = data(r)(c)

  def row(r: Int): Vector[String]= data(r)

  def col(c: Int): Vector[String] = data.map(r => r(c))
```

```
  lazy val indexOfHeading: Map[String, Int] = headings.zipWithIndex.toMap

  def col(h: String): Vector[String] = col(indexOfHeading(h))

  def values(h: String): Vector[String] = col(h).distinct.sorted

  override def toString: String =
    val s = sep.toString
    headings.mkString(s) + "\n" +data.map(_.mkString(s)).mkString("\n")

object Table:
  def fromFile(fileName: String, sep: Char = ';'): Table =
    val lines = scala.io.Source.fromFile(fileName).getLines.toVector
    val matrix= lines.map(_.split(sep).toVector)
    new Table(matrix.tail, matrix.head, sep)
```

b)

```
@main
def run(fileName: String, separator: String): Unit =
  require(separator.length == 1, "separator ska vara exakt ett tecken")
  val t = Table.fromFile(fileName, separator.head)
  val counts: Vector[Vector[String]] =
    (0 until t.dim._2)
      .map(i => t.values(t.headings(i))
      .map(x => s"$x: ${t.col(i).count(_ == x)}"))
      .toVector
  for (i <- 0 until t.dim._2) do
    println(s"\nColumn: ${i + 1}, ${t.headings(i)}:")
    for (j <- 0 until counts(i).length) do
      println(counts(i)(j))
```

Lösn. uppg. 7. *Skapa ett yatzy-spel för användning i terminalen.*

–

L.8.3 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 8. *Generiska funktioner.*

a)

1. –
2. Strängrepresentationen av `42` spegelvänds
3. `"hej"` spegelvänds - `toString` av en sträng ger en likadan sträng
4. –
5. Gurk-objektets strängrepresentation spegelvänds
6. Funktionens typparameter matchar inte parameterns typ: `42` är ingen sträng
7. Implicit typkonvertering till `Double` sker för att stämma överens med typparametern, vilket ger en strängrepresentation med decimal

b)

1. En funktion definieras så att den tar emot två andra funktioner som argument, sätter ihop dem, och matar in ett tredje argument till den den sammansatta funktionen.
2. En funktion som inkrementerar ett heltal med 1 definieras.
3. En funktion som halverar ett flyttal definieras.
4. `42` matas in i `inc()` och resultatet (`43`) matas vidare till `half()`. Inuti `half()` sker implicit typkonvertering till `Double` då talet divideras med ett flyttal (`2.0`) och resultatet blir `43.0 / 2.0`, alltså `21.5`.
5. Resultatet från `half()` är av typ `Double`, medan `inc()` tar emot ett argument av typ `Int`. Då flyttal generellt inte kan konverteras till heltal utan informationsförlust sker ingen implicit konvertering, istället sker ett kompileringsfel.

c)

```
def inc(x: Double): Double = x + 1.0
```

Nu ges kompileringsfel på rad 4 istället, vilket kan lösas med följande ändring:

```
def half(x: Double): Double = x / 2.0
```

Lösn. uppg. 9. *Generiska klasser.*

a) –

b)

```
class Cell[T](var value: T):
  override def toString = "Cell(" + value + ")"
  def concat[U](that: Cell[U]): Cell[String] =
    Cell(s"$value${that.value}")
```

c) Endast celler med samma typparameter kan nu konkateneras. Eftersom `concat()` returnerar ett objekt av typ `Cell[String]` kan ett ojämnt antal celler med någon annan typparameter än `String` alltså inte längre konkateneras. Är antalet jämnt går det att konkatenera dem parvis och sedan konkatenera de returnerade `Cell[String]`-objekten, men det är något omständigt.

Lösn. uppg. 10. *Implementera fler generiska metoder i* `Matrix[T]`.

```
case class Matrix[T](data: Vector[Vector[T]]):
  require(data.forall(row => row.size == data(0).size))

  val dim: (Int, Int) = (data.length, data(0).length)

  def apply(row: Int, col: Int): T = data(row)(col)

  def updated(row: Int, col: Int)(value: T): Matrix[T] =
    Matrix(data.updated(row, data(row).updated(col, value)))

  def foreach(f: T => Unit): Unit = data.foreach(_.foreach(f))

  def foreachIndex(f: (Int, Int) => Unit): Unit =
    for r <- data.indices; c <- data(r).indices do f(r, c)

  def map[U](f: T => U): Matrix[U] = Matrix(data.map(_.map(f)))

  def mapIndex[U](f: (Int, Int) => U): Matrix[U] =
    var result = Matrix.fill(dim)(f(0,0))
    for
      r <- data.indices
      c <- data(r).indices
    do
      result = result.updated(r, c)(f(r, c))
    end for
    result

  override def toString =
    s"""Matrix of dim $dim:\n${ data.map(_.mkString(" ")).mkString("\n") }"""

object Matrix:
  def fill[T](dim: (Int, Int))(value: T): Matrix[T] =
    Matrix[T](Vector.fill(dim._1, dim._2)(value))
```

L.9 Lösning `lookup`

L.9.1 Grunduppgifter; förberedelse inför laboration

Lösn. uppg. 1. *Para ihop begrepp med beskrivning.*

mängd	1	⇝	H	oordnad samling med unika element
nyckel-värde-tabell	2	⇝	F	oordnad samling av mappningar med unika nycklar
mappning	3	⇝	G	`nyckel -> värde`
nyckel	4	⇝	C	en unik identifierare
persistens	5	⇝	D	egenskapen att finnas kvar efter programmets avslut
serialisera	6	⇝	E	koda objekt till avkodningsbar sekvens av symboler
de-serialisera	7	⇝	B	avkoda symbolsekvens och återskapa objekt i minnet
linjärsöka	8	⇝	A	leta i sekvens tills sökkriteriet är uppfyllt

Lösn. uppg. 2. *Vad är en mängd?* En mängd är en samling som snabbt kan ge svaret på frågan om ett visst element ingår i samlingen eller ej. Elementen i en mängd är unika. Tilläg av redan existerande element ignoreras. En mängd är inte en sekvens, eftersom traversering med t.ex. `map` eller `foreach` inte (nödvändigtvis) sker i den ordning som elementen gavs när mängden konstruerades eller uppdaterades.

Lösn. uppg. 3. *Använda mängder.*

`Set(1, 2) ++ Set(1, 2)`	1	⇝	I	`Set(1, 2)`
`(1 to 3).toSet`	2	⇝	G	`Set(1) + 2 + 3`
`Vector.fill(3)(1).toSet`	3	⇝	F	`Set(1, 2) - 2`
`Set(1, 2, 3) diff Set(1, 2)`	4	⇝	B	`Set(3)`
`(1 to 7).toSet.apply(8)`	5	⇝	H	`false`
`Set(1, 2, 3).sorted`	6	⇝	D	`error: ...`
`Set(2,4) subsetOf (1 to 7).toSet`	7	⇝	E	`true`
`Set(1, -1, 2, -2).map(_.abs).sum`	8	⇝	A	`3`
`Set(1, 1, 1, 1, 1, 5).sum`	9	⇝	C	`6`

Lösn. uppg. 4. *Räkna unika ord med hjälp av en mängd.*

a)

```
scala> val hej = "hej hej hemskt mycket hej"
scala> val n = hej.split(' ').toSet.size
n: Int = 3
```

b) Metoden `distinct` returnerar en sekvens med unika element och bibehållen ursprunglig ordning.

Lösn. uppg. 5. *Skapa 2-tupler med metoden `->` som kan uttalas ”mappas till”.*

a) Ja, fabriksmetoden returnerar ett helt vanligt par:

```
scala> val härBorJag = "Skåne" -> "Lund"
val härBorJag: (String, String) = (Skåne,Lund)
```

```
scala> härBorJag._1
val res0: String = Skåne

scala> härBorJag._2
val res1: String = Lund
```

b)

```
val huvudstad = Vector(
  "Sverige"  -> "Stockholm",
  "Danmark"  -> "Köpenhamn",
  "Grönland" -> "Nuuk",
  "Skåne"    -> "Lund"
)
```

c)

```
scala> huvudstad(3)._2
val res2: String = Lund
```

Lösn. uppg. 6. *Linjärsöka efter nyckel i sekvens av mappningar.*

a)

```
def lookupIndex(xs: Vector[(String, String)])(key: String): Int =
  xs.indexWhere(_._1 == key)
```

b)

```
scala> val i = lookupIndex(huvudstad)("Skåne")
val i: Int = 3

scala> huvudstad(i)._2
val res2: String = Lund
```

Eller med funktioner som återanvändbara dellösningar:

```
scala> val indexOf = lookupIndex(huvudstad) _

scala> def capital(key: String) = huvudstad(indexOf(key))._2

scala> capital("Skåne")
val res3: String = Lund

scala> capital("Sverige")
val res4: String = Stockholm
```

Lösn. uppg. 7. *Nyckel-värde-tabell.*

```
scala> telnr("Fröken Ur")
val res0: Long = 464690510

scala> :type telnr
Map[String,Long]
```

```
scala> :type telnr.toVector
Vector[(String, Long)]
```

Lösn. uppg. 8. *Använda nyckel-värdetabell.*

a) Nej nyckel-värde-paren lagras i någon speciell ordning som bestäms av en intern, smart lagringsprincip enligt en s.k. hashfunktion[2], för att åstadkomma snabba uppslagningar av värden från nycklar och vilket normalt inte sammanfaller med ordningen i den sekvens som de skapades ur.

b)

`xs(2) + xs(4)`	1	⇝ A	`8`
`ys(0)`	2	⇝ C	`(10, 11)`
`xs(0)`	3	⇝ I	`NoSuchElementException`
`(xs + (0 -> 1)).apply(0)`	4	⇝ D	`1`
`xs.keySet.apply(2)`	5	⇝ G	`true`
`xs isDefinedAt 0`	6	⇝ H	`false`
`xs.getOrElse(0, 7)`	7	⇝ B	`7`
`xs.maxBy(_._2)`	8	⇝ E	`(16, 17)`
`xs.map(p => p._1 -> -p._2)(8)`	9	⇝ F	`-9`

Lösn. uppg. 9. *Registrering i förändringsbar nyckel-värde-tabell.*

```
class FreqMapBuilder:
  private val register = scala.collection.mutable.Map.empty[String,Int]
  def toMap: Map[String, Int] = register.toMap
  def add(s: String): Unit =
    register.addOne(s -> (register.getOrElse(s, 0) + 1))

object FreqMapBuilder:
  def apply(xs: String*): FreqMapBuilder =
    val result = new FreqMapBuilder
    xs.foreach(result.add)
    result
```

Lösn. uppg. 10. *Metoden `sliding`.*

a)

```
scala> val xs = Vector("fem", "gurkor", "är", "fler", "än", "fyra", "tomater")
val xs: Vector[String] =
  Vector(fem, gurkor, är, fler, än, fyra, tomater)

scala> xs.sliding(2).toVector
val res9: Vector[Vector[String]] =
  Vector(Vector(fem, gurkor), Vector(gurkor, är), Vector(är, fler), Vector(fler

scala> xs.sliding(3).toVector
```

[2] https://sv.wikipedia.org/wiki/Hashfunktion

```
val res10: Vector[Vector[String]] =
  Vector(Vector(fem, gurkor, är), Vector(gurkor, är, fler), Vector(är, fler, är

scala> xs.sliding(10).toVector
val res11: Vector[Vector[String]] =
  Vector(Vector(fem, gurkor, är, fler, än, fyra, tomater))
```

`xs.sliding(n).toVector` skapar en sekvens som innehåller sekvenser av längden `n` som bildas genom att ta varje element och dess `n - 1` efterföljande element.

b)

```
scala> xs.sliding(2).map(ys => ys(0) -> ys(1)).toMap
val res0: Map[String,String] =
  Map(är -> fler,
      än -> fyra,
      fyra -> tomater,
      gurkor -> är,
      fem -> gurkor,
      fler -> än
  )
```

Man kan använda tabellen till att slå upp vilket som är efterföljande ord. Det fungerar eftersom alla ord är unika. Om det funnits flera likadana ord med olika efterföljande ord så hade vi behövt skapa en tabell med nycklar som mappar till en samling som registrerar efterföljande ord. Detta ska vi göra på veckans laboration.

Lösn. uppg. 11. *Läsa text från fil och webbservrar.*

a)

```
val populationOf = data.tail.map(v => v(0) -> v(1).toInt).toMap
val sizeOf       = data.tail.map(v => v(0) -> v(2).toInt).toMap
val capitalOf    = data.tail.map(v => v(0) -> v(3)).toMap
```

```
scala> capitalOf("Sverige")
res2: String = Stockholm

scala> populationOf("Sverige")
res3: Int = 9223766

scala> sizeOf("Sverige")
res4: Int = 449964
```

```
scala> val filename = "europa.txt"
scala> val xs = io.Source.fromFile(filename, "UTF-8").getLines.toVector
scala> val data = xs.map(_.split(';').toVector)
scala> data.map(_.map(_.take(15).padTo(15,' ')).mkString(" ")).foreach(println)
```

L.9.2 Extrauppgifter; träna mer

Lösn. uppg. 12. –

L.9.3 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 13. *Registrering med `groupBy`.*

a) Metoden `groupBy` skapar en nyckel-värde-tabell där värdena i tabellen är en sekvens med elementen grupperade på ett speciellt sett. Mer precist:

Resultatet av `xs.groupBy(f: K => V)` för en sekvens `xs` av typen `Vector[K]` blir en tabell av typen `Map[V,Vector[K]]` där varje element `e` i `xs` är grupperade i samma tabellvärde om de lika är enligt `f(e)`. Varje grupp får tabellnyckeln `f(e)`.

Listigt trick: Om man låter funktionen `f` vara enhetsfunktionen som avbildar varje element på sig själv, alltså `x => x`, så grupperas värdena i samma sekvens om de är lika.

```
scala> val xs = Vector(1, 1, 2, 2, 4, 4, 4).groupBy(x => x > 2)
val xs: Map[Boolean,Vector[Int]] =
  Map(false -> Vector(1, 1, 2, 2), true -> Vector(4, 4, 4))

scala> val ys = Vector(1, 1, 2, 2, 4, 4, 4).groupBy(x => x)
val ys: Map[Int,Vector[Int]] =
  Map(2 -> Vector(2, 2), 4 -> Vector(4, 4, 4), 1 -> Vector(1, 1))
```

b)

```
def freq(xs: Vector[Int]): Map[Int, Int] =
  xs.groupBy(x => x).map(p => p._1 -> p._2.size)
```

Förklaring: metoden `groupBy` skapar en tabell med par `k, v` där `v` är en sekvens med så många `k` som antalet gånger `k` förekommer i `xs`. Genom att omvandla alla värden `p._2` till storleken `p._2.size` får vi en frekvenstabell.

```
scala> freq(kasta(1000))
val res0: Map[Int,Int] =
  Map(5 -> 163, 1 -> 174, 6 -> 161, 2 -> 169, 3 -> 167, 4 -> 166)

scala> freq(kasta(1000)).toVector.sortBy(_._1).foreach(println)
(1,183)
(2,167)
(3,169)
(4,179)
(5,154)
(6,148)
```

Lösn. uppg. 14. –

L.10 Lösning `inheritance`

L.10.1 Grunduppgifter; förberedelse inför laboration

Lösn. uppg. 1. *Para ihop begrepp med beskrivning.*

bastyp	1	⇝	N	den mest generella typen i en arvshierarki
supertyp	2	⇝	O	en typ som är mer generell
subtyp	3	⇝	D	en typ som är mer specifik
körtidstyp	4	⇝	J	kan vara mer specifik än den statiska typen
dynamisk bindning	5	⇝	L	körtidstypen avgör vilken metod som körs
polymorfism	6	⇝	B	kan ha många former, t.ex. en av flera subtyper
trait	7	⇝	I	är abstrakt, kan mixas in, kan ha parametrar
inmixning	8	⇝	H	tillföra egenskaper med `with` och en trait
överskuggad medlem	9	⇝	M	medlem i subtyp ersätter medlem i supertyp
anonym klass	10	⇝	C	klass utan namn, utvidgad med extra implementation
skyddad medlem	11	⇝	K	är endast synlig i subtyper
abstrakt medlem	12	⇝	F	saknar implementation
abstrakt klass	13	⇝	E	kan ha parametrar, kan ej instansieras, kan ej mixas in
förseglad typ	14	⇝	P	subtypning utanför denna kodfil är förhindrad
referenstyp	15	⇝	A	har supertypen `AnyRef`, allokeras i heapen via referens
värdetyp	16	⇝	G	har supertypen `AnyVal`, lagras direkt på stacken

Lösn. uppg. 2. *Gemensam bastyp.*

a) `Vector[Object]`. Typen `Object` i JVM är motsvarar typen `AnyRef` som är bastyp för alla referenstyper.

b) Felmeddelande:

```
scala> grönsaker.map(_.vikt).sum
-- Error:
1 |grönsaker.map(_.vikt).sum
  |              ^^^^^^
  |             value vikt is not a member of Object - did you mean wait?
-- Error:
1 |grönsaker.map(_.vikt).sum
  |                       ^
  |ambiguous implicit arguments: both object DoubleIsFractional in object Numer
```

Det första felmeddelandet beror på att vektorns element är av typen `Object` och medlemmen `vikt` är inte definierat för denna typ. Det andra felmeddelandet är ett följdfel som beror på att en sekvens med element av typen `Object` inte kan summeras eftersom kompilatorn inte kan härleda att elementtypen är numerisk.

c) Attributet `vikt` initialiseras vid konstruktion av `Gurka` resp. `Tomat`. Värdet ges av resp. klassparameter.

d) `Vector[Grönsak]`.

e) Ja. Eftersom den statiska typen för elementen i sekvensen är `Grönsak` (den dynamiska typen kan vara godtycklig subtyp av `Grönsak`) och alla instanser av denna typ garanterat har attributet `vikt` som är av typen `Int` så kan kompilatorn vid

kompileringstid dra slutsatsen att summeringen är giltig och därmed kan kompilatorn kompilera koden till körbar maskinkod.

f)

```
scala> new Grönsak
-- Error:
1 |new Grönsak
  |    ^^^^^^^
  |    Grönsak is a trait; it cannot be instantiated
```

g)

```
scala> val anonymGrönsak = new Grönsak { val vikt = 42 }
val anonymGrönsak: Grönsak = anon$1@1edde8b6
scala> anonymGrönsak.toString
val res0: String = anon$1@1edde8b6
```

Typen är `Grönsak` och blir här en s.k. *anonym klass*, eftersom vi inte har använt en namngiven klass med **`extends`**, utan bara ”hängt på” en klasskropp inom klammerparenteser direkt vid konstruktion. När du skapar anonyma klasser måste du använda nyckelordet **`new`**.

Kompilatorn hittar på ett unikt klassnamn, här anon$1, för att hålla reda på den anonyma klassen under kompilering till maskinkod. Strängrepresentationen innehåller ett hexadecimalt heltal som är unikt för instansen, här `1edde8b6`.

h)

```
scala> new Grönsak { }
-- Error:
1 |new Grönsak { }
  |^
  |object creation impossible, since val vikt: Int in trait Grönsak is not defined
```

Lösn. uppg. 3. *Polymorfism vid arv, s.k. subtypspolymorfism.*

a)

```
def skapaDjur(): Djur =
  if math.random() > 0.5 then Ko() else Gris()
```

b)

```
class Häst extends Djur:
  def väsnas = println("Gnäääääägg")

def skapaDjur(): Djur =
   math.random() match
    case r if r < 0.33 => Ko()
    case r if r < 0.67 => Gris()
    case _             => Häst()
```

Lösn. uppg. 4. *Olika typer av heltalspar till laborationen `snake0`.*

a)

```
trait Pair[T]:
  def x: T
  def y: T
  def tuple: (T, T) = (x, y)
```

b)

- Fungerar koden ovan även utan nyckelordet **override**? Varför?

  Ja den fungerar eftersom **override** ej måste anges när ärvda *abstrakta* medlemmar implementeras i en subtyp. Abstrakta medlemmar saknar implementation och det finns inget som behöver överskuggas.

- När *måste* **override** användas?

  Det krävs **override** om du vill ge en ärvd medlem en *annan* implementation i subtypen, om denna medlemmen redan *har* en implementation i supertypen. Din nya implementation överskuggar (ersätter) den ärvda medlemmens implementation.

- Vad är fördelen resp. nackdelen med att använda **override** även när det inte är nödvändigt?

  Fördel: du får hjälp av kompilatorn att kontrollera att du verkligen implementerar en ärvd medlem och inte t.ex. råkat stava medlemmens namn fel.

  Nackdel: mer att skriva och därmed även längre att läsa.

c)

```
case class Dim(x: Int, y: Int) extends Pair[Int]
object Dim:
  def apply(dim: (Int, Int)): Dim = Dim(dim._1, dim._2)
```

d)

```
case class Pos private (x: Int, y: Int, dim: Dim) extends Pair[Int]:
  def +(p: Pair[Int]): Pos = Pos(x + p.x, y + p.y, dim)
  def -(p: Pair[Int]): Pos = Pos(x - p.x, y - p.y, dim)

object Pos:
  def apply(x: Int, y: Int, dim: Dim): Pos =
    import java.lang.Math.floorMod as mod
    new Pos(mod(x, dim.x), mod(y, dim.y), dim) //OBS: new nödvändig här!

  def random(dim: Dim): Pos =
    import scala.util.Random.nextInt as rni
    Pos(rni(dim.x), rni(dim.y), dim)
```

e) Om du glömmer skriva **new** explicit i kompanjonsobjektets `apply`-metod så blir det ett rekursivt anrop som resulterar i en oändlig loop vid körtid. Med **new** så är det garanterat den privata primärkonstruktorn för `Pos` som anropas.

I `Dim.apply` så skiljer sig parametertyperna åt mellan fabriksmetoden och primärkonstruktorn och kompilatorn väljer då primärkonstruktorn eftersom den passar med de givna två separata heltalen och inte med en 2-tupel.

f)

```
enum Dir(val x: Int, val y: Int) extends Pair[Int]:
  case North extends Dir( 0, -1)
  case South extends Dir( 0,  1)
  case East  extends Dir( 1,  0)
  case West  extends Dir(-1,  0)
export Dir.*  // gör så att North etc blir synliga i paketet snake
```

Lösn. uppg. 5. *Supertyp med parameter.*

a)

```
val person = new Person("Person1")
val akademiker = new Akademiker("Person2", "LTH")
val student = new Student("Person3", "LTH", "D")
val forskare = new Forskare("Person4", "LTH", "Doktorand")
```

b)

```
val vec = Vector(person, akademiker, student, forskare)
for(i <- vec){ print(i.toString + i.namn) }
```

c) Felmeddelande vid instansiering av `abstract class Akademiker`:
`Akademiker is abstract; it cannot be instantiated`

Det går *inte* lika bra med en `trait` i det speciella fallet `Akademiker`, eftersom en trait inte får skicka vidare parametrar till en supertyp. Felmeddelande:
`trait Akademiker may not call constructor of trait Person`

```
trait Person(val namn: String)

abstract class Akademiker(
  namn: String,
  val universitet: String) extends Person(namn)

class Student(
  namn: String,
  universitet: String,
  program: String) extends Akademiker(namn, universitet)

class Forskare(
  namn: String,
  universitet: String,
  titel: String) extends Akademiker(namn, universitet)
```

d)

```
scala>
     |trait Person(val namn: String)
     |
     | abstract class Akademiker(
     |   namn: String,
     |   val universitet: String) extends Person(namn)
     |
```

```
     | case class Student(
     |   namn: String,
     |   universitet: String,
     |   program: String) extends Akademiker(namn, universitet)
     |
     | case class Forskare(
     |   namn: String,
     |   universitet: String,
     |   titel: String) extends Akademiker(namn, universitet)
-- Error:
8 |  namn: String,
  |  ^
  |  error overriding value namn in trait Person of type String;
  |    value namn of type String needs `override` modifier
-- Error:
9 |  universitet: String,
  |  ^
  |  error overriding value universitet in class Akademiker of type String;
  |    value universitet of type String needs `override` modifier
-- Error:
13 |  namn: String,
   |  ^
   |  error overriding value namn in trait Person of type String;
   |    value namn of type String needs `override` modifier
-- Error:
14 |  universitet: String,
   |  ^
   |  error overriding value universitet in class Akademiker of type String;
   |    value universitet of type String needs `override` modifier
```

```
trait Person(val namn: String)

abstract class Akademiker(
  namn: String,
  val universitet: String) extends Person(namn)

case class Student(
  override val namn: String,
  override val universitet: String,
  program: String) extends Akademiker(namn, universitet)

case class Forskare(
  override val namn: String,
  override val universitet: String,
  titel: String) extends Akademiker(namn, universitet)
```

```
scala> val ps = Vector(Student("Kim", "Lund", "D"), Forskare("Herz", "Lund", "Dr"))
val ps: Vector[Akademiker] = Vector(Student(Kim,Lund,D), Forskare(Herz,Lund,Dr))
scala> ps :+ new Person("Abstrakt") {}
val res0: Vector[Person] =
  Vector(Student(Kim,Lund,D), Forskare(Herz,Lund,Dr), anon1@1941bbf3)
```

e)

```
trait Person:
  val namn: String
  val nbr: Long

trait Akademiker extends Person:
  val universitet: String

case class Student(
  namn: String,
  nbr: Long,
  universitet: String,
  program: String) extends Akademiker

case class Forskare(
  namn: String,
  nbr: Long,
  universitet: String,
  titel: String) extends Akademiker

case class IckeAkademiker(
    namn: String,
    nbr: Long) extends Person
```

L.10.2 Extrauppgifter; träna mer

Lösn. uppg. 6. *Bastypen* `Shape` *och subtyperna* `Rectangle` *och* `Circle`.

a)

```
val c1 = Circle(Point(1, 1), 42)
val r1 = Rectangle(Point(3, 3), 20, 30)
c1.move(2, 3)
r1.move(3, 2)
```

b)

```
case class Point(x: Double, y: Double):
  def move(dx: Double, dy: Double): Point = Point(x + dx, y + dy)
  def moveTo(x: Double, y: Double): Point = Point(x, y)

trait Shape:
  def pos: Point
  def move(dx: Double, dy: Double): Shape
  def moveTo(x: Double, y: Double): Shape

case class Rectangle(pos: Point, width: Double, height: Double) extends Shape:
  def move(dx: Double, dy: Double): Shape = copy(pos = pos.move(dx, dy))
  def moveTo(x: Double, y: Double): Shape = copy(pos.moveTo(x, y))

case class Circle(pos: Point, radius: Double) extends Shape:
  def move(dx: Double, dy: Double): Shape = copy(pos = pos.move(dx, dy))
  def moveTo(x: Double, y: Double): Shape = copy(pos.moveTo(x, y))
```

c) `def distanceTo(that: Point): Double = math.hypot(that.x - x, that.y - y)`

d) `def distanceTo(that: Shape): Double = pos.distanceTo(that.pos).`

e)

```
case class Point(x: Double, y: Double):
  def move(dx: Double, dy: Double): Point = Point(x + dx, y + dy)
  def moveTo(x: Double, y: Double): Point = Point(x, y)
  infix def distanceTo(that: Point): Double = math.hypot(that.x - x, that.y - y)

trait Shape:
  def pos: Point
  def move(dx: Double, dy: Double): Shape
  def moveTo(x: Double, y: Double): Shape
  infix def distanceTo(that: Shape): Double = pos.distanceTo(that.pos)

case class Rectangle(pos: Point, width: Double, height: Double) extends Shape:
  def move(dx: Double, dy: Double): Shape = copy(pos = pos.move(dx, dy))
  def moveTo(x: Double, y: Double): Shape = copy(pos.moveTo(x, y))

case class Circle(pos: Point, radius: Double) extends Shape:
  def move(dx: Double, dy: Double): Shape = copy(pos = pos.move(dx, dy))
  def moveTo(x: Double, y: Double): Shape = copy(pos.moveTo(x, y))
```

L.10.3 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 7. *Inmixning.*

a) Det finns många olika sätt, några exempellösningar:

```
val antalFlygånkor: Int =
  fyle.count(f => f.isInstanceOf[Ånka] && f.ärFlygkunnig)
```

```
val antalFlygånkor: Int =
  fyle.filter(f => f.isInstanceOf[Ånka] && f.ärFlygkunnig).size
```

```
val antalFlygånkor: Int =
  fyle.collect{case f: Ånka if f.ärFlygkunnig}.size
```

```
val antalFlygånkor: Int = fyle.map(_ match
  case f: Ånka if f.ärFlygkunnig => 1
  case _ => 0
).sum
```

b)

```
val antalKrax: Int = fyle.filter(f => !f.ärSimkunnig).size * 2
val antalKvack: Int = fyle.filter(f => f.ärSimkunnig).size * 4
```

Lösn. uppg. 8. *Finala klasser.*

a) Sätt `final` framför `class` i klasserna.

b) error: illegal inheritance from final class Kråga.

Lösn. uppg. 9. *Accessregler vid arv och nyckelordet* ***`protected`****.*

a)

```
2 |  def avslöja = minHemlis
  |                ^^^^^^^^^
  |                Not found: minHemlis
```

b)

```
scala> class Sub extends Super:
         def kryptisk = vårHemlis * math.Pi
scala> (new Sub).vårHemlis
-- Error:
1 |(new Sub).vårHemlis
  |^^^^^^^^^^^^^^^^^^^
  |value vårHemlis in trait Super cannot be accessed as a member of Sub.
  | Access to protected value vårHemlis not permitted because enclosing object
  | is not a subclass of trait Super where target is defined
```

c) Ja.

Lösn. uppg. 10. *Användning av* ***`protected`****.*

a) I Fyle:

```
protected var räknaLäte: Int = 0
def väsnas: Unit = { print(läte * 2); räknaLäte += 2 }
```

I Ånka: `override def väsnas = { print(läte * 4); räknaLäte += 4 }`

b) `def antalLäten: Int = räknaLäte`

c) Om en klass som representerar en fågel som skulle ge ifrån sig fler/färre läten än en vanlig `Fyle`, behöver `väsnas` ändras. Denna metod behöver tillgång till `räknaLäte`, vilken inte får vara `private`.

d) Räknar-variabeln ska inte kunna påverkas i någon annan del av programmet.

Lösn. uppg. 11. *Inmixning av egenskaper.*

```
trait Fyle:
  val läte: String
  def väsnas: Unit = { print(läte * 2); räknaLäte += 2 }
  protected var räknaLäte: Int = 0
  val ärSimkunnig: Boolean
  val ärFlygkunnig: Boolean
  val ärStor : Boolean
  def antalLäten: Int = räknaLäte

trait KanSimma { val ärSimkunnig = true }
trait KanInteSimma { val ärSimkunnig = false }
trait KanFlyga { val ärFlygkunnig = true }
```

```
trait KanKanskeFlyga { val ärFlygkunnig = math.random() < 0.8 }
trait KanKanskeSimma { val ärSimkunnig = math.random() < 0.2 }
trait ÄrStor { val ärStor = true }
trait ÄrLiten { val ärStor = false }

final class Kråga extends Fyle, KanFlyga, KanInteSimma, ÄrStor:
  val läte = "krax"

final class Ånka extends Fyle, KanSimma, KanKanskeFlyga, ÄrStor:
  val läte = "kvack"
  override def väsnas = { print(läte * 4); räknaLäte += 4 }

final class Pjodd extends Fyle, KanFlyga, KanKanskeSimma, ÄrLiten:
  val läte = "kvitter"
  override def väsnas = { print(läte * 8); räknaLäte += 8 }
```

I REPL:

```
val fyle = Vector.fill(42)(
  if math.random() < 0.33 then Kråga()
  else if math.random() < 0.5 then Ånka()
  else Pjodd()
)
fyle.filter(f => f.isInstanceOf[Kråga]).size * 2
fyle.filter(f => f.isInstanceOf[Ånka]).size * 4
fyle.filter(f => f.isInstanceOf[Pjodd]).size * 8
```

L.11 Lösning `context`

L.11.1 Grunduppgifter; förberedelse inför laboration

Lösn. uppg. 1. *Kontextparameter.*

a)

```
scala> given Int = 0
val res0: Int = 83

scala> f(using 41)
val res1: Int = 42

scala> g(41)(using 42)
val res2: Int = 83
```

Om man glömmer **`using`** vid explicit kontextargument blir det kompileringsfel. Kompilatorn blir ”förvirrad” och tror att du försöker ge ett ”vanligt” argument till en (i detta fallet) icke-existerande ”vanlig” parameterlista.

```
scala> f(41)
-- [E050] Type Error: ----------------------------------------------
1 |f(41)
  |^
  |method f does not take more parameters
  |
  | longer explanation available when compiling with `-explain`
1 error found

scala> :setting -explain

scala> f(41)
-- [E050] Type Error: ----------------------------------------------
1 |f(41)
  |^
  |method f does not take more parameters
  |-----------------------------------------------------------------
    | Explanation (enabled by `-explain`)
  |- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  | You have specified more parameter lists than defined in the
    method definition(s).
   -----------------------------------------------------------------

scala> g(41)(42)
-- [E050] Type Error: ----------------------------------------------
1 |g(41)(42)
  |^^^^^
  |method g does not take more parameters
  |-----------------------------------------------------------------
    | Explanation (enabled by `-explain`)
  |- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  | You have specified more parameter lists than defined in the
    method definition(s).
   -----------------------------------------------------------------
```

Det är inte vanligt att ange **using**-parametrar explicit; det vanligaste är att låta kompilatorn framkalla ett givet värde.

b) Det givna värdet `0` binds till motsvarande kontextparameter, som ska vara deklarerad i en egen parameterlista som börjar med **using**.

```
scala> f
val res3: Int = 1

scala> g(42)
val res4: Int = 42
```

c) Nej, det blir kompileringsfel om man försöker blanda vanliga parametrar och kontextparametrar i en och samma parameterlista:

```
scala> def h(i: Int, using j: Int) = i + j
-- [E040] Syntax Error: ----------------------------------------------
1 |def h(i: Int, using j: Int) = i + j
  |                   ^
  |                   ':' expected, but identifier found
```

Det är ett medvetet val att kräva separata parameterlistor, så att det inte ska uppstå förvirring om huruvida en vanlig parameter eller kontextparameter avses.

Lösn. uppg. 2. *Flera olika givna värden i lokal kontext.*

a) 2

b) 43

c) När kompilatorn försöker framkalla ett givet värde att automatiskt använda som argument till **using**-parametern `dx`, så letar den i den kontext som är närmast anropet först. Om det finns ett givet värdet i kompanjonsobjektet för parametertypen så tar kompilatorn detta i sista hand, om inget annat givet värde hittas närmare anropet.

Lösn. uppg. 3. *Lösning på konfigurationsproblemet med hjälp av givna värden.*

Nedan visas test av de tre olika lösningarna som givits i uppg. a)

Efter varje test diskuteras tillhörande för- och nackdelar, som efterfrågas i uppg. b)

```
def testGlobalVar(useDefault: Boolean = true) =
  import GlobalVar.*
  if useDefault then println(greetMsg) else
    GreetConfig.config = GreetConfig("Godmorgon", "världen")
    println(greetMsg)
```

Eftersom `config` här är en förändringsbar variabel, så kan en ändring på ett ställe påverka helt andra delar av programmet, vilket ibland kan vara en fördel, men ofta en nackdelen eftersom det kan vara svårt att förstå vad som händer bara genom att läsa en enskild del av programmet – en förändring av `config` kan ju ske varsomhelst. Det är lätt att glömma ändra till baka till default-värdet, om det är det som förväntas.

```
1 scala> testGlobalVar(); testGlobalVar(false); testGlobalVar()
2 Hello World!
3 Godmorgon världen!
4 Godmorgon världen!
```

```
1 def testDefaultArgs(useDefault: Boolean = true) =
2   import DefaultArgs.*
3   if useDefault then println(greetMsg()) else
4     println(greetMsg(GreetConfig("Godmorgon","världen")))
```

Här sker ingen tillståndsförändring och default-användning är enkel, men det går inte enkelt att göra avsteg från default som gäller i en lokal kontext; vid *varje* enskilt anrop behöver du explicit ange alla de argument som inte ska vara default, så som visas på rad 4 ovan. Ändring av default har bara lokal påverkan. Om alla argument ska följa default, så gäller det att inte glömma anropa med tomt parentespar: `greetMsg()`. (Vad händer annars?)

```
1 scala> testDefaultArgs(); testDefaultArgs(false); testDefaultArgs()
2 Hello World!
3 Godmorgon världen!
4 Hello World!
```

Med kontextparametrar är flexibiliteten större; **`using`**-parametrar låter användaren själv styra vad som gäller i olika sammanhang och själva anropet blir enkelt oavsett om det är default-värdet eller andra, i den lokala kontexten, givna värden som önskas. Ändring av default har bara lokal påverkan, men den har påverkan på godtyckligt många anrop i den lokala kontexten – argument som skiljer sig kan alltså vara givna en gång utan att behöva upprepas vid varje anrop. Vid anrop där man vill låta kompilatorn framkallar givna värden för kontextparametern ska inga parenteser användas, och anropen bli därmed korta och enkla.

```
def testGivenVal(using g: GivenVal.GreetConfig) = println(g.greetMsg)
```

```
1  scala> testGivenVal
2  Hello World
3
4  scala> def localContext =
5           import GivenVal.*
6           given GreetConfig = GreetConfig("Godmorgon","världen")
7           testGivenVal
8
9  scala> localContext
10 Godmorgon världen
11
12 scala> testGivenVal
13 Hello World
```

c) Kompilatorn framkallar ett givet värde i den lokala kontexten:

```
1 scala> summon[GivenVal.GreetConfig]
2 val res0: GivenVal.GreetConfig = GreetConfig(Hello,World)
```

Kompilatorn följer denna prioritetsordning i sökandet efter ett unikt givet värde:

1. **Explicita** argument till kontextparametrar märkta med **`using`**
2. **`given`** och **`import given`** ... i aktuell namnrymd (eng. *current scope*)
3. **`given`**-värden i **kompanjonsobjekt** för den använda typen.

Om flera givna värden kan framkallas för typer som ingår i en gemensam arvshierarki så väljer kompilatorn det givna värdet som är av den *mest specifika* typen.

d) Det blir kompileringsfel om kompilatorn inte hittar ett givet värde för den typ som avses.

```
scala> summon[Long]
-- Error: ----------------------------------------------------------
1 |summon[Long]
  |           ^
  |           no given instance of type Long was found for parameter x of
  |           method summon in object Predef
1 error found
```

e) Ja! Det får *inte* vara tvetydigt vilket givet värde som ska framkallas:

```
scala> def tvetydigt =
     |   given a: Int = 42
     |   given b: Int = 43
     |   summon[Int]
     |
-- Error: ----------------------------------------------------------------
4 |  summon[Int]
  |            ^
  |ambiguous given instances: both given instance b and given instance a
  |match type Int of parameter x of method summon in object Predef
1 error found
```

Läs mer om kontexuella abstraktioner här:
https://docs.scala-lang.org/scala3/reference/contextual/

L.11.2 Extrauppgifter; träna mer

Lösn. uppg. 4. *Kontextparameter och givet värde.*

a)

```
scala> add(1)
-- Error: ----------------------------------------------------------
1 |add(1)
  |     ^
  |    no given instance of type Int was found for parameter y of method add
1 error found
```

b) Nu finns ett givet värde som kompilatorn automatiskt kan fylla i på platsen vid anropet.

c) `def sub(x: Int)(using y: Int) = x - y`

L.11.3 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 5. *Varians och typgränser.*

a) Gör lådan flexibel i sin typparameter med ett + före typparametern enligt nedan.

```
case class Box[+A](x: A)
```

Kompilatorn tillämpar reglerna för kovarians eftersom typparametern har ett plustecken framför sig: `Box[Cat]` är en suptyp till `Box[Any]` om `Cat` är en subtyp till `Any`, vilket den ju är eftersom alla typer är subtyp till `Any`.

b) Förklaringen till beteendet har med olika varians att göra:

- Samlingen `Vector` är kovariant och därmed flexibel i sin typparameter (liksom andra oföränderliga sekvenser i Scalas standardbibliotek). Kompilatorn betraktar därmed `Vector[Cat]` som en subtyp till `Vector[Pet]` eftersom `Cat` är en subtyp till `Pet`. På platser i koden där en `Vector[Pet]` krävs så anses `Vector[Cat]` överensstämma med (eng. *conforms to*) `Vector[Pet]` och får därmed duga på dessa platser.
- En mängd har en apply-metod från elemttypen till `Boolean` som ger innehållstest. Av det skälet har man låtit `Set[T]` ärva `Function1[T, Boolean]` som är deklarerad kontravariant i `T`, så att en mängd kan användas där en `T => Boolean` förväntas. Även om det skulle vara praktiskt om Set[T] vore kovariant i `T`, i likhet med `Vector`, `List`, `Seq` etc, så kan inte `T` vara både kovariant och kontravariant på en och samma gång. Man har därför valt att göra `Set` invariant och därmed är mängder ej flexibla i sin typparameter. `Set[Cat]` är alltså *inte* en subtyp till `Set[Pet]` *även* om `Cat` är en subtyp till `Pet`, vilket ger kompileringsfel i uppgiftens exempel. Se även https://stackoverflow.com/questions/676615/why-is-scalas-immutable-set-not-covariant-in-its-type
- Med `:settings -explain` ger kompilatorn en längre utskrift som förklarar den bevisföring som skedde under kompileringens typkontroll.

c) Det blir kompileringsfel då metoden `isHealthy` ej existerar för godtycklig typ.

d) Lägg till en övre gräns som garanterar att metoden `isHealthy` finns för alla typer som kan bindas till typparametern `A`:

```
class Vet[-A <: Pet]:
  def treat(x: A): Unit = x.isHealthy = true
```

Kompilatorn garanterar alltså att typparametern `A` är ”mindre än eller lika med” `Pet`.

e) Veterinären `Vet` är flexibel i sin typparameter och minustecknet anger kontravarians och därmed att `Vet[Pet]` är en subtyp till `Vet[Cat]` då `Cat` är en subtyp till `Pet`. Detta kan demonstreras med nedan exempel:

```
scala> val pinkPanther = Cat()
val pinkPanther: Cat = Cat@33e7ece5

scala> val somePet: Pet = Cat()
val somePet: Pet = Cat@57f1cb96

scala> val catVet = Vet[Cat]()
val catVet: Vet[Cat] = Vet@1060e784

scala> pinkPanther.isHealthy = false

scala> catVet.treat(pinkPanther)

scala> pinkPanther.isHealthy
val res2: Boolean = true

scala> somePet.isHealthy = false

scala> catVet.treat(somePet)
-- [E007] Type Mismatch Error: --------------
```

```
1 |catVet.treat(somePet)
  |             ^^^^^^^
  |             Found:    (somePet : Pet)
  |             Required: Cat

scala> val powerVet = Vet[Pet]()
val powerVet: Vet[Pet] = Vet@2eb90ae9

scala> pinkPanther.isHealthy = false

scala> powerVet.treat(pinkPanther)

scala> pinkPanther.isHealthy
val res3: Boolean = true

scala> val pluto = Dog()
val pluto: Dog = Dog@6f27db5d

scala> pluto.isHealthy = false

scala> powerVet.treat(pluto)

scala> pluto.isHealthy
val res4: Boolean = true
```

Lösn. uppg. 6. *Typklasser och kontextparametrar.*

a)

- Först deklarerar vi en **`trait`**, `CanCompare`, med en generisk typparameter `T`. Den innehåller en abstrakt metod `compare` som tar två parametrar av typen `T` och returnerar en `Int`.
- Sedan definieras en metod `sort` som också tar en generisk typparameter `T`. Metoden tar två parametrar, a och b av typen T, samt en **`using`** parameter cc som måste vara en instans av `CanCompare[T]`. Inuti metoden används compare-metoden från CanCompare för att bestämma om a och b ska byta plats eller inte.
- När vi försöker köra `sort(42, 41)` så får vi felmeddelande av kompilatorn. Anledning till detta är att det inte finns en given instans av CanCompare[Int].
- Vi löser detta på nästa rad med **`given`** `intComparator` som är av typen CanCompare[Int]. Vi definierar även vår abstrakta metod `compare` från CanCompare med **`override def`** `compare`... När vi kör `sort(42,41)` på nästa rad fungerar det nu som det ska och vi får tillbaka `(Int, Int) = (41, 42)`
- När vi försöker köra sort med argument av typen `Double` får vi ett liknande felmeddelande som vi fick tidigare, och av samma anledning att det inte finns en CanCompare för typen Double.

b)

```
scala> given doubleComparator: CanCompare[Double] with
         override def compare(a: Double, b: Double): Int = (a - b).toInt
```

c)

```
scala> given stringComparator: CanCompare[String] with
         override def compare(a: String, b: String): Int = (a.compareTo(b))
```

Lösn. uppg. 7. *Användning av given ordning.*— TODO!!!

Lösn. uppg. 8. *Skapa egen implicit ordning med `Ordering`.*

a) TODO!!!

b) TODO!!!

c) TODO!!!

d) TODO!!!

Lösn. uppg. 9. *Specialanpassad ordning genom att ärva från `Ordered`*

a)

```
case class  Team(name: String, rank: Int) extends Ordered[Team]:
  override def compare(that: Team): Int = -rank.compare(that.rank)
```

b)

```
scala> Team("fnatic",1499) < Team("gurka", 2)
val res1: Boolean = true
```

c) Ad hoc polymorfism är mer flexibel. TODO!!! mer diskussion om likheter och skillnader här...

L.12 Lösning `extra`

L.12.1 Uppgifter om sökning och sortering

Lösn. uppg. 1. *Tidmätning.*

a) Exekvera koden och du bör finna att det tar längre tid att hitta värdet 1 i vårt Set s än i vektorn v.

b)

En vektor har en sekventiell ordning som find kan använda, medan `Set` är internt ordnad på ett annat sätt för att innehållskontroll ska gå extra snabbt. Anledningen att det tar tid för `find` på `Set` är att det först måste skapas en iterator innan vår mängd kan gås igenom från början till slut. Metoden `contains` på `Set` däremot är rasande snabb beroende på den interna strukturen hos objekt av typen `Set` (som är smart designad med s.k. hash-koder, där det går lika snabbt att hitta ett element oavsett vart det befinner sig).

Lösn. uppg. 2. *Sökning med inbyggda sökmetoder.*

a) Förslag på test av `indexOfSlice`:

```
scala> List(1,2,3,35,1,23).indexOfSlice(List(35,1,23))
res73: Int = 3
scala> List(1,2,3,35,1,23).indexOfSlice(List(35,1,3))
res74: Int = -1
```

b) Förslag på test av `lastIndexOfSlice`:

```
Vector(1,2,3,4,1,2).lastIndexOfSlice(Vector(1,2))
res2: Int = 4
Vector("apa", "banan", "majs", "banan").lastIndexOfSlice(Vector("banan"))
res3: Int = 3
Vector("apa", "banan", "majs", "banan").lastIndexOfSlice(Vector("banand"))
res4: Int = -1
```

c) Observera att metoden `search` antar att samlingen är sorterad i stigande ordning. När vi inverterar ordningen kan `search` oftast inte hitta det vi letar efter, eftersom den kommer leta i fel halva av samlingen.

```
scala> val udda = (1 to 1000000 by 2).toVector
scala> import scala.collection.Searching._
scala> udda.search(udda.last)
res18: collection.Searching.SearchResult = Found(499999)
//Search hittar det sista elementet på plats 499999 i samlingen.

scala> udda.search(udda.last + 1)
res19: collection.Searching.SearchResult = InsertionPoint(500000)
//Search kan inte hitta udda.last + 1 eftersom det inte existerar i samlingen
//och returnerar således ett objekt av typen InsertionPoint med värdet 500000.
//Vårt element udda.last + 1 hade alltså legat på plats 500000 om det funnits.

scala> udda.reverse.search(udda(0))
res20: collection.Searching.SearchResult = InsertionPoint(0)
//Som förklarat innan så förutsätter search att listan är sorterad i stigande
//ordning, så den kan inte hitta elementet udda(0) = 1 när listan är inverterad
```

```
scala> def lin(x: Int, xs: Seq[Int]) = xs.indexOf(x)
scala> def bin(x: Int, xs: Seq[Int]) = xs.search(x) match
  case Found(i) => i
  case InsertionPoint(i) => -i

//Definierar en metod bin som använder sig av metoden search på en sekvens.
//Den ser sedan till med hjälp av "pattern matching" att bara returnera positio
//i, och inte ett objekt av typen Found eller InsertionPoint.

scala> timed{ lin(udda.last, udda) }
time: 42.294821 ms
res22: (Int, Long) = (499999,42294821)
//För att hitta udda.last = 499999 med linjärsökning tog det ca 42ms.

scala> timed{ bin(udda.last, udda) }
time: 0.147314 ms
res23: (Int, Long) = (499999,147314)
//Binärsökning för att hitta värdet 499999 tog extremt mycket kortare tid.
//Detta för att vid varje steg i binärsökningen halveras mängden tal som
//sökningen måste kolla i. Detta är dock ett extremfall eftersom vi söker
//talet längst bak i listan. Om vi istället gjort en linjärsökning efter
//det första talet 1, hade detta gått minst lika snabbt som binärsökning.
```

d) Det behövs $log_2(n)$ jämförelser. Detta eftersom att vi hela tiden halverar antalet element i listan vi behöver söka igenom. Så efter första jämförelsen har vi $\frac{n}{2}$ element kvar. Efter andra jämförelsen har vi $\frac{n}{2*2}$ element kvar etc. När vi bara har ett element kvar har vi hittat det vi söker efter, och har då gjort b antal jämförelser. Ekvationen ser då ut på följande vis:

$$\frac{n}{2^b} = 1$$

Enligt lagarna för logaritmer kan vi nu komma fram till vad b är:

$$log_2(n) = b$$

Lösn. uppg. 3. *Sök bland LTH:s kurser med linjärsökning.*

a) Första raden innehåller kolumnnamnen `Kurskod KursSve KursEng Hskpoang Niva`. Därefter kommer en rad för varje kurs med kursdata enligt kolumnnamnen.

b) Koden laddar ner data och skapar en vektor med instanser av case-klassen `Course` med hjälp av metoden `fromLine`. Eftersom variabeln `lth` är deklarerad som **`lazy`** kommer inte `download()` bli anropad förrän första gången som variablen `lth` refereras. Antalet kurser ges av:

```
scala> val n = courses.lth.length
n: Int = 1104
```

c)

```
1 scala> def isCS(s: String) = s.startsWith("EDA") || s.startsWith("ETS")
2 scala> val x = courses.lth.find(c => isCS(c.code) && c.level == "G2")
3 x: Option[courses.Course] = Some(Course(EDAF05,Algoritmer, datastrukturer och
4    komplexitet,Algorithms, Data Structures and Complexity,5.0,G2))
```

d)

```
def linearSearch[T](xs: Seq[T])(p: T => Boolean): Int =
   var i = 0
   while(i < xs.length && !p(xs(i))) i += 1
   if (i < xs.length) i else -1
```

e)

```
def rndCode: String =
   //randomizes from 0 to n (inclusive)
   def rnd(n: Int) = (math.random() * (n + 1)).toInt

   def letter = (rnd('Z' - 'A') + 'A').toChar
   def dig = ('0' + rnd(9)).toChar
   Seq(letter, letter, letter, letter, dig, dig).mkString
```

f)

```
val xs = Vector.fill(500000)(rndCode)
val(ixs, elapsedLin) =
  timed { xs.map(x => linearSearch(courses.lth)(_.code == x)) }
val found = ixs.filterNot(_== -1).size
```

g)

```
def linearSearch[T](xs: Seq[T])(p: T => Boolean): Int = xs.indexWhere(p)
```

Lösn. uppg. 4. *Sök bland LTH:s kurser med binärsökning.*

a) —

b)

```
def binarySearch(xs: Seq[String], key: String): Int =
  var (low, high) = (0, xs.size - 1)
  var found = false
  var mid = -1

  while (low <= high && !found) do
    mid = (low + high) / 2
    if xs(mid) == key then found = true
    else if xs(mid) < key then low = mid + 1
    else high = mid - 1
  end while
  if found then  mid else -(low + 1)
```

c) Med en i7-3770K @ 3.50Hz tog sökningarna följande tid:

- Binärsökning: `time: 142.6 ms`
- Linjärsökning: `time: 3316.5 ms`

Med en i7-8700T @ 2.40GHz tog sökningarna följande tid:

- Binärsökning: `time: 81.5 ms`
- Linjärsökning: `time: 5138.6 ms`

d) Binärsökningen var ca 23 gånger snabbare på en i7-3770K @ 3.50Hz och ca 63 gånger snabbare på en i7-8700T CPU @ 2.40GHz.

Lösn. uppg. 5. *Insättningssortering.*

a) —

b)

```
def insertionSort(xs: Seq[Int]): Seq[Int] =
  val result = scala.collection.mutable.ArrayBuffer.empty[Int]
  for e <- xs do
    var pos = 0
    while pos < result.size && result(pos) < e do pos += 1
    result.insert(pos,e)
  end for
  result.toVector
```

Lösn. uppg. 6. *Sortering på plats.*

```
def selectionSortInPlace(xs: Array[String]): Unit =
  def indexOfMin(startFrom: Int): Int =
    var minPos = startFrom
    var i = startFrom + 1
    while (i < xs.size) do
      if (xs(i) < xs(minPos)) minPos = i
      i += 1
    end while
    minPos
  end indexOfMin

  def swapIndex(i1: Int, i2: Int): Unit =
    val temp = xs(i1)
    xs(i1) = xs(i2)
    xs(i2) = temp
  end swapIndex

  for i <- 0 to xs.size - 1 do swapIndex(i, indexOfMin(i))
end selectionSortInPlace
```

Lösn. uppg. 7. *Undersök om en sekvens är sorterad.*

a) Det tar i värsta fall $O(n * log(n))$ för timsort att sortera listan med n element. Sedan krävs n stycken jämförelser mellan den sorterade och osorterade listan. Det totala antalet jämförelser i värstafallet uppgår därför till max $n + n * log(n)$. För 10^6 element blir det ca 10^7 jämförelser.

```
scala> val n = 1E6
val n: Double = 1000000.0

scala> def worstCase(n: Double) = n + n * math.log(n)
def worstCase(n: Double): Double

scala> println(s"i värsta fall med n=$n så blir det ${worstCase(n)} jämförelse
i värsta fall med n=1000000.0 så blir det 1.4815510557964273E7 jämförelser
```

b) En mer effektiv version av `isSorted` som avbryter sökningen när ett osorterat element upptäcks:

```
def isSorted(xs: Vector[Int]): Boolean =
  if xs.length > 1 then
    var i = 0
    var result = true
    while i < xs.length-1 && result do
      if xs(i) > xs(i+1) then result = false
      i += 1
    end while
    result
  else true
end isSorted
```

c) I värsta fall behöver man göra $n - 1$ parvisa jämförelser, om alla ligger i sorterad ordning utom den sista.

d) 2-tupeln är av typen `(Int, Int)`.

```
def isSorted(xs: Vector[Int]): Boolean =
  xs.zip(xs.tail).forall(x => x._1 <= x._2)
```

Lösn. uppg. 8. *Insättningssortering på plats.*

```
def insertionSort(xs: Array[Int]): Unit =
  for elem <- 1 until xs.length if xs.length > 0 do
    var pos = elem
    while pos > 0 && xs(pos) < xs(pos - 1) do
      val temp = xs(pos -1)
      xs(pos -1) = xs(pos)
      xs(pos) = temp
      pos -= 1
    end while
  end for
end insertionSort
```

Lösn. uppg. 9. *Sortering till ny sekvens med urvalssortering.*

```
def selectionSort(xs: Seq[String]): Seq[String] =
  def indexOfMin(xs: Seq[String]): Int = xs.indexOf(xs.min)
  val unsorted = xs.toBuffer
  val result = scala.collection.mutable.ArrayBuffer.empty[String]
  while !unsorted.isEmpty do
    val minPos = indexOfMin(unsorted)
    val elem = unsorted.remove(minPos)
    result.append(elem)
  end while
  result.toVector
end selectionSort
```

L.12.2 Uppgifter om trådar och jämlöpande exekvering

Lösn. uppg. 10. *Trådar.*

a) -

b) `java.lang.IllegalThreadStateException`. Det går inte att starta en tråd mer än en gång. Tråden kan därför inte startas om när den redan har exekverats.

c) När `start` anropas exekveras koden i `run` parallellt. Därför skrivs `Gurka` och `Tomat` ut omlöpande. Om istället `run` anropas direkt blir det inte jämnlöpande exekvering och `Gurka` skrivs ut 100 gånger, sedan skrivs `Tomat` ut 100 gånger.

d) `Thread.sleep` pausar inte tråden i exakt den tiden som angets. Alltså kommer det skrivas ut `zzz snark hej!` i de flesta fall, men det är inte garanterat.

Lösn. uppg. 11. *Jämlöpande variabeluppdatering.*

a) I `slösaSpara` hämtas saldot, ändras och placeras tillbaka i minnet - fördröjs - upprepas. Om `bamse` blir klar med att ladda, ändra och lagra innan skutt gör detsamma blir det problem, då de tävlar om vem som får uppdatera (eng. *race contiion*). Problemet innan en tråd kan lagra det förändrade värdet laddar den andra tråden det gamla värdet. Bara en av dessa trådar vinner racet och får lagra sitt ändrade tal och den andra ändringen går förlorad. `skutt` och `bamse` blir alltså upprörda för att inte alla dess uttag och insättningar registreras.

Lösn. uppg. 12. *Trådsäkra `AtomicInteger`.*

Nu är `farmor`-tråden garanterad att kunna ladda saldot, ta ut pengar/ändra och lagra innan `vargen`-tråden kan skriva över resultatet. I `slösaSpara` pausas tråden i en millisekund så `vargen`-tråden kan hinna ta ut pengar innan `farmor`-sätter hinner sätta in pengar igen och saldot blir negativt. Dock kommer alla uttag och insättningar registreras eftersom operationerna är atomära och saldot kommer återställas till noll, utan att insättningar går förlorade.

Lösn. uppg. 13. *Jämlöpande exekvering med `scala.concurrent.Future`.*

a) error: Cannot find an implicit ExecutionContext. Future behöver en ExecutionContext för att kunna köras. `f` är av typen Future[Unit].

b) Funktionen `printLater` har en Future, vilket innebär att när både `printLater` och `println` anropas i foreach-loopen exekveras de jämnlöpande. Eftersom det tar

längre tid att starta upp en Future för datorn är `println` snabbare och skriver ut att alla är igång först. Sedan skrivs siffrorna från 1 - 42 ut med oregelbundna mellanrum eftersom tråden pausas olika länge.

c) `big` är en Future[Int]. Det stora talet har 7 520 383 siffror. `r` är av typen Try[Int] (se dokumentationen för Future om du är osäker)

d) Eftersom exekveringen blockas tills den har fått ett resultat går det inte att fortsätta skriva i REPL medan uträkningen pågår. Väntar man för kort tid får man ett TimeOutException och uträkningen avbryts.

Lösn. uppg. 14. *Använda `Future` för att göra flera saker samtidigt.*

a) -

b) -

c) Varje sida fördröjs med mellan 2 upp till 3 sekunder (2000-3000 millisekunder). Så i medeltal tar det 2.5 sekunder för varje sida att laddas. Vektorn måste fyllas innan exekveringen kan fortsätta. Därför laddas alla 10 stycken sidor in innan man kan se första websidan. Det tar därför i medeltal 2.5 x 10 = 25 sekunder.

d) `f` ger en Vektor fylld med strängar där varje element ges av en rad på hemsidan. Då `f` körs i bakgrunden kan programmet fortlöpa medan innehållet räknas ut. Du kan därför skriva `f` i REPL:n men det är inte säkert att processen är klar och det slutgiltiga resultatet visas.

e) Samma som ovan, förutom att det blir en vektor där varje element är i sig en vektor med strängar.

f) Ladda data parallellt så att nedladdningen sker samtidigt, men det går olika snabbt pga metoden seg.

g) Eftersom datan laddas i parallella trådar utan blockering blir de inte klara i ordning, utan i den ordningen tråden körs klart. Till slut blir alla klara och resultatet visar en vektor med **`true`** värden.

h) Metoden `lycka` är väldefinerad och kastar därför inga undantag. Den skriver alltid ut `:)`. Metoden `olycka` är inte väldefinierad då division med 0 ger `java.lang.ArithmeticException`. Detta fångas upp vid callbacken och det skrivs ut `:(` samt det specificerade undantaget.

L.12.3 Extrauppgifter; träna mer

Lösn. uppg. 15.

```
def isPrime(n: BigInt): Boolean = n match
  case _ if (n <= 1) => false
  case _ if (n <= 3) => true
  case _ if n % 2 == 0 || n % 3 == 0 => false
  case _ =>
    var i = BigInt(5)
    while i * i < n do
      if (n % i == 0 || n % (i + 2) == 0) false
      i += 6
    end while
    true
end isPrime
```

```
import scala.concurrent.*
import ExecutionContext.Implicits.global

val primes = Vector.fill(10)(Future{nextPrime(randomBigInt(16))})
primes.foreach(_.onSuccess{case i => println(i)})
```

Lösn. uppg. 16. *Svara på teorifrågor.*

a) Stackoverflow ger följande förklaring:
A thread is an independent set of values for the processor registers (for a single core). Since this includes the Instruction Pointer (aka Program Counter), it controls what executes in what order. It also includes the Stack Pointer, which had better point to a unique area of memory for each thread or else they will interfere with each other.

b)

```
val thread = new Thread(new Runnable{
  def run(){println(''Det här är en tråd'')}
})
```

c) `thread.start`

d) Det kan bli kapplöpning(race conditions) om vilken tråds resurser blir sparade. Vilket leder till att de andra trådarnas ändringar blir ignorerade.

e) Trådsäkerhet innebär att flera trådar kan köras parallellt utan felaktigheter i resultatet. Exempelvis får man vara väldigt försiktig om man vill ha en muterbar variabel som alla trådar ska ändra samtidigt.

f) Till exempel slipper man skapa instanser av klassen Thread eftersom man kan placera koden i en Future istället. Den löser även mycket under huven för kodaren.

Lösn. uppg. 17. –

Lösn. uppg. 18. *Skapa din egen multitrådade webbserver.*

a) `abbasillen` skrivs ut baklänges till `nellisabba`.

b)

c)

d)

e)

f)

g)

h)

i)
Lösningsförslag:

```
1 package fibserver.threaded.memcache.whileloop
2
3 import java.net.{ServerSocket, Socket}
4 import java.io.OutputStream
5 import java.util.Scanner
6 import scala.util.{Try, Success, Failure}
7 import scala.concurrent._
```

```
8  import ExecutionContext.Implicits.global
9
10 object html:
11   def page(body: String): String =  //minimal web page
12     s"""<!DOCTYPE html>
13        |<html><head><meta charset="UTF-8"><title>Min Sörver</title></head>
14        |<body>
15        |$body
16        |</body>
17        |</html>
18        """.stripMargin
19
20   def header(length: Int): String = //standardized header of reply to client
21     s"HTTP/1.0 200 OK\nContent-length: $length\nContent-type: text/html\n\n"
22
23   def insertBreak(s: String, n: Int = 80): String =
24     if s.size < n then s else s.take(n) + "</br>" + insertBreak(s.drop(n),n)
25
26 object compute:
27   import java.util.concurrent.ConcurrentHashMap
28   val memcache = new ConcurrentHashMap[BigInt, BigInt]
29
30   def fib(n: BigInt): BigInt =
31     if memcache.containsKey(n) then
32       println("CACHE HIT!!! no need to compute: " + n)
33       memcache.get(n)
34     else
35       println("cache miss :( must compute fib:  " + n)
36       val f = superFib(n)
37       memcache.put(n, f)
38       f
39
40   private def superFib(n: BigInt): BigInt =
41     if n <= 0 then 0
42     else if n == 1 || n == 2 then 1
43     else
44       var secondLast: BigInt = 1
45       var last: BigInt = 1
46       var sum: BigInt = secondLast + last
47       var i = 3
48       while i < n do
49         if memcache.containsKey(i) then
50           sum = memcache.get(i)
51         else
52           secondLast = last
53           last = sum
54           sum = secondLast + last
55           memcache.put(i, sum)
56         i += 1
57       sum
58
59
60 object start:
61
62   def fibResponse(num: String) =
63     num.toIntOption match
64       case Some(n) => html.page(s"fib($n) == " + compute.fib(n))
```

```
65        case None    => html.page(s"FEL: skriv ett heltal, inte $num")
66
67    def errorResponse(uri:String) = html.page(s"Error: $uri </br> use /fib/heltal")
68
69    def handleRequest(cmd: String, uri: String, socket: Socket): Unit =
70      val os = socket.getOutputStream
71      val afterSlash = uri.toString.drop(1) // skip initial slash
72      println(s"afterSlash:$afterSlash")
73      val response: String =
74        if afterSlash.startsWith("fib/") then fibResponse(afterSlash.stripPrefix("fib/"))
75        else errorResponse(uri)
76      os.write(html.header(response.size).getBytes("UTF-8"))
77      os.write(response.getBytes("UTF-8"))
78      os.close
79      socket.close
80    end handleRequest
81
82    def serverLoop(server: ServerSocket): Unit =
83      println(s"http://localhost:${server.getLocalPort}/hej")
84      while true do
85        Try {
86          var socket = server.accept  // blocks thread until connect
87          val scan = new Scanner(socket.getInputStream, "UTF-8")
88          val (cmd, uri) = (scan.next, scan.next)
89          println(s"Request: $cmd $uri")
90          Future { handleRequest(cmd, uri, socket) }.recover {
91            case e => println(s"Reqest failed: $e")
92          }
93        }.recover{ case e: Throwable => s"Connection failed: $e" }
94
95    def main(args: Array[String]) =
96      val port = Try{ args(0).toInt }.getOrElse(8089)
97      serverLoop(new ServerSocket(port))
```

L.13 Lösning examprep

Lösn. uppg. 1. *Gör klart ditt projekt.* —

Lösn. uppg. 2. *Gör en extenta.* —

Lösn. uppg. 3. *Förbered din projektredovisning.* –

Lösn. uppg. 4. *Skapa dokumentation för ditt projekt.*–

Lösn. uppg. 5. *Repetera övningar och laborationer.* –

L.14 Lösning `java`

L.14.1 Grunduppgifter; förberedelse inför laboration

Lösn. uppg. 1. *Översätta metoder från Java till Scala.*

a)

```
import java.net.URL;
import java.util.ArrayList;
import java.util.{Set => JSet};
import java.util.{HashSet => JHashSet};
import java.util.Scanner;

object Hangman {  // This is Java-like, non-idiomatic Scala code!
    private var hangman: Array[String] = Array[String](
        " ======  ",
        " |/   |  ",
        " |    O  ",
        " |   -|- ",
        " |   / \\ ",
        " |       ",
        " |       ",
        " ==========================   RIP  :(");

    private def renderHangman(n: Int): String = {
        var result: StringBuilder = new StringBuilder();
        for (i: Int <- 0 until n){
            result.append(hangman(i));
            if (i < n - 1) {
                result.append("\n");
            }
        }
        return result.toString();
    }

    private def hideSecret(secret: String,
                           found: JSet[Character]): String = {
        var result: String = "";
        for (i: Int <- 0 until secret.length()) {
            if (found.contains(secret.charAt(i))) {
                result += secret.charAt(i);
            } else {
                result += '_';
            }
        }
        return result;
    }

    private def foundAll(secret: String,
                         found: JSet[Character]): Boolean = {
        var foundMissing: Boolean = false;
        var i: Int = 0;
        while (i < secret.length() && !foundMissing) {
            foundMissing = !found.contains(secret.charAt(i));
            i += 1;
        }
```

```
50          return !foundMissing;
51      }
52
53      private def makeGuess(): Char = {
54          var scan: Scanner = new Scanner(System.in);
55          var guess: String = "";
56          while ({
57             System.out.println("Gissa ett tecken: ");
58             guess = scan.next();
59             guess.length() > 1;
60          }) ()
61          return Character.toLowerCase(guess.charAt(0));
62      }
63
64      def download(address: String, coding: String): String = {
65          var result: String = "lackalänga";
66          try {
67              var url: URL = new URL(address);
68              var words: ArrayList[String] = new ArrayList[String]();
69              var scan: Scanner = new Scanner(url.openStream(), coding);
70              while (scan.hasNext()) {
71                  words.add(scan.next());
72              }
73              var rnd: Int = (Math.random() * words.size()).asInstanceOf[Int];
74              result = words.get(rnd);
75          } catch { case e: Exception =>
76              System.out.println("Error: " + e);
77          }
78          return result;
79      }
80
81      def play(secret: String): Unit = {
82          var found: JSet[Character] = new JHashSet[Character]();
83          var bad: Int = 0;
84          var won: Boolean = false;
85          while (bad < hangman.length && !won){
86              System.out.println(renderHangman(bad));
87              System.out.print("Felgissningar: " + bad + "\t");
88              System.out.println(hideSecret(secret, found));
89              var guess: Char = makeGuess();
90              if (secret.indexOf(guess) >= 0) {
91                  found.add(guess);
92              } else {
93                bad += 1;
94              }
95              won = foundAll(secret, found);
96          }
97          if (won) {
98              System.out.println("BRA! :)");
99          } else {
100             System.out.println("Hängd! :(");
101         }
102         System.out.println("Rätt svar: " + secret);
103         System.out.println("Antal felgissningar: " + bad);
104     }
105
106     def main(args: Array[String] ): Unit = {
```

```
        if (args.length == 0) {
            var runeberg: String =
                "http://runeberg.org/words/ord.ortsnamn.posten";
            play(download(runeberg, "ISO-8859-1"));
        } else {
            var rnd: Int = (Math.random() * args.length).asInstanceOf[Int];
            play(args(rnd));
        }
    }
}
```

b)

```
object hangman:
  val hangman = Vector(
    " ======  ",
    " |/   |  ",
    " |    O  ",
    " |   -|- ",
    " |   / \\",
    " |       ",
    " |       ",
    " ==========================   RIP  :(")

  def renderHangman(n: Int): String = hangman.take(n).mkString("\n")

  def hideSecret(secret: String, found: Set[Char]): String =
      secret.map(ch => if found(ch) then ch else '_')

  def makeGuess(): Char =
    val guess = scala.io.StdIn.readLine("Gissa ett tecken: ")
    if guess.length == 1 then guess.toLowerCase.charAt(0)
    else makeGuess()

  def download(address: String, coding: String): Option[String] =
    scala.util.Try {
      import scala.io.Source.fromURL
      val words = fromURL(address, coding).getLines.toVector
      val rnd = (math.random() * words.size).toInt
      words(rnd)
    }.toOption

  def play(secret: String): Unit =
    def loop(found: Set[Char], bad: Int): (Int, Boolean) =
      if secret forall found then (bad, true)
      else if bad >= hangman.length then (bad, false)
      else
        println(renderHangman(bad) + s"\nFelgissningar: $bad\t")
        println(hideSecret(secret, found))
        val guess = makeGuess()
        if secret contains guess then loop(found + guess, bad)
        else loop(found, bad + 1)

    val (badGuesses, won) = loop(Set(),0)
```

```
42      val msg = if won then "BRA! :)" else "Hängd! :("
43      println(s"$msg\nRätt svar: $secret")
44      println(s"Antal felgissningar: $badGuesses")
45
46    def main(args: Array[String] ): Unit =
47      if args.length == 0 then
48        val runeberg = "http://runeberg.org/words/ord.ortsnamn.posten"
49        val secret = download(runeberg, "ISO-8859-1").getOrElse("läckalånga")
50        play(secret)
51      else play(args((math.random() * args.length).toInt))
```

Lösn. uppg. 2. *Översätta mellan klasser i Scala och klasser i Java.*

a)

```
1  import java.util.List;
2  import java.util.ArrayList;
3
4  public class JPoint {
5      private int x, y;
6
7      public JPoint(int x, int y){
8        this.x = x;
9        this.y = y;
10     }
11
12     public JPoint(int x, int y, boolean save){
13         this(x, y);
14         if (save) {
15             saved.add(0, this);
16         }
17     }
18
19     public JPoint(){
20         this(0, 0);
21     }
22
23     public int getX(){
24         return x;
25     }
26
27     public int getY(){
28         return y;
29     }
30
31     public double distanceTo(JPoint that) {
32       return distanceBetween(this, that);
33     }
34
35     @Override public String toString() {
36         return "JPoint(" + x + ", " + y + ")";
```

```
    }

    private static List<JPoint> saved = new ArrayList<JPoint>();

    public static Double distanceBetween(JPoint p1, JPoint p2) {
        return Math.hypot(p1.x - p2.x, p1.y - p2.y);
    }

    public static void showSaved() {
        System.out.print("Saved: ");
        for (int i = 0; i < saved.size(); i++){
          System.out.print(saved.get(i));
          if (i < saved.size() - 1) {
            System.out.print(", ");
          }
        }
        System.out.println();
    }
}
```

b) -

c)

```
case class Person(name: String, age: Int = 0)
```

d) p.*TAB* - copy, producArity, ProductIterator, productElement, productPrefix
Person.*TAB* - apply, curried, tupled, unapply

```
scala> p.copy
   def copy(name: String,age: Int): Person

scala> p.copy()
res0: Person = Person(Björn,49)

scala> p.copy(age = p.age + 1)
res1: Person = Person(Björn,50)

scala> Person.unapply(p)
res2: Option[(String, Int)] = Some((Björn,49))
```

Lösn. uppg. 3. *Oföränderlig Java-klass.*

```
// Översätt:  class Point3D(val x: Int, val y: Int, val z: Int = 0)

public class JPoint3D {
  private int x;       // Attributen måste vara privata eftersom
  private int y;       //   val-attribut ej finns i Java.
  private int z;       // Typen skrivs före namnet i deklarationer.

  public JPoint3D(int x, int y, int z){// Konstruktor heter som klassen.
    this.x = x;                        // Satser måste sluta med ;
    this.y = y;                        // this behövs p.g.a. skuggning.
```

```
    this.z = z;
  }

  public JPoint3D(int x, int y){  // I stället för default-argument.
    this(x, y, 0);                // Anropa konstruktor i konstruktor.
  }
  public int getX(){    // I Java brukar man ha get i namnet på getter.
    return x;           // Metoder som ej är procedurer måste ha return.
  }

  public int getY(){    // Metoder måste ha parenteser, även getters.
    return y;
  }

  public int getZ(){   // Synlighet public måste anges explicit.
    return z;
  }
}
```

```
> code JPoint3D.java
> javac JPoint3D.java
> ls
JPoint3D.class  JPoint3D.java
> scala

scala> val p = new JPoint3D(1,2)
val p: JPoint3D = JPoint3D@53b1a3f8

scala> p.x
1 |p.x
  |^^^
  |value x is not a member of JPoint3D

scala> p.getX
val res0: Int = 1
```

Lösn. uppg. 4. *Förändringsbar Java-klass.*

```
//Översätt class MutablePoint3D(var x: Int, var y: Int, var z: Int = 0)

public class JMutablePoint3D {
  private int x;  // Attributen brukar vara privata även i föränngsbara
  private int y;  //   klasser eftersom enhetlig access och syntax för
  private int z;  //   setter med tilldelning ej finns i Java.

  public JMutablePoint3D(int x, int y, int z){
    this.x = x;          // Satser måste sluta med ;
    this.y = y;          // this behövs p.g.a. skuggning.
    this.z = z;
  }

  public JMutablePoint3D(int x, int y){  // Motsv. default-argument.
```

```
    this(x, y, 0);      // Anropa konstruktor i konstruktor.
  }
  public int getX(){    // I Java brukar man ha get i namnet på getter.
    return x;           // Metoder som ej är procedurer måste ha return.
  }

  public int getY(){    // Metoder måste ha parenteser, även getters.
    return y;
  }

  public int getZ(){   // Synlighet public måste anges explicit.
    return z;
  }

  public void setX(int x){ // I Java brukar man ha set i namnet på setter.
    this.x = x;
  }

  public void setY(int y){ // Nyckelordet void liknar Unit i Scala, men
    this.y = y;            //   det finns inget äkta tomt värde som ()
  }

  public void setZ(int zz){ // Om namn ej krockar behövs inte this, men
    z = zz;                 // man brukar ha samma parameternamn som
  }                         // attributnamn i setters trots skuggning så
}                           // att man slipper hitta på nytt namn.
```

```
> code JMutablePoint3D.java
> javac JMutablePoint3D.java
> ls
JMutablePoint3D.class  JMutablePoint3D.java
> scala

scala> val p = new JMutablePoint3D(1,2)
val p: JMutablePoint3D = JMutablePoint3D@625b215b

scala> p.x
1 |p.x
  |^^^
  |value x is not a member of JMutablePoint3D

scala> p.getZ
val res0: Int = 0

scala> p.setZ(3)

scala> p.getZ
val res1: Int = 3
```

Lösn. uppg. 5. *Jämföra strängar i Java.*

a)

```
String = java.lang.String
Boolean = true
Int = 0
```

b) Exempel på 3 olika uttryck för att testa `compareTo`:

1. Hej kommer först då `H < h`.

```
"hej".compareTo("Hej")
res: Int = 32
```

2. Dessa är ekvivalenta, så `compareTo` returnerar 0.

```
"hej".compareTo("hej")
res: Int = 0
```

3. *h* kommer före *ö*.

```
"hej".compareTo("ö")
res: Int = -142
```

c) Exempel på 3 olika uttryck för att testa `compareToIgnoreCase`:

1.

```
"hej".compareToIgnoreCase("HEj")
res: Int = 0
```

2.

```
"hej".compareToIgnoreCase("Ö")
res: Int = -142
```

3. Samma som ovan, då Ö omvandlas till ö innan jämförelse.

```
"hej".compareToIgnoreCase("ö") \\ res: Int = -142
```

d)

```
false
true
0
```

Lösn. uppg. 6. *Linjärsökning i Java.*

a)

```
public static boolean isYatzy(int[] dice){
    int col = 1;
    boolean allSimilar = true;
    while(col < dice.length && allSimilar){
        allSimilar = (dice[0] == dice[col]);
        col++; //denna raden saknades
    }
    return allSimilar;
}
```

b)

```
public static int findFirstYatzyRow(int[][] m){
    int row = 0;
    int result = -1;
    while(row < m.length){
        if(isYatzy(m[row])){
           result = row;
           break;
        }
        row++;
    }
    return result;
}
```

Lösn. uppg. 7. *Jämförelsestöd i Java.*

a)

b)

```
val teamComparator = new Comparator[Team]{
  def compare(o1: Team, o2: Team) = o2.rank - o1.rank
}
```

c)

d)

e)

```
case class Point(x: Int, y: Int) extends Comparable[Point] {
  def distanceFromOrigin: Double = math.hypot(x, y)
  def compareTo(that: Point): Int =
    (distanceFromOrigin - that.distanceFromOrigin).round.toInt
}
```

Lösn. uppg. 8. *`java.util.Arrays.binarySearch`*

Lösn. uppg. 9. *Auto(un)boxing.*

a) -

b) Cell har typen java.lang.Integer. När man hämtar ut värdet med `c.value` hämtas den primitiva typ `int` ut.

c) Med hjälp av autoboxing förvandlas 42 till typen `Integer` och kan därför jämföras med en annan `Integer`.

d) i.compareTo(42) fungerar på grund av autoboxing. Då JVM packar in den primitiva typ int i en Integer-objekt automatiskt.

e)

```
0 10 20 30 40 50 60 ... 390 400 410

[0]: 0
[42]: 0
```

```
NOT EQUAL
```

f)

```
1  import java.util.ArrayList;
2
3  public class Autoboxing2 {
4      public static void main(String[] args) {
5          ArrayList<Integer> xs = new ArrayList<Integer>();
6          for (int i = 0; i < 42; i++) {
7              xs.add(i);
8          }
9          for (int x: xs) {
10             int y = x * 10;
11             System.out.print(y + " ");
12         }
13         int pos = xs.size();
14         xs.add(pos, 0);
15         System.out.println("\n\n[0]: " + xs.get(0));
16         System.out.println("[" + pos + "]: " + xs.get(pos));
17         if (xs.get(0).equals(xs.get(pos))) {
18             System.out.println("EQUAL");
19         } else {
20             System.out.println("NOT EQUAL");
21         }
22     }
23 }
```

g) 42 kommer läggas längst fram i listan istället för längst bak, då autounboxing kommer göra Integer(0) till 0 och tvärtom med variablen `pos`.

h) Om man ska undersöka om två int-variabler är lika ska man använda ==, men om variablerna är av typen Integer måste man använda `equals`.

JVM kommer inte varna om man vänder på `Integer` och `int`, som i `xs.add(0, pos)`.

Lösn. uppg. 10. *CollectionConverters.*

a)

Vector[Int] -> java.util.List[Int]
Set[Char] -> java.util.Set[Char]
Map[String, Int] -> java.util.Map[String, Int]

b)

ArrayList[Int] -> scala.collection.mutable.Buffer[Int]
HashSet[Char] -> scala.collection.mutable.Set[Char]

Båda blir föränderliga motsvarigheter. Det visas genom att de till hör `scaka.collection.mutable` och både `ArrayList` och `HashSet` är förändrliga i Java.

c) `scala.collection.immutable.Set`

d) `sm.asJava.asScala` ger typen `scala.collection.mutable.Map[String,Int]`
`sm.asJava.asScala.toMap` ger typen `scala.collection.immutable.Map[String,Int]`

e) -

Lösn. uppg. 11. *Hur fungerar en* `switch`*-sats i Java (och flera andra språk)?*

a) Beroende på första bokstaven i din favoritgrönsak får du olika svar såsom *gurka är gott!* vid första bokstaven *g*.
Javas `switch`-sats testar den första bokstaven på favoritgrönsaken genom att stegvis jämföra den med `case`-uttrycken. Om första bokstaven `firstChar` matchar bokstaven efter ett `case` körs koden efter kolonet till `switch`-satsens slut eller tills ett `break` avbryter `switch`-satsen.
Matchar inte `firstChar` något `case` så finns även `default`, som körs oavsett vilken första bokstaven är, ett generellt fall.

b) Om `case 't'` körs kommer både *tomat är gott!* och *broccoli är gott!* skrivas ut, man säger att koden "faller igenom". Utan `break`-satsen i Java körs koden i efterkommande `case` tills ett `break` avbryter exekveringen eller `switch`-satsen tar slut.

Lösn. uppg. 12. *Fånga undantantag i Java med en* `try-catch`*-sats.*

a)

1. Eftersom första argumentet inte är strängen *safe* görs en oskyddad division av 42 med 42 där slutsvaret 1 visas.

2. Eftersom första argumentet inte är strängen *safe* görs en oskyddad division av 42 med 0 som ger `ArithmeticException` eftersom ett tal inte kan delas med noll.

3. Eftersom första argumentet är strängen *safe* görs en skyddad division av 42 med 42 där slutsvaret 1 visas.

4. Eftersom första argumentet är strängen *safe* görs en skyddad division av 42 med 0. Denna gång fångas `ArithmeticException` av `try-catch`-satsen vilket ersätter den gamla division med en säker division med 1 där slutsvaret 42 visas.

5. Eftersom inga argument givits kastas ett `ArrayIndexOutOfBoundsException` när programmet försöker anropa `equals` metoden hos en sträng som inte finns. Detta kunde också kontrollerats av en `try-catch`-sats.

b)

```
1 TryCatch.java:16: error: variable input might not have been initialized
```

Ett kompileringsfel uppstår på grund av risken att `input` inte blivit definierad vid division.

Lösn. uppg. 13. *Matriser med array i Java.*

a) Vid initialisering fylls alla element i `xss` med standardvärdet för typen, `0` i fallet med `int`. Den yttre `for`-loopen i `showMatrix()` itererar över raderna i `xss`. Den inre `for`-loopen itererar i sin tur längs med elementen på den aktuella raden och skriver ut rad, kolumn och innehåll. Efter varje rad sker en radbrytning, så att en rad i utskriften även motsvarar en rad i matrisen.
Exempel på skillnader mellan användning av matriser i scala och java:

- åtkomst: `minArray(rad)(kolumn)` respektive `minArray[rad][kolumn]`

- typnamn: `Array[Array[elementTyp]]` respektive `elementTyp[][]`

- allokering: `Array.ofDim[typ](xDim,yDim)` respektive `new typ[xDim][yDim]`

b)

```
public class JavaArrayTest {

  public static void showMatrix(int[][] m){
    System.out.println("\n--- showMatrix ---");
    for (int row = 0; row < m.length; row++){
      for (int col = 0; col < m[row].length; col++) {
        System.out.print("[" + row + "]");
        System.out.print("[" + col + "] = ");
        System.out.print(m[row][col] + ";");
      } System.out.println();
    }
  }

  public static void fillRnd(int[][] m, int n){
    for (int row = 0; row < m.length; row++){
      for (int col = 0; col < m[row].length; col++) {
        m[row][col] = (int) (Math.random() * n + 1);
      }
    }
  }

  public static void main(String[] args) {
    System.out.println("Hello JavaArrayTest!");
    int[][] xss = new int[10][5];
    fillRnd(xss, 6);
    showMatrix(xss);
  }
}
```

Lösn. uppg. 14. *Översätta från Java till Scala.*

```
1  object showInt {
2    def show(obj: Any, msg: String = ""): Unit = println(msg + obj)
3
4    def repeatChar(ch: Char, n: Int): String = ch.toString * n
5
6    def showInt(i: Int): Unit = {
7      val leading = Integer.numberOfLeadingZeros(i)
8      val binaryString = repeatChar('0', leading) + i.toBinaryString
9      show(i,               "Heltal : ")
10     show(i.asInstanceOf[Char],         "Tecken : ")
11     show(binaryString,    "Binärt : ")
12     show(i.toHexString,   "Hex    : ")
13     show(i.toOctalString, "Oktal  : ")
14   }
15
16
```

```
  import scala.io.StdIn.readLine
  import scala.util.{Try,Success,Failure}

  def loop: Unit =
    Try { readLine("Heltal annars pang: ").toInt } match {
      case Failure(e) => show(e); show("PANG!")
      case Success(i) => showInt(i); loop
    }

  def main(args: Array[String]): Unit =
    if(args.length > 0) args.foreach(i => showInt(i.toInt))
    else loop
}
```

Lösn. uppg. 15. *Innehållslikhet och referenslikhet i Java.*

Lösn. uppg. 16. *Implementera innehållslikhet i Java.*

Lösn. uppg. 17. *Array och* `for`*-sats i Java.*

a) Programmet simulerar 10000 tärningskast (med slumptalsfrö 42) och skriver ut förekomsten av respektive tärningskast.

```
Rolling the dice 10000 times with seed 42
Number of 1's: 1654
Number of 2's: 1715
Number of 3's: 1677
Number of 4's: 1629
Number of 5's: 1643
Number of 6's: 1682
```

b) I Java används hakparenteser medan Scala har ”vanliga” parenteser. En array i scala deklareras så här:
`val scalaArray = Array.ofDim[Int](6)`
vilket i java motsvarar: `int[] javaArray = new int[6];`
`for`-sats i scala skrivs: `for(i <- 0 to n) {...}` medan i Java skrivs:
`for (int i = 0; i < n; i++) { ... }`.

I Java måste semikolon skrivas efter varje sats och typen måste anges explicit vid varje variabeldeklaration.

I scala behövs inte semikolon (förutom för att separera satser på samma rad) och typer kan ofta härledas i Scala av kompilatorn och behöver inte alltid skrivas explicit.

c) Lägg till `System.out.println(i);` i for-looparna

d)

```
// DiceReg2.java
import java.util.Random;
public class DiceReg2{
  public static int[] diceReg = new int[6];
  private static Random rnd = new Random();

  public static int parseArguments(String[] args){
    int n = 100;
```

```
    if(args.length > 0) {
      n = Integer.parseInt(args[0]);
    }
    if(args.length > 1) {
      int seed = Integer.parseInt(args[1]);
      rnd.setSeed(seed);
    }
    return n;
  }

  public static void registerPips(int n) {
    for(int i = 0; i<n; i++) {
      int pips = rnd.nextInt(6);
      diceReg[pips]++;
    }
  }

  public static void main(String[] args) {
    int n = parseArguments(args);
    registerPips(n);
    printReg();
  }
}
```

e)

```
  // Skriver ut förekomsten av 1000 tärningskast med slumptalsfrö 42.
Number of 1's: 165
Number of 2's: 163
Number of 3's: 178
Number of 4's: 183
Number of 5's: 156
Number of 6's: 155

  // Skriver ut diceReg-attributet
res1: Array[Int] = Array(165, 163, 178, 183, 156, 155)

  // Skriver ut diceReg-attributet efter 1000 till kast.
res2: Array[Int] = Array(329, 325, 349, 360, 324, 313)

  // Skriver ut diceReg-attributet efter 1000 till kast.
res3: Array[Int] = Array(498, 484, 531, 513, 485, 489)

  // Det blir kompileringsfel då attributet rnd är privat
<console>:11: error: value rnd is not a member of object DiceReg2
  DiceReg2.rnd
           ^
```

Lösn. uppg. 18. *Läsa in sekvens av tal med `Scanner` i Java.*

a)

`hasNextInt()` kollar om det finns ett till tal och returnerar **`true`** eller **`false`**. `nextInt()` läser nästa tal.

Se https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html#hasNextInt%28%29 och https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html#nextInt%28%29.

b)

```
1  import java.util.Random;
2  import java.util.Scanner;
3
4  public class DiceScanBuggy {
5    public static int[] diceReg = new int[6];
6    public static Scanner scan = new Scanner(System.in);
7
8    public static void registerPips() {
9      System.out.println("Enter pips separated by blanks: ");
10     System.out.println("End with -1 and <Enter>.");
11     boolean isPips = true;
12     while(isPips && scan.hasNextInt()){
13       int pips = scan.nextInt();
14       if(pips >= 1 && pips <= 6) {
15         diceReg[pips-1]++;
16       } else {
17         isPips = false;
18       }
19     }
20   }
21
22   public static void printReg(){
23     for(int i = 1; i<7; i++) {
24     System.out.println("Number of " + i + "'s: " + diceReg[i-1]);
25     }
26   }
27
28   public static void main(String[] args) {
29     registerPips();
30     printReg();
31   }
32 }
```