Programmering, grundkurs

Lösningar till övningar

Datavetenskap, LTH
Lunds Universitet

Kompilerad: 8 januari 2017

http://cs.lth.se/pgk

Innehåll

1. Lösningar till övning `expressions`

1.1 Grunduppgifter

Uppgift 1.

a) skriver ut `"hejsan REPL"`

b) man får fortsätta skriva på nästa rad

c) värde: `gurkatomat`, typ: `String`

d) värde: `gurkatomatgurkatomatgurkatomatgurkatomat`, typ: `String`

Uppgift 2. En literal anger ett specifikt datavärde.

Uppgift 3.

a) `Int`

b) `Long`

c) `char`

d) `String`

e) `Double`

f) `Double`

g) `Double`

h) `Float`

i) `Float`

j) `Boolean`

k) `Boolean`

Uppgift 4.

```
hejsan
42
gurka
```

- Klammerparenteser används för att gruppera flera satser. Klammerparenteser behövs om man vill definiera en funktion som består av mer än en sats.
- Semikolon avskiljer flera satser. Semikolon behövs om man vill skriva många satser på samma rad.

Uppgift 5.

a) Ett utryck kan evalueras och resulterar då i ett användbart värde. En sats *gör* något (t.ex. skriver ut något), men resulterat inte i något användbart värde.

b) `println()`

c)

Värdesaknas innehåller Unit

Skriver ut `Unit`
Skriver ut `"()"`
Skriver ut `"()"`
Skriver först ut hej med det innersta anropet och sen `()` med det yttre anropet

d) `Unit`

e) `Unit`

Uppgift 6.

a) `Int, 42`

b) `Float,19`

c) `Double,42`

d) `Double,42`

e) `Float,1.042E42`

f) `Long, 12E6`

g) `String, gurka`

h) `Char, 'A'`

i) `Int,65`

j) `Int,48`

k) `Int,49`

l) `Int,57`

m) `Int, 113`

n) `Char, 'q'`

o) `Char, '*'`

Uppgift 7.

a) `Int, 84`

b) `Float, 21`

c) `Float, 41.8`

d) `Double, 12`

Uppgift 8.

a) `Int,27`

b) `Int,50`

c) `Double, 13.3`

d) `Int, 13`

Uppgift 9.

a) `Int, 21`

b) `Int, 10`

c) `Float,10.5`

d) `Int, 0`
e) `Int, 1`
f) `Int,3`
g) `Int, 0`
h) `((5793 - 1000 * (5793 / 1000)) / 100`

Uppgift 10.

a) 127,-128
b) 32767, -32768
c) 2147483647,-2147483648
d) 9223372036854775807,-9223372036854775808

Uppgift 11.

a) java: PI scala: Pi
b) andvänder sig utav pythagoras sats
c) `scalb()`

Uppgift 12.

a) den blir `Int.MinValue`
b) kastar exeption
c) `1.0000000000000001E8`
d) avrundas till `1E8`
e) `45.00000000000001`
f) returnerar en double som är oändlig
g) `Int.MaxValue`
h) `NaN`
i) `NaN`
j) Man kastar ett nytt exception.

Uppgift 13.

a) **`true`**
b) **`false`**
c) **`false`**
d) **`false`**
e) **`true`**
f) **`true`**
g) **`true`**
h) **`false`**
i) **`true`**
j) **`false`**

k) `false`

l) `true`

m) `true`

n) `false`

o) `true`

p) `true`

q) `true`

r) `false`

s) `true`

t) `false`

u) `true`

v) `true`

Uppgift 14.

```
a = 13
b = 14
c = Double 54
b = 0
a = 0
c = Double 55
```

Uppgift 15.

a)

x blir 30
x blir 31
skriv ut x
x = 32
skriv ut x
false
constant värde y blir 20
fungerar ej
skriv ut gurka och z blir 10
funktionen w blir det inom måsvingarna
skriv ut z
skriv ut z
z blir 11
anropa w
anropa w
fungerar ej

b) Rad 8 och 16. y är konstant och kan ej modifieras. kan ej modifiera en funktion

c)

- `var` används för att deklarera en variabel som kan tilldelas nya värden efter att den initialiserats

- `val` används för att deklarera en variabel som kan tilldelas ett endast en gång (initialseras men sedan inte ändras)
- `def` används för att deklarera en funktion som evalueras varje gång den anropas.

d) `val even = n % 2 == 0`

e) `val odd = n % 2 != 0`

Uppgift 16.

a blir 40
b blir 80
a blir 50
b blir 70
a blir 100
b blir 35

Uppgift 17.

a)

```
Namnet 'Kim Finkodare' har 12 bokstäver.
Efternamnet 'Finkodare' har 9 bokstäver.
```

b)

```
val fTot = f.size
val eTot = e.size
println(s"$f har  $fTot bokstäver.")
println(s"$e har  $eTot bokstäver.")
```

Uppgift 18.

1. skriver ut ”sant” (else-grenen görs ej)
2. skriver ut ”falskt” (else-grenen görs)
3. skriver ut ”faskt” (else-grenen görs)
4. skriver ut ”sant” (else-grenen görs ej)
5. definerar en funktion som ...
6. ... skriver ut antingen krona eller klave med lika stor sannolikhet
7. singlar slant tre gånger

Uppgift 19.

a) `String`, inte gott

b) `String`, gott

c) `String`, likastora

d) `String`, gurka

e) `String`, banan

Uppgift 20.

a)

1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
1, 2, 3, 4, 5, 6, 7, 8, 9,
2, 4, 6, 8, 10,
1, 11, 21, 31, 41, 51, 61, 71, 81, 91,
10, 9, 8, 7, 6, 5, 4, 3, 2, 1,

b)

```
scala> for(i <- 1 to 43 by 3) print("A" + i + ", ")
```

Uppgift 21.

a)

9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33,

b)

B33, B30, B27, B24, B21, B18, B15, B12, B9, B6, B3, B0,

Uppgift 22.

a)

0 till 9 0, 2, 4, 6, 8, 10, 12

b)

```
var k = 0
while(k <= 43)
{
print("A" + k + ", ")
k = k + 3
}
```

c) `foreach`

Uppgift 23.

a) `Double`

b) 0, less than 1.0

c) Nej

d) Man får 20 olika slumptal mellan 1 och 3 då `math.random` ger ett slumptal av typen `Double` mellan `0.0` och nästan, men inte inklusive, `1.0`. Om man multiplicerar med 3 och adderar 1 och anropar `toInt` så blir det ett heltal i intervallet [1,3].

e) `for (i <- 1 to 100) println((math.random * 9).toInt)`

f) `for (i <- 1 to 100) println((math.random * 5 + 1).toInt)`

g) `for (i <- 1 to 100) println((math.random * 6).toInt + 1 )`

h) gurka skrivs ut olika antal gånger

i) `while (math.random > 0.01) println("gurka")`

j) Samma sak som i dem förra fast man skriver ut slumptalet

Uppgift 24.

a) `poäng > 1000`

b) `poäng > 100`

c) `poäng < highscore`

d) `poäng < 0 || poäng > highscore`

e) `poäng > 0 \&\& poäng < highscore`

f) `klar`

g) `!klar`

1.2 Extrauppgifter

Uppgift 25.

a)

```
def tärning = (math.random * 6 + 1).toInt
```

b)

```
def rnd = math.round(math.random * 10) / 10.0f
```

c) Talen avrundas neråt istället för till närmsta tal vilket gör att det är omöjligt att slumpa fram `1.0`

Uppgift 26.

Uppgift 27.

```
if (x != 42)  println(":(") else println("the meaning of it all")
```

Uppgift 28.

Uppgift 29.

```
println("My heart will go on")
while(true)  println("and on")
```

Uppgift 30.

1.3 Fördjupningsuppgifter

Uppgift 31. Scala använder sig utav Remainderdet vill säga resten vi heltals division. Detta gör att -3 % 4 blir -3 och inte 1 som det blir i matematiken.

Uppgift 32.

a) Variabeln får namnet "konstig val", backtick gör att man kan namge variabler till annars otilåtna namn t.ex. med mellanrum eller nyckelord i sig.

b) Backticks tillåter en att anropa metoder som heter samma som nyckelord i scala. I java får man döpa en metod till t.ex. yield men ska man anropa metoden i scala krävs då backticks för att yield är ett nyckel ord. java.Thread.‘yield‘()

Uppgift 33. a) `toBinaryString` gör om heltalet till en sträng med ettor och nollor som är den binära versionen utav talet. `toHexString` gör sama sak fast till ett hexadecimalt tal. b) 42

Uppgift 34.

Uppgift 35. först blir `i 42`. `i` blir sedan 43 och multipliceras med 2 och blir 86. Efter den delas med 3 blir den 28 eftersom `Int` inte har några decimaler.

Uppgift 36.

Uppgift 37.

a) Den första raden returnerar 84. Den andra kastar ett exception.

b) För att kunna hantera situationer när bydelängden på variabler inte är lång nog för värden.

c) Overflow är när en variabel inte kan inehålla ett värde då det är för stort och istället blir ett värde som variabeln egentligen inte ska få.

Uppgift 38.

a) `4.9E-3240`

b) `-1.7976931348623157E308`

c) `4.9E-324`

Uppgift 39.

Uppgift 40.

```
val s = f"Gurkan är $g meter lång"
```

2. Lösningar till övning `programs`

Uppgift 1.

a) värde: `Range(1,2,3,4,5,6,7,8,9)`
typ: `scala.collection.immutable.Range`

b) värde: `Range(1,2,3,4,5,6,7,8,9,10)`
typ: `scala.collection.immutable.Range`

c) värde: `Range(0,5,10,15,20,25,30,35,40,45)`
typ: `scala.collection.immutable.Range`

d) värde: `10`, typ: `Int`

e) värde: `Range(0,5,10,15,20,25,30,35,40,45,50)`
typ: `scala.collection.immutable.Range`

f) värde: `11`, typ: `Int`

g) värde: `Range(0,1,2,3,4,5,6,7,8,9)`
typ: `scala.collection.immutable.Range`

h) värde: `Range(0,1,2,3,4,5,6,7,8,9)`
typ: `scala.collection.immutable.Range`

i) värde: `Range(0,1,2,3,4,5,6,7,8,9)`
typ: `scala.collection.immutable.Range`

j) värde: `Range(0,1,2,3,4,5,6,7,8,9,10)`
typ: `scala.collection.immutable.Range.Inclusive`

k) värde: `Range(0,1,2,3,4,5,6,7,8,9,10)`
typ: `scala.collection.immutable.Range.Inclusive`

l) värde: `Range(0,5,10,15,20,25,30,35,40,45)`
typ: `scala.collection.immutable.Range`

m) värde: `Range(0,5,10,15,20,25,30,35,40,45,50)`
typ: `scala.collection.immutable.Range`

n) värde: `11`, typ: `Int`

o) värde: `500500`, typ: `Int`

Uppgift 2.

a) Ett objekt av typen `Array[String]` skapas med värdet `Array(hej, på, dej, !)` och med namnet `xs`.

b) Returnerar en sträng med värdet `hej`.

c) Returnerar en sträng med värdet `!`.

d) Ett exception genereras. Skriver ut:
`java.lang.ArrayIndexOutOfBoundsException: 4`

e) Returnerar en sträng med värdet `på dej`.

f) Returnerar en sträng med värdet `hejpådej!`.

g) Returnerar en sträng med värdet `hej på dej !`.

h) Returnerar en sträng med värdet `(hej,på,dej,!)`.

i) Returnerar en sträng med värdet `Array(hej,på,dej,!)`.

j) Ett fel uppstår av typen **`type`** `mismatch`. Konsollen talar om för oss vad den fick, dvs värdet `42` av typen `Int`. Den talar även om för oss vad den ville ha, dvs något värde av typen `String`. Till sist skriver den ut vår kodrad och pekar ut felet.

k) Det första elementet i `xs` ändras till värdet `42`. Därefter skrivs det första värdet i `xs` ut.

l) Ett objekt av typen `Array[Int]` skapas med värde
t `Array(42, 7, 3, 8` och med namnet `ys`.

m) Returnerar summan av elementen i `ys`. Resultatet är `60`.

n) Returnerar det minsta värdet i `ys`. Resultatet är `3`.

o) Returnerar det största värdet i `ys`. Resultatet Ã¤r `42`.

p) Ett nytt värde av typen `Array[Int]` skapas med `10` stycken element, alla med värdet `42`.

q) Returnerar summan av elementen i `zs`. Resultatet blir 420 (42 multiplicerat med 10).

r) `r` tar upp 12 bytes. `a` tar upp ca 4 miljarder bytes.

Uppgift 3.

a) Ett objekt av typen `scala.collection.immutable.Vector[String]` initieras med värdet `Vector(hej, på dej, !)`.

b) Returnerar det nollte elementet i `words`, dvs strängen `hej`.

c) Returnerar det tredje elementet i `words`, dvs strängen `!`.

d) Omvandlar vektorn till en Sträng.

e) Samma som ovan, fast den här gången används mellanrum för att seperera elementen.

f) Samma som ovan, fast den här gången sepereras elementen av kommatecken istället för mellanrum och dessutom börjar och slutar den resulterande strängen med parenteser.

g) Samma som ovan, fast med ordet `Ord` tillagt i början av den resulterande strängen.

h) Ett fel uppstår. Typen `Vector` är immutable. Dess element kan alltså inte bytas ut.

i) En ny `Vector[Int]` skapas med värdet `Vector(42, 7, 3, 8)`.

j) Returnerar summan av vektorn `numbers`.

k) Returnerar vektorns minsta element.

l) Returnerar vektorns största element.

m) En ny vektor skapas innehållandes tiotusen 42or.

n) Returnerar summan av vektorns element.

o) Byta ut element.

Uppgift 4.

a) typ: `scala.collection.immutable.IndexedSeq[Int]`
värde: `Vector(1, 2, 3, 4, 5, 6, 7, 8, 9)`

b) typ: `scala.collection.immutable.IndexedSeq[Int]`
värde: `Vector(1, 2, 3, 4, 5, 6, 7, 8, 9)`

c) typ: `scala.collection.immutable.IndexedSeq[Int]`
värde: `Vector(2, 3, 4, 5, 6, 7, 8, 9, 10)`

d) typ: `scala.collection.immutable.IndexedSeq[Int]`
värde: `Vector(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)`

e) typ: `scala.collection.immutable.IndexedSeq[Int]`
värde: `Vector(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)`

f) typ: `scala.collection.immutable.IndexedSeq[Int]`
värde: `Vector(2, 3, 4, 5, 6, 7, 8, 9, 10, 11)`

g) typ: `Int`, värde: `Vector(65)`

h) typ: `scala.collection.immutable.IndexedSeq[Int]`
värde: `Vector(0.0, 0.707, 1.0, 0.707, 0.0, -0.707, -1.0, -0.707)`

Uppgift 5.

a) typ: `scala.collection.immutable.IndexedSeq[Int]`
värde: `Vector(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)`

b) typ: `scala.collection.immutable.IndexedSeq[Int]`
värde: `Vector(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)`

c) typ: `scala.collection.immutable.IndexedSeq[Int]`
värde: `Vector(2, 4, 6, 8, 10, 12, 14, 16, 18, 20)`

d) typ: `scala.collection.immutable.IndexedSeq[Int]`
värde: `Vector(2, 4, 6, 8, 10, 12, 14, 16, 18, 20)`

e) typ: `scala.collection.immutable.Vector[Int]`
värde: En vector av tiotusen 85or (85 = 42 + 43).

Uppgift 6.

a) En `Range` skapas och dess element skrivs ut ett och ett.

b) Samma sak händer.

c) De tio första tio första jämna talen (noll ej inräknat) skrivs ut med ett "hejframför.

d) Talen 1 till 10 skrivs ut.

e) Tiotusen slumptal mellan 0 och 1 genereras. Varje gång ett tal är större än 0.99 kommer det ett pling.

Uppgift 7.

a) Pseudokoden kan se ut såhär:
Skapa heltalsvariabel temp. Flytta värdet från x till temp. Flytta värdet från y till x. Flytta värdet från temp till y.

b)

```
scala> var (x, y) = (42, 43)
x: Int = 42
y: Int = 43
scala> var temp = x; x = y; y = temp;
temp: Int = 42
x: Int = 43
y: Int = 42
scala> println("x är " + x + ", y är " + y)
x är 43, y är 42
```

Uppgift 8.

a) Skriver ut "hej skript".

b) Ett felmeddelande skrivs ut.

c) Lägg till raden: `println((2 to 1001).sum)` eller motsvarande.

d) Filen ska se ut ungefär såhär:

```
val n = args(0).toInt
println("hej skript")
println((1 to n).sum)
```

e) `java.lang.ArrayIndexOutOfBoundsException: 0`

Uppgift 9.

a) Hello.class och Hello$.class

b) Ta bort en av hakparenteserna i slutet.

c) I ett skript behöver man inte skriva någon main-metod. Kompilatorn lägger till en automatiskt precis när koden ska köras. I en applikation behöver man däremot det. För att göra en applikation definierar vi ett objekt som vi i det här fallet kallar för `Hello`. Från början gör inte objekt någonting. De bara finns. För att objekt ska kunna göra något behövs det metoder. I vanliga fall utförs inte metoder förrän en annan metod roparpå metoden. main-metoden ropas dock automatiskt när en applikation startas. Annars hade ju ingenting hänt, eftersom alla metoderna väntar på att någon annan metod ska börja.

d) Första gången man ska köra en applikation måste den först kompileras innan den exekveras. Skript kompileras automatiskt samtidigt som de exekveras, vilket totalt sett görs på kortare tid. Därför tar det längre tid att starta en applikation första gången än att starta ett skript första gånge. När en applikation väl har kompileras och kan exekveras, går det dock mycket fortare. Fördelen med applikationer är att de kan exekveras flera gånger utan att kompileras om.

Uppgift 10.

a) Hi.class

b) I javas syntax början man med orden `public static`. I scala uteblir dessa. I scala är alla metoder automatiskt publika om inget annat används. Därför behövs aldrig ordet `public` i scala. I scala finns det tekniskt sett inga statiska metoder. Men i praktiken fungerar vanliga metoder i ett scala-objekt på ungefär samma sätt som statiska metoder i en java-klass. I scala används ordet **`def`** varje gång en funktion ska definieras. I java slipper man det. I java skriver man returtypen (`void`) innan parametrarna. I scala kommer istället metodens returtyp (`Unit`) i slutet. Javas `void` motsvarar scalas `Unit`. I scalas syntax kommer parameterns namn (`args`) före parameterns typ (`Array[String]`), separerat med ett kolon. I java kommer typen (`String[]`) först och sen kommer namnet (`args`). `String[]` i java betyder ungefär samma sak som `Array[String]` i scala.

c) -

Uppgift 11. a) Bugg: Eftersom `i` inte ökar, fastnar programmet i en oändlig loop. Fix: Lägg till en sats i slutet av while-blocket som ökar värdet på i med 1. Bugg: Eftersom man bara ökar summan med 1 varje gång, kommer resultatet att bli summan av n stycken 1or, inte de n första heltalen. Fix: Ändra så att summan ökar med `i` varje gång, istället för 1. För -1, blir resultatet 0. Förklaring: i börjar på 1 och är alltså aldrig mindre än n som ju är -1. while-blocket genomförs alltså noll gånger, och efter att `sum` får sitt ursprungsvärde förändras den aldrig. b) 39502716 c) - d) Såhär kan implementationen se ut:

```
public class SumN {
  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    int sum = 0;
    int i = 1;
    while(i <= n){
      sum = sum + i;
      i = i + 1;
      }
    }
    System.out.println(sum);
}
```

Uppgift 12. a) Bugg: i ökar aldrig. Programmet fastnar i en oändlig loop. Fix: Lägg till en sats som ökar i med 1, i slutet av while-blocket.

b) Så här kan implementationen se ut:

```
object Max {
  def main(args: Array[String]): Unit = {
    var max = Int.MinValue
    val n = args.size
```

```
    var i = 0
    while(i < n) {
      val x = args(i).toInt
      if(x > max) {
        max = x
      }
      i = i + 1
    }
    println(max)
  }
}
```

c) Raden där max initieras ändras till `var max = args(0).toInt`

d) `java.lang.ArrayIndexOutOfBoundsException: 0`

Uppgift 13. a) Skriver ut talet 8. `a` får värdet `4 + 4` eftersom detta är den sista satsen i blocket. Man får också tre stycken varningar. Detta beror på att det förekommer tre satser i blocket som inte gör någon skillnad.

b) Skriver ut talet 5. De tre första satserna i det yttre blocket ignoreras. `b` får värdet som returneras av det yttre blocket. Det yttre blocket returnerar värdet som returneras i den sista satsen i blocket, som i sin tur är ett block. I det inre blocket skapas en ny `val` som också får namnet `b`. Notera att detta alltså inte är samma värde, även om det har samma namn. Den andra satsen räknar summan av `b` med sig själv. Eftersom vi nu befinner oss i det block där det andra bet precis har definieras så är det detta `b` som används och summan blir alltså åtta. Detta är dock helt irrelevant eftersom resultatet inte sparas någonstans. I den sista satsen blir resultatet 5 (eftersom `b` är fyra och vi adderar ett). Detta resultatet returneras från det innre blocket och vidare ur det yttre blocket.

c) Skriver ut talet 42. Blockets satser exekveras i ordning.

d) Skriver inte ut 42. I blocket skapas ett `val` med namnet `a` och värdet 42. Detta värde finns inte utanför blocket och kommer därför inte att skrivas ut. Om du däremot definierat `a` som något annat tidigare så kommer istället det värdet att skrivas ut.

e) Skriver först ut `43` och sedan `42`. Förklaring:

`a` initieras med värdet `42`. Ett nytt värde som också har namnet `a` initieras med värdet `43`. Eftersom detta sker innanför ett nytt block, befinner vi oss i ett annat namespaceöch det gör alltså inget att vi använder samma namn. `a` skrivs ut. Eftersom vi befinner oss i det innre blocket är det `43` som skrivs ut, inte `42`. Scala kollar först efter värden som heter `a` i det innre namespacet". Det är först i andra hand som den skulle upptäcka att det finns ett `a` i det yttre blocket. Till sist körs den sista satsen i det yttre blocket. Då skrivs `a` ut. Eftersom vi nu befinner oss i det yttre blocket, vet inte ens scala om att det anda aet existerar. Resultatet av den här utskriften blir allts `42`.

f) Ett fel uppstår. Variabeln `a` initieras två gånger i samma namespace.

Förklaring till felet:

I det yttre blockets första sats initieras variablen `a` med värdet `42`. I det yttre blockets tredje sats försöker vi definiera en ny variabel med samma namn. I och med att vi befinner oss i samma namespace, krockar namnen.

Förklaring till vad som händer i sats två:

I det innre blocket har vi inte definierat någon variabel `a`. Till en början hittar alltså inte scala något sådant. Då letar scala vidare i det namespace som finns utanför det innre blocket och hittar variabeln som vi definierade i det yttre blockets första sats. Denna variabel får sitt värde förändrat.

g) Fel. Framåtreferens. Förklaring:

Det är inte tillåtet att referera till variabler som initieras senare i koden.

h) Skriver ut `85`. Förklaring:

I och med att vi den här gången initierade variabeln `b` och gav den ett värde innan vi använder oss av den, slipper vi problemet ovan.

i) Skriver ut `85`. Förklaring:

Det är tillåtet att referera till funktioner som definieras senare i koden.

j) Skriver ut `85`. Förklaring:

`a.b` refererar till variabeln `b` som ingår i objektet `a`. `a.a.a` refererar till variabeln `a`, som ingår i ett objekt som heter `a` som i sin tur befinner sig i ett annat objekt som också heter `a`.

k) Skriver ut `85`. Förklaring:

Koden är identisk med förra deluppgiften förutom att ny rad används istället för semikolon.

l) I stora projekt med mycket kod, kan det vara svårt att hitta unika namn till alla sina variabler. Då är det en fördel om man kan hålla sina variabler i begränsade namespaces, så att de bara är tillgängliga precis när de behöver användas.

Uppgift 14.

a) `script` `security` `smartcardio` `sound` `sql` `swing`

b) Radernas funktion i ordning:

1. Importerar JOptionPane från javax.swing

2. Definierar en metod som tar en sträng och öppnar en dialogruta med strängen.

3. Testar funktionen med argumentet "Hej på dej!". En dialogruta öppnas med texten "Hej på dej!".

4. Definierar en metod som tar emot en sträng som argument och öppnar en input-dialogruta med strängen.

5. Testar funktionen med argumentet "Vad heter du?". En dialogruta öppnas med texten "Vad heter du?". I ett fält kan man fylla i sitt namn. Funktionen returnerar namnet.

6. Importerar showOptionDialog från JOptionPane under namnet optDlg.

7. Definierar en metod som tar emot en sträng och en Array som argument och öppnar en flervalsdialog. Strängen ska innehålla frågan som flervalsdialo-

gen visar upp. Arrayn ska innehålla alternativen som användaren ska välja mellan.

8.Testar funktionen med argumenten `"Vad väljer du?"` och `Array("Sten, "Sax", "Påse")`. En dialogruta kommer upp och man får möjlighet att välja sten sax eller påse. Funktionen returnerar valet som man gör.

c) På alla ställen där `JOptionPane` förekommer, hade man istället fått skriva `javax.swing.JOptionPane`.

d) -

Uppgift 15. a) jar cvf [namn på skapad fil] [namn på input-filer]

b) -

Uppgift 16. a) -

b) -

3. Lösningar till övning `functions`

Uppgift 1.

a)

```
def öka(x: Int): Int = x+1
```

b) `Int`

c) Kompilatorn försöker lista ut vad för returtyp det är och lägger till det själv.

d) För om det skulle bli error och du förväntar dig en returtyp men får en annan så kan det betyda att du har en bug i programmet. Det gör även programmet mer lättläsligt.

e) När man pratar om parameter menar man variablen som används som indata till funktionen, medans argument är den faktiska indatan. Så i detta fallet så är x parametern, vilket är namnet vi get parametervariablen, medans 42 är argumentet.

f) `46`

g)

```
def minska(x: Int): Int = x-1
```

h) `42`

Uppgift 2.

a) `-100`

b) `15`

c) `185`

d) `256`

Uppgift 3. Funktionen lägger helt enkelt ihop två värden och om det andra saknas så lägger den bara på 1 till första värdet, d.v.s. om andra värdet saknas antas det vara 1.

Så det första anropet returnerar 44, det andra 43 och det sista returnerar även det 43 då j antas vara 1.

Uppgift 4.

a) Utskriften blir följande:

```
Namn: Finkodare, Kim
Namn: Oval, Viktor
Namn: Triangelsson, Stina
```

Eftersom vi har namngett argumenten så behöver vi inte nödvändigtvis skriva argumenten i rätt ordning. Att namnge argumenten i andra anropet gör ingen skillnad, men i tredje så skriver vi egentligen in argumenten i fel ordning.

b) Det blir lättare att hålla reda på vad som är vad både när man skriver och sedan när man ska läsa koden. Det låter oss även skriva argument i den ordningen som känns naturligast för oss istället för vad funktionen har dikterat.

Uppgift 5. Detta är alla olika sätt att köra en funktion över alla element i en samling

a) `Vector(1, 2, 3, 4, 5)`

b) `Vector(0, 1, 2, 3, 4)`

c) `Vector(1, 2, 3, 4, 5)`

d) `Vector(0, 1, 2, 3, 4)`

e) `Vector(1, 2, 3, 4, 5)`

f) `Vector(0, 1, 2, 3, 4)`

g) `Vector(13, 4, 42, -7)`

h) `Vector(11, 2, 40, -9)`

Uppgift 6.

a) `dallas`

b) `dallas`

c) Koden skriver ut `dallas` när du sparar variablen för proceduren skriver ut när den anropas, men den returnerar inget så `print(x)` skriver inte ut något

d) Det är en procedur så den returnerar inget att skriva ut

e) Det är en procedur så den returnerar inget att skriva ut

f) Det är en procedur så den returnerar inget att skriva ut

g) För att man ska vara säker på att den faktiskt är en procedur och inte gör något man inte väntat sig. Det är ett bra sätt att bugtesta för ifall det blir ett exception så kanske det finns något fel man borde kolla på.

Uppgift 7.

a) `snark, Int = 42`

b) `snark snark snark, Int = 42`

Den evaverar inte uttrycket förrän det behövs så flera snark ger bara ett resultat för det är bara det som begärs

c) `2`

d) `2`

Det är ingen större skillnad när vi bara skickar in en etta.

e) `snark, Int = 84`

f) När vi kallar på `callByValue` så skickar vi in snark som ett värde, och därmed behöver x bara beräknas en gång medans när vi kallar på `callByName` så skickar vi den som en funktion och därmed kallas snark på två gånger i funktionen för att beräkna x båda gångerna

`snark snark, Int = 84`

g) `görDetta` är en kontrolstruktur som helt enkelt exekverar koden den matas med, `görDettaTvåGånger` gör just vad den säger. Detta är dock grunderna för vad som kan bli mycket mer advancerat om man kombinerar detta med loopar eller if-satser, som i sig är kontrolstrukturer.

Uppgift 8. Först så adderas 22 och 20 för att bli 42

Sedan adderas först 1 och 19 och det adderas sen med 22 för att tillslut bli 42.

Uppgift 9.

a) -

b) I den första parameterlistan så anges hur många gånger koden i den andra parameterlistan ska exekveras. så `upprepa(10)(println("hej"))` printar hej 10 gånger

Uppgift 10.

a) Man kan spara en funktion som en variabel och funktioner kan likt andra värden sparade i variabler användas i andra funktioner så länge det är godkänd indata. Så femte raden blir identisk med tredje.

b) Kompilatorn saknar en parameterlista så den kastar ett fel, genom att ange _ så säger vi åt den att vi tar emot en godtycklig parameter.

c)

```
def dec(x: Int): Int = x - 1
val g = dec _
Vector(12, 3, 41, -8).map(g)
```

d) Int => Int

g har samma typ som f alltså Int => Int

e) d = 84, h = 21. Räkna tar en funktion och ger den det första argumentet som indata.

Uppgift 11. När man gör curryfunktioner så skjuter man upp att ange det andra värdet till senare och på så sätt gör nyafunktioner så att säga. När vi sparar undan variablen `f` så har vi angett första argumentet men den väntar fortfarande på det andra som vi anger sen vilket ger ett resultatvärde.

Samma sak senare, genom att skapa variablerna inc och dec som summan av +1 respektive -1 så har vi skapat"våra `inc` och `dec` funktioner från tidigare funktioner.

Uppgift 12.

a) -

b) -

c)

```

---- Frekvenser ----
```

```
Antal tecken: 1932
Antal ord: 337
Antal meningar: 84

---- Frekvenser ----
Antal tecken: 1890
Antal ord: 295
Antal meningar: 126

---- Frekvenser ----
Antal tecken: 3824
Antal ord: 633
Antal meningar: 210
```

Först och främst, vi har default argument 42 så det är det som används vid anropet. Detta används för att multiplicera `s1` och `s2` i `Test` när man anropar `printFreq`. `statistics` objektet använder i sin tur funktionerna i `stringfun` för att räkna ord och meningar. Resultatet presenteras av `statistics`, vilket är det vi ser.

d) vi ser att stringfun är enbart funktioner och har därmed ingt tillstånd.

statistics har ett tillstånd eftersom variablen history sparar vad som skett, så när man anropar funktionener i objektet så kan objektet ändra beteende beroende på dess tillstånd, tillståndet består alltså av history.

Uppgift 13.

a) inc, addY och isPalindrome. Notera att y sätts till x's värde i början vilket är 0 och sedan kan det inte ändras eftersom det är en **`val`**.

b) Försök att med samma argument få olika resultat med samma funktion genom att ändra x

c) Vad är x och y?

d) y

Uppgift 14.

a) Först notera att `plus` med bara paranteser och med `apply` är indentiska anrop, för det är det som är tricket med ett object med `apply` metod man kan kalla på den som en funktion.

Sedan sparar vi en funktion som add, men eftersom alla fukntioner också är object så så har de några andra metoder man också kan anropa vilket vi ser med TAB.

Till sist så sparar vi en ny funktion inc som en curry funktion av add med 1 som argument.

b)

```
1 scala> object slumptal{ def apply(a: Int, b: Int) = (math.random * b + a).toInt
2 defined object slumptal
3
4 scala> (1 to 100).foreach{i => print(slumptal.apply(1,6) + " ")}
5 3 5 6 6 5 3 4 3 2 3 1 3 1 2 2 5 1 2 6 2 1 1 4 5 5 3 4 6 5 1 1 2 3 1 1 1 4 4 6 1
6 scala> (1 to 100).foreach{i => print(slumptal(1,6) + " ")}
```

```
7 4 2 1 6 5 2 5 2 2 3 1 3 3 3 5 1 6 1 1 2 2 2 2 2 3 6 1 5 6 4 2 3 3 2 2 4 1 5 4 6
```

Uppgift 15.

a) Notera vid deklarationen av `nu`, `sen` och `igen` så är det bara `nu` som tar tid och skriver ut sin text. För `nu` evalueras men de andra väntar.

Men när vi ska kalla på dem så tar `nu` ingen tid och skriver inte ut nu medans `sen` och `igen` nu tar tid och skriver ut sin text. För `nu` har redan evaluerats men de andra behöver evalueras för de kallas på.

Och när vi kallar på dem för andra gången så är det bara `igen` som tar tid. För `nu` och `sen` är evaluerade och vid det här laget identiska medans `igen` behöver evalueras varje gång man kallar på den.

Vid deklaration av objektet så går allt bra, när vi kallar på `liten` så får vi 42 men när vi kallar på `stor` så får vi exception. Eftersom objekt är lata så evalueras inte objektet förrän vi anropar något ur det men `stor` är också lazy så den blir inte ett problem förrän just den anropas.

b) **`val`** evalueras direkt, **`lazy val`** evalueras när det behövs medans **`def`** evalueras varje gång det behövs.

c) När vi skapar objektÄrLata så skrivs inget ut för det evalueras inte förrän vi kallar på det, som vi gör nästa rad då skrivs nu ut. När vi sedan anropar sen evalueras inget och nu skrivs inte ut.

{**`val`** `x=y;` **`val`** `y = 42`} kastar error eftersom y inte är evaluerad när vi försöker evaluera x.

När vi skapar `buggig` så varnar den och vi ser varför på nästa rad då att `a` verkar vara lika med 0. Eftersom `b` inte är evaluerad än så blir det 0 men till skillnad från tidigare vet **`val`** `a` iallafall att den är en Int eftersom objectet med `b` är åtminstone definerat.

`funkar` fungerar just för att `a` är en **`lazy val`** och evalueras då inte förrän efter `b` evaluerats och ger därför 42 som den ska.

`nowarning` har samma problem som `buggig` men ger ingen varning och därför försöker skriva ut innan `one` är evaluerad och ger därför nollor.

d) När man vill skapa ett objekt som funkar eller fixa problemet i no warning, man kan vilja ha sina variabler i just den ordningen p.g.a. läslighet.

Men även om man inte vill slöa ner ett system med en massa arbete när allt deklareras på en gång utan tar det hellre gradvis allteftersom det används.

Uppgift 16.

a) Utskrift:
inc[x = 0]
dec[x = 1]
inc[x = 0]
1

`inc` kallar på `dec` som i sin tur kallar på `inc` med argumentet 0. den sista funktionen returnerar sedan 1 varpå nästa minskar till 0 och den sista ökar till 1.

add[x = 1, y = -2]
dec[x = 1]
dec[x = 0]
add[x = 1, y = -1]
dec[x = 1]
add[x = 1, y = 0]
1

add kallas på tre gånger, add längst ner kallar på dec två gånger på varandra för att returnera -1 till andra inc som tar det som argument.

Den andra add kallar på dec en gång och returnerar 0 till första add

Den ursprungliga add returnerar till sist 1 utan att kalla på några andra funktioner

b) Stacken för x när den är som djupast

inc
dec
inc

När vi har nått tredje add och den kallat på dec så ser den ut så här för två iterationer.

dec
add
add
add

Sedan vid det andra add så ser den ut så här i en iteration varpå den sedan rensas

dec
add
add

Uppgift 17.

a) 1
add[x = 1, y = -2]
dec[x = 1]
dec[x = 0]
add[x = 1, y = -1]
dec[x = 1]
add[x = 1, y = 0]
1
Vi saknar utskrift när den arbetar med x för den använder de funktioner som den känner till vilket är de versioner utan utskrift, när vi kallar på add så får vi dock utskrift när vi kallar på inc samt dec för add använder sin lokala variant.

b) Den stora fördelen är att man kan separera kod i funktioner lokalt utan att påverka något utanför och därmed göra sin kod mer lättläst. Det kommer även dock till stor nytta ifall man vill ha annan funktionalitet lokalt än allmänt, då kan man definera en ny lokal funktion med samma namn som gör något lite anorlunda.

Uppgift 18.

a) `Vector(2, 3, 4, 5` Lägger till 1 på varje
`Vector(2, 3, 4, 5` Identisk med ovan
`Vector(2.0, 4.0, 8.0, 16.0` Ger 2 upphöjt med talen från vektorn
`Vector(1.0, 4.0, 9.0, 16.0` Talen upphöjt med 2
`Vector(1, 2, 3, 4` Skriver ut talen som sträng
`Vector(1, 2, 3, 4` Identisk med ovan

b) Int => Int Int => Int Int => Double Int => Double Int => String Int => String
Den vet vad den behöver utgå ifrån och på samma sätt som man i vanliga fall inte explicit behöver ange returtyp för funktioner i scala så ser den vad för typ som returneras t.ex. `i.toString` returnerar en String

c) Missing Parameter type och Missing Parameter type for expanded function

Uppgift 19.

a) `countdown` skriver ut x och kallar på `countdown` igen med x-1 som argument om det är större än noll vilket innebär att samma sak görs igen tills x når 0.

`finalCountdown` gör samma sak fast med en Byte och den fortsätter även om x passerar 0 med de rekursiva funktionsanropen.

b) Eftersom vi hade `1/x` efter rekursionsanropet innan så kom vi aldrig dit för vi returnerade aldrig något utan gick bara djupare i stacken. Om vi placerar `1/x` tidigare så når vi den raden kod och den kastar ett exception då det är division med noll.

c) Den sista raden leder till mycket fler rekursiva anrop, för rekursionen avslutas när y är noll, inte om x är det.

Uppgift 20.

```
1 def avg(x: Int, y: Int): Double = (x+y)/2
```

Uppgift 21.

```
1 def dist(x1: Int, y1: Int, x2: Int, y2: Int): Double =
2  Math.sqrt(Math.pow((x1+x2), 2)+(Math.pow((y1+y2), 2)))
```

4. Lösningar till övning `data`

Uppgift 1.

a)

```
scala> val pt = (15.9, 28.9)

scala> math.hypot(pt._1, pt._2)
res0: Double = 32.98514817307935
```

b) `val (x, y) = pt`

c) `(String, String, Double, Boolean)`

d) `Vector[(Double, Double)]`

e) `huvudstäder :+ ''Danmark'' -> ''Köpenhamn''`

f) `Vector[Double]`

g) `val antalUdda = (1234 to 3456).map(i => div(i, 2)._2).sum`

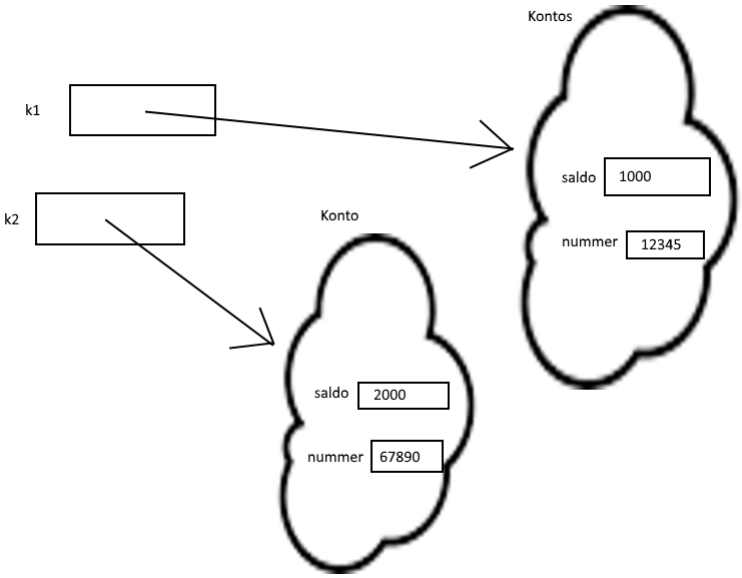
h) 0-tupel

Uppgift 2.

a) `mittKonto.saldo = (math.random * 1000000).toInt`

b) Går ej eftersom val är oförändlig, man får alltså ett Error.

Uppgift 3.



a)

b) Tilldelningen på rad 8 `k1.nummer = 12345L` ger felmeddelande eftersom variablen är oförändlig.

Uppgift 4.

a) `String = Konto@cd576`, där `Konto@cd576` är ett unikt namn som identifierar instansen.

b) Ja.

c)

```
scala> k.saldo = 42
scala> k2.saldo = 67
```

d) Eftersom variablen är oförändlig ges ett felmeddelande.

e) En fördel med klass är att man kan specificera att variablen ska kunna vara förändlig. En till är att man kan inkludera metoder i klassen som man vill kunna använda på värdena.

Uppgift 5.

a) Det går bra att ändra på variablen saldo i instansen av Konto1 men inte av Konto2 där man får ett error på raden ”k2.saldo += 1000”

b) -

c) ”println(k.saldo)” och ”k.saldo += 1000” ger båda error, pga privat attribut.

d)

```
def ut(belopp: Int): (Int, Int) = {
  if(saldo >= belopp) {
    saldo -= belopp
    (belopp, saldo)
  } else {
    val temp = saldo
    saldo = 0
    (temp, 0)
  }
}
```

e) Lägg till en if-sats i båda funktionerna som omsluter den gamla koden.

```
def ut(belopp: Int): (Int, Int) = {
  if(belopp >= 0) {
    if(saldo >= belopp) {
      saldo -= belopp
      (belopp, saldo)
    } else {
      val temp = saldo
      saldo = 0
      (temp, 0)
    }
  }
}

def in(belopp: Int): Unit = {
  if(belopp >= 0) {
    saldo += belopp
```

```
  }
}
```

f) Genom att göra attributet privat och gör egna metoder kan man se till att attriuten endast ändras på säkra sätt. Så inte fel uppstår.

Uppgift 6.

a) ”val i: Int = pt.x” error: type mismatch; Eftersom typen Int ej är kompatibel med ett värde av typen Double.

”val p: Double = new Punkt(5.0, 5.0)” error: type mismatch; Eftersom typen Double ej är kompatibel med ett värde av typen Punkt.

”val p = new Punkt(5.0, 5.0): Double” error: type mismatch; Eftersom typen Double ej är kompatibel med ett värde av typen Punkt.

b) Rad 3 till 7 i respektive ordning: true, false, false, true och false.

Uppgift 7.

En variabel med namn pt skapas med typen Punkt.
true
true
String = 1.0
skriver ut: 1.0
error: not found: value a
String = 2.0
error: not found: value a

Uppgift 8.

a) ”println(pt)” kallar på pt.toString, och eftersom metoden är överskriven kallas den nya version.

b) `override def toString: String = ''Punkt('' + x + '', '' + y + '').''`

c) error: overriding method toString in class Object of type ()String;

Uppgift 9.

a)

```
1  scala> val pt = Pt(1.0, 2.0)
2  pt: Pt = Pt(x=1.0,y=2.0)
3
4  scala> Pt(4.0, 2.0)
5  res0: Pt = Pt(x=4.0,y=2.0)
6
7  scala> Pt(6.0, 3.0)
8  res1: Pt = Pt(x=6.0,y=3.0)
9
10 scala> Pt(666.0, 1337.0)
11 res2: Pt = Pt(x=666.0,y=1337.0)
```

b) `def apply(): Pt = new Pt(0, 0)`

c) `class Rational(val nom: Int, val denom: Int)`

d)

```
object Rational {
def apply(nom: Int, denom: Int): Rational = new Rational(nom, denom)
}
```

e)

```
scala> Rational(2, 5)
scala> Rational(2, 7)
scala> Rational(7, 4)
scala> Rational(666, 1337)
```

Uppgift 10. a) `case class Rational(nom: Int, denom: Int)`

Uppgift 11.

a)

```
scala> Point(3, 4).distToOrigin
res0: Double = 5.0
```

b) p3.x = 8 p3.y = 10

Uppgift 12.

a)
Operatornotation: 4, 6, 10, 12
Punktnotation: 3, 5, 8, 9, 11, 13
Felmeddelande: 9

b)

```
case class Point(x: Double, y: Double) {
  def distToOrigin: Double = math.hypot(x, y)
  def add(p: Point): Point = Point(x + p.x, y + p.y)
  def +(p: Point): Point = add(p)
  def sub(p: Point): Point = Point(x - p.x, y - p.y)
  def -(p: Point): Point = sub(p)
}
```

```
scala> val p1: Point = Point(1, 9)
scala> val p2: Point = Point(9, 6)
scala> p1.sub(p2)
scala> p1.-(p2)
scala> p2 sub p1
scala> p2 - p2
scala> p1.add(p2.sub(p1))
scala> p1 + (p2 - p1)
```

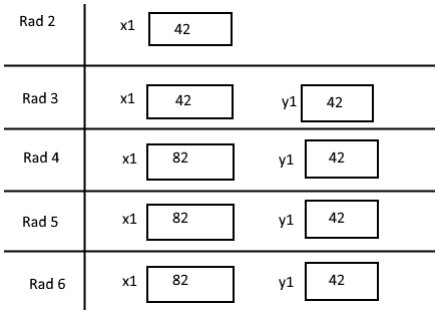
c)

```
case class Point(x: Double, y: Double) {
  def distToOrigin: Double = math.hypot(x, y)
  def add(p: Point): Point = Point(x + p.x, y + p.y)
  def +(p: Point): Point = add(p)
  def sub(p: Point): Point = Point(x - p.x, y - p.y)
  def -(p: Point): Point = sub(p)
  def scale(a: Double, b: Double) = Point(x * a, y * b)
}
```

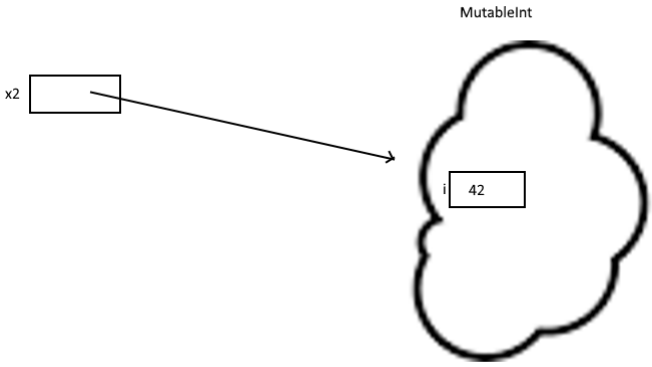
```
1 scala> val p: Point(13,  37)
2 scala> p.scale(4, 2)
3 scala> p scale (3, 7)
```

Uppgift 13.

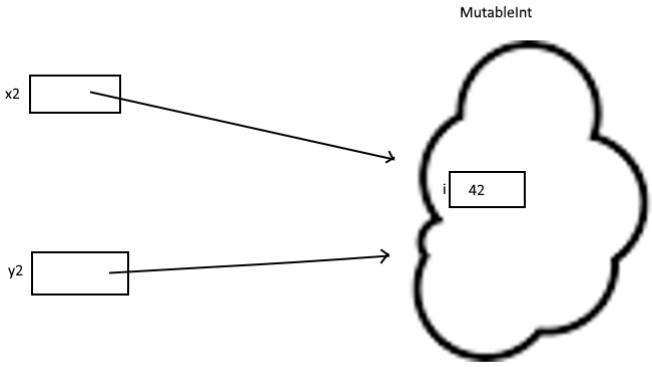
a)



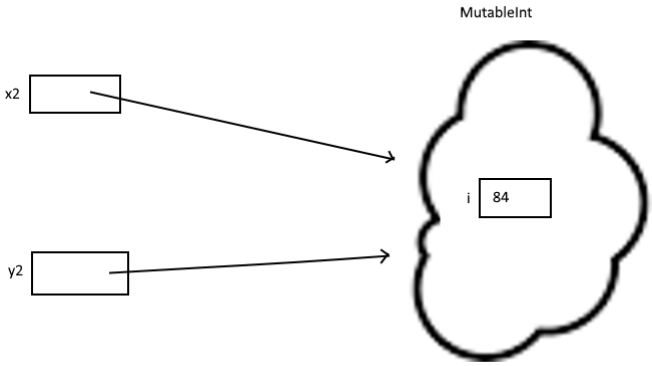
b)



1.

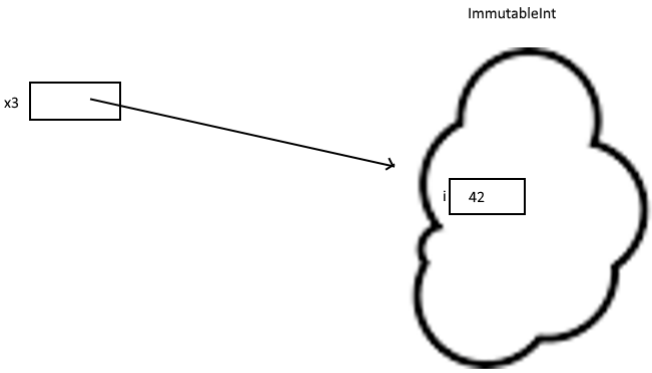

MutableInt
x2
i
42
y2

2.

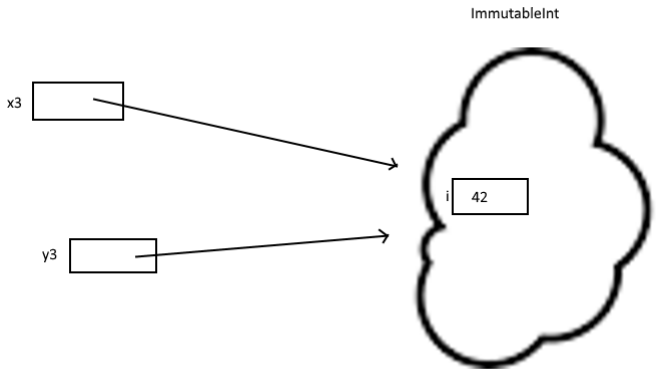

MutableInt
x2
i
84
y2

3.

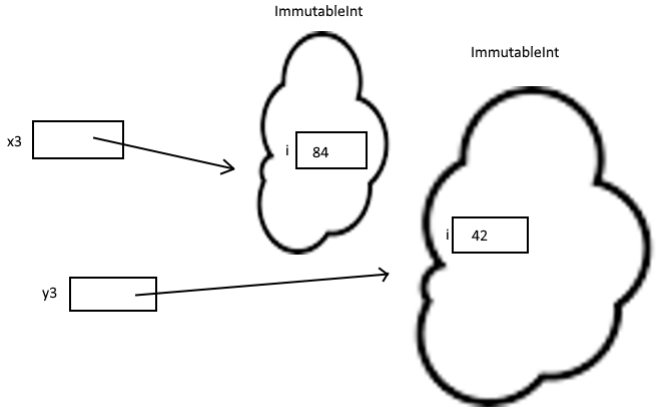
c)


ImmutableInt
x3
i
42

1.



2.



3.

d) En stor fördel är att vi till exempel kan skicka med en immutable som argument till en metod och vara säkra på att metoden inte ändrar på värdet.

Uppgift 14.

a) Ungefär 150 metoder.

b) Ungefär lika många.

Uppgift 15.

a)
1. Instansierar en tom vektor med element av typen int och tilldelar värdet till en variabel xs.
2. Error eftersom `xs :+ ''42''` ger en Vector[Any] när Vector[Int] krävs.
3. xs tilldelas ett nytt värde av Vector(43, 64, 46)
4. xs skrivs ut.
5. Lägger till talet 42 i xs.
6. Error: type mismatch
7. Skapar en tom Vector i variablen ingenting
8. ingenting får värdet Vector(1, 2, 3)

b) Tre metoder skapas, första för att få första elementet i en lista, och eftersom den defineras med special typen T går den att använda med alla Vek-

torer oavsätt typen av variable i vektorn. Den andra får fram sista elementet och sista hämtar båda två.

En till function defineras längre ner ”wrap”, som tar en lista och lägger till ett element längst fram och ett längst bak.

Uppgift 16.

a) String = ”ka2”

b) String = ”abra”

c) false

d) false

e) 100000

f) 100000

g) minsta talet i listan

h) största talet i listan

i) 1

j) 3

k) Vektor b fast med ”först” som första element

l) Vektor a fast med ”sist” som sista element.

m) plats 3 i vektorn xs får värdet 42

n) En ny vektor fylld med ”!” från och med plats 4 till 10. Men de andra värdena samma som i a.

o) b sorterad i bokstävsordning

p) b baklänges

q) true

r) true

s) en vektor med alla unika element i b.

Uppgift 17.

a) Metoden ger tillbaka en ny Vector[String] som nu består av alla element i a plus alla element i b. I samma ordning med elementen i a först.

b) Samma som i uppgift a fast vektorn som returnas är av typen Vector[Any]. Det är eftersom Any är den närmsta typen som String och Double delar. Elementen från vektor a är fortfarande först och uppföljt av elementen i stor.

c) Variablen ys får värdet av en Vector[Int] som innehåller alla talen från xs fast multiplicerade med 5. Alltså ys = 5, 10, 15..., osv.

d) Functionen tar alla värden från en Vektor och sätter in i ett Set (mängd). Eftersom en mängd ej har dubletter så försvinner ett ”sala” och ett ”bim”, Vector[String] som returneras blir därför (”sim”, ”sala”, ”bim”).

e) Metoden head ger första elementet i en samling, och last sista. Därför blir kombinationen av a.head och b.last en ny Vector[String] som består av a:s första element, och b:s första element.

f) Ger en Vector[String] som innehåller alla element efter det första. Alltså i detta fallet ”ka” och ”dabra”.

g) True, eftersom head ger första elementet och tail ger resten, sedan sätter metoden +: ihop dem till en vektor med samma värden som a.

h) Eftersom ++ sätter ihop alla värden från två vektorer måste vi först omvandla från en sträng till vektor. Resultatet blir en ny vektor av samma typ som innan med a:s första element och b:S sista.

i) Samma resultat som i h, metoden take börjar från vänster och tar så många element som man skickar med som parameter och gör till en ny lista. Med 1 som parameter motsvarar det att göra Vector(a.head). Metoden takeRight gör samma sak fast från höger.

j) Metoden drop är motsvarigheten till take fast excuderar de speciferade elementen istället för att inkludera dem i vektorn.

k) Eftersom a endast innehåller 3 element returnerar drop(100) en tom vektor.

l) Returnerar en tom vektor med element typen String

m) returnerar Vector(true, false)

n) True, metoden contains kollar om en samling innehåller ett specifikt element.

o) True. Eftersom en sträng även kan ses som Vector[Char].

p) Filtrerar vektorn a till att endast innehålla strängar som innehåller k.

q) Exakt samma som i p

r) map(_.toUpperCase) omvandlar alla strängar i a till stora bokstäver filterNot(_.contains(”K”)) tar resultatet vi precis fick och tar bort alla strängar som innehåller stora K.

s) filtrerar så att endast jämna tal finns kvar.

t) Exakt samma som i s

Uppgift 18.

a) Vi instansierar en vektor xs med talen 1, 2 och 3. sedan definerar vi en metod blandat som ger oss en randomiserad version av xs. sedan definerar vi en till metod som testar om xs är lika med resultatet från blandat. Om det är så returnerar den strängen ”lika” annars ”olika”. Sist kör vi en for-loop där vi 100 gånger kör testet, samtidigt räknas hur många gånger ”lika” returneras.

Vårt resultat är en siffra på hur många gånger xs var samma som en blandad version av sig själv, eftersom det finns 6 permutationer med 3 variabler så borde det vara ungefär 1/6 chans.

b) -

c)
`map(\_.trim)` tar bort alla onödiga mellanrum i början och slutet på varje rad
`filterNot(\_.startsWith(''<''))` filtrerar bort alla rader som börjar med strängen ”<”

`filterNot(\_.isEmpty)` filtrerar bort alla tomma rader.
`foreach(println)` skriver ut alla rader.

Uppgift 19.

a) I princip alla metoder delas, en lista har några fler t. ex. ”::”, ”:::”, ”mapConserve” osv.

b) Först skapas en lista med 4 sträng värden och instansierar variablen xs med det värdet. sedan skapar vi en ny lista, som består av ”zero” + den gamla listan och ger värdet till xs. Sist instansierar vi en ny variabel ys, som får värdet av xs omvänd plus xs.

Uppgift 20.

a) true, Boolean

b) En samling av alla värden i s och t, Set[String]

c) true, Boolean

d) false, Boolean

e) false, Boolean

f) false, Boolean

g) true, Boolean

h) Samlingen s utan elementet ”Stockholm”, Set[String]

i) Samlingen t utan elementen ”Norge” och ”Danmark”, Set[String]

j) returnerar s, Set[String]

k) Samlingen s utan ”Malmö” och ”Oslo”, Set[String]

l) Set(2, 3), Set[Int]

m) se deluppgift l

n) Set(1, 2, 3 ,4), Set[Int]

o) se deluppgift n

Uppgift 21.

a) Returnerar strängen ”Malmö” eftersom det värdet är indexerat på platsen ”Skåne”.

b) Returnerar strängen ”Stockholm” eftersom det värdet är indexerat på platsen ”Sverige”.

c) true, eftersom huvudstar innehåller indexet ”Skåne”

d) false, eftersom huvudstad ej innehåller indexet ”Malmö”. Notera att det är index och inte värden vi kollar om det finns.

e) Lägger till indexet ”Danmark” med värdet ”Köpenhamn” i samlingen.

f) Skriver ut alla 2-tupler.

g) Returnerar ”Oslo”, Note: Om indexet ”Norge” inte hade funnits hade ”???” returnerats istället.

h) Returnerar ”???”

i) Returnerar en sorterar vektor med alla index.

j) Returnerar en sorterar vektor med alla värden.

k) Returnerar en ny mängd men med ”Skåne” -> ”Malmö” borttaget.

l) Returnerar huvudstad mängden eftersom det inte finns ett ”Jylland” index att ta bort.

m) Uppdaterar indexet ”Skåne” till att istället leda till värdet ”Lund”

Uppgift 22.

a)

```
pairs: scala.collection.immutable.Vector[(String, Long)] =
         Vector((Björn,444), (Maj,441), (Lucy,666))
```

b) Map[String, Long]

c)

```
scala> telnr(''Maj'')
res0: Long = 441

scala> telnr.get(''Maj'')
res1: Option[Long] = Some(441)

scala> telnr(''Kim'')
java.util.NoSuchElementException: key not found: 'Kim
  at scala.collection.MapLike$class.default(MapLike.scala:228)
  at scala.collection.AbstractMap.default(Map.scala:59)
  at scala.collection.MapLike$class.apply(MapLike.scala:141)
  at scala.collection.AbstractMap.apply(Map.scala:59)
  ... 32 elided

scala> telnr.get(''Kim'')
res2: Option[Long] = None
```

d)

```
scala> telnr.getOrElse(''Maj'', -1L)
res0: Long = 441

scala> telnr.getOrElse(''Kim'', -1L)
res1: Long = -1
```

e) telnr += ”Fröken Ur” -> 464690510L

f) telnr.toVector.map(p => p._1 -> (”0” + p._2.toString.substring(2)))

g) Använd metoden toMap och apply.

Uppgift 23.

a) Metoden maxBy hämtar det element som är ”störst”, på rad två gör `x => x._1` att första värdet i tuplerna används för att bestämma vilken som är störst. Likt gör `x => x._2` på rad tre att istället det andra värdet används.

b)

```
scala> xs.maxBy(_._1)
scala> xs.maxBy(_._2)
```

c)

```
scala> xs.minBy(_._1)
scala> xs.minBy(_._2)
```

Uppgift 24.

Metoden `sliding(i: Int).toVector` skapar en ny vector med alla möjliga permutationer av elementen i en vektor med storleken i.

5. Lösningar till övning sequences

5.1 Grunduppgifter

Uppgift 1.

a) 42;
1
2
7
42;
WrappedArray(hej, på, dej)

b) WrappedArray

c) def printAll(xs: Int*) = println(xs.size); xs.foreach(println)

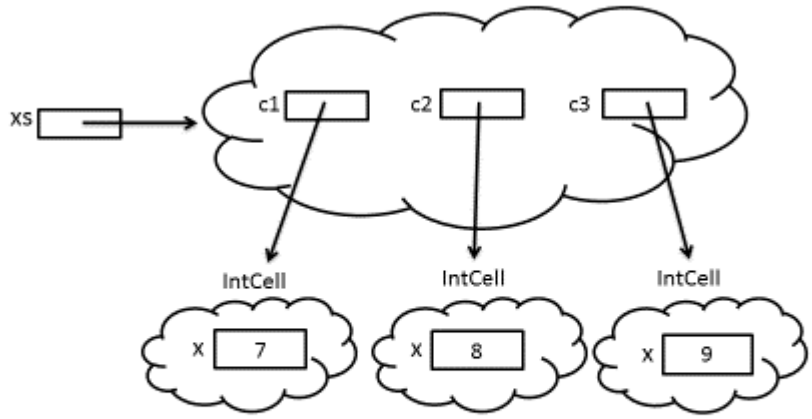
d) Storleken "0" skrivs ut och inget annat.

Uppgift 2.

a)

```
scala.collection.immutable.Vector[IntCell] =
    Vector([Int](7), [Int](42), [Int](9))
```

Referensena till c2 och xs ändras aldrig. xs kommer fortfarande ha tre vektorer som refererar till c1, c2, c3, däremot refererar dessa i sin tur till var sin int som är Mutable. I detta fallet ändras c2.x:s referens från 8 till 42.



b)

c) Istället för att skriva `IntCell(var x: Int)` så kan man skriva `IntCell(val x: Int)` där varje cells intvärde kommer vara oförändlig. Alltså då attributen till objek-

ten är “Val” så kommer även de att vara oförändliga.

Uppgift 3.

a)

```
def copyAppend(xs: Array[Int], x: Int): Array[Int] = {
  val n = xs.size
  val ys = new Array[Int](n+1)
  var i = 0
  while(i < n) {
    ys(i) = xs(i)
    i += 1
  }
  ys(n) = x
  ys
}
```

b)

```
xs: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer()
ArrayBuffer(1, 1, 2)
ArrayBuffer(1, 1, 2, 3)
ArrayBuffer(1, 1, 2, 3, 5)
ArrayBuffer(1, 1, 2, 3, 5, 8)
ArrayBuffer(1, 1, 2, 3, 5, 8, 13)
ArrayBuffer(1, 1, 2, 3, 5, 8, 13, 21)
ArrayBuffer(1, 1, 2, 3, 5, 8, 13, 21, 34)
ArrayBuffer(1, 1, 2, 3, 5, 8, 13, 21, 34, 55)
ArrayBuffer(1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89)
ArrayBuffer(1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144)
Int = 144
Int = 12
```

c) `xs.size = 46`
`xs(45) = 1836311903`
(Ha en arrayBuffer av typen Long istället och byt 100 mot Int.MaxValue och ta det nästsista elementet i sekvensens (det sista kommer vara över))

Uppgift 4.

a) Nej det gör det inte. Då ys tilldelas xs.toArray kopieras datan från xs över i en array (som är mutable) vilket är en annan referens än den till xs. Detta innebär att xs och ys inte “pekar” på samma objekt längre.

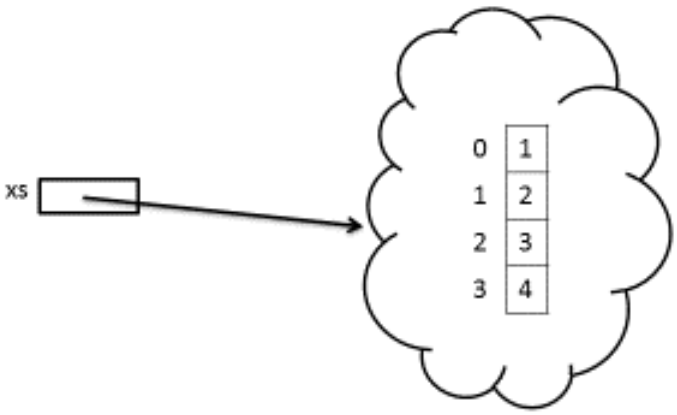
b) Ja därför båda är array och nu kopieras referensen till ys över till zs. Därför kommer alla ändringar i zs att påverka ys (så länge de pekar på samma referens).

c) Nej det gör det inte. Se a).

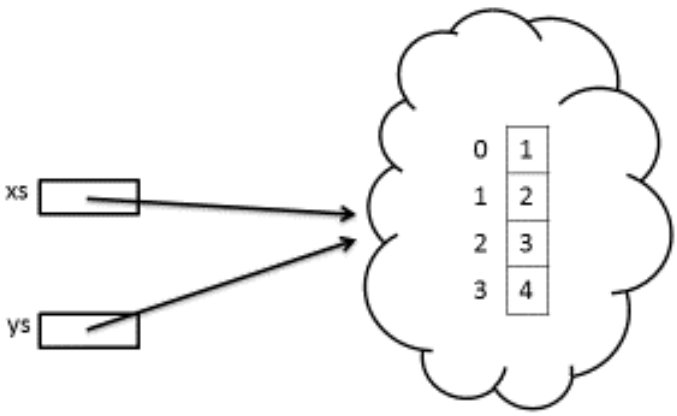
Uppgift 5.

a) Den andra parametern anger hur stor den nya vektorn som returneras ska vara.

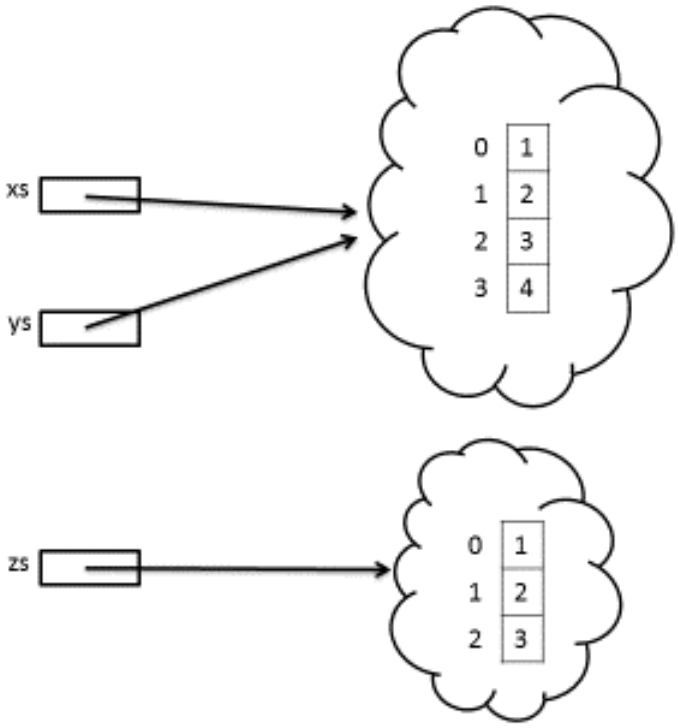
b)



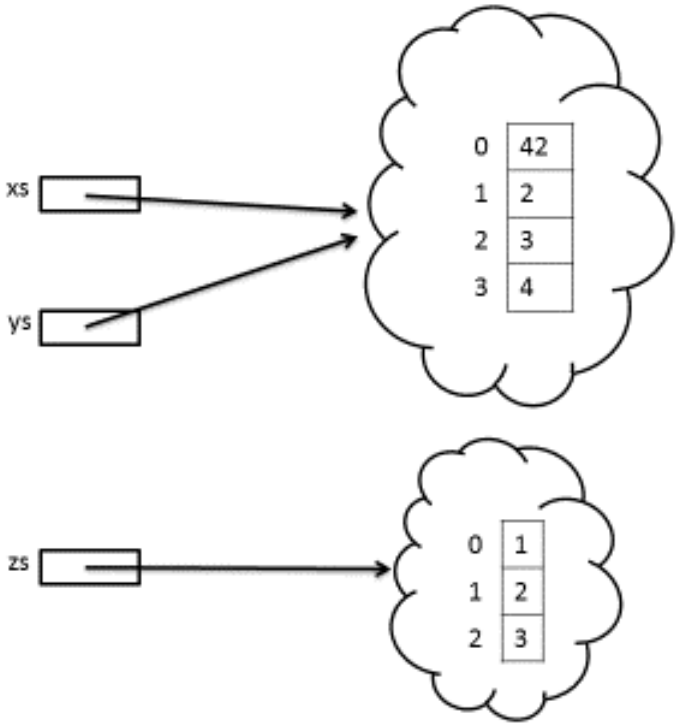
1.



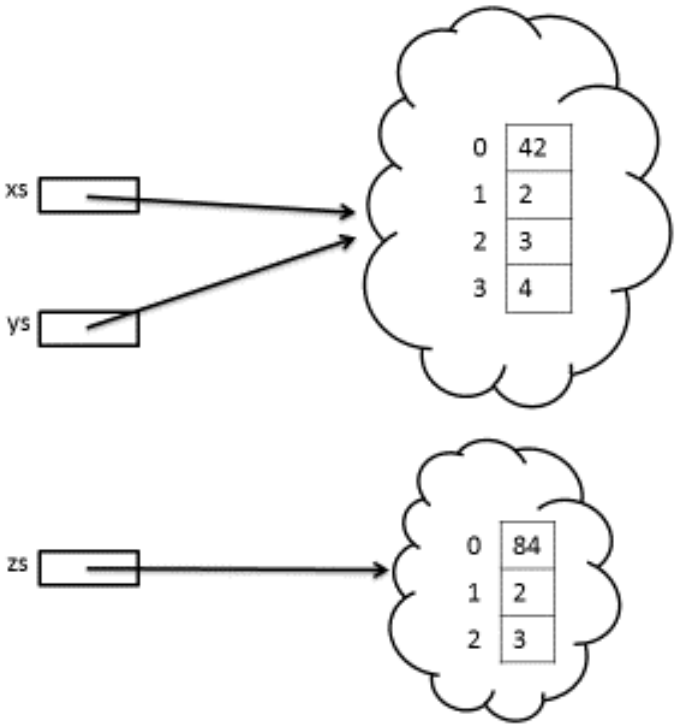
2.



3.



4.



5.

xs = Array(42, 2, 3, 4)
ys = Array(42, 2, 3 ,4)
zs = Array(84, 2, 3)
xs och yx refererar till samma objekt och då deras första IntCell:s värde ändras till 42, så kommer förändringen att ske för båda. zs har en referens till ett annat objekt med ett mindre element. Att zs:s första element ändras, påverkar inte xs och ys.

Uppgift 6.

a)

```
def seqReverseCopy(xs: Array[Int]): Array[Int] = {
  val n = xs.size
  val ys = Array.fill[Int](n)(0)
  var i = 0
  while(i < n) {
    ys(n-i-1) = xs(i)
    i += 1
  }
  ys
}
```

b)

```
def seqReverseCopy(xs: Array[Int]): Array[Int] = {
```

```
  val n = xs.size
  val ys = Array.fill[Int](n)(0)
  for(i <- n-1 to 0 by -1) ys(n-i-1) = xs(i)
  ys
}
```

c) Se b).

Uppgift 7.

a)

```
def reverseString(s: String): String = {
  val sb = new StringBuilder(s)
  val n = sb.length
  for (i <- 0 until n / 2) {
    val temp = sb(i)
    sb(i) = sb(n - i - 1)
    sb(n - i - 1) = temp
  }
  sb.toString
}
```

b)

```
def isPalindrome(s: String): Boolean = {s == reverseString(s)}
```

c)

```
def isPalindrome(s: String): Boolean = {
  val n = s.length
  var foundDiff = false
  var i = 0
  while (i < n/2 && !foundDiff)  {
    foundDiff = s(i) != s(n - i - 1)
    i += 1
  }
  !foundDiff
}
```

Uppgift 8.

a) `xs.filter(_ == 6).size`

b) `xs.filter(_ % 2 == 0).size`

c)

```
scala.collection.immutable.Map[Int,scala.collection.immutable.Vector[Int]] =
  Map(1 -> Vector(5, 3, 1, 1, 3, 5, 1, 1, 3), 0 -> Vector(6, 6, 2, 6))
scala.collection.immutable.Map[Int,scala.collection.immutable.Vector[Int]] =
```

```
  Map(1 -> Vector(5, 3, 1, 1, 3, 5, 1, 1, 3), 0 -> Vector(6, 6, 2, 6))
scala.collection.immutable.Map[Int,scala.collection.immutable.Vector[Int]] =
  Map(2 -> Vector(5, 5, 2), 1 -> Vector(1, 1, 1, 1), 0 -> Vector(3, 6, 3, 6, 3,
(2,Vector(5, 5, 2))
(1,Vector(1, 1, 1, 1))
(0,Vector(3, 6, 3, 6, 3, 6))
freqEvenOdd: scala.collection.immutable.Map[Int,Int] = Map(1 -> 9, 0 -> 4)
nEven: Int = 4
nOdd: Int = 9
```

d) `xs.groupBy(i => i)` skapar en map där nycklarna är alla unika element och värdena är av samma värde som respektive nyckel.

e) `val freq: Map[Int, Int] = xs.groupBy(i => i).map(p => (p._1, p._2.size))`

f)

```
def tärningsRegistrering(xs: Array[Int]): Array[Int] = {
  val f = Array.fill(7)(0)
  f(0) = xs.size
  var i = 0
  while (i < f(0)) {
    f(xs(i)) += 1
    i += 1
  }
  f
}
```

Uppgift 9.

a)

Indata : En sekvens xs av typen `Array[Int]` och pos
Resultat : En ny sekvens av typen `Array[Int]` som är en kopia av xs fast med elementet på plats pos borttaget

1 $n \leftarrow$ antalet element xs
2 $ys \leftarrow$ en ny `Array[Int]` med plats för $n-1$ element
3 **for** $i \leftarrow 0$ **to** $pos-1$ **do**
4 $\quad ys(i) \leftarrow xs(i)$
5 **end**
6 $ys(pos) \leftarrow x$
7 **for** $i \leftarrow pos+1$ **to** $n-1$ **do**
8 $\quad ys(i-1) \leftarrow xs(i)$
9 **end**
10 **return** ys

b)

```
def removeCopy(xs: Array[Int], pos: Int): Array[Int] = {
  val n = xs.size
  val ys = Array.fill(n - 1)(0)
  for (i <- 0 until pos) ys(i) = xs(i)
```

```
  for (i <- pos+1 until n) ys(i - 1) = xs(i)
  ys
}
```

Uppgift 10.

a)

Indata : En sekvens xs av typen `Array[Int]` och pos
Resultat : En uppdaterad sekvens av xs där elementet på plats pos tagits bort och efterföljande element flyttas ett steg mot lägre index med ett sista elementet som är 0

1 $n \leftarrow$ antalet element xs
2 **for** $i \leftarrow pos + 1$ **to** $n - 1$ **do**
3 $\quad xs(i-1) \leftarrow xs(i)$
4 **end**
5 $xs(n-1) \leftarrow 0$

b)

```
def remove(xs: Array[Int], pos: Int): Unit = {
  val n = xs.size
  for (i <- pos+1 until n) xs(i - 1) = xs(i)
  xs(n-1) = 0
}
```

Uppgift 11.

a) Antingen kan du skapa en ny instans av `java.util.Random` genom att skriva: `val r1 = new java.util.Random`. Men om `java.util.Random` importeras kan "java.util" skippas och istället skrivs: `val r2 = new Random`. Som valfritt argument kan ett slumptalsfrö av typen Long skickas med när en ny instans skapas, e.g. `val r3 = new Random(42L)`. `nextInt(x)` skapar ett slumptal från och med 0, upp till x (exklusive x).

b)

```
1  import java.util.Random // Importerar Random
2
3  frö: Long = 42 // Ett slumptalsfrö av värdet 42L skapas.
4
5    // Skapar ett Random objekt med slumptalsfrö "frö".
6  rnd: java.util.Random = java.util.Random@2f410acf
7
8  res0: Int = 7 // Slumpade fram ett tal från 0 till och med 9.
9
10 9 8 8 8 9 7 2 1 4 0 0 3 8 8 4 5 9 1 3 3 5 1 1
11 3 3 3 6 3 4 7 5 7 8 7 6 9 7 0 3 0 6 6 1 0 8 1
12 1 1 0 5 3 5 1 5 3 5 9 9 5 1 8 9 0 6 4 7 5 7 9
13 6 4 0 8 1 0 9 6 6 3 2 7 9 2 7 0 6 9 8 5 0 0 8
14 9 2 7 7 3 5 1 3 // Slumpar och skriver ut 100 tal från 0 till och med 9.
15
```

```
  // Skapar ett Random objekt med slumptalsfrö "frö".
rnd1: java.util.Random = java.util.Random@31e4bb20

  // Skapar ett Random objekt med slumptalsfrö "frö".
rnd2: java.util.Random = java.util.Random@45e37a7e

  // Skapar ett Random objekt med slumptalsfrö med
  // värdet av vad tiden är just nu i nanosekunder.
rnd3: java.util.Random = java.util.Random@57eda880

  // Skapar ett Random objekt med slumptalsfrö
  // med värdet (math.random * Long.MaxValue).toLong.
rnd4: java.util.Random = java.util.Random@79da1ec0

flip: (r: java.util.Random)String // Skapar en funktion som singlar slant.

  // Singlar slant med alla fyra Random
  // objekt 100 gånger samt printar ut resultatet.
xs: scala.collection.immutable.IndexedSeq[(String, String, String, String)] =
Vector((krona,krona,krona,klave), (klave,klave,krona,krona), (krona,krona,klave
(klave,klave,krona,klave), (klave,klave,krona,krona), (krona,krona,klave,krona)
(klave,klave,klave,klave), (krona,krona,klave,krona), (krona,krona,klave,krona)
(klave,klave,krona,klave), (krona,krona,krona,klave), (klave,klave,krona,klave)
(klave,klave,krona,krona), (klave,klave,klave,krona), (klave,klave,klave,krona)
(krona,krona,klave,klave), (klave,klave,klave,klave), (krona,krona,klave,krona)
(krona,krona,klave,klave), (krona,krona,klave,klave), (krona,krona,klave,krona)
(klave,klave,klave,klave), (klave,klave,krona,krona), (klave,klave,klave,klave)
(krona,krona,krona,krona), (krona,krona,krona,klave)...

  // Kollar om det finns något värde som rnd1
  // har genererat men som inte rnd2 genererat.
res1: Boolean = false

  // Kollar om det finns något värde som rnd1
  // har genererat men som inte rnd3 genererat.
res2: Boolean = true
```

c) Vid felsökning och vid simulering där man vill att samma “slumpmässiga” sekvens uppstår varenda gång.

d) Ja.

e) https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html#random%28%29-- säger att den skapar ett nytt java.util.Random-objekt.

f) Den skapar ett slumpmässigt slumptalsfrö. För mer information, se: https://docs.oracle.com/javase/8/docs/api/java/util/Random.html#Random--

Uppgift 12.

```
def testRandom(r: Random, n: Int): Unit = {
  val xs = Array.fill(n)(r.nextInt(6) + 1)
  val f = tärningsRegistrering(xs)
  println("Antal kast: " + f(0))
  for (i <- 1 to 6) println(s"Antal $i:or: " + f(i))
}
```

Uppgift 13.

a) -

b)

```
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
```

Simulerar 10000 tärningskast (med slumptalsfrö 42) och skriver ut förekomsten av respektive tärningskast.

c) Array i scala deklararas: `val scalaArray = Array.ofDim[Int](6)` medan i java skrivs: `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 semicolon skrivas efter varje operation samt att typen måste explicit definieras vid variabeldeklaration. I scala behövs inga semicolon (förutom för att separera operationer på samma rad) och scala bestäms typen implicit, alltså att kompilatorn "gissar" typen av variabeln som deklareras.

d) Lägg till `System.out.println(i);` i for-looparna

e)

```
// 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();
  }
}
```

f)

```
  // 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 runtime error då attributet rnd är
  // private och kan inte nås via REPL:n.
<console>:11: error: value rnd is not a member of object DiceReg2
  DiceReg2.rnd
           ^
```

g)

```
value [diceReg/rnd] is not a member of object DiceReg2
```

h) Om man ska spara under någon data som man inte vill att användaren, eller någon annan, inte ska kunna komma åt. T.ex. om du gör en bankapp vill du inte att nyckeln som du använder för att autorisera en användare ska vara tillgänglig för då kan hackare använda det för att ta sig in på kontot och stjäla pengar!

Uppgift 14.

a) `hasNextInt()` kollar enbart om det finns ett till tal och returnerar **`true`**/**`false`**. `nextInt()` "hoppar" till nästa tal och returnerar det. Se https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html#hasNextInt%28%29 och https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html#nextInt%

28%29.

b) -

c) -

d)

```
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 }
```

Uppgift 15.

a) `ArrayBuffer`.

b) `ArrayBuffer` eller `Array`.

c) `Array`.

d) `Vector`.

5.2 Extrauppgifter

Uppgift 16.

a)

```
def insertCopy(xs: Array[Int], x: Int, pos: Int): Array[Int] = {
  val n = xs.size
  val ys = Array.ofDim[Int](n + 1)
  for (i <- 0 until pos) ys(i) = xs(i)
  ys(pos) = x
  for (i <- pos until n) ys(i + 1) = xs(i)
  ys
}
```

b) `pos` måste vara `0`.

c)

```
1 java.lang.ArrayIndexOutOfBoundsException: -1
```

d) Elementet `x` läggs till på slutet av arrayen, alltså kommer den returnerande arrayen vara större än den som skickades in.

e)

```
1 java.lang.ArrayIndexOutOfBoundsException: 5
```

Man får `ArrayIndexOutOfBoundsException` då indexeringen är utanför storleken hos arrayen.

Uppgift 17.

a)

Indata : En sekvens xs av typen `Array[Int]` och heltalen x och pos
Resultat : En uppdaterad sekvens av xs där elementet x har satts in på platsen pos och efterföljande element flyttas ett steg där sista elementet försvinner

1 $n \leftarrow$ antalet element xs
2 $ys \leftarrow$ en klon av xs
3 $xs(pos) \leftarrow x$
4 **for** $i \leftarrow pos + 1$ **to** $n - 1$ **do**
5 $\quad xs(i) \leftarrow ys(i - 1)$
6 **end**

b)

```
def insert(xs: Array[Int], x: Int, pos: Int): Unit = {
  val n = xs.size
  val ys = xs.clone
  xs(pos) = x
  for (i <- pos + 1 until n) xs(i) = ys(i - 1)
```

```
}
```

Uppgift 18.

```
def tärningsRegistrering(xs: Array[Int]): Array[Int] = {
  val f = Array.fill(7)(0)
  f(0) = xs.size
  for(i <- 0 until f(0)) f(xs(i)) += 1
  f
}
```

Uppgift 19.

5.3 Fördjupningsuppgifter

Uppgift 20.

a)

b)

Uppgift 21.

a)

b)

c)

Uppgift 22.

Uppgift 23.

a)

b)

c)

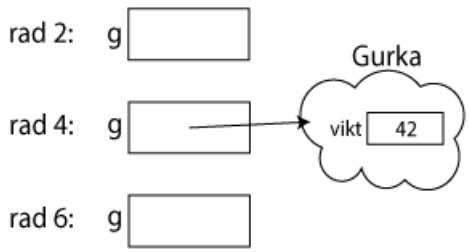
6. Lösningar till övning `classes`

6.1 Grunduppgifter

Uppgift 1.

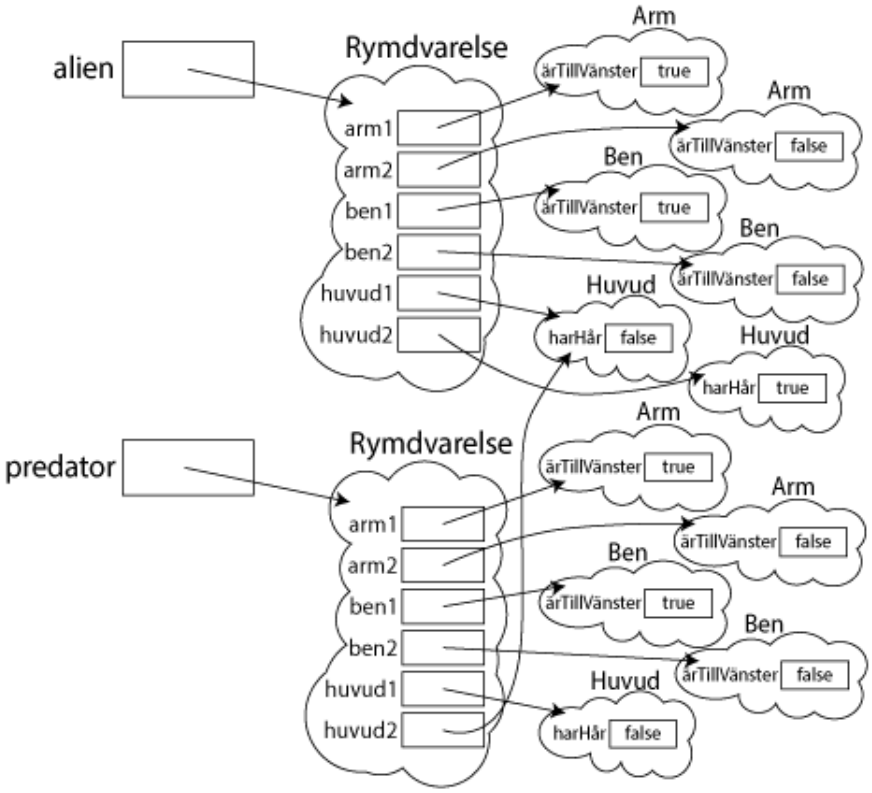
a) Rad 3 och 7 ger båda felmeddelandet "java.lang.NullPointerException". Detta eftersom `g` i båda fallen inte innehåller en referens till en `Gurka` utan pekar på inget – null".

b)



Uppgift 2.

a) Vi skapar två rymdvarelser, `alien` och `predator`, med två ben, två armar samt två huvuden (där det ena är skalligt och det andra har hår) vardera. Efter det är varken `alien` eller `predator` skallig eftersom båda har ett huvud med hår. Sen låter man referensen till `predators` huvud med hår referera till aliens huvud utan hår. Nu är predator helt skallig.



b)

c) Eftersom det inte längre finns någon referens som pekar på det objektet kommer Garbage Collector ta hand om det och kommer förr eller senare skrivas över av något annat som behöver sparas. Nej, det går inte att komma åt.

Uppgift 3. Rad 2:

```
error: value vikt is not a member of Gurka1
```

Detta eftersom om man varken väljer att skriva **`val`** eller **`var`** skapar inte scala någon getter eller setter (metoder för att läsa/ändra en variabel) och därför ser det ut som att vikt inte finns för kompilatorn.

Rad 4: Denna rad skapar inte en error eftersom om man skriver **`val`** innan variabeln skapas en getter automatiskt och man kan därför komma åt `vikt`.

Rad 6:

```
error: value vikt in class Gurka3 cannot be accessed in Gurka3
```

I detta fallet skapas en `getter` men eftersom accessnivån sätts till **`private`** vet kompilatorn att man inte får komma åt variabeln utifrån.

Rad 11:

```
java.lang.NullPointerException
```

Detta eftersom `kompis` är `ingenGurka` som inte pekar på något objekt och när man då försöker komma åt ett attribut från den kommer det inte funka.

Rad 12: Kommer inte generera en error eftersom när man kallar `kompisVikt` (som är `public`) försöker den komma åt `Gurka4(84, null).vikt`. `vikt` är `private val` vilket innebär att det har en getter och eftersom huvudobjektet också är av typen `Gurka4` är accessnivån tillräckligt hög.

Rad 13:

```
error: value vikt is not a member of Gurka5
```

När man sätter ett attribut till `private[this]` tillåts inte ens objekt av samma typ att komma åt variabeln och därför får man en error som säger att den inte finns.

Rad 17:

```
error: constructor Gurka6 in class Gurka6 cannot be accessed in object
```

Eftersom man satt klassparametrarna till `private` kan man inte komma åt konstruktorn och därför får man en error.

Rad 26:

```
error: constructor Gurka7 in class Gurka7 cannot be accessed in object
```

Samma anledning som på rad 17.

Rad 27:

```
java.lang.IllegalArgumentException: requirement failed: negativ vikt: -42
```

Kompanjonsobjektet har en requirement på att `vikt >= 0` vilket innebär att om det inte stämmer kommer man få en error av typen `IllegalArgumentException`.

Rad 30: Anledningen till att man kan sätta vikten till något negativt är att checken om det är negativt endast görs när man skapar `Gurka7` vilket innebär att i efterhand kan man ändra den till vilket värde som helst (av typen `Int`).

Uppgift 4.

a) Rad 16:

```
java.lang.IllegalArgumentException: requirement failed: negativ vikt: -42
```

Kompanjonsobjektet har en requirement på att `vikt >= 0` vilket innebär att om det inte stämmer kommer man få en error.

Rad 20:

```
java.lang.IllegalArgumentException: requirement failed: negativ vikt: -1
```

Eftersom settern har implementerat ett krav på att vikten måste vara större eller lika med 0 får man en error när man försöker sätta den till -1.

Rad 22:

```
java.lang.IllegalArgumentException: requirement failed: negativ vikt: -958
```

Eftersom 42-1000 är mindre än noll får man en error.

b) Man kan sätta egna mer specifika krav på vad som får göras med värdena så man har större koll på att inget oväntat händer.

Uppgift 5.

```
class Square(val x: Int, val y: Int, val side: Int) {
  val area: Int = side*side

  def move(dx: Int, dy: Int): Square = new Square(x + dx, y + dy, side)

  def isEqualSizeAs(that: Square): Boolean = this.side == that.side

  def scale(factor: Double): Square = new Square(x, y, (side*factor).toInt)

  override def toString: String = s"Square(x: $x, y: $y, side: $side)"
}

object Square {
  val unit: Square = new Square(0, 0, 1)

  def apply(x: Int, y: Int, side: Int): Square = new Square(x, y, side)

  def apply(): Square = new Square(0, 0, 1)
}
```

Eftersom `s1`, `s2`, `s3` och `Square.unit` alla har en sida med längden 1 så kommer rad 3-5 returnera **`true`**. Rad 6 kommer returnera **`false`** eftersom `s2.scale(math.Pi)` sida är π och `s2` fortfarande har sidan 1. Rad 7 kommer däremot returnera **`true`** då båda har sidan π.

Uppgift 6.

a) Variablerna `a` och `b` är båda objekt av en vanlig klass vilket kommer innebära att de jämförs med referenslikhet och eftersom de inte är samma objekt retunerar == **`false`**. `c` och `d` är däremot objekt av en case klass så de jämförs med strukturlikhet och eftersom de har samma vikt returnerar == **`true`**.

b) Både `a eq b` och `c eq d` ska returnera **`false`** eftersom de alla är olika objekt och det är referenslikhetsom gäller.

Uppgift 7.

a) se e) för komplett lösning

b) se e) för komplett lösning

c) se e) för komplett lösning

d) se e) för komplett lösning

e)

```
case class Point(x: Int, y: Int) {

  def distanceTo(that: Point): Double = math.hypot(that.x - x, that.y -y)

  def distanceTo(x: Int, y: Int): Double = distanceTo(Point(x, y))

  def move(dx: Int, dy: Int): Point = Point(x + dx, y + dy)
}
```

```
object Point {
  //val origin: Point = new Point(0, 0)
  def origin: Point = Point(0, 0)
}
```

f) == och != kollar strukturlikhet så om två objekt innehåller samma värden kommer == returnera **true** och != **false** och vise versa. eq och ne kollar referenslikhet så om två variabler pekar på samma objekt kommer eq returnera **true** och ne **false** och vise versa.

g) **false**. Detta eftersom om origin implementeras som en metod som returnerar en ny Point varje gång den kallas kommer Point.origin inte peka på samma objekt varje gång metoden kallas (eq är referenslikhet).

h) Sturkturlikhet bryr sig endast om innehållet i objekten och jämför det. Det kvittar alltså om det är samma objekt eller två olika så länge de innehåller samma värden. Referenslikhet kollar endast på om det är samma objekt variablerna pekar på och struntar fullständigt i om de innehåller samma värden.

Uppgift 8.

```
class Square(val p: Point, val side: Int) {
  val area: Int = side*side

  def move(dx: Int, dy: Int): Square = new Square(p.move(dx, dy), side)

  def isEqualSizeAs(that: Square): Boolean = this.side == that.side

  def scale(factor: Double): Square = new Square(p, (side*factor).toInt)

  override def toString: String = s"Square(p: $p, side: $side)"
}

object Square {
  val unit: Square = new Square(new Point(0, 0), 1)

  def apply(x: Int, y: Int, side: Int): Square =
    new Square(new Point(x, y), side)

  def apply(): Square = new Square(new Point(0, 0), 1)
}
```

Uppgift 9.

```
case class Point(p:(Int,Int)) {
  val x: Int = p._1

  val y: Int = p._2

  def distanceTo(that: Point): Double = math.hypot(that.x - x, that.y -y)

  def distanceTo(that: (Int, Int)): Double = distanceTo(Point(that))
```

```
  def move(dx: Int, dy: Int): Point = Point(x + dx, y + dy)
}

object Point {
  val origin: Point = new Point(0, 0)
}
```

Uppgift 10. Inget! Eftersom både `Point(1,2)` och `Point((1,2))` är okej sätt att komma åt den nya klassen så kommer det se likadant utifrån och därför behöver man inte ändra något i `Square`.

Uppgift 11.

a)

```
class Frog private (initX: Int = 0, initY: Int = 0) {
  private var _x: Int = initX
  private var _y: Int = initY
  private var _distanceJumped: Double = 0

  def jump(dx: Int, dy: Int): Unit = {
    _x += dx
    _y += dy
    _distanceJumped += Math.hypot(dx, dy)
  }

  def x: Int = _x
  def y: Int = _y

  def randomJump: Unit = {
    val r = scala.util.Random
    val xtmp = r.nextInt(10)+1
    val ytmp = r.nextInt(10)+1
    _x += xtmp
    _y += ytmp
    _distanceJumped += Math.hypot(xtmp, ytmp)
  }

  def distanceToStart: Double = Math.hypot(_x,_y)
  def distanceJumped: Double = _distanceJumped
  def distanceTo(that: Frog): Double = Math.hypot(_x - that.x, _y - that.y)
}

object Frog {
  def spawn(): Frog = new Frog()
}
```

b)

```
val f1 = Frog.spawn()
//test requirement 1 and 4
assert(f1.x == 0 && f1.y == 0, "Either x or y isn't 0")

f1.jump(4,3)
//test requirement 1 and 5
```

```
assert(f1.x == 4 && f1.y == 3, "Either x isn't 4 or y isn't 3")

f1.jump(4,3)
//test requirement 2
var text = "distanceJumped is " + f1.distanceJumped + ". Should be 10"
assert(f1.distanceJumped == 10, text)

f1.jump(-4,-3)
//test requirement 3
text = "distanceToStart is " + f1.distanceJumped + ". Should be 5"
assert(f1.distanceToStart == 5, text)

var f2 = Frog.spawn()
for (x <- 1 to 1000) {
  f2.randomJump
  //test requirement 5
  text = "Either x or y isn't in [1,10]. x:" + f2.x + ", y: " + f2.y
  assert(f2.x > 0 && f2.x <= 10 && f2.y > 0 && f2.y <= 10, text)
  f2 = Frog.spawn()
}

val f3 = Frog.spawn()
f3.jump(1,1)
val f4 = Frog.spawn()
f4.jump(4,5)
// Test distanceT()
text = "distanceTo is " + f3.distanceTo(f4) + ". Should be 5"
assert(f3.distanceTo(f4) == 5, text)
```

c) Getter

d) Om metoden har parametrar och retur-typen `Unit`. Det betyder troligen att parametrarna ändrar något istället för att skapa något nytt.

e)

```
class Frog private (initX: Int = 0, initY: Int = 0) {
  private var _x: Int = initX
  private var _y: Int = initY
  private var _distanceJumped: Double = 0

  def jump(dx: Int, dy: Int): Unit = {
    _x += dx
    _y += dy
    _distanceJumped += Math.hypot(dx, dy)
  }

  def x: Int = _x
  def y: Int = _y

  def x_= (newX: Int): Unit = {
    _distanceJumped += Math.abs(_x - newX)
    _x = newX
  }
  def y_= (newY: Int): Unit = {
    _distanceJumped += Math.abs(_y - newY)
    _y = newY
```

```
  }

  def randomJump: Unit = {
    val r = scala.util.Random
    val xtmp = r.nextInt(10)+1
    val ytmp = r.nextInt(10)+1
    _x += xtmp
    _y += ytmp
    _distanceJumped += Math.hypot(xtmp, ytmp)
  }

  def distanceToStart: Double = Math.hypot(_x,_y)
  def distanceJumped: Double = _distanceJumped
  def distanceTo(that: Frog): Double = Math.hypot(_x - that.x, _y - that.y)
}

object Frog {
  def spawn(): Frog = new Frog()
}
```

f)

```
var noCollision = true
var counter = 0
val numberOfFrogs = 100
val distanceBetweenFrogs = 8
val frogArray = Array.fill(numberOfFrogs){Frog.spawn()}
(0 until numberOfFrogs).foreach(i => frogArray(i).x(i*distanceBetweenFrogs))
while (noCollision) {
  frogArray.foreach(frog => frog.randomJump)
  for (frog <- frogArray) {
    for (frog2 <- frogArray) {
      if (frog != frog2 && frog.distanceTo(frog2) < 0.5) {
        noCollision = false
      }
    }
  }
  counter += 1
}
print(counter)
```

6.2 Extrauppgifter

Uppgift 12.

```
/** A mutable and expensive Square. */
class Square private (val initX: Int, val initY: Int, val initSide: Int) {

  private var nMoves = 0;
  private var sumCost = 0.0;
  private var _x = initX;
  private var _y = initY;
  private var _side = initSide;

  private def addCost: Unit = {
   sumCost += math.hypot(x - initX, y - initY) * side
  }

  /** The current position on the x axis */
  def x: Int = _x

  /** The current position on the y axis */
  def y: Int = _y

  /** The size of the side */
  def side = _side

  /** Scales the size of this square and rounds it to nearest integer */
  def scale(factor: Double): Unit = { _side = (_side * factor).round.toInt }

  /** Moves this square to position (x + xd, y + dy) */
  def move(dx: Int, dy: Int): Unit = {
    _x += dx; _y += dy;
    nMoves += 1
    addCost
  }

  /** Moves this square to position (x, y) */
  def moveTo(x: Int, y: Int): Unit = {
    _x = x; _y = y;
    nMoves += 1
    addCost
  }

  /** The accumulated cost of this Square */
  def cost: Double = sumCost

  /** Reset the cost of this Square */
  def pay: Unit = {sumCost = 0}

  /** A string representation of this Square */
  override def toString: String =
    s"Square[($x, $y), side: $side, #moves: $nMoves times, cost: $sumCost]"
}
```

```
object Square {
  private var created = Vector[Square]()

  /** Constructs a new Square object at (x, y) with size side */
  def apply(x: Int, y: Int, side: Int): Square = {
    require(side >= 0, s"side must be positive: $side")
    val sq = (new Square(x, y, side))
    created :+= sq
    sq
  }

  /** Constructs a new Square object at (0, 0) with side 1 */
  def apply(): Square = apply(0, 0, 1)

  /** The total number of moves that have been made for all squares. */
  def totalNumberOfMoves: Int = created.map(_.nMoves).sum

  /** The total cost of all squares. */
  def totalCost: Double = created.map(_.cost).sum
}
```

6.3 Fördjupningsuppgifter

TODO!!!

7. Lösningar till övning `traits`

7.1 Grunduppgifter

Uppgift 1.

a) `Vector[Object]`.

b) Det beror på att vektorns element är av typen `Object`. `vikt` är inte definierat för denna typ.

c) -.

d) `Vector[Grönsak]`.

e) Ja.

f) -.

g) `Grönsak`. $anon$1@88dfbe.

Uppgift 2.

a)

```
def skapaDjur: Djur =
   {if(math.random > 0.5) new Ko else new Gris}
```

b)

```
class Häst extends Djur{ def väsnas = println("Gnääääägg") }
def skapaDjur: Djur = {val r = math.random;
   if(r < 0.33) new Ko else if(r < 0.67) new Gris else new Häst}
```

Uppgift 3.

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) För `Point`: `def moveTo(dx: Double, dy: Double): Point = Point(dx, dy)`.
För `Shape`: `def moveTo(dx: Double, dy: Double): Shape`.
För `Rectangle`: `override def moveTo(dx: Double, dy: Double): Rectangle = Rectangle(pos.moveTo(dx, dy), this.dx, this.dy)`.
För `Circle`: `override def moveTo(dx: Double, dy: Double): Circle = Circle(pos.moveTo(dx, dy), radius)`.

c) `def distanceTo(that: Point): Double = math.hypot(that.x - x, that.y - y)`.

d) `def distanceTo(that: Shape): Double = pos.distanceTo(that.pos)`.

Uppgift 4.

a)

```
fyle.filter(f => f.isInstanceOf[Ånka] && f.ärFlygkunnig).size
```

b)

```
val antalKrax: Int = fyle.filter(f => !f.ärSimkunnig).size * 2
val antalKvack: Int = fyle.filter(f => f.ärSimkunnig).size * 4
```

Uppgift 5.

a) Sätt `final` framför `class` i klasserna.

b) error: illegal inheritance from final class Kråga.

Uppgift 6.

a) error: not found: value minHemlis.

b) error: value vårHemlis in class Super$class cannot be accessed in Sub.

c) Ja.

Uppgift 7.

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.

Uppgift 8.

a) B ärver A. C och D ärver B.

b) 1. True eftersom c är av typen C.
2. False eftersom c inte är av typen D.
3. True eftersom d är av typen D som är en subtyp av B.
4. True eftersom c är av typen C som är en subtyp av B, som i sin tur är en subtyp av A.
5. True eftersom b är av typen D, som är en subtyp av B, som i sin tur är en subtyp av A.
6. True eftersom b är av typen D.
7. True eftersom a är av typen C som är en subtyp av B.
8. True eftersom c är av typen C som är en subtyp av AnyRef.
9. True eftersom c är av typen C som är en subtyp av Any.
10. Error eftersom `isInstanceOf` inte kan använda sig av `AnyVal`.

11. True eftersom c är av typen C som är en subtyp av Object (Object är java-representationen av AnyRef).
12. Error eftersom `isInstanceOf` inte kan testa om värdetyper (i detta fallet 42) är referenstyper.
13. True eftersom 42 är av typen `Int` som är en subtyp av Any.

c) 3. Går inte eftersom c inte är av typen D, utan typen C.
6. Går inte eftersom a inte är av typen D, utan typen C.
7. Går inte eftersom typen E inte finns.

Uppgift 9.

a) 2. Måste ha **override** framför `b` för att kunna ändra på metoden.
4. `c` är **private**, vilket betyder att den är gömd för subklasserna. Därför kan den inte överskuggas. Genom att ta bort **override** fungerar klassen.
5. En **final**-medlem måste ha ett bestämt värde. Kan lösas genom att tilldela **final** `a` ett värde eller ta bort **final**.
6. En **final**-medlem kan inte överskuggas, varken med eller utan **override**. Här får konflikterna tas bort.
7. Se 6.
8. Eftersom `c` inte finns i `Super5` kan den inte överskuggas. Genom att ta bort **override** fungerar klassen.
10. Överskuggningen av **val** måste vara oföränderlig (immutable); detta är inte nödvändigtvis **def**. Löses genom att byta ut **def** `a` mot **val** `a` hos `Sub10`.
11. Samma problem som i 10.; **lazy val** kan vara föränderlig. Löses genom att ta bort **lazy**.
12. Samma problem igen! **var** är föränderlig, vilket bryter mot typsäkerheten när man försöker överskugga en **val**. Löses genom att ändra **var** till **val**.
15.**def** `a = 43` och **val** `c = "?"` täcker inte allt som **var** kräver. Det behövs en setter för att kunna uppfylla kraven för överskuggning för en **var**. Dessutom finns det ingen anledning för en **val** att överskuggas; man kan ju ändra på den lite hur man vill!

b) Sub3: a = 43, b = 43 eftersom medlemmen är överskuggad. c hittas inte eftersom den är **private**.
Sub13: a = 43, b = 42, c = still lazy"eftersom medlemmen överskuggas.
SubSub: a = 44 eftersom medlemmen överskuggas, b = 42, c = still lazy".

c) -.

Uppgift 10.

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) error: class Person is abstract; cannot be instantiated.

d) error: overriding value namn in class Person of type String; value namn needs 'override' modifier.
toString för Student: Student(Person3,LTH,D).
toString för Forskare: Student(Person4,LTH,Doktorand).

e)

```
trait Person {val namn: String; val nbr: Int}
trait Akademiker extends Person {val universitet: String}
case class Student(
  namn: String,
  nbr: Int,
  universitet: String,
  program: String) extends Akademiker
case class Forskare(
  namn: String,
  nbr: Int,
  universitet: String,
  titel: String) extends Akademiker
case class IckeAkademiker(
    namn: String,
    nbr: Int) extends Person
```

f) Man måste använda en klass om man behöver klassparametrar. Man måste använda en trait om man vill göra in-mixning med `with`.
Se http://www.artima.com/pins1ed/traits.html#12.7.

Uppgift 11.

a) Sättet är säkrare då man inte kan tilldela korten en färg som inte finns. Med heltalskonstanterna kan man till exempel ge ett kort färgen 5, vilken inte korresponderar till någon riktig färg.

b) `for (f <- Färg.values; v <- 1 to 13) yield Kort(f,v)`

c)

```
def blandadKortlek: Vector[Kort] = {
  val kortlek =
    for (f <- Färg.values; v <- 1 to 13) yield Kort(f,v)
  scala.util.Random.shuffle(kortlek)
}
```

d) `def färgPoäng(xs: Vector[Kort]): Int = xs.map(_.färg.toInt).sum`

7.2 Extrauppgifter

Uppgift 12.

```
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
  with KanFlyga
  with KanInteSimma
  with ÄrStor{
  val läte = "krax"
}

final class Ånka
  extends Fyle
  with KanSimma
  with KanKanskeFlyga
  with ÄrStor{
  val läte = "kvack"
  override def väsnas = { print(läte * 4); räknaLäte += 4 }
}

final class Pjodd
  extends Fyle
  with KanFlyga
  with KanKanskeSimma
  with ÄrLiten{
  val läte = "kvitter"
  override def väsnas = { print(läte * 8); räknaLäte += 8 }
}
```

I REPL:

```
val fyle = Vector.fill(42)(
  if(math.random < 0.33) new Kråga else
  if (math.random < 0.5) new Ånka else
  new 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
```

7.3 Fördjupningsuppgifter

8. Lösningar till övning `matching`

8.1 Grunduppgifter

Uppgift 1.

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.

Uppgift 2.

a) Svaret blir identiskt mot föregående uppgiften i Java.
Scalas **`match`**-uttryck fungerar väldigt likt Javas **`switch`**. Den jämför stegvis värdet med varje **`case`** för att sedan returnera ett värde tillhörande motsvarande **`case`**.

b)

```
1 scala.MatchError (of class java.lang.Character)
```

Exekveringsfel, uppstår av en viss input under körningen.

c) Scalas **`match`** ersätter kolonet (:) i **`switch`** med Scalas högerpil (=>).
`match` returnerar ett värde till skillnad från **`switch`** som inte returnerar något.
`match` kan inte "falla igenom" så ett **`break`** efter varje **`case`** är inte nödvändigt.
Till skillnad från **`switch`**-satsen kastar **`match`** ett `MatchError` om ingen matchning skulle ske.

Uppgift 3.

Garden som införts vid **`case`** `'g'` slumpar fram ett tal mellan 0 och 1 och om talet inte är större än 0.5 så blir det ingen matchning med **`case`** `'g'` och programmet testar vidare tills default-caset.
Gardens krav måste uppfyllas för att det ska matcha som vanligt.

Uppgift 4.

a) G100true. Vid byte av plats: Gtrue100.
`match` testar om kompanjonsobjektet `Gurka` är av typen `Gurka` med två parametervärden. De angivna parametrarna tilldelas namn, `vikt` får namnet `v` och `ärRutten` namnet `rutten` och skrivs sedan ut. Byts namnen dessa ges skrivs de ut i den omvända ordningen.

b) `Option[(Int, Boolean)]`

c) `Some((100, true))`, en `Option` med en tupel av parametrarna från g.

d) `ärÄtvärd` testar om `Grönsak g` är av typen `Gurka(v, rutten)` eller `Tomat`. Dessa har sedan garder.
`Gurka` måste ha `vikt` över 100 och `ärRutten` vara `false` för att `case Gurka` ska returnera `true`.
`Tomat` måste ha `vikt` över 50 och `ärRutten` vara `false` för att `case Tomat` ska returnera `true`.
Matchas inte `Grönsak g` med någon av dessa returneras default-värdet `false`.

Uppgift 5.

a)

```
package vegopoly

trait Grönsak {
  def vikt: Int
  def ärRutten: Boolean
  def ärÄtbar: Boolean
}

case class Gurka(vikt: Int, ärRutten: Boolean) extends
  Grönsak { val ärÄtbar: Boolean = (!ärRutten && vikt > 100)}
case class Tomat(vikt: Int, ärRutten: Boolean) extends
  Grönsak { val ärÄtbar: Boolean = (!ärRutten && vikt > 50)}

object Main{
  def slumpvikt: Int = (math.random*500 + 100).toInt
  def slumprutten: Boolean = math.random > 0.8
  def slumpgurka: Gurka = Gurka(slumpvikt, slumprutten)
  def slumptomat: Tomat = Tomat(slumpvikt, slumprutten)
  def slumpgrönsak: Grönsak = if (math.random > 0.2) slumpgurka
    else slumptomat

  def main(args: Array[String]): Unit = {
    val skörd = Vector.fill(args(0).toInt)(slumpgrönsak)
    val ätvärda = skörd.filter(_.ärÄtbar)
    println("Antal skördade grönsaker: " + skörd.size)
    println("Antal ätvärda grönsaker: " + ätvärda.size)
  }
}
```

b) Följande `case class` läggs till:

```
case class Broccoli(vikt: Int, ärRutten: Boolean)
    extends Grönsak {
  val ärÄtbar: Boolean = (!ärRutten && vikt > 80)
```

```
}
```

Därefter läggs följande till i **object** Main innan **def** slumpgrönsak:

```
def slumpbroccoli: Broccoli = Broccoli(slumpvikt, slumprutten)
```

Slutligen ändras **def** slumpgrönsak till följande:

```
def slumpgrönsak: Grönsak = {    // välj t.ex. denna fördelning:
  val rnd = math.random
  if (rnd > 0.5) slumpgurka      // 50% sannolikhet för gurka
  else if (rnd > 0.2) slumptomat // 30% sannolikhet för tomat
  else slumpbroccoli             // 20% sannolikhet för broccoli
}
```

c) Fördelarna med **match**-versionen, och mönstermatchning i sig, är att det är väldigt lätt att göra ändringar på hur matchningen sker. Detta innebär att det skulle vara väldigt lätt att ändra definitionen för ätbarheten. Skulle dock dessa inte ändras ofta utan snarare grönsaksutbudet så kan det polyformistiska alternativet vara att föredra. Detta eftersom det skulle implementeras och ändras lättare än mönstermatchningen vid byte av grönsaker.

Uppgift 6.

a)

```
def parafärg(f: Färg): Färg = f match {
  case Spader  => Klöver
  case Hjärter => Ruter
  case Ruter   => Hjärter
  case Klöver  => Spader
}
```

b)

```
1 <console>:17: warning: match may not be exhaustive.
2 It would fail on the following input: Ruter
```

Varningen kommer redan vid kompilering.

c)

```
1 scala.MatchError: Ruter (of class Ruter$)
2   at .parafärg(<console>:17)
```

Detta är ett körtidsfel.

d) Om en klass är **sealed** innebär det att om ett element ska matchas och är en subtyp av denna klass så ger Scala varning redan vid kompilering om det finns en risk för ett MatchError, alltså om **match**-uttrycket inte är uttömmande och det finns fall som inte täcks av ett **case**.

En förseglad supertyp innebär att programmeraren redan vid kompileringstid får en varning om ett fall inte täcks och i sånt fall vilket av undertyperna, liksom annan hjälp av kompilatorn. Detta kräver dock att alla subtyperna delar samma fil som den förseglade klassen.

Uppgift 7.

a) Både `str` och `vadsomhelst` matchar med inputen, oavsett vad denna är på grund av att de har en liten begynnelsebokstav.
`str` har dock en gard att strängen måste börja med *g* vilket gör så endast `val g = "gurka"` matchar med denna. `val x = "urka"` plockas dock upp av `vadsomhelst` som är utan gard.

b)

```
<console>:16: warning: patterns after a variable pattern cannot match (SLS 8.1
.1)
```

och

```
<console>:17: warning: unreachable code due to variable patter 'tomat' on line
16
```

Trots att en klass `tomat` existerar så tolkar Scalas `match` den som en `case`-gren som fångar allt på grund av en liten begynnelsebokstav. Detta gör så alla objekt som inte är av typen `Gurka` kommer ge utskriften *tomat* och att sista caset inte kan nås.

c)

```
case `tomat` => println("tomat")
```

Uppgift 8.

a)

1. `var kanske` blir en `Option` som håller `Int` men är utan något värde, kallas då `None`.
2. Eftersom `var kanske` är utan värde är storleken av den 0.
3. `var kanske` tilldelas värdet 42 som förvaras i en `Some` som visar att värde finns.
4. Eftersom `var kanske` nu innehåller ett värde är storleken 1.
5. Eftersom `var kanske` innehåller ett värde är den inte tom.
6. Eftersom `var kanske` innehåller ett värde är den definierad.
7. `def ökaOmFinns` matchar en `Option[Int]` med dess olika fall.
   Finns ett värde, alltså `opt: Option[Int]` är en `Some`, så returneras en `Some` med ursprungliga värdet plus 1.
   Finns inget värde, alltså `opt: Option[Int]` är en `None`, så returneras en `None`.

8. -
9. -
10. -
11. `def` `ökaOmFinns` appliceras på `kanske` och returnerar en `Some` med värdet hos `kanske` plus 1, alltså 43.
12. `def` `öka` tar emot värdet av en `Int` och returnerar värdet av denna plus 1.
13. `map` applicerar `def` `öka` till det enda elementen i `kanske`, 42. Denna funktion returnerar en `Some` med värdet 43 som tilldelas `merKanske`.

b)

1. `val` `meningen` blir en `Some` med värdet 42.
2. `val` `ejMeningen` blir en `Option[Int]` utan något värde, en `None`.
3. `map(_ + 1)` appliceras på `meningen` och ökar det existerande värdet med 1 till 43.
4. `map(_ + 1)` appliceras på `ejMening` men eftersom inget värde existerar fortsätter denna vara `None`.
5. `map(_ + 1)` appliceras ännu en gång på `ejMening` men denna gång inkluderas metoden `orElse`. Om ett värde inte existerar hos en `Option`, alltså är av typen `None`, så utförs koden i `orElse`-metoden som i detta fall skriver ut *saknas* för värdet som saknas.
6. Samma anrop från föregående rad utförs denna gång på `meningen` och eftersom ett värde finns utförs endast första biten som ökar detta värde med 1.

Denna metod kan användas i stället för `match`-versionen i föregående exempel i och med dennas simplare form. En `Option` innehåller ju antingen ett värde eller inte så ett längre `match`-uttryck är inte nödvändigt.

c)

1. En vektor `xs` skapas med var femte tal från 42 till 82.
2. En tom `Int`-vektor `e` skapas.
3. `headOption` tar ut första värdet av vektorn `xs` och returnerar den sparad i en `Option`, `Some(42)`.
4. Första värdet i vektorn `xs` sparas i en `Option` och hämtas sedan av `get`-metoden, 42.

5. Som i föregående rad men denna gång används `getOrElse` som om den `Option` som returneras saknar ett värde, alltså är av typen `None`, returnerar 0 istället.
   Eftersom `xs` har minst ett värde så är den `Option` som returneras inte `None` och ger samma värde som i föregående, 42.

6. Som föregående rad fast istället för att returnera 0 om värde saknas så returneras en `Option[Int]` med 0 som värde.

7. `headOption` försöker ta ut första värdet av vektorn `e` men eftersom denna saknar värden returneras en `None`.

8. ```
   java.util.NoSuchElementException: None.get
   ```

   Liksom föregående rad returnerar `headOption` på den tomma vektorn `e` en `None`. När `get`-metoden försöker hämta ett värde från en `None` som saknar värde ger detta upphov till ett körtidsfel.

9. Liksom i föregående returneras `None` av `headOption` men eftersom `getOrElse`-metoden används på denna `None` returneras 0 istället.

10. Liksom föregående används `getOrElse`-metoden på den `None` som returneras. Denna gång returneras dock en `Option[Int]` som håller värdet 0.

11. En vektor innehållandes elementen `xs`-vektorn och 3 `e`-vektorer skapas.

12. `map` använder metoden `lastOption` på varje delvektor från vektorn på föregående rad. Detta sammanställer de sista elementen från varje delvektor i en ny vektor. Eftersom vektor `e` är tom returneras `None` som element från denna.

13. Samma sker som i föregående rad men `flatten`-metoden appliceras på slutgiltiga vektorn som rensar vektorn på `None` och lämnar endast faktiska värden.

14. `lift`-metoden hämtar det eventuella värdet på plats 0 i `xs` och returnerar den i en `Option` som blir `Some(42)`.

15. `lift`-metoden försöker hämta elementet på plats 1000 i `xs`, eftersom detta inte existerar returneras `None`.

16. Samma sker som i föregående fast applicerat på vektorn `e`. Sedan appliceras `getOrElse(0)` som, eftersom `lift`-metoden returnerar `None`, i sin tur returnerar 0.

17. `find`-metoden anropas på `xs`-vektorn. Den letar upp första talet över 50 och returnerar detta värde i en `Option[Int]`, alltså `Some(52)`.
   ```

18. `find`-metoden anropas på `xs`-vektorn. Den letar upp första värdet under 42 men eftersom inget värde existerar under 42 i `xs` returneras `None` istället.

19. `find`-metoden anropas på `e`-vektorn och skriver ut *HITTAT!* om ett element under 42 hittas. Eftersom `e`-vektorn är tom returneras `None` vilket `foreach` inte räknar som element och därav inte utförs på.

d) Användning av -1 som returvärde vid fel eller avsaknad på värde kan ge upphov till körtidsfel som är svåra att upptäcka. **`null`** kan i sin tur orsaka kraschar om det skulle bli fel under körningen. `Option` har inte samma problem som dessa, används ett `getOrElse`-uttryck eller dylikt så kraschar inte heller programmet.
Dessutom behöver inte en funktion som returnerar en `Option` samma dokumentation av returvärdena. Istället för att skriva kommentarer till koden på vilka värden som kan returneras och vad dessa betyder så syns det direkt i koden.
Slutgiltligen är `Option` mer typsäkert än **`null`**. När du returnerar en `Option` så specificeras typen av det värde som den kommer innehålla, om den innehåller något, vilket underlättar att förstå och begränsar vad den kan returnera.

Uppgift 9.

a)

1. Ett `Exception` kastas med felmeddelandet *PANG!*.

2. Flera olika typer av `Exception` visas.

3. En typ av `Exception`, `IllegalArgumentException`, kastas med felmeddelandet *fel fel fel*.

4. Ett stycke kod testas med **`try`**. Ett `Exception` med felmeddelandet *stormvind!* kastas som fångas av **`catch`**-uttrycket. Den matchar felmeddelandet såsom ett **`match`**-uttryck och det godtyckliga fallet `e` skriver ut det `Exception` som fångats och returnerar -1.

b) Exempelvis:
`OutOfMemoryError`, om programmet får slut på minne.
`IndexOutOfBoundsException`, om en vektorposition som är större än vad som finns hos vektorn försöker nås.
`NullPointerException`, om en metod eller dylikt försöker användas hos ett objekt som inte finns och därav är en nullreferens.

c) Eftersom värdet som skulle vara av typen `Int` känner **`try`**-funktionen igen returtypen hos **`case`** `e` och `carola` blir av typen `Int`. Skulle **`catch`**-grenen returnera en sträng istället vet programmet inte vilken typ denna är av och `carola` blir av typen `Any`.

Uppgift 10.

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.

c) Den mest markanta skillnaden mellan språken är att Scala varken kräver att ett undantag fångas av en **`catch`** eller att ett undantag behöver deklareras innan det kastas med en `@throws`. Dessutom saknar **`catch`**-metoden hos Java de **`match`**-egenskaper Scala har. Inte heller returnerar **`catch`** hos Java något värde vilket gör det nödvändigt att definiera variabler för detta innan. I övrigt är semantiken och syntaxen väldigt lika mellan båda språken. De använder samma struktur och samma ord, dessutom har de en hel del `Exception` gemensamt.

Uppgift 11.

a)

1. **`def`** `pang` skapas som kastar ett `Exception` med felmeddelandet *PANG!*.
2. Scalas verktyg `Try`, `Success` och `Failure` importeras.
3. **`def`** `pang` anropas i `Try` som fångar undantaget och kapslar in den i en `Failure`.
4. Metoden `recover` matchar undantaget i `Failure` från föregående rad med ett **`case`** och gör om föredetta `Failure` till `Success` vid matchning, liknande **`catch`**.

5. Strängen *tyst* körs i föregående test men eftersom inget undantag kastas blir den inkapslad i en `Success` och `recover` behöver inte göra något. Den tar endast hand om undantag.

6. `def kanskePang` skapas som har lika stor chans att returnera strängen *tyst* såsom anropa `def pang`.

7. `def kanskeOk` skapas som testar `def kanskePang` med `Try`.

8. En vektor `xs` fylls med resultaten, `Success` och `Failure`, från 100 körningar av `kanskeOk`.

9. Elementet på plats 13 i vektor `xs` matchas med något av 2 `case`. Om det är en `Success` skrivs *:)* ut, om en `Failure` skrivs *:(* plus felmeddelandet ut.

10. -

11. -

12. -

13. Metoden `isSuccess` testar om elementet på plats 13 i `xs` är en `Success` och returnerar `true` om så är fallet.

14. Metoden `isFailure` testar om elementet på plats 13 i `xs` är en `Failure` och returnerar `true` om så är fallet.

15. Metoden `count` räknar med hjälp av `isFailure` hur många av elementen i `xs` som är `Failure` och returnerar detta tal.

16. Metoden `find` letar upp med hjälp av `isFailure` ett element i `xs` som är `Failure` och returnerar denna i en `Option`.

17. `badOpt` tilldelas den första `Failure` som hittas i `xs`.

18. `goodOpt` tilldelas den första `Success` som hittas i `xs`.

19. Resultatet badOpt skrivs ut, `Option[scala.util.Try[String]] = Some(Failure(java.lang.Exception: PANG!))`

20. Metoden `get` hämtar från `badOpt` den `Failure` som förvaras i en `Option`.

21. Metoden `get` anropas ännu en gång på resultatet från föregående rad, alltså en `Failure`, som hämtar undantaget från denna och som då i sin tur kastas.

22. Metoden `getOrElse` anropas på den `Failure` som finns i `badOpt`. Eftersom detta är en `Exception` utförs `orElse`-biten istället för att undantaget försöker hämtas. Då returneras strängen *bomben desarmerad!*.

23. Metoden `getOrElse` anropas på den `Success` som finns i `goodOpt`. Eftersom detta är en `Success` med en normal sträng sparad i sig returneras denna sträng, *tyst*.

24. Metoden från föregående används denna gång på alla element i `xs` där resultatet skrivs ut för varje.

25. Metoden `toOption` appliceras på alla `Success` och `Failure` i `xs`. De med ett exception, alltså `Failure`, blir en `None` medan de med värden i `Success` ger en `Some` med strängen *tyst* i sig.

26. Metoden `flatten` appliceras på vektorn fylld med `Option` från föregående rad för att ta bort alla `None`-element.

27. Metoden `size` används på slutgiltiga listan från föregående rad för att räkna ut hur många `Some` som resultatet innehåller. Den har alltså beräknat antalet element i `xs` som var av typen `Success` med hjälp av `Option`-typen.

b) `pang` har returtypen `Nothing`, en specialtyp inom Scala som inte är kopplad till `Any`, och som inte går att returnera.

c) Typen `Nothing` är en subtyp av varenda typ i Scalas hierarki. Detta innebär att den även är en subtyp av `String` vilket implicerar att `String` inkluderar både strängar och `Nothing` och därav blir returtypen.

Uppgift 12.

a)

1. En klass `Gurka` skapas med parametrarna `vikt` av typen `Int` och ärÄtbar av typen `Boolean`.

2. `g1` tilldelas en instans av `Gurka`-klassen med `vikt = 42` och `ärÄtbar = true`.

3. `g2` tilldelas samma `Gurka`-objekt som g1.

4. `g3` tilldelas en ny instans av `Gurka`-klassen med motsvarande parametrar som g1.

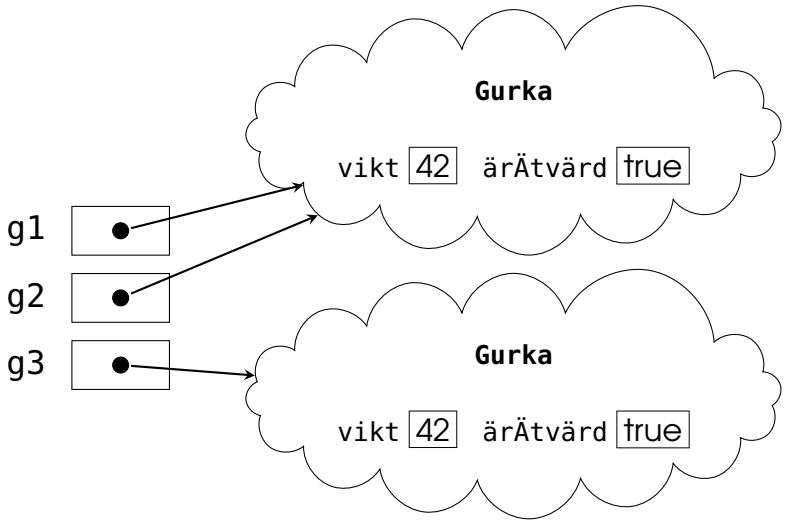
5. ==(`equals`)-metoden jämför g1 med g2 och returnerar `true`.

6. ==(`equals`)-metoden jämför g1 med g3 och returnerar `false`.

7. `def equals(x\$1: Any): Boolean`

Som kan ses ovan är elementet som jämförs i `equals` av typen `Any`. Eftersom programmet inte känner till klassen så används `Any.equals` vid jämförelsen. Till skillnad från de primitiva datatyperna som vid jämförelse med `equals` jämför innehållslikhet, så jämförs referenslikheten hos klasser om inget annat är specificerat. `g1` och `g2` refererar till samma objekt medan `g3` pekar på ett eget sådant vilket innebär att `g1` och `g3` inte har referenslikhet.

b)



c) -

d) I de första 3 raderna sker samma som i deluppgift *a*. När nu dessa jämförelser görs mellan `Gurka`-objekten så överskuggas `Any.equals` av den `equals` som är specificerad för just `Gurka`. Eftersom båda objekten `g1` jämförs med också är av typen `Gurka` så matchar den med `case that: Gurka`. Denna i sin tur jämför vikterna hos de båda gurkorna och returnerar en `Boolean` huruvida de är lika eller inte, vilket de i båda fallen är.

e) I deluppgift a gav `g1 == g3` **`false`** trots innehållslikhet. Efter skuggningen ger dock detta uttryck **`true`** vilket påvisar jämförelse av innehållslikhet.

8.2 Extrauppgifter

Uppgift 13.

a)

b)

Uppgift 14.

Uppgift 15.

8.3 Fördjupningsuppgifter

Uppgift 16.

Uppgift 17.

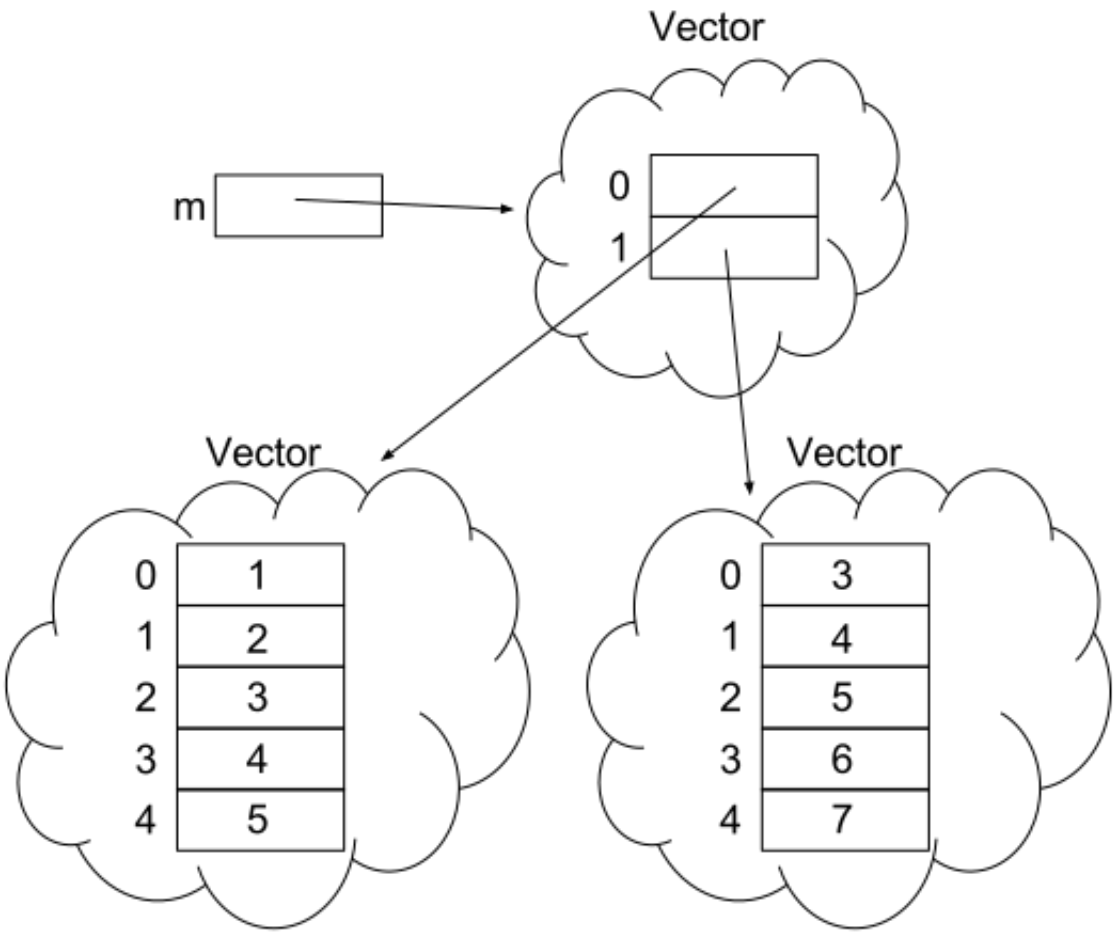
Uppgift 18.

Uppgift 19.

9. Lösningar till övning `matrices`

9.1 Grunduppgifter

Uppgift 1.



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)
2: `Int`
3: `Vector[Int]`
4: `Int`

c)
m2: `Vector[Vector[Int]]`
m3: `Vector[Vector[AnyVal]]`
m4: `Vector[Vector[Any]]`

[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

m5: `Vector[Vector[Int]]`

d) TODO

e) m5, 42×2

Uppgift 2.

a)

```
def throwDie: Int = (math.random * 6).toInt + 1
```

b) 1000×5

c) –

d)

```
def roll(n: Int) = Vector.fill(n)(throwDie).sorted
```

e)

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

f)

```
def isYatzy(xs: Vector[Int]): Boolean = {
  var foundDiff = false
  var i = 0
  while (i < xs.size && !foundDiff) {
    foundDiff = xs(i) != xs(0)
    i += 1
  }
  !foundDiff
}
```

g)

```
def diceMatrix(m: Int, n: Int): Vector[Vector[Int]] =
  Vector.fill(m)(roll(n))
```

h)

```
def diceMatrixToString(xss: Vector[Vector[Int]]): String =
  xss.map(_.mkString(" ")).mkString("\n")
```

i) 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 utvärderas `0 until xss.size` till en tom `Range` eftersom `xss.size` blir `0` och `until` är exkluderande. Innehållet i 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.
Med `StringBuilder` behöver inte hela innehållet kopieras vid varje tillägg, vilket spar prestanda vid många tillägg, men eftersom det är ett föränderligt objekt kan innehållet ändras av någon annan del av programmet som också har tillgång till referensen; objektet kan helt plötsligt innehålla någonting annat, trots att referensen är densamma.

j)

```
def filterYatzy(xss: Vector[Vector[Int]]): Vector[Vector[Int]] =
  xss.filter(isYatzy)
```

k)

```
def filterYatzy(xss: Vector[Vector[Int]]): Vector[Vector[Int]] = {
  var result: Vector[Vector[Int]] = Vector()
  for (i <- 0 until xss.size) {
    if (isYatzy(xss(i))) result = result :+ xss(i)
  }
  result
}
```

l) –

m)

```
def yatzyPips(xss: Vector[Vector[Int]]): Vector[Int] =
  xss.filter(isYatzy).map(_.head)
```

Uppgift 3.

a)

```
case class Table(
  data: Vector[Vector[String]],
  headings: Vector[String],
  sep: String){

  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 lazy val toString: String =
    headings.mkString(sep) + "\n" +data.map(_.mkString(sep)).mkString("\n")
```

```
}
object Table {
  def fromFile(fileName: String, separator: Char = ';'): Table = {
    val lines = scala.io.Source.fromFile(fileName).getLines.toVector
    val matrix= lines.map(_.split(separator).toVector)
    new Table(matrix.tail, matrix.head, separator.toString)
  }
}
```

b)

```
object RegTable {
  def main( args:Array[String]): Unit = {
    val t = Table.fromFile(args(0), args(1)(1))
    val counts: Vector[Vector[String]] =
      (0 until t.dim._2)
        .map(i => t.values(t.headings(i))
        .map(x => x + ": " + t.col(i).count(_ == x)))
        .toVector

    for (i <- 0 until t.dim._2) {
      println(s"\nColumn: ${i + 1}, ${t.headings(i)}:")
      for (j <- 0 until counts(i).length) {
        println(counts(i)(j))
      }
    }
  }
}
```

Uppgift 4.

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
```

Uppgift 5.

a) –

b)

```
class Cell[T](var value: T){
  override def toString = "Cell(" + value + ")"
  def concat[U](that: Cell[U]): Cell[String] =
    new Cell(value.toString + that.value.toString)
}
```

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.

d) –

Uppgift 6.

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 auktuella 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 ArrayMatrix {

  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("ArrayMatrix test");
    int[][] xss = new int[10][5];
    showMatrix(xss);
    fillRnd(xss, 6);
    showMatrix(xss);
  }
}
```

9.2 Extrauppgifter

Uppgift 7.

a)

```
/** En skiss på en klass som kan användas till ett förenklat yatzy-spel */
case class YatzyRows(val rows: Vector[Vector[Int]]) {

  private def throwDie: Int = (math.random * 6).toInt + 1

  /** A new YatzyRows with a new row of 5 dice rolls appended to rows */
```

```
  def roll: YatzyRows = new YatzyRows(rows :+ Vector.fill(5)(throwDie))

  /** A new YatzyRow with some indices of the last row re-rolled */
  def reroll(indices: Vector[Int]): YatzyRows =
    new YatzyRows(rows :+ rows(rows.length - 1).zipWithIndex.map {
      case (x, i) => if (indices.contains(i)) throwDie else x
    })
}
object YatzyRows {

  def isYatzy(xs: Vector[Int]): Boolean = xs.forall(_ == xs(0))

  def isThreeOfAKind(xs: Vector[Int]): Boolean =
    xs.exists(x => xs.count(_ == x) >= 3)

  def isFourOfAKind(xs: Vector[Int]): Boolean =
    xs.exists(x => xs.count(_ == x) >= 4)

  def isFullHouse(xs: Vector[Int]): Boolean =
    xs.exists(x => xs.count(_ == x) == 3) &&
    xs.exists(x => xs.count(_ == x) == 2)

  def isSmallStraight(xs: Vector[Int]): Boolean =
    xs.forall(x => xs.count(_ == x) == 1) && !xs.exists(_ == 6)

  def isLargeStraight(xs: Vector[Int]): Boolean =
    xs.forall(x => xs.count(_ == x) == 1) && !xs.exists(_ == 1)
}
```

Observera att fem stycken 2:or uppfyller kraven för Yatzy, men även för triss och fyrtal.

b) Slumpen gör att utfallet inte kommer stämma exakt överens med teorin, men för ett stort antal kast bör resultaten hamna ganska nära. De teoretiska sannolikheterna (utan omkast) finns i 1.

Tabell 1: Sannolikhet för olika Yatzy-resultat

Yatzy	0,077%
≥ 3 av samma	21%
≥ 4 av samma	2,0%
Kåk	3,9%
Liten stege	1,5%
Stor stege	1,5%

Kodexempel:

```
import YatzyRows._

object YatzyStats extends App {
  val n = 1000000.0
  var yr = YatzyRows(Vector(Vector[Int]()))
  for (i <- 1 to n.toInt) yr = yr.roll
  println(s"Yatzy: ${yr.rows.count(isYatzy(_)) / n * 100}%")
```

```
  println(s"Three of a kind: ${yr.rows.count(isThreeOfAKind(_)) / n * 100}%")
  println(s"Four of a kind: ${yr.rows.count(isFourOfAKind(_)) / n * 100}%")
  println(s"Full house: ${yr.rows.count(isFullHouse(_)) / n * 100}%")
  println(s"Small straight: ${yr.rows.count(isSmallStraight(_)) / n * 100}%")
  println(s"Large straight: ${yr.rows.count(isLargeStraight(_)) / n * 100}%")
}
```

c) –

9.3 Fördjupningsuppgifter

Uppgift 8.

a) –

Uppgift 9.

a) –

b) –

c) –

d) –

e) –

f) –

Uppgift 10.

a) –

10. Lösningar till övning `sorting`

10.1 Grunduppgifter

Uppgift 1.

a)

```
true
true
true
true
true
false
```

b) *s1* kommer först.

Uppgift 2. 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
```

Uppgift 3.

a)

1. Returnerar en sorterad `Vector` av `double`-värden
2. Skapar en variabel xs och sparar en `Array` med `Int`-värden mellan 100000 till 1.
3. Sorterar `xs = 1,2,3...`
4. Konverterar xs till en `Array` av `String`-värden och sorterar dem lexikografiskt: `xs = "1", "10", "100"` etc.
5. Konverterar xs till en `Array` av `Byte`-värden (max 127, min -128) och sorterar dem, samt tar bort dubletter: `xs = -128, -127, -1...`
6. Skapar en ny klass `Person` som tar 2 `String`-argument i konstruktorn
7. Sparar en Vector med två `Person`-objekt i en variabel ps
8. Försöker kalla på `sorted`-metoden för klassen `Person`. Eftersom vi skrivit denna klass själva och inte berättat för Scala hur `Person`-objekt ska sorteras, resulterar detta i ett felmeddelande.

b)

1. —
2. —
3. Sorterar `Person`-objekten i ps med avseende på värdet i `firstName`
4. Sorterar `Person`-objekten i ps med avseende på värdet i `familyName`
5. `sortBy` tar en funktion f som argument. f ska ta ett argument av typen `Person` och returnera en generisk typ B.
6. Sortera `Person`-objekten i ps med avseende på `firstName` i sjunkande ordning (omvänt från tidigare alltså)
7. `sortWith` tar en funktion lt som argument. lt ska i sin tur ta två argument av typen `Person` och returnera ett boolskt värde.

8. Sorterar en vektor så att värdena som är minst delbara med 2 hamnar först, och de mest delbara med 2 hamnar sist. Detta delar alltså upp udda och jämna tal.

c) Klassens signatur blir då:

```
case class Person(firstName: String, lastName: String, age: Int)
```

Lägg in dem i en vektor:

```
val ps2 = Vector(Person("a", "asson", 34), Person("asson", "assonson", 1234),
Person("anna", "Book", 2))
```

Sortera dem på olika sätt:

1. Vektorn blir sorterad med avseende på personernas ålder i stigande ordning

```
scala> ps2.sortWith((p1, p2) => p1.age < p2.age)
res40: scala.collection.immutable.Vector[Person] = Vector(Person(anna,Boo
Person(a,asson,34), Person(asson,assonson,1234))
```

2. Sorterar vektorn med avseende på namn, men också med avseende på ålder (i sjunkande ordning). För att komma före någon i ordningen måste alltså både namnet komma före, och åldern vara högre.

```
scala> ps2.sortWith((p1, p2) => (p1.firstName < p2.firstName) &&
(p1.age > p2.age))
res42: scala.collection.immutable.Vector[Person] = Vector(Person(a,asson,
Person(asson,assonson,1234), Person(anna,Book,2))
```

Uppgift 4.

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).

Uppgift 5.

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:

$$log_2(n) = b$$

Uppgift 6.

a) Den finns som värde för en *td* tagg, på följande vis: `<td class="mitt">2</td>`.

b) Koden laddar ner html-koden för sidan
http://kurser.lth.se/lot/?lasar=16_17&soek_text=&sort=kod&val=kurs&soek=t
och sparar den i en vektor. Sedan filtreras ut endast de rader som innehåller strängen ”kurskod” så att all onödig HTML-kod försvinner. Sedan konverteras detta, för varje rad, till `Course`-objekt med hjälp av metoden `fromHtml`. Eftersom variabeln `lth2016` är deklarerad som **`lazy`** kommer inte `download()` bli anropad förrän vi vill komma åt variabeln. Vi startar alltså processen genom att referera variabeln `lth2016` i objektet `courses`:

```
courses.lth2016
```

Detta genererar en lång lista med `Course`-objekt. Antalet kurser är således lika med storleken på vektorn `lth2016`.

```
courses.lth2016.size
res38: Int = 1097
```

c)

```
1 scala> def isCS(s: String) = s.startsWith("EDA") || s.startsWith("ETS")
2 scala> val x = courses.lth2016.find(c => isCS(c.code) && c.level == "G2").get
3 x: courses.Course = Course(EDA031,C++ - programmering,C++ Programming,7.5,G2)
```

Obs: metoden `find` returnerar ett objekt av typen `Option`. För att få värdet som är lagrat i detta objekt krävs det att man kallar på `get`.

d)

```
def linearSearch[T](xs: Seq[T])(p: T => Boolean): Int = {
   var i = 0
   while(i < xs.size && !p(xs(i))) i += 1
   if (i < xs.size) 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
   val special = "ACFGLMNP0123456789"
   def digLetter = special(rnd(special.size - 1))
   Seq(letter, letter, letter, digLetter, dig, dig).mkString
}
```

f)

```
val lthCourses = courses.lth2016 //avoid including download time
val xs = Vector.fill(500000)(rndCode)
val(ixs, elapsedLin) = timed{
xs.map(x => linearSearch(lthCourses)(_.code == x))}
val found = ixs.filterNot(_== -1).size
```

g)

```
def linearSearch[T](xs: Seq[T])(p: T => Boolean): Int =
  xs.indexWhere(p)
```

Uppgift 7.

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) {
    mid = (low + high) / 2
    if (xs(mid) == key) found = true
    else if (xs(mid) < key) low = mid + 1
    else high = mid - 1
  }
  if (found)
    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`

d) Binärsökningen var ca 23 gånger snabbare.

Uppgift 8.

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;
}
```

Uppgift 9.

a) —

b)

```
def insertionSort(xs: Seq[Int]): Seq[Int] = {
  val result = scala.collection.mutable.ArrayBuffer.empty[Int]
  for (e <- xs) {
    var pos = 0
    while (pos < result.size && result(pos) < e) pos += 1
    result.insert(pos,e)
  }
```

```
  result.toVector
}
```

Uppgift 10.

```
def selectionSortInPlace(xs: Array[String]): Unit = {

  def indexOfMin(startFrom: Int): Int = {
    var minPos = startFrom
    var i = startFrom + 1
    while (i < xs.size) {
      if (xs(i) < xs(minPos)) minPos = i
      i += 1
    }
    minPos
  }

  def swapIndex(i1: Int, i2: Int): Unit = {
    val temp = xs(i1)
    xs(i1) = xs(i2)
    xs(i2) = temp
  }

  for (i <- 0 to xs.size - 1) swapIndex(i, indexOfMin(i))
}
```

10.2 Extrauppgifter

Uppgift 11.

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 $n + n * log(n)$.

b)

En mer effektiv version av `isSorted` som stoppar direkt när den upptäcker att ett element inte är sorterat.

```
def isSorted(xs: Vector[Int]): Boolean = {

  if(xs.length > 1){
    for(i <- 0 until xs.length-1 if xs(i) > xs(i+1)){
      return false
    }
  }
  true
```

```
}
```

c)

2-tupeln är av typen `(Int, Int)`.

```
def isSorted(xs: Vector[Int]): Boolean =
  xs.zip(xs.tail).forall(x => x._1 <= x._2)
```

Uppgift 12.

a)

```
def insertionSort(xs: Array[Int]): Unit = {

  for(elem <- 1 until xs.length if xs.length > 0){
    var pos = elem
    while(pos > 0 && xs(pos) < xs(pos - 1)){
      val temp = xs(pos -1)
      xs(pos -1) = xs(pos)
      xs(pos) = temp
      pos -= 1
    }
  }
}
```

b)

```
public static void insertionSort(int[] xs) {

    if (xs.length < 1)
        return;

    for (int i = 1; i < xs.length; i++) {
        int pos = i;

        for (; pos > 0 && xs[pos] < xs[pos - 1]; pos--) {
            int temp = xs[pos - 1];
            xs[pos - 1] = xs[pos];
            xs[pos] = temp;
        }
    }
}
```

Uppgift 13.

```
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) {
    val minPos = indexOfMin(unsorted)
    val elem = unsorted.remove(minPos)
    result.append(elem)
  }

  result.toVector
}
```

10.3 Fördjupningsuppgifter

Uppgift 14.

a) —

b) —

c) Tänk på att det fortfarande måste returneras en Int.

d) Undersök i Javas API hur metoden `compareTo` är implementerad för strängar.

Uppgift 15.

—

Uppgift 16.

—

Uppgift 17.

Tänk på att för att sortering i omvänd ordning (alltså högst rank först) ska fungera så måste jämförelsen returnera **`false`**.

```
case class  Team(name: String, rank: Int) extends Ordered[Team]{
  override def compare(that: Team): Int = -rank.compare(that.rank)
}
```

Uppgift 18.

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
}
```

Uppgift 19.

Uppgift 20.

Uppgift 21.

11. Lösningar till övning `scalajava`

11.1 Grunduppgifter

Uppgift 1.

a)

```
1  import java.net.URL;
2  import java.util.ArrayList;
3  import java.util.{Set => JSet};
4  import java.util.{HashSet => JHashSet};
5  import java.util.Scanner;
6
7  object Hangman {  // This is Java-like, non-idiomatic Scala code!
8      private var hangman: Array[String] = Array[String](
9          " ======  ",
10         " |/   |  ",
11         " |    O  ",
12         " |   -|- ",
13         " |   / \\ ",
14         " |       ",
15         " |       ",
16         " ==========================   RIP  :(");
17
18     private def renderHangman(n: Int): String = {
19         var result: StringBuilder = new StringBuilder();
20         for (i: Int <- 0 until n){
21             result.append(hangman(i));
22             if (i < n - 1) {
23                 result.append("\n");
24             }
25         }
26         return result.toString();
27     }
28
29     private def hideSecret(secret: String,
30                            found: JSet[Character]): String = {
31         var result: String = "";
32         for (i: Int <- 0 until secret.length()) {
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 def foundAll(secret: String,
43                          found: JSet[Character]): Boolean = {
44         var foundMissing: Boolean = false;
45         var i: Int = 0;
46         while (i < secret.length() && !foundMissing) {
47             foundMissing = !found.contains(secret.charAt(i));
```

```
48              i += 1;
49          }
50          return !foundMissing;
51      }
52
53      private def makeGuess(): Char = {
54          var scan: Scanner = new Scanner(System.in);
55          var guess: String = "";
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      def download(address: String, coding: String): String = {
64          var result: String = "lackalänga";
65          try {
66              var url: URL = new URL(address);
67              var words: ArrayList[String] = new ArrayList[String]();
68              var scan: Scanner = new Scanner(url.openStream(), coding);
69              while (scan.hasNext()) {
70                  words.add(scan.next());
71              }
72              var rnd: Int = (Math.random() * words.size()).asInstanceOf[Int];
73              result = words.get(rnd);
74          } catch { case e: Exception =>
75              System.out.println("Error: " + e);
76          }
77          return result;
78      }
79
80      def play(secret: String): Unit = {
81          var found: JSet[Character] = new JHashSet[Character]();
82          var bad: Int = 0;
83          var won: Boolean = false;
84          while (bad < hangman.length && !won){
85              System.out.println(renderHangman(bad));
86              System.out.print("Felgissningar: " + bad + "\t");
87              System.out.println(hideSecret(secret, found));
88              var guess: Char = makeGuess();
89              if (secret.indexOf(guess) >= 0) {
90                  found.add(guess);
91              } else {
92                bad += 1;
93              }
94              won = foundAll(secret, found);
95          }
96          if (won) {
97              System.out.println("BRA! :)");
98          } else {
99              System.out.println("Hängd! :(");
100         }
101         System.out.println("Rätt svar: " + secret);
102         System.out.println("Antal felgissningar: " + bad);
103     }
```

```
104
105     def main(args: Array[String] ): Unit = {
106         if (args.length == 0) {
107             var runeberg: String =
108                 "http://runeberg.org/words/ord.ortsnamn.posten";
109             play(download(runeberg, "ISO-8859-1"));
110         } else {
111             var rnd: Int = (Math.random() * args.length).asInstanceOf[Int];
112             play(args(rnd));
113         }
114     }
115 }
```

b)

```
1  object hangman {
2    val hangman = Vector(
3      " ======  ",
4      " |/   |  ",
5      " |    O  ",
6      " |   -|- ",
7      " |   / \\",
8      " |       ",
9      " |       ",
10     " ========================   RIP  :(")
11
12   def renderHangman(n: Int): String = hangman.take(n).mkString("\n")
13
14   def hideSecret(secret: String, found: Set[Char]): String =
15       secret.map(ch => if (found(ch)) ch else '_')
16
17   def makeGuess(): Char = {
18     val guess = scala.io.StdIn.readLine("Gissa ett tecken: ")
19     if (guess.length == 1) guess.toLowerCase.charAt(0)
20     else makeGuess()
21   }
22
23   def download(address: String, coding: String): Option[String] =
24     scala.util.Try {
25       import scala.io.Source.fromURL
26       val words = fromURL(address, coding).getLines.toVector
27       val rnd = (math.random * words.size).toInt
28       words(rnd)
29     }.recover{ case e: Exception =>
30       println(s"Error: $e")
31       "lackalänga"
32     }.toOption
33
34   def play(secret: String): Unit = {
35     def loop(found: Set[Char], bad: Int): (Int, Boolean) =
36       if (secret forall found) (bad, true)
37       else if (bad >= hangman.length) (bad, false)
38       else {
```

```
        println(renderHangman(bad) + s"\nFelgissningar: $bad\t")
        println(hideSecret(secret, found))
        val guess = makeGuess()
        if (secret contains guess) loop(found + guess, bad)
        else loop(found, bad + 1)
      }

    val (badGuesses, won) = loop(Set(),0)
    val msg = if (won) "BRA! :)" else "Hängd! :("
    println(s"$msg\nRätt svar: $secret")
    println(s"Antal felgissningar: $badGuesses")
  }

  def main(args: Array[String] ): Unit = {
    if (args.length == 0) {
      val runeberg = "http://runeberg.org/words/ord.ortsnamn.posten"
      download(runeberg, "ISO-8859-1").foreach(play)
    } else play(args((math.random * args.length).toInt))
  }
}
```

Uppgift 2.

a)

```
import java.util.List;
import java.util.ArrayList;

public class JPoint {
    private int x, y;

    public JPoint(int x, int y){
      this.x = x;
      this.y = y;
    }

    public JPoint(int x, int y, boolean save){
        this(x, y);
        if (save) {
            saved.add(0, this);
        }
    }

    public JPoint(){
        this(0, 0);
    }

    public int getX(){
        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 + ")";
37      }
38
39      private static List<JPoint> saved = new ArrayList<JPoint>();
40
41      public static Double distanceBetween(JPoint p1, JPoint p2) {
42          return Math.hypot(p1.x - p2.x, p1.y - p2.y);
43      }
44
45      public static void showSaved() {
46          System.out.print("Saved: ");
47          for (int i = 0; i < saved.size(); i++){
48            System.out.print(saved.get(i));
49            if (i < saved.size() - 1) {
50              System.out.print(", ");
51            }
52          }
53          System.out.println();
54      }
55  }
```

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))
```

Uppgift 3.

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 `Ingeger`.

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)`.

Uppgift 4.

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) -

11.2 Extrauppgifter

Uppgift 5.

```
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
17   import scala.io.StdIn.readLine
18   import scala.util.{Try,Success,Failure}
19
20   def loop: Unit =
21     Try { readLine("Heltal annars pang: ").toInt } match {
```

```
22        case Failure(e) => show(e); show("PANG!")
23        case Success(i) => showInt(i); loop
24      }
25
26    def main(args: Array[String]): Unit =
27      if(args.length > 0) args.foreach(i => showInt(i.toInt))
28      else loop
29  }
```

Uppgift 6.

11.3 Fördjupningsuppgifter

Uppgift 7.

Uppgift 8.

Uppgift 9.

12. Lösningar till övning `threads`

12.1 Grunduppgifter

Uppgift 1.

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.

Uppgift 2.

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 med den muterbara variablen hade det inte varit perfekt. Problemet ligger i när en tråd laddar och innan den kan lagra det förändrade värdet laddar den andra tråden samma värde. Bara en av dessa trådar vinner racet och får lagra sitt ändrade tal. `skutt` och `bamse` blir alltså upprörda för att inte alla dess uttag och insättningar registreras.

Uppgift 3.

Nu är `farmor` garanterad att kunna ladda saldot, ta ut pengar/ändra och lagra innan `vargen` kan överskriva resultatet. I `slösaSpara` pausas tråden i en millisekund så `vargen` kan fortfarande ta ut pengar innan `farmor` hinner sätta in pengar igen. Dock kommer alla uttag och insättningar registreras eftersom operationerna är atomära.

Uppgift 4.

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 dom 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.

Uppgift 5.

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 proccessen är klar och det slutgilltiga resultatet visas.

e) Samma som ovan, förutom att det blir en vektor där varje element är i sig en vektor med strängar.

f) Laddar in datan parallelt så nedladdingen sker samtidigt, men det går olika snabbt pga metoden seg.

g) Eftersom datan laddas i parallella trådar utan blockering blir dom 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 väldefinerad då division med 0 ger `java.lang.ArithmeticException`. Detta fångas upp vid callbacken och det skrivs ut `:(` samt det specifierade undantaget.

12.2 Extrauppgifter

Uppgift 6.

```
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) {
      if (n % i == 0 || n % (i + 2) == 0) false
      i += 6
    }
    true
}

import scala.concurrent._
```

```
import ExecutionContext.Implicits.global

val primes = Vector.fill(10)(Future{nextPrime(randomBigInt(16))})
primes.foreach(_.onSuccess{case i => println(i)})
```

Uppgift 7.

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.

Uppgift 8.

-

12.3 Fördjupningsuppgifter

Uppgift 9.

a) `abbasillen` skrivs ut baklänges till `nellisabba`.

b)

c)

d)

e)

f)

g)

h)

i)

Lösningsförslag:

```
package fibserver

import java.net.{ServerSocket, Socket}
import java.io.OutputStream
import java.util.Scanner
import scala.util.{Try, Success, Failure}
import scala.concurrent._
import ExecutionContext.Implicits.global

object html {
  def page(body: String): String =  //minimal web page
    s"""<!DOCTYPE html>
       |<html><head><meta charset="UTF-8"><title>Min Sörver</title></head>
       |<body>
       |$body
       |</body>
       |</html>
       """.stripMargin

  def header(length: Int): String = //standardized header of reply to client
    s"HTTP/1.0 200 OK\nContent-length: $length\nContent-type: text/html\n\n"

  def insertBreak(s: String, n: Int = 80): String =
    if (s.size < n) s else s.take(n) + "</br>" + insertBreak(s.drop(n),n)
}

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 = superFib(n)
      memcache.put(n, f)
      f
    }

  private def superFib(n: BigInt): BigInt = {
    if (n <= 0) 0
    else if (n == 1 || n == 2) 1
    else {
      var secondLast: BigInt = 1
      var last: BigInt = 1
      var sum: BigInt = secondLast + last
      var i = 3
      while (i < n) {
        if (memcache.containsKey(i)) {
          sum = memcache.get(i)
        } else {
          secondLast = last
```

```
          last = sum
          sum = secondLast + last
          memcache.put(i, sum)
        }
        i += 1
      }
      sum
    }
  }
}


object start {

  def fibResponse(num: String) = Try { num.toInt } match {
    case Success(n) =>
      val result = html.insertBreak(compute.fib(n).toString)
      html.page(s"<p>fib($n) == " + result + "</p>")
    case Failure(e) => html.page(s"FEL $e: skriv heltal, inte $num")
  }

  def errorResponse(uri:String) = html.page("FATTAR NOLL: " + uri)

  def handleRequest(cmd: String, uri: String, socket: Socket): Unit = {
    val os = socket.getOutputStream
    val parts = uri.split('/').drop(1) // skip initial slash
    val response: String = (parts.head, parts.tail) match {
      case (head, Array(num)) => fibResponse(num)
      case _                  => errorResponse(uri)
    }
    os.write(html.header(response.size).getBytes("UTF-8"))
    os.write(response.getBytes("UTF-8"))
    os.close
    socket.close
  }

  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" }
    }
  }

  def main(args: Array[String]) {
    val port = Try{ args(0).toInt }.getOrElse(8089)
    serverLoop(new ServerSocket(port))
  }
}
```

Uppgift 10.

—

Uppgift 11.

—

Uppgift 12.

—

Uppgift 13.

a)

b)

c)

Uppgift 14.

a) —

b) —

c) —

d) —