

Introduktion till programmering med Scala och Java

Grundkurs



EDAA45, Lp1-2, HT 2016
Datavetenskap, LTH
Lunds Universitet

Kompileringsdatum: 27 oktober 2016
<http://cs.lth.se/pgk>

Editor: Björn Regnell

Contributors in alphabetical order: Anders Buhl, Anna Axelsson, Anna Palmqvist Sjövall, Anton Andersson, Björn Regnell, Casper Schreiter, Cecilia Lindskog, Emelie Engström, Emil Wihlander, Erik Bjäreholt, Erik Gramp, Fredrik Danebjer, Gustav Cedersjö, Henrik Olsson, Jakob Hök, Johan Ravnborg, Jonas Danebjer, Måns Magnusson, Maj Stenmark, Oscar Sigurdsson, Oskar Berg, Oskar Widmark, Patrik Persson, Per Holm, Sandra Nilsson, Sebastian Hegardt, Simon Persson, Stefan Jonsson, Tom Postema, Valthor Halldorsson, Viktor Claesson,

Home: <https://cs.lth.se/pgk>

Repo: <https://github.com/lunduniversity/introprog>

This compendium is on-going work.

Contributions are welcome!

Contact: bjorn.regnell@cs.lth.se

Cover art: Björn Regnell (inspired by Poul Ströyer's illustration of Lennart Hellsing's lyrics to the childrens song "Herr Gurka" with music by Knut Brodin)

You can use this work if you respect this *LICENCE*: CC BY-SA 4.0

<http://creativecommons.org/licenses/by-sa/4.0/>

Please do *not* distribute your solutions to lab assignments and projects.

Copyright © 2015-2016.

Dept. of Computer Science, LTH, Lund University. Lund. Sweden.

Framstegsprotokoll

Genomförda övningar

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

Övning	Grund	Extra	Fördjupning
expressions	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
programs	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
functions	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
data	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
sequences	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
classes	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
traits	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
matching	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
matrices	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
sorting	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
scalajava	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
threads	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Godkända obligatoriska moment

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

Namn:

Namnteckning:

Lab	Datum gk	Handledares namnteckning
kojo
blockmole
pirates
shuffle
turtlegraphics
turtlerace-team
chords-team
maze
survey
lthopoly-team

Projektuppgift (välj en)

Om egendef., ge kort beskrivning:

- life
- bank
- imageprocessing
- tictactoe
- egendefinerad

Förord

Programmering är inte bara ett sätt att ta makten över de människoskapade system som är förutsättningen för vårt moderna samhälle. Programmering är också ett kraftfullt verktyg för tanken. Med kunskap i programmeringens grunder kan du påbörja den livslånga läranderesa som det innebär att vara systemutvecklare och abstraktionskonstnär. Programmeringsspråk och utvecklingsverktyg kommer och går, men de grundläggande koncepten bakom *all* mjukvara består: sekvens, alternativ, repetition och abstraktion.

Detta kompendium utgör kursmaterial för en grundkurs i programmering, som syftar till att ge en solid bas för ingenjörsstudenter och andra som vill utveckla system med mjukvara. Materialet omfattar en termins studier på kvartsfart och förutsätter kunskaper motsvarande gymnasienivå i svenska, matematik och engelska.

Kompendiet är framtaget för och av studenter och lärare, och distribueras som öppen källkod. Det får användas fritt så länge erkännande ges och eventuella ändringar publiceras under samma licens som ursprungsmaterialet. På kurshemsidan cs.lth.se/pgk och i kursrepoet github.com/lunduniversity/introprog finns instruktioner om hur du kan bidra till kursmaterialet.

Läromaterialet fokuserar på lärande genom praktiskt programmeringsarbete och innehåller övningar och laborationer som är organiserade i moduler. Varje modul har ett tema och en teoridel som bearbetas på föreläsningar.

I kursen använder vi språken Scala och Java för att illustrera grunderna i imperativ och objektorienterad programmering, tillsammans med elementär funktionsprogrammering. Mer avancerad objektorientering och funktionsprogrammering lämnas till efterföljande fördjupningskurser.

Den kanske viktigaste framgångsfaktorn vid studier i programmering är att du bejakar din egen upptäckarglädje och experimentlusta. Det fantastiska med programmering är att dina egna intellektuella konstruktioner faktiskt gör något som just *du* har bestämt! Ta vara på det och prova dig fram genom att koda egna idéer – det är kul när det funkar, men minst lika lärorikt är felsökning, buggrättande och alla misslyckade försök som, ibland efter hårt arbete vänds till lyckade lösningar och/eller bestående lärdomar.

Välkommen i datavetenskapens fascinerande värld och hjärtligt lycka till med dina studier!

Lund, 27 oktober 2016, Björn Regnell

Innehåll

Framstegsprotokoll	iii
Förord	v
I Om kursen	1
-1 Kursens arkitektur	3
0 Anvisningar	11
0.1 Samarbetsgrupper	11
0.1.1 Samarbetskontrakt	12
0.1.2 Grupplaborationer	13
0.1.3 Samarbetsbonus	13
0.2 Föreläsningar	14
0.3 Övningar	14
0.4 Resurstider	16
0.5 Laborationer	17
0.6 Kontrollskrivning	18
0.7 Projektuppgift	19
0.8 Tentamen	20
II Moduler	21
1 Introduktion	23
1.1 Övning: expressions	30
1.1.1 Grunduppgifter	30
1.1.2 Extrauppgifter	39
1.1.3 Fördjupningsuppgifter	40
1.2 Laboration: kojo	42
1.2.1 Obligatoriska uppgifter	42
1.2.2 Frivilliga extrauppgifter	48
2 Kodstrukturer	53
2.1 Övning: programs	54
2.1.1 Grunduppgifter	54
2.1.2 Extrauppgifter	64

2.1.3 Fördjupningsuppgifter	65
3 Funktioner, objekt	67
3.1 Övning: functions	68
3.1.1 Grunduppgifter	68
3.1.2 Extrauppgifter	77
3.1.3 Fördjupningsuppgifter	77
3.2 Laboration: blockmole	79
3.2.1 Obligatoriska uppgifter	79
3.2.2 Frivilliga extrauppgifter	85
4 Datastrukturer	87
4.1 Övning: data	89
4.1.1 Grunduppgifter	89
4.1.2 Extrauppgifter	103
4.1.3 Fördjupningsuppgifter	104
4.2 Laboration: pirates	108
4.2.1 Bakgrund	108
4.2.2 Obligatoriska uppgifter	109
4.2.3 Frivilliga extrauppgifter	114
5 Sekvensalgoritmer	115
5.1 Övning: sequences	119
5.1.1 Grunduppgifter	119
5.1.2 Extrauppgifter	129
5.1.3 Fördjupningsuppgifter	130
5.2 Laboration: shuffle	133
5.2.1 Bakgrund	133
5.2.2 Obligatoriska uppgifter	135
5.2.3 Frivilliga extrauppgifter	135
5.2.4 Bilder med exempel på olika pokerhänder	136
6 Klasser	139
6.1 Övning: classes	144
6.1.1 Grunduppgifter	144
6.1.2 Extrauppgifter	151
6.1.3 Fördjupningsuppgifter	153
6.2 Laboration: turtlegraphics	154
6.2.1 Bakgrund	154
6.2.2 Obligatoriska uppgifter	156
6.2.3 Frivilliga extrauppgifter	160
7 Arv	163
7.1 Övning: traits	166
7.1.1 Grunduppgifter	166
7.1.2 Extrauppgifter	175
7.1.3 Fördjupningsuppgifter	176

7.2 Grupplaboration: <i>turtlerace</i> -team	179
7.2.1 Bakgrund	179
7.2.2 Obligatoriska uppgifter	179
8 Repetition, specialundervisning	183
9 Mönster, undantag	185
9.1 Övning: <i>matching</i>	187
9.1.1 Grunduppgifter	187
9.1.2 Extrauppgifter	197
9.1.3 Fördjupningsuppgifter	200
9.2 Grupplaboration: <i>chords</i> -team	203
9.2.1 Bakgrund	203
9.2.2 Obligatoriska uppgifter	204
9.2.3 Extrauppgifter	207
10 Matriser, typparametrar	209
10.1 Övning: <i>matrices</i>	210
10.1.1 Grunduppgifter	210
10.1.2 Extrauppgifter	218
10.1.3 Fördjupningsuppgifter	219
10.2 Laboration: <i>maze</i>	222
10.2.1 Bakgrund	222
10.2.2 Obligatoriska uppgifter	225
10.2.3 Frivillig extrauppgift	227
11 Sökning, sortering	231
11.1 Övning: <i>sorting</i>	232
11.1.1 Grunduppgifter	232
11.1.2 Extrauppgifter	241
11.1.3 Fördjupningsuppgifter	242
11.2 Laboration: <i>survey</i>	249
11.2.1 Bakgrund	249
11.2.2 Given kod	250
11.2.3 Obligatoriska uppgifter	251
11.2.4 Frivilliga extrauppgifter	255
12 Scala och Java	257
12.1 Övning: <i>scalajava</i>	258
12.1.1 Grunduppgifter	258
12.1.2 Extrauppgifter	268
12.1.3 Fördjupningsuppgifter	270
12.2 Grupplaboration: <i>lthopoly</i> -team	271
12.2.1 Bakgrund	271
12.2.2 Kodstruktur	273
12.2.3 Obligatoriska uppgifter	279
12.2.4 Frivilliga extrauppgifter	282

13 Extra: design, api, trådar, webb	283
13.1 Övning: threads	284
13.1.1 Grunduppgifter	284
13.1.2 Extrauppgifter	289
13.1.3 Fördjupningsuppgifter	290
13.2 Projektuppgift: life	295
13.2.1 Bakgrund	295
13.2.2 Reglerna i Life	295
13.2.3 Beskrivning av Workspace	296
13.2.4 Obligatoriska uppgifter	296
13.2.5 Frivilliga extrauppgifter	299
13.2.6 Extra läsning	301
13.3 Projektuppgift: bank	303
13.3.1 Fokus	303
13.3.2 Bakgrund	303
13.3.3 Krav	303
13.3.4 Design	304
13.3.5 Tips	307
13.3.6 Obligatoriska uppgifter	307
13.3.7 Frivilliga extrauppgifter	309
13.3.8 Exempel på körning av programmet	309
13.4 Projektuppgift: tictactoe	316
13.4.1 Bakgrund	316
13.4.2 Regler	316
13.4.3 Teori	316
13.4.4 Design	317
13.4.5 Obligatoriska uppgifter	318
13.5 Projektuppgift: imageprocessing	322
13.5.1 Bakgrund	322
13.5.2 Uppgiften	322
13.5.3 Frivilliga extrauppgifter	329
14 Tentaträning	331
III Appendix	333
A Terminalfönster	335
A.1 Vad är ett terminalfönster?	335
A.2 Vad är en path/sökväg?	337
A.3 Några viktiga terminalkommando	338
B Editera	339
B.1 Vad är en editor?	339
B.2 Välj editor	340

C Kompilera och exekvera	343
C.1 Vad är en kompilator?	343
C.2 Java JDK	344
C.2.1 Kontrollera om du har JDK installerat	344
C.2.2 Installera JDK	345
C.3 Scala	346
C.3.1 Installera Scala	346
C.3.2 Scala Read-Evaluate-Print-Loop (REPL)	347
D Integrerad utvecklingsmiljö	351
D.1 Vad är en integrerad utvecklingsmiljö?	351
D.2 Kojo	352
D.2.1 Installera Kojo	352
D.2.2 Använda Kojo	353
D.3 Eclipse och ScalaIDE	356
D.3.1 Installera Eclipse Mars och ScalaIDE	356
D.3.2 Anpassa Eclipse och ScalaIDE	358
D.3.3 Använda Eclipse och ScalaIDE	359
D.4 IntelliJ IDEA	366
D.4.1 Installera IntelliJ med Scala-plugin	366
D.4.2 Anpassa IntelliJ	366
D.4.3 Använda IntelliJ	367
E Fixa buggar	377
E.1 Vad är en bugg?	377
E.1.1 Olika sorters fel	378
E.2 Att förebygga fel	382
E.3 Vad är debugging?	384
E.3.1 Hur hitta felorsaken?	384
E.4 Åtgärda fel	385
E.5 Använda en debugger	386
E.5.1 Debuggern i Eclipse med ScalaIDE	387
E.5.2 Debuggern i IntelliJ IDEA med Scala-plugin	387
F Dokumentation	389
F.1 Vad gör ett dokumentationsverktyg?	390
F.2 scaladoc	390
F.2.1 Använda dokumentation från scaladoc	390
F.2.2 Skriva dokumentationskommentarer för scaladoc	392
F.2.3 Generera dokumentation med scaladoc	393
F.2.4 Lära mer om scaladoc	393
F.3 javadoc	396
F.3.1 Använda dokumentation genererad med javadoc	396
F.3.2 Skriva dokumentationskommentarer för javadoc	396
F.3.3 Generera dokumentationskommentarer för javadoc	398

G Byggverktyg	399
G.1 Vad gör ett byggverktyg?	399
G.2 Byggverktyget sbt	401
G.2.1 Installera sbt	401
G.2.2 Använda sbt	401
H Versionshantering och kodlagring	405
H.1 Vad är versionshantering?	405
H.2 Versionshanteringsverktyget Git	406
H.2.1 Installera git	407
H.2.2 Anpassa Git	407
H.2.3 Använda git	407
H.3 Kodlagringsplatser på nätet	409
I Virtuell maskin	411
I.1 Vad är en virtuell maskin?	411
I.2 Vad innehåller kursens vm?	412
I.3 Installera kursens vm	412
J Hur bidra till kursmaterialet?	415
J.1 Bidrag är varmt välkomna!	415
J.2 Instruktioner	415
J.2.1 Vad behövs för att kunna bidra?	415
J.2.2 Svenska eller engelska?	415
J.3 Exempel	416

Del I

Om kursen

Kapitel - 1

Kursens arkitektur

Veckoöversikt

W	Modul	Övn	Lab
W01	Introduktion	expressions	kojo
W02	Kodstrukturer	programs	–
W03	Funktioner, objekt	functions	blockmole
W04	Datastrukturer	data	pirates
W05	Sekvensalgoritmer	sequences	shuffle
W06	Klasser	classes	turtlegraphics
W07	Arv	traits	turtlerace-team
KS	KONTROLLSKRIVN.	–	–
W08	Repetition, specialundervisning	Repetera	Kom-i-kapp
W09	Mönster, undantag	matching	chords-team
W10	Matriser, typparametrar	matrices	maze
W11	Sökning, sortering	sorting	survey
W12	Scala och Java	scalajava	lthopoly-team
W13	Extra: design, api, trådar, webb	threads	Projekt
W14	Tentaträning	Extenta	–
T	TENTAMEN	–	–

Kursen består av en **modul** per läsvecka med två **föreläsningar**, en **övning** och en **laboration** (undantaget W02, W13 & W14 som saknar labb och/eller övning). Föreläsningarna ger en översikt av den teori som ingår i varje modul. Genom att göra övningarna bearbetar du teorin och förebereder dig inför laborationerna. När du klarat övningen och laborationen i en modul är du redo att gå vidare till nästa. Tabellen på nästa uppslag visar begrepp som ingår i varje modul.

Kursen är uppdelad i två läsperioder. Efter första läsperioden gör du en diagnostisk **kontrollskrivning** som kontrollerar ditt kunskapsläge. Andra läsperioden avslutas med ett större **projekt** och en skriftlig **tentamen**.

W01	Intro- duktion	sekvens, alternativ, repetition, abstraktion, programmeringsspråk, programmeringsparadigmer, editera-kompilera-exekvera, datorns delar, virtuell maskin, REPL, literal, värde, uttryck, identifierare, variabel, typ, tilldelning, namn, val, var, def, inbyggda grundtyper, Int, Long, Short, Double, Float, Byte, Char, String, println, typen Unit, enhetsvärdet (), stränginterpolatorn s, if, else, true, false, MinValue, MaxValue, aritmetik, slumptal, math.random, logiska uttryck, de Morgans lagar, while-sats, for-sats
W02	Kod- strukturer	iterering, for-uttryck, map, foreach, Range, Array, Vector, algoritm vs implementation, pseudokod, algoritm: SWAP, algoritm: SUM, algoritm: MIN/MAX, algoritm: MININDEX, block, namnsynlighet, namnöverskuggning, lokala variabler, paket, import, filstruktur, jar, dokumentation, programlayout, JDK, main i Java vs Scala, java.lang.System.out.println
W03	Funktioner, objekt	definera funktion, anropa funktion, parameter, returtyp, värdeandrop, namnanrop, default-argument, namngivna argument, applicera funktion på alla element i en samling, procedur, värdeanrop vs namnanrop, uppdelad parameterlista, skapa egen kontrollstruktur, objekt, modul, punktnotation, tillstånd, metod, medlem, funktionsvärde, funktionstyp, äkta funktion, stegad funktion, apply, lazy val, lokala funktioner, anonyma funktioner, lambda, aktiveringspost, anropsstacken, objektheopen, rekursion cslib.window.SimpleWindow
W04	Data- strukturer	attribut (fält), medlem, metod, tupel, klass, Any, instanceof, toString, case-klass, samling, scala.collection, föränderlighet vs oföränderlighet, List, Vector, Set, Map, typparameter, generisk samling som parameter, översikt samlingsmetoder, översikt strängmetoder, läsa/skriva textfiler, Source.fromFile, java.nio.file
W05	Sekvens- algoritmer	sekvensalgoritm, algoritm: SEQ-COPY, in-place vs copy, algoritm: SEQ-REVERSE, algoritm: SEQ-REGISTER, sekvenser i Java vs Scala, for-sats i Java, java.util.Scanner, scala.collection.mutable.ArrayBuffer, StringBuilder, java.util.Random, slumptalsfrö
W06	Klasser	objektorientering, klass, Point, Square, Complex, new, null, this, inkapsling, accessregler, private, private[this], kompanjonsobjekt, getters och setters, klassparameter, primär konstruktör, objektfabriksmetod, överlägning av metoder, referenslikhet vs strukturlikhet, eq vs ==
W07	Arv	arv, polymorfism, trait, extends, instanceof, with, inmixning, supertyp, subtyp, bastyp, override, klasshierarkin i Scala: AnyRef Object AnyVal Null Nothing, referenstyper vs värdetyper, klasshierarkin i scala.collection, Shape som bastyp till Rectangle och Circle, accessregler vid arv, protected, final, klass vs trait, abstract class, case-object, typer med uppräknade värden
KS	KONTROLLSKRIVN.	

W08	Repetition, specialundervisning	REBOOT CAMP: repetera, identifiera kunskapsluckor, kom-i-kapp med övningar och labbar, specialundervisning med hårdträning för behövande
W09	Mönster, undantag	mönstermatchning, match, Option, throw, try, catch, Try, unapply, sealed, flatten, flatMap, partiella funktioner, collect, speciella matchningar: wildcard pattern; variable binding; sequence wildcard; back-ticks, equals, hashCode, exempel: equals för klassen Complex, switch-sats i Java
W10	Matriser, typparametrar	matris, nästlad samling, nästlad for-sats, typparameter, generisk funktion, generisk klass, fri vs bunden typparameter, matriser i Java vs Scala, allokering av nästlade arrayer i Scala och Java
W11	Sökning, sortering	strängjämförelse, compareTo, implicit ordning, linjärsökning, binärsökning, algoritm: LINEAR-SEARCH, algoritme: BINARY-SEARCH, algoritmisk komplexitet, sortering till ny vektor, sortering på plats, insättningssortering, urvalssortering, algoritm: INSERTION-SORT, algoritm: SELECTION-SORT, Ordering[T], Ordered[T], Comparator[T], Comparable[T]
W12	Scala och Java	syntaxskillnader mellan Scala och Java, klasser i Scala vs Java, referensvariabler vs enkla värden i Java, referensstil delning vs värdetilldelning i Java, alternativ konstruktor i Scala och Java, for-sats i Java, java for-each i Java, java.util.ArrayList, autoboxing i Java, primitiva typer i Java, wrapperklasser i Java, samlingar i Java vs Scala, scala.collection.JavaConverters, namnkonventioner för konstanter
W13	Extra: design, api, trådar, webb	utvecklingsprocessen, krav-design-implementation-test, gränssnitt, trait vs interface, programmeringsgränssnitt (api), designexempel, tråd, jämlöpande exekvering, icke-blockerande anrop, callback, java.lang.Thread, java.util.concurrent.atomic.AtomicInteger, scala.concurrent.Future, kort om html+css+javascript+scala.js och webbprogrammering
W14	Tentaträning	
T	TENTAMEN	

Vad lär du dig?

- Grundläggande principer för programmering:
Sekvens, Alternativ, Repetition, Abstraktion (SARA)
⇒ Inga förkunskaper i programmering krävs!
- Implementation av algoritmer
- Tänka i abstraktioner, dela upp problem i delproblem
- Förståelse för flera olika angreppssätt:
 - **imperativ programmering**
 - **objektorientering**
 - **funktionsprogrammering**
- Programspråken **Scala** och **Java**
- Utvecklingsverktyg (editor, kompilator, utvecklingsmiljö)
- Implementera, testa, felsöka

Varför Scala + Java som förstaspråk?

- Varför Scala?
 - Enkel och enhetlig syntax => lätt att skriva
 - Enkel och enhetlig semantik => lätt att fatta
 - Kombinerar flera angreppsätt => lätt att visa olika lösningar
 - Statisk typning + typhärledning => färre buggar + koncis kod
 - Scala Read-Evaluate-Print-Loop => lätt att experimentera
- Varför Java?
 - Det mest spridda språket
 - Massor av fritt tillgängliga kodbibliotek
 - Kompabilitet: fungerar på många platormar
 - Effektivitet: avancerad & mogen teknik ger snabba program
- Java och Scala fungerar utmärkt tillsammans
- Illustrera likheter och skillnader mellan olika språk
=> Djupare lärande

Hur lär du dig?

- Genom praktiskt **eget arbete**: **Lära genom att göra!**
 - Övningar: applicera koncept på olika sätt
 - Laborationer: kombinera flera koncept till en helhet
- Genom studier av kursens teori: **Skapa förståelse!**
- Genom samarbete med dina kurskamrater: **Gå djupare!**

Kurslitteratur



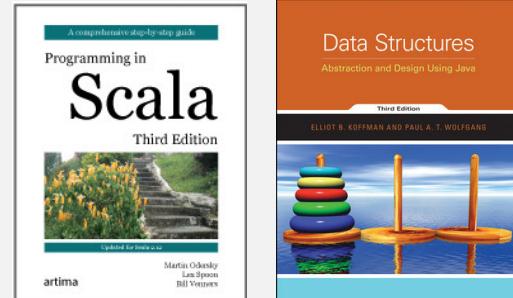
- **Kompendium** med övningar & laborationer, trycks & säljs av inst. på beställning
- Föreläsningsbilder
- Nätresurser enl. länkar

Bra, men ej nödvändigt, **bredvidläsning**:

– för **nybörjare**:



– för de som **redan kodat** en del:



Kompendiet är den huvudsakliga kurslitteraturen och definierar kursinnehållet. Föreläsningar, övningar och laborationer i kompendiet är kursens primära kunskapskällor, tillsammans med de öppna resurser på nätet som kompendiet hänvisar till. Kompendiet är öppen källkod och du välkomnas varmt att bidra!

Om du gärna vill ha en eller flera mer traditionella läroböcker som bredvidläsning rekommenderas följande:

- För de som aldrig kodat, och vill läsa om kodning från grunden:
 - "Introduction to Programming and Problem-Solving Using Scala, Second Edition", Mark C. Lewis, Lisa Lacher. www.crcpress.com/Introduction-to-Programming-and-Problem-Solving-Using-Scala-Second-Edition/Lewis-Lacher/p/book/9781498730952
 - "Objektorienterad programmering och Java", Per Holm, Tredje upplagan (2007). www.studentlitteratur.se/#6735
- För de som redan kodat en hel del i ett objektorienterat språk:
 - "Programming in Scala, Third Edition – A comprehensive step-by-step guide", Martin Odersky, Lex Spoon, and Bill Venners. www.artima.com/shop/programming_in_scala_3ed
 - "Data Structures: Abstraction and Design Using Java, 3rd Edition", Elliot B. Koffman, Paul A. T. Wolfgang. <http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1119186528.html>

Dessa läroböcker följer inte direkt kursens upplägg vad gäller omfång och progression och du får själv göra den nyttiga hemläxan att koppla deras innehåll till det vi går igenom i kursens olika moduler.

Föreläsningsanteckningar

- Föreläsningbilder utvecklas under kursens gång.
- Alla bilder läggs ut här:
github.com/lunduniversity/introprog/tree/master/slides
och uppdateras kontinuerligt allt eftersom de utvecklas.
- Förslag på innehåll välkomna!

Kursmoment — varför?

- **Föreläsningar:** skapa översikt, ge struktur, förklara teori, svara på frågor, motivera varför.
- **Övningar:** bearbeta teorins steg för steg, **grundövningar** för alla, **extraövningar** om du vill/behöver öva mer, **fördjupningsövningar** om du vill gå djupare; **förberedelse inför laborationerna**.
- **Laborationer:** **obligatoriska**, sätta samman teorins delar i ett större program; lösningar redovisas för handledare; gk på alla för att få tenta.
- **Resurstider:** få hjälp med övningar och labortionsförberedelser av handledare, fråga vad du vill.
- **Samarbetsgrupper:** gruppplärande genom samarbete, hjälpa varandra.
- **Kontrollskrivning:** **obligatorisk**, diagnostisk, kamraträttad; kan ge samarbetsbonuspoäng till tentan.
- **Individuell projektuppgift:** **obligatorisk**, du visar att du kan skapa ett större program självständigt; redovisas för handledare.
- **Tentamen:** **obligatorisk**, skriftlig, enda hjälpmittel: snabbreferensen.
<http://cs.lth.se/pgk/quickref>

Varför studera i samarbetsgrupper?

Huvudsyfte: **Bra lärande!**

- Pedagogisk forskning stödjer tesen att lärandet blir mer djupinriktat om det sker i utbyte med andra
- Ett studiesammanhang med **höga ambitioner** och **respektfull gemenskap** gör att vi **når mycket längre**
- Varför ska du som redan kan mycket aktivt dela med dig av dina kunskaper?
 - Förstå bättre själv genom att förklara för andra
 - Träna din pedagogiska förmåga
 - Förbered dig för ditt kommande yrkesliv som mjukvaruutvecklare

En typisk kursvecka

1. Gå på **föreläsningar** på **måndag–tisdag**
2. **Jobba individuellt** med teori, övningar, labbförberedelser på **måndag–torsdag**
3. Kom till **resurstdiderna** och få hjälp och tips av handledare och kurskamrater på **onsdag–torsdag**
4. Genomför den obligatoriska **laborationen** på **fredag**
5. **Träffas i samarbetsgruppen** och hjälp varandra att förstå mer och fördjupa lärandet, förslagsvis på återkommande tider varje vecka då alla i gruppen kan

Se detaljerna och undantagen i schemat: cs.lth.se/pgk/schema

Kapitel 0

Anvisningar

Detta kapitel innehåller anvisningar och riktlinjer för kursens olika delar. Läs noga så att du inte missar viktig information om syftet bakom kursmomenten och vad som förväntas av dig.

0.1 Samarbetsgrupper

Ditt lärande i allmänhet, och ditt programmeringslärande i synnerhet, fördjupas om det sker i dialog med andra. Dessutom är din samarbetsförmåga och din pedagogiska förmåga avgörande för din framgång som professionell systemutvecklare. Därför är kursdeltagarna indelade i *samarbetsgrupper* om 4-6 personer där medlemmarna samverkar för att alla i gruppen ska nå så långt som möjligt i sina studier.

För att hantera och dra nytta av skillnader i förkunskaper är samarbetsgrupperna indelade så att deltagarnas har *varierande förkunskaper* baserat på en förkunskapsenkät. De som redan har provat på att programmera får då chansen att träna på sin pedagogiska förmåga som är så viktig för systemutvecklare, medan de som ännu inte kommit lika långt kan dra nytta av gruppmedlemmarnas samlade kompetens i sitt lärande. Kompetensvariationen i gruppen kommer att förändras under kursens gång, då olika individer lär sig olika snabbt i olika skeden av sitt lärande; de som till att börja med har ett försprång kanske senare får kämpa för att komma över en viss lärandetröskel.

Samarbetsgrupperna organiserar själva sitt arbete och varje grupp får finna de samarbetsformer som passar medlemmarna bäst. Här följer några erfarenhetsbaserade tips:

1. Träffas så fort som möjligt i hela gruppen och lär känna varandra. Ju snabbare ni kommer samman som grupp och får den sociala interaktionen att fungera desto bättre. Ni kommer att ha nytta av denna investering under hela terminen och kanske under resten av er studietid.
2. Kom överens om stående mötestider och mötesplatser. Det är viktigt med kontinuiteten i arbetet för att samarbetet i gruppen ska utvecklas och fördjupas. Träffas minst en gång i veckan. Ha en stående agenda, t.ex.

en runda runt bordet där var och en berättar hur långt hen kommit och listar de begreppen som hen för tillfället behöver fokusera på.

3. Hjälps åt att tillsammans identifiera och diskutera era olika individuella studiebehov och studieambitioner. När man ska lära sig att programmera stöter man på olika lärandeträsklar som man kan få hjälp att ta sig över av någon som redan är förbi tröskeln. Men det gäller då för den som hjälper att först förstå exakt vad det är som är svårt, eller vilka specifika pusselbitar som saknas, för att på bästa sätt kunna underlätta för en medstudent att ta sig över tröskeln. Det gäller att hjälpa *lagom* mycket så att var och en självständigt får chansen att skriva sin egen kod.
4. Var en schysst kamrat och agera professionellt, speciellt i situationer där gruppdeltagarna vill olika. Kommunicera på ett respektfullt sätt och sök konstruktiva kompromisser. Att utvecklas socialt är viktigt för din framtida yrkesutövning som systemutvecklare och i samarbetsgruppen kan du träna och utveckla din samarbetsförmåga.

0.1.1 Samarbetskontrakt

Ni ska upprätta ett samarbetskontrakt redan under första veckan och visa för en handledare. Alla gruppmedlemmarna ska skriva under kontrakten. Handledaren ska också skriva under som bekräftelse på att ni visat kontrakten.

Syftet med kontrakten är att ni ska diskutera igenom i gruppen hur ni vill arbeta och vilka regler ni tycker är rimliga. Ni bestämmer själva vad kontrakten ska innehålla. Nedan finns förslag på punkter som kan ingå i ert kontrakt. En kontraktsmall finns här: <https://github.com/lunduniversity/introprog/blob/master/admin/collaboration-contract.tex>

Samarbetskontrakt

Vi som skrivit under detta kontrakt lovar att göra vårt bästa för att följa samarbetsreglerna nedan, så att alla ska lära sig så mycket som möjligt.

1. Komma i tid till gruppmöten.
2. Vara väl förberedda genom självstudier inför gruppmöten.
3. Hjälpa varandra att förstå, men inte ta över och lösa allt åt någon annan.
4. Ha ett respektfullt bemötande även om vi har olika åsikter.
5. Inkludera alla i gemenskapen.
6. ...

0.1.2 Grupplaborationer

Det finns två typer av laborationer: individuella laborationer och grupplaborationer. Det flesta av kursens laborationer är individuella, medan laborationerna i veckorna W07, W08 och W11 genomförs av respektive samarbetsgrupp gemensamt. Följande anvisningar gäller speciellt för grupplaborationer. (Allmänna anvisningar för laborationer finns i avsnitt 0.5.)

1. Diskutera i din samarbetsgrupp hur ni ska dela upp koden mellan er i flera olika delar, som ni kan arbeta med var för sig. En sådan del kan vara en klass, en trait, ett objekt, ett paket, eller en funktion.
2. Varje del ska ha en *huvudansvarig* individ.
3. Arbetsfördelningen ska vara någorlunda jämt fördelad mellan gruppmedlemmarna.
4. När ni redovisar er lösning ska ni börja med att redogöra för handledaren hur ni delat upp koden och vem som är huvudansvarig för vad.
5. Den som är huvudansvarig för en viss del redovisar den delen.
6. Grupplaborationer görs i huvudsak som hemuppgift. Salstiden används primärt för redovisning.

0.1.3 Samarbetsbonus

Alla tjänar på att samarbeta och hjälpa varandra i lärandet. Som extra incitament för grupplärande utdelas *samarbetsbonus* baserat på resultatet från den diagnostiska kontrollskrivningen efter halva kurserna (se avsnitt 0.6). Bonus ges till varje student enligt gruppmedelvärdet av kontrollskrivningspoängen och räknas ut med funktionen `collaborationBonus` nedan, där `points` är en sekvens med heltal som utgör gruppmedlemmars individuella poäng från kontrollskrivningen.

```
def collaborationBonus(points: Seq[Int]): Int =  
    (points.sum / points.size.toDouble).round.toInt
```

Samarbetsbonusen viktas så att den högsta möjliga bonusen maximalt utgör 5% av maxpoängen på tentan och adderas till det individuella tentaresultatet om du är godkänd på kursens sluttentamen. Samarbetsbonusen kan alltså påverka om du når högre betyg, men den påverkar *inte* om du får godkänt eller ej. Detta gör att alla i gruppen gynnas av att så många som möjligt lär sig på djupet inför kontrollskrivningen. Din eventuella samarbetsbonusen räknas dig tillgodo endast vid det första, ordinarie tentamenstillfället.

0.2 Föreläsningar

En normal läsperiodsvecka börjar med två föreläsningspass om 2 timmar vardera. Föreläsningarna ger en översikt av kursens teoretiska innehåll och går igenom innehörden av de begrepp du ska lära dig. Föreläsningarna innehåller många programmeringsexempel och föreläsaren ”lajvkodar” då och då för att illustrera den kreativa problemlösningsprocess som ingår i all programmering. Föreläsningarna berör även kursens organisation och olika praktiska detaljer.

På föreläsningarna ges goda möjligheter att ställa allmänna frågor om teorin och att i plenum diskutera specifika svårigheter (individuell lärarhjälp ges på resurstider, se avsnitt 0.4, och på laborationer, se avsnitt 0.5). Även om det är många i föreläsningssalen, *tveka inte att ställa frågor* – det är säkert fler som undrar samma sak som du!

Föreläsningarna är inte obligatoriska, men det är mycket viktigt att du går dit, även om du i perioder känner att du har bra koll på all teori. På föreläsningarna får du en övergripande ämnesstruktur och en konkret programmeringsupplevelse, som du delar med dina kursare och kan diskutera i samarbetsgrupperna. Föreläsningarna ger också en prioritering av materialet och förbereder dig inför examinationen med praktiska råd och tips om hur du bör fokusera dina studier.

0.3 Övningar

I en normal läsperiodsvecka ingår en övning med flera uppgifter och deluppgifter. Övningarna utgör basen för dina programmeringsstudier och erbjuder en systematisk genomgång av kursteorins alla delar genom praktiska kodexempel som du genomför steg för steg vid datorn med hjälp av ett interaktivt verktyg som kallas Read-Evaluate-Print-Loop (REPL). Om du gör övningarna i REPL säkerställer du att du skaffar dig tillräcklig förståelse för alla begrepp som ingår i kursen och att du inte missar någon viktigt pusselbit.

Övningarna utgör också förberedelse inför laborationerna. Om du inte gör veckans övning är det inte troligt att du kommer att klara veckans laboration inom rimlig tid.

Dessa två punkter är speciellt viktiga när du ska lära sig att programmera:

- **Programmera!** Det räcker inte med att bara passivt läsa om programmering; du måste *aktivt* själv skriva mycket kod och genomföra egna programmeringsexperiment. Det underlättar stort om du bejakar din nyfikenhet och experimentlusta. Alla programmeringsfel som du gör och alla dina misstag, som i efterhand verkar enkla, är i själva verket oumbärliga steg på vägen och ger avgörande ”*Aha!*”-upplevelser. Kursens övningarna är grunden för denna form av lärande.
- **Ha tålmod!** Det är först när du har förmågan att aktivt kombinera *många* olika programmeringskoncept som du själv kan lösa lite större programmeringsuppgifter. Det kan vara frustrerande i början innan du

når så långt att din verktygslåda med begrepp är tillräckligt stor för att du ska kunna skapa den kod du vill. Ibland krävs det extra tålmod innan allt plötslig lossnar. Många programmeringslärare och -studenter vittnar om att ”polletten plötsligt trillar ner” och allt faller på plats. Övningarna syftar till att, steg för steg, bygga din verktygslåda så att den till slut blir tillräckligt kraftfull för mer avancerad problemlösning.

Olika studenter har olika ambitionsnivå, olika arbetskapacitet, mer eller mindre välutvecklad studietecknik och olika lätt för att lära sig att programmera. För att hantera denna variation erbjuds övningsuppgifter av tre olika typer:

- **Grunduppgifter.** Varje veckas grunduppgifter täcker basteorin och hjälper dig att säkerställa att du kan gå vidare utan kunskapsluckor. Grunduppgifterna utgör även basen för laborationerna. Alla studenter bör göra alla grunduppgifter. En bra förståelse för innehållet i grunduppgifterna ger goda förutsättningar att klara godkänt betyg på sluttentamen.
- **Extrauppgifter.** Om du upplever att grunduppgifterna är svåra och du vill öva mer, eller om du vill vara säker på att du verkligen befäster dina grundkunskaper, då ska du göra extrauppgifterna. Dessa är på samma nivå som grunduppgifterna och ger extra träning.
- **Fördjupningsuppgifter.** Om du vill gå djupare och har kapacitet att lära dig ännu mer, gör då fördjupningsuppgifterna. Dessa kompletterar grunduppgifterna med mer avancerade exempel och går utöver vad som krävs för godkänt på kursen. Om du satsar på något av de högre betygen ska du göra fördjupningsuppgifterna.



Vissa uppgifter har en penna i marginalen. Denna symbol indikerar att du ska räkna ut något, rita en figur över minnessituationen, söka information på nätet eller på annat sätt komma fram till ett resultat och gärna skriva ner resultatet (snarare än att ”bara” köra kodexempel i REPL).

Till varje övning finns lösningar som du hittar på kursens hemsida. Titta *inte* på lösningen innan du själv först försökt lösa uppgiften. Ofta innehåller lösningarna kommentarer och tips så glöm inte att kolla igenom veckans lösningar innan du börjar förbereda dig inför veckans laboration.

Tänk på att det ofta finns *många olika lösningar* på samma programmeringsproblem, som kan vara likvärdiga eller ha olika fördelar och nackdelar beroende på sammanhanget. Diskutera gärna olika lösningsvarianter med dina kursare och handledare – att prova många olika sätt att lösa en uppgift fördjupar ditt lärande avsevärt!

Många uppgifter lyder ”testa detta i REPL och förklara vad som händer” och svårigheten ligger ofta inte i att skapa själva koden utan att förstå hur den fungerar och *varför*. På detta sätt tränar du ditt programmeringstänkande med hjälp av en växande begreppsapparat. Syftet är ofta att illustrera ett allmänt giltigt koncept och det är därför extra bra om du skapar egna övningsuppgifter på samma tema och experimenterar med nya varianter som ger dig ytterligare förståelse.

Övningsuppgifterna innehåller ofta färdiga kodsnuttar som du ska skriva in i REPL medan den kör i ett terminalfönster. REPL-kod visas i övningsuppgifterna med ljus text på mörk bakgrund, så här:

```
1 scala> val msg = "Hello world!"  
2 scala> println(msg)
```

Prompten `scala>` indikerar att REPL är igång och väntar på indata. Du ska skriva den kod som står *efter* prompten. Mer information om hur du använder REPL hittar du i appendix [C.3.2](#).

Även om kompendiet finns tillgängligt för nedladdning, frestas *inte* att klippa ut och klistra in alla kodsnuttar i REPL. Ta dig istället den ringa tiden det tar att skriva in koden rad för rad. Medan du själv skriver hinner du tänka efter, och det egna, aktiva skrivandet främjar ditt lärande och gör det lättare att komma ihåg och förstå.

0.4 Resurstider

Under varje läsperiodsvecka finns ett flertal resurstider i schemat. Det finns minst en tid som passar din schemagrupp, men du får gärna gå på andra och/eller flera tider i mån av plats. Resurstiderna är schemalagda i datorsal med Linuxdatorer och i varje sal finns en handledare som är redo att svara på dina frågor.

Följande riktlinjer gäller för resurstiderna:

1. **Syfte.** Resurstiderna är primärt till för att hjälpa dig vidare om du kör fast med övningarna eller laborationsförberedelserna, men du får fråga om vad som helst som rör kursen i den mån handledaren kan svara och hinner med.
2. **Samarbete.** Hjälp gärna varandra under resurstiderna! Om någon kursare kör fast är det utvecklande och lärorikt att hjälpa till. Om schema och plats tillåter kan du gärna gå på samma resurstdistillfälle som någon medlem i din samarbetsgrupp, men ni kan också lika gärna hjälpas åt tvärs över gruppgränserna.
3. **Hänsyn.** När du hjälper andra, tänk på att prata riktigt tyst så att du inte stör andras koncentration. Tänk också på att alla behöver träna mycket själv utan att bli alltför stydda av en ”baksätesförare”. Ta inte över tangentbordet från någon annan; ge hellre väldenomtänkta tips på vägen och låt din kursare behålla kontrollen över uppgiftlösningen.
4. **Fokus.** Du ska *inte* göra och redovisa laborationen på resurstiderna; dessa ska göras och redovisas på laborationstid. Men om du varit sjuk eller ej blivit godkänd på någon enstaka laborationerna kan du, om handledaren så hinner, be att få redovisa din restlaboration på en resurstdistillfälle.

5. **Framstegsprotokoll.** På sidan [iii](#) finns ett framstegsprotokoll för övningarna. Håll detta uppdaterat allteftersom du genomför övningarna och visa protokollet när du frågar om hjälp av handledare. Då blir det lättare för handledaren att se vilka kunskaper du förvärvat hittills och anpassa dialogen därefter.

0.5 Laborationer

En normal läsperiodsvecka avslutas med en lärarhandledd laboration. Medan övningar tränar teorins olika delar i många mindre uppgifter, syftar laborationerna till träning i att kombinera flera begrepp och applicera dessa tillsammans i ett större program med flera samverkande delar.

En laboration varar i 2 timmar och är schemalagd i salar med datorer som kör Linux. Följande anvisningar gäller för laborationerna:

1. **Obligatorium.** Laborationerna är obligatoriska och en viktig del av kurssens examination. Godkända laborationer visar att du kan tillämpa den teori som ingår i kursen och att du har tillgodogjort dig en grundläggande förmåga att självständigt, och i grupp, utveckla större program med många delar. *Observera att samtliga laborationer måste vara godkända innan du får tentera!*
2. **Individuellt arbete.** Du ska lösa de individuella laborationerna *självständigt* genom eget, enskilt arbete. Det är tillåtet att under förberedelserna diskutera övergripande principer för laborationernas lösningar i samarbetsgruppen, men var och en måste skapa sin egen lösning. (Speciella anvisningar för grupplaborationer finns i avsnitt [0.1.2](#).) *Du ska absolut inte lägga ut laborationslösningar på nätet.* Läs noga på denna webbsida om var gränsen går mellan samarbete och fusk: <http://cs.lth.se/utbildning/samarbete-eller-fusk/>
3. **Förberedelser.** Till varje laboration finns förberedelser som du ska göra *före* laborationen. Detta är helt avgörande för att du ska hinna göra laborationen inom 2 timmar. Ta hjälp av en kamrat eller en handledare under resurstdiderna om det dyker upp några frågor under ditt förberedelsearbete. Innan varje laboration skall du ha:
 - (a) studerat relevanta delar av kompendiet;
 - (b) gjort grunduppgifterna som ingår i veckans övning, och gärna även (några) extraövningar och/eller fördjupningsövningar;
 - (c) läst igenom *hela* laborationen noggrant;
 - (d) löst förberedelseuppgifterna. I labbförberedelserna ska du i förekommande fall skriva delar av den kod som ingår i laborationen. Det krävs inte att allt du skrivit är helt korrekt, men du ska ha gjort ett rimligt försök. Ta hjälp om du får problem med uppgifterna, men låt inte någon annan lösa uppgiften åt dig.

Om du inte hinner med alla obligatoriska labbuppgifter, får du göra de återstående uppgifterna på egen hand och redovisa dem vid påföljande labbtillfälle eller resurstid, och förbereda dig *ännu* bättre till nästa laboration...

4. **Sjukanmälan.** Om du är sjuk vid något laborationstillfälle måste du anmäla detta till *kursansvarig* via mejl *före* laborationen. Om du varit sjuk ska du försöka göra uppgiften på egen hand och sedan redovisa den vid nästa labbtillfälle eller resurstid. Om du behöver hjälp att komma ikapp efter sjukdom, kom till en eller flera resurstider och prata med en handledare. Om du uteblir utan att ha anmält sjukdom kan kursansvarig besluta att du får vänta till nästa läsår med redovisningen, och då får du inte något slutbetyg i kurserna under innevarande läsår.
5. **Skriftliga svar.** Vid några laborationsuppgifter finns en penna i marginalen. Denna symbol indikerar att du ska skriva ner och spara ett resultat som du behöver senare, och/eller som du ska visa upp för labbhandledaren vid en efterföljande kontrollpunkt. 
6. **Kontrollpunkter.** Vid några laborationsuppgifter finns en ögonsymbol ✓  med en bock i marginalen. Detta innebär att du nått en kontrollpunkt där du ska diskutera dina resultat med en handledare. Räck upp handen och visa vad du gjort innan du fortsätter. Om det är lång väntan innan handledaren kan komma så är det ok att ändå gå vidare, men glöm inte att senare diskutera med handledaren så att ni gemensamt säkerställer att du förstått alla delresultat. Dialogen med din handledare är en viktig chans till återkoppling på din kod – ta vara på den!

0.6 Kontrollskrivning

Efter första halvan av kurserna ska du göra en *obligatorisk kontrollskrivning*, som genomförs individuellt på papper och penna, och liknar till formen den ordinarie tentan. Kontrollskrivningen är *diagnostisk* och syftar till att hjälpa dig att avgöra ditt kunskapsläge när halva kurserna är över. Ett annat syfte är att ge träning i att lösa skrivningsuppgifter med papper och penna utan datorhjälpmedel.

Kontrollskrivningen rättas med *kamratbedömning* under själva skrivningstillfället. Du och en kurskamrat får efter att skrivningstiden är ute två andra skrivningar att poängbedöma i enlighet med en bedömningsmall. Syftet med detta är att du ska få träning i att bedöma kod som andra skrivit och att resonera kring kodkvalitet. När rättningen är klar får du se poängsättningen av din skrivning och kan i händelse av avgörande felaktigheter överklaga bedömningen till kursansvarig.

Den diagnostiska kontrollskrivningen påverkar inte om du blir godkänd eller ej på kurserna, men det samlade poängresultatet för din samarbetsgrupp ger möjlighet till *samarbetsbonus* som kan påverka ditt betyg på kurserna (se avsnitt [0.1.3](#)).

0.7 Projektuppgift

Efter avslutad labbserie följer en projektuppgift där du på egen hand ska skapa ett stort program med många olika samverkande delar. Det är först när mängden kod blir riktigt stor som du verkligen har nytta av de olika abstraktionsmekanismer du lärt dig under kursens gång och din felsökningsförmåga sätts på prov. Följande anvisningar gäller för projektuppgiften:

1. **Val av projektuppgift.** Du väljer själv projektuppgift. I kapitel 13 finns flera förslag att välja bland. Läs igenom alla uppgiftsalternativ innan du väljer vilken du vill göra. Du kan också i samråd med en handledare definiera en egen projektuppgift, men innan du börjar på en egendefinierad projektuppgift ska en skriftlig beskrivning av uppgiften godkännas av handledare, senast två veckor innan redovisningstillfället. Välj uppgift efter vad du tror du klarar av och undvik både en för simpel uppgift och att ta dig vatten över huvudet.
2. Anvisningarna 1 och 2 för laborationer (se avsnitt 0.5) gäller också för projektuppgiften: den är **obligatorisk** och arbetet ska ske **individuellt**. Du får diskutera din projektuppgift på ett övergripande plan med andra och du kan be om hjälp av handledare på resurstdid med enskilda detaljer om du kör fast, men lösningen ska vara *din* och du ska ha skrivit hela programmet själv.
3. **Omfattning.** Skillnaden mellan projektuppgiften och labbarna är att den ska vara *väsentligt* mer omfattande än de största laborationerna och att du färdigställer den kompletta lösningen *innan* redovisningstillfället. Du behöver därför börja i god tid, förslagsvis två veckor innan redovisningstillfället, för att säkert hinna klart. Det är viktigt att du tänker igenom omfattningen noga, i förhållande till ditt val av projektuppgift, gärna utifrån din självinsikt om vad du behöver träna på. Det är också bra (men inte obligatoriskt) om du blandar Scala och Java i din projektuppgift. Diskutera gärna med en handledare hur du använder projektuppgiften på bästa sätt för ditt lärande.
4. **Dokumentation.** Inför redovisningen ska färdigställa automatiskt genererad dokumentation utifrån relevanta dokumentationskommentarer, så som beskrivs i appendix F.
5. **Redovisning.** Vid redovisningen använder du tiden med handledaren till att gå igenom din lösning och redogöra för hur din kod fungerar och diskutera för- och nackdelar med ditt angreppssätt. Du ska också beskriva framväxten av ditt program och hur du stegvis har avlusat och förbättrat implementationen. På redovisningen ska du även gå igenom dokumentationen av din kod.

0.8 Tentamen

Kursen avslutas med en skriftlig tentamen med en snabbreferens¹ som enda tillåtna hjälpmedel. Tentamensuppgifterna är uppdelade i två delar, del A och del B. Följande preliminära gränser gäller:

- Del A omfattar 20% av den maximala poängsumman.
- Om du efter bedömning av del A erhållit färre än 80% av A-delens maxpoäng underkänns din tentamen utan att del B bedöms.
- Betygsgränser:
 - Du måste ha totalt minst 50% av maxpoängen, exklusive eventuell samarbetsbonus, för att bli godkänd.
 - För betyg 4 krävs minst 70% av maxpoängen, inklusive eventuell samarbetsbonus.
 - För betyg 5 krävs minst 90% av maxpoängen, inklusive eventuell samarbetsbonus.

¹<http://cs.lth.se/pgk/quickref>

Del II

Moduler

Kapitel 1

Introduktion

Begrepp som ingår i denna veckas studier:

- sekvens
- alternativ
- repetition
- abstraktion
- programmeringsspråk
- programmeringsparadigmer
- editera-kompilera-exekvera
- datorns delar
- virtuell maskin
- REPL
- literal
- värde
- uttryck
- identifierare
- variabel
- typ
- tilldelning
- namn
- val
- var
- def
- inbyggda grundtyper
- Int
- Long
- Short
- Double
- Float
- Byte
- Char
- String
- println
- typen Unit
- enhetsvärdet ()
- stränginterpolatorn s
- if
- else
- true
- false
- MinValue
- MaxValue
- aritmetik
- slumptal
- math.random
- logiska uttryck
- de Morgans lagar
- while-sats
- for-sats

Vad är programmering?

- Programmering innebär att ge instruktioner till en maskin.
- Ett **programmeringsspråk** används av människor för att skriva **käll-kod** som kan översättas av en **kompilator** till **maskinspråk** som i sin tur **exekveras** av en dator.

- Ada Lovelace skrev det första programmet redan på 1800-talet ämnat för en kugghjulsdator.
- sv.wikipedia.org/wiki/Programmering
- en.wikipedia.org/wiki/Computer_programming
- Ha picknick i Ada Lovelace-parken på Brunnshög!



Vad är en kompilator?



Grace Hopper uppfann första kompilatorn 1952.
en.wikipedia.org/wiki/Grace_Hopper



Vad består ett program av?

- Text som följer entydiga språkregler (gramatik):
 - **Syntax**: textens konkreta utseende
 - **Semantik**: textens betydelse (vad maskinen gör/beräknar)
- **Nyckelord**: ord med speciell betydelse, t.ex. **if, else**
- **Deklarationer**: definitioner av nya ord: **def** gurka = 42
- **Satser** är instruktioner som **gör** något: **print("hej")**
- **Uttryck** är instruktioner som beräknar ett **resultat**: **1 + 1**
- **Data** är information som behandlas: t.ex. heltalet 42
- Instruktioner ordnas i kodstrukturer: (SARA)
 - **Sekvens**: ordningen spelar roll för vad som händer
 - **Alternativ**: olika saker händer beroende på uttrycks värde
 - **Repetition**: satser upprepas många gånger
 - **Abstraktion**: nya byggblock skapas för att återanvändas

Exempel på programmeringsspråk

Det finns massor med olika språk och det kommer ständigt nya.

Exempel:

- Java
- C
- C++
- C#
- Python
- JavaScript
- Scala

Topplistor:

- TIOBE Index
- PYPL Index



Olika programmeringsparadigm

- Det finns många olika **programmeringsparadigm** (sätt att programmera på), till exempel:
 - **imperativ programmering**: programmet är uppbyggt av sekvenser av olika satser som påverkar systemets tillstånd
 - **objektorienterad programmering**: en sorts imperativ programmering där programmet består av objekt som sammanför data och operationer på dessa data
 - **funktionsprogrammering**: programmet är uppbyggt av samverkande (matematiska) funktioner som undvikar förändringsdata och tillståndsänderingar
 - **deklarativ programmering, logikprogrammering**: programmet är uppbyggt av logiska uttryck som beskriver olika fakta eller villkor och exekveringen utgörs av en bevisprocedur som söker efter värden som uppfyller fakta och villkor

Hello world

```
scala> println("Hello World!")
Hello World!
```

```
// this is Scala

object Hello {
    def main(args: Array[String]): Unit = {
        println("Hejsan scala-appen!")
    }
}
```

```
// this is Java

public class Hi {
    public static void main(String[] args) {
        System.out.println("Hejsan Java-appen!");
    }
}
```

Utvecklingscykeln

editera; kompilera; hitta fel och förbättringar; editera; kompilera; hitta fel och förbättringar; editera; kompilera; hitta fel och förbättringar; editera; kompile-ra; hitta fel och förbättringar; editera; kompilera; hitta fel och förbättringar; editera; kompilera; hitta fel och förbättringar; ...

```
upprepa(1000) {
    editera
    kompilera
    testa
}
```

Utvecklingsverktyg

- Din verktygskunskap är mycket viktig för din produktivitet.
- Lär dig kortkommandon för vanliga handgrep.
- Verktyg vi använder i kursen:
 - Scala **REPL**: från övn 1
 - **Texteditor** för kod, t.ex gedit eller atom: från övn 2
 - Kompilera med **scalac** och **javac**: från övn 2
 - Integrerad utvecklingsmiljö (IDE)
 - * **Kojo**: från lab 1
 - * **Eclipse+ScalaIDE** eller **IntelliJ IDEA** med Scala-plugin: från lab 3 i vecka 4
 - **jar** för att packa ihop och distribuera klassfiler
 - **javadoc** och **scaladoc** för dokumentation av kodbibliotek
- Andra verktyg som är bra att lära sig:
 - git för versionshantering
 - GitHub för kodlagring – men **inte** av lösningar till labbar!

Literaler

- Literaler representerar ett fixt **värde** i koden och används för att skapa **data** som programmet ska bearbeta.
- Exempel:
 - 42 heltalslitteral
 - 42.0 decimaltalslitteral
 - '!' teckenlitteral, omgärdas med 'enkelfnuttar'
 - "hej" stränglitteral, omgärdas med "dubbelfnuttar"
 - true** litteral för sanningsvärdet "sant"
- Literaler har en **typ** som avgör vad man kan göra med dem.

Exempel på inbyggda datatyper i Scala

- Alla värden, uttryck och variabler har en **datatyp**, t.ex.:
 - Int för heltal
 - Long för extra stora heltal (tar mer minne)
 - Double för decimaltal, så kallade flyttal med flyttande decimalpunkt
 - String för strängar
- Kompilatorn håller reda på att uttryck kombineras på ett **typsäkert** sätt. Annars blir det **kompileringsfel**.
- Scala och Java är s.k. **statiskt typade** språk, vilket innebär att **all** typinformation måste finnas redan vid kompilering (eng. *compile time*)^a.
- Scala-kompilatorn gör **typhärledning**: man **slipper skriva typerna** om kompilatorn kan lista ut dem med hjälp av typerna hos deluttrycken.

^aAndra språk, t.ex. Python och Javascript är **dynamiskt typade** och där skjuts typkontrollen upp till körningsdags (eng. *run time*)

Vilka är för- och nackdelarna med statisk vs. dynamisk typning?

Grundtyper i Scala

Dessa **grundtyper** (eng. *basic types*) finns inbyggda i Scala:

Svenskt namn	Engelskt namn	Grundtyper
heltalstyp	integral type	Byte, Short, Int, Long, Char
flyttalstyp	floating point number types	Float, Double
numeriska typer	numeric types	heltalstyper och flyttalstyper
strängtyp (teckensekvens)	string type	String
sanningsvärdestyp (booelsk typ)	truth value type	Boolean

Grundtypernas implementation i JVM

Grundtyp i Scala	Antal bitar	Omfång minsta/största värde	primitiv typ i Java & JVM
Byte	8	$-2^7 \dots 2^7 - 1$	byte
Short	16	$-2^{15} \dots 2^{15} - 1$	short
Char	16	$0 \dots 2^{16} - 1$	char
Int	32	$-2^{31} \dots 2^{31} - 1$	int
Long	64	$-2^{63} \dots 2^{63} - 1$	long
Float	32	$\pm 3.4028235 \cdot 10^{38}$	float
Double	64	$\pm 1.7976931348623157 \cdot 10^{308}$	double

Grundtypen String lagras som en *sekvens* av 16-bitars tecken av typen Char och kan vara av godtycklig längd (tills minnet tar slut).

1.1 Övning: expressions

Mål

- Förstå vad som händer när satser exekveras och uttryck evalueras.
- Förstå sekvens, alternativ och repetition.
- Känna till literalerna för enkla värden, deras typer och omfång.
- Kunna deklarera och använda variabler och tilldelning, samt kunna rita bilder av minnessituationen då variablers värden förändras.
- Förstå skillnaden mellan olika numeriska typer, kunna omvandla mellan dessa och vara medveten om noggrannhetsproblem som kan uppstå.
- Förstå booleska uttryck och värdena **true** och **false**, samt kunna förenkla booleska uttryck.
- Förstå skillnaden mellan heltalsdivision och flyttalsdivision, samt användning av rest vid heltalsdivision.
- Förstå precedensregler och användning av parenteser i uttryck.
- Kunna använda **if**-satsen och **if**-uttryck.
- Kunna använda **for**-satsen och **while**-satsen.
- Kunna använda `math.random` för att generera slumpat i olika intervaller.

Förberedelser

- Studera begreppen i kapitel 1.
- Du behöver en dator med Scala installerad, se appendix C.

1.1.1 Grunduppgifter

Uppgift 1. Starta Scala REPL (eng. *Read-Evaluate-Print-Loop*) och skriv satsen `println("hejsan REPL")` och tryck på *Enter*.

```
> scala
Welcome to Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8).
Type in expressions to have them evaluated.
Type :help for more information.

scala> println("hejsan REPL")
```

- Vad händer?
- Skriv samma sats igen (eller tryck pil-upp) men "glöm bort" att skriva högerparentesen innan du trycker på *Enter*. Vad händer?
- Evaluera uttrycket "gurka" + "tomat" i REPL. Vad har uttrycket för värde och typ? Vilken siffra står efter ordet `res` i variabeln som lagrar resultatet?

```
scala> "gurka" + "tomat"
```

- Evaluera uttrycket `res0 * 4` (byt ev. ut `0`:an mot siffran efter `res` i utskriften från förra evalueringen). Vad har uttrycket för värde och typ?

```
scala> res0 * 4
```

**Uppgift 2.** Vad är en *literal*?

[en.wikipedia.org/wiki/Literal_\(computer_programming\)](https://en.wikipedia.org/wiki/Literal_(computer_programming))

Uppgift 3. Vilken typ har följande literaler? Försök först gissa vilken typen blir; testa sedan i REPL och notera vad det blev för typ.

- a) 15
- b) 32L
- c) '*'
- d) "*"
- e) 42.0
- f) 84D
- g) 32d
- h) 23F
- i) 18f
- j) **true**
- k) **false**

**Uppgift 4.** Vad gör dessa satser? Till vad används klammer och semikolon?

```
scala> def p = { print("hej"); println("san"); println(42); println("gurka") }
scala> p;p;p;p
```

**Uppgift 5.** Satser versus uttryck.

- a) Vad är det för skillnad på en sats och ett uttryck?
- b) Ge exempel på satser som inte är uttryck?
- c) Förklara vad som händer för varje evaluerad rad:

```
1 scala> def värdeSaknas = ()
2 scala> värdeSaknas
3 scala> värdeSaknas.toString
4 scala> println(värdeSaknas)
5 scala> println(println("hej"))
```

- d) Vilken typ har literalen ()?
- e) Vilken returytpe har println?

Uppgift 6. Vilken typ och vilket värde har följande uttryck? Försök först gissa vilket värde och vilken typ det blir; testa sedan i REPL och notera resultatet.

- a) 1 + 41
- b) 1.0 + 18
- c) 42.toDouble

- d) `(41 + 1).toDouble`
- e) `1.042e42`
- f) `12E6.toLong`
- g) `"gurk" + 'a'`
- h) `'A'`
- i) `'A'.toInt`
- j) `'0'.toInt`
- k) `'1'.toInt`
- l) `'9'.toInt`
- m) `'A' + '0'`
- n) `('A' + '0').toChar`
- o) `"*!%#".charAt(0)`

Uppgift 7. *De fyra räknesätten.* Vilket värde och vilken typ har följande uttryck?

- a) `42 * 2`
- b) `42.0 / 2`
- c) `42 - 0.2`
- d) `9L + 3d`

Uppgift 8. *Precedensregler.* Evalueringsordningen kan styras med parenteser. Vilket värde och vilken typ har följande uttryck?

- a) `23 + 2 * 2`
- b) `(23 + 2) * 2`
- c) `(-(2 - 42)) / (1 + 1 + 1).toDouble`
- d) `((-(2 - 42)) / (1 + 1 + 1).toDouble).toInt`

Uppgift 9. *Heltalsdivision.* Vilket värde och vilken typ har uttrycken i deluppgifterna **a** till **h** nedan?

- a) `42 / 2`
- b) `42 / 4`
- c) `42.0 / 4`
- d) `1 / 4`
- e) `1 % 4`
- f) `45 % 42`
- g) `42 % 2`
- h) `41 % 2`
- i) Skriv ett uttryck som ”plockar ut” siffran 7 ur talet 5793 med hjälp av  heltalsdivision och modulräkning.

Uppgift 10. *Heltalsomfång.* För var och en av heltaletyperna i deluppgifterna nedan: undersök i REPL med operationen `.MaxValue` resp. `.MinValue`, vad som är största och minsta värde, till exempel `Int.MaxValue` etc.

- a) Byte
- b) Short
- c) Int
- d) Long

Uppgift 11. Klassen `java.lang.Math` och paketobjektet `scala.math`. Genom att trycka på tab tangenten kan man se vad som finns i olika paket.

```
1 scala> java.      //tryck TAB efter punkten
2 applet    awt    beans   io    lang    math    net    nio    rmi    security    sql
3
4 scala>
```

- a) Undersök genom att trycka på Tab-tangenten, vilka funktioner som finns i `Math` och `math`. Vad heter konstanten π i `java.lang.Math` respektive `scala.math`?

```
1 scala> java.lang.Math.    //tryck TAB efter punkten
2 scala> scala.math.        //tryck TAB efter punkten
```

- b) Undersök dokumentationen för klassen `java.lang.Math` här:
<https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>
Vad gör `java.lang.Math.hypot`?
- c) Undersök dokumentationen för paketobjektet `scala.math` här:
<http://www.scala-lang.org/api/current/#scala.math.package>
Ge exempel på någon funktion i `java.lang.Math` som inte finns i `scala.math`.

Uppgift 12. Vad händer här? Notera undantag (eng. *exceptions*) och noggrannhetsproblem.

- a) `Int.MaxValue + 1`
- b) `1 / 0`
- c) `1E8 + 1E-8`
- d) `1E9 + 1E-9`
- e) `math.pow(math.hypot(3,6), 2)`
- f) `1.0 / 0`
- g) `(1.0 / 0).toInt`
- h) `math.sqrt(-1)`
- i) `math.sqrt(Double.NaN)`
- j) `throw new Exception("PANG!!!")`

Uppgift 13. *Booleska uttryck.* Vilket värde och vilken typ har följande uttryck?

- a) `true && true`
- b) `false && true`

- c) `true && false`
- d) `false && false`
- e) `true || true`
- f) `false || true`
- g) `true || false`
- h) `false || false`
- i) `42 == 42`
- j) `42 != 42`
- k) `42.0001 == 42`
- l) `42.000000000000001 == 42`
- m) `42.0001 > 42`
- n) `42.000000000000001 > 42`
- o) `42.0001 >= 42`
- p) `42.000000000000001 <= 42`
- q) `true == true`
- r) `true != true`
- s) `true > false`
- t) `true < false`
- u) `'A' == 65`
- v) `'S' != 66`

Uppgift 14. *Variabler och tilldelning.* Rita en ny bild av datorns minne efter varje evaluerad rad nedan. Bilderna ska visa variablers namn, typ och värde. 

```

1 scala> var a = 13
2 scala> var b = a + 1
3 scala> var c = (a + b) * 2.0
4 scala> b = 0
5 scala> a = 0
6 scala> c = c + 1

```

Efter första raden ser minnessituationen ut så här:

a: Int	13
--------	----

Uppgift 15. *Deklarationer: var, val, def.* Evaluera varje rad nedan i tur och ordning i Scala REPL.

```

1 scala> var x = 30
2 scala> x + 1
3 scala> x
4 scala> x = x + 1
5 scala> x
6 scala> x == x + 1
7 scala> val y = 20
8 scala> y = y + 1

```

```

9  scala> var z = {println("gurka"); 10}
10 scala> def w = {println("gurka"); 10}
11 scala> z
12 scala> z
13 scala> z = z + 1
14 scala> w
15 scala> w
16 scala> w = w + 1

```

- a) För varje rad ovan: förklara för vad som händer.
- b) Vilka rader ger kompileringsfel och i så fall vilket och varför?
- c) Vad är det för skillnad på **var**, **val** och **def**?
- d) Tilldela variabeln **val even** värdet av ett uttryck som med modulo-operatorn `%` och likhetsoperatorn `==` testar om ett tal `n` är jämnt.
- e) Tilldela variabeln **val odd** värdet av ett uttryck som med modulo-operatorn `%` och olikhetsoperatorn `!=` testar om ett tal `n` är udda.
- Uppgift 16.** *Tilldelningsoperatorer.* Man kan förkorta en tilldelningssats som förändrar en variabel, t.ex. `x = x + 1`, genom att använda så kallade tilldelningsoperatorer och skriva `x += 1` som betyder samma sak. Rita en ny bild av datorns minne efter varje evaluerad rad nedan. Bilderna ska visa variablers namn, typ och värde.

```

1  scala> var a = 40
2  scala> var b = a + 40
3  scala> a += 10
4  scala> b -= 10
5  scala> a *= 2
6  scala> b /= 2

```

Uppgift 17. *Stränginterpolatorn s.* Man behöver ofta skapa strängar som innehåller variabelvärdet. Med ett `s` framför en strängliteral får man hjälp av kompilatoren att, på ett typsäkert sätt, infoga variabelvärdet i en sträng. Variablernas namn ska föregås med ett dollartecken, t.ex. `s"Hej $namn"`. Om man vill evaluera ett uttryck placeras detta inom klammer direkt efter dollartecknet, t.ex. `s"Dubbla längden: ${namn.size * 2}"`

```

1  scala> val f = "Kim"
2  scala> val e = "Finkodare"
3  scala> val tot = f.size + e.size
4  scala> println(s"Namnet '$f $e' har $tot bokstäver.")
5  scala> println(s"Efternamnet '$e' har ${e.size} bokstäver.")

```

- a) Vad skrivs ut ovan?
- b) Skapa följande utskrifter med hjälp av stränginterpolatorn `s` och lämpliga variabler.

```

1  Namnet 'Kim' har 3 bokstäver.
2  Namnet 'Finkodare' har 9 bokstäver.

```

Uppgift 18. **if-sats.** För varje rad nedan; förklara vad som händer.

```

1 scala> if (true) println("sant") else println("falskt")
2 scala> if (false) println("sant") else println("falskt")
3 scala> if (!true) println("sant") else println("falskt")
4 scala> if (!false) println("sant") else println("falskt")
5 scala> def singlaSlant =
6   scala>   if (math.random > 0.5) print(" krona") else print(" klave")
7   scala> singlaSlant; singlaSlant; singlaSlant

```

Uppgift 19. **if-uttryck.** Deklarera följande variabler med nedan initialvärden:

```

scala> var grönsak = "gurka"
scala> var frukt = "banan"

```

Vad har följande uttryck för värden och typ?

- a) **if** (grönsak == "tomat") "gott" **else** "inte gott"
- b) **if** (frukt == "banan") "gott" **else** "inte gott"
- c) **if** (frukt.size == grönsak.size) "lika stora" **else** "olika stora"
- d) **if** (**true**) grönsak **else** frukt
- e) **if** (**false**) grönsak **else** frukt

Uppgift 20. **for-sats.** Med bakåtpilen <- kan man i en **for**-sats ange vilka värden som ska gås igenom i sekvensen. Vid varje runda i loopen får en lokal variabel ett nytt värde i sekvensen.

- a) Vad ger nedan **for**-satser för utskrift?

```

1 scala> for (i <- 1 to 10) print(i + ", ")
2 scala> for (i <- 1 until 10) print(i + ", ")
3 scala> for (i <- 1 to 5) print((i * 2) + ", ")
4 scala> for (i <- 1 to 92 by 10) print(i + ", ")
5 scala> for (i <- 10 to 1 by -1) print(i + ", ")

```

- b) Skriv en **for**-sats som ger följande utskrift:

```
A1, A4, A7, A10, A13, A16, A19, A22, A25, A28, A31, A34, A37, A40, A43,
```

Uppgift 21. Repetition med metoden **foreach**. Efter framåtpilen => (se nedan) anges vad som ska hända för varje element som går igenom sekventiellt. Vid varje runda i loopen får en lokal variabel ett nytt värde i sekvensen.

- a) Vad ger nedan satser för utskrifter?

```

1 scala> (9 to 19).foreach{i => print(i + ", ")}
2 scala> (1 until 20).foreach{i => print(i + ", ")}
3 scala> (0 to 33 by 3).foreach{i => print(i + ", ")}

```

- b) Använd **foreach** och skriv ut följande:

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

Uppgift 22. **while**-sats. En sats eller ett block med satser upprepas så länge ett villkor är sant.

- a) Vad ger nedan satser för utskrifter?

```
1 scala> var i = 0
2 scala> while (i < 10) { println(i); i = i + 1 }
3 scala> var j = 0; while (j <= 10) { println(j); j = j + 2 }; println(j)
```

- b) Skriv en **while**-sats som ger följande utskrift. Använd en variabel k som initialiseras till 1.

```
A1, A4, A7, A10, A13, A16, A19, A22, A25, A28, A31, A34, A37, A40, A43,
```

- c) Vilken av **for**, **while** och **foreach** är kortast att skriva om man vill repetera mer än en sats 100 gånger? Vilken tycker du är lättast att läsa?

Uppgift 23. Slumptal. Undersök vad dokumentationen säger om funktionen `scala.math.random`:

<http://www.scala-lang.org/api/current/#scala.math.package>

- a) Vilken typ har värdet som returneras av funktionen `random`?
 b) Vilket är det minsta respektive största värde som kan returneras?
 c) Är `random` en *äkta* funktion (eng. *pure function*) i matematisk mening?
d) Anropa funktionen `math.random` upprepade gånger och notera vad som händer. Använd pil-upp-tangenten.

```
scala> math.random
```

- e) Vad händer? Använd *pil-upp* och kör nedan **for**-sats flera gånger. Förklara vad som sker.

```
scala> for (i <- 1 to 20) println((math.random * 3 + 1).toInt)
```

- f) Skriv en **for**-sats som skriver ut 100 slumpmässiga heltal från 0 till och med 9 på var sin rad.

```
scala> for (i <- 1 to 100) println(???)
```

- g) Skriv en **for**-sats som skriver ut 100 slumpmässiga heltal från 1 till och med 6 på samma rad.

```
scala> for (i <- 1 to 100) print(???)
```

- h) Använd *pil-upp* och kör nedan **while**-sats flera gånger. Förklara vad som sker.

```
scala> while (math.random > 0.2) println("gurka")
```

- i) Ändra i **while**-satsen ovan så att sannolikheten ökar att riktigt många strängar ska skrivas ut.

- j) Förklara vad som händer nedan.

```
1 scala> var slumptal = math.random  
2 scala> while (slumptal > 0.2) { println(slumptal); slumptal = math.random }
```

Uppgift 24. *Logik och De Morgans Lagar.* Förenkla följande uttryck. Antag att poäng och highscore är heltalsvariabler medan klar är av typen Boolean. 

- a) poäng > 100 && poäng > 1000
- b) poäng > 100 || poäng > 1000
- c) !(poäng > highscore)
- d) !(poäng > 0 && poäng < highscore)
- e) !(poäng < 0 || poäng > highscore)
- f) klar == **true**
- g) klar == **false**

1.1.2 Extrauppgifter

Uppgift 25. Slumptal.

- a) Ersätt ??? nedan med literaler så att tärning returnerar ett slumpmässigt heltal mellan 1 och 6.

```
scala> def tärning = (math.random * ??? + ???).toInt
```

- b) Ersätt ??? med literaler så att rnd blir ett decimaltal med max en decimal mellan 0.0 och 1.0.

```
scala> def rnd = math.round(math.random * ???) / ???
```

- c) Vad blir det för skillnad om `math.round` ersätts med `math.floor` ovan? (Se dokumentationen av `java.lang.Math.round` och `java.lang.Math.floor`.)

Uppgift 26. Undersök vad som finns i paketet `scala.math` genom att studera dess dokumentation: www.scala-lang.org/api/current/#scala.math.package och gör några matematiska beräkningar i REPL som använder olika funktioner i `math`-paketet.

-  **Uppgift 27.** Antag att du byter plats mellan satsen efter villkoret och satsen efter `else` i `if`-satsen nedan. Hur kan du ändra i villkoret så att det ändå skrivas ut samma sak som före bytet?

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

-  **Uppgift 28.** Rita en ny bild av datorns minne efter varje evaluerad rad nedan. Bilderna ska visa variablers namn, typ och värde.

```
1 scala> var x = 42
2 scala> var y = x + 1
3 scala> x += -1
4 scala> y -= 1
```

Uppgift 29. Skapa med hjälp av `while` några olika oändliga loopar som skriver ut olika saker vid varje loop-runda.

Uppgift 30. Hitta på några egna övningar för att träna mer på De Morgans lagar.

1.1.3 Fördjupningsuppgifter

Uppgift 31. Läs om moduläräkning här en.wikipedia.org/wiki/Modulo_operation och undersök hur det blir med olika tecken (positivt resp. negativt) på divisor och dividend.

Uppgift 32. Läs om identifierare i Scala och speciellt *literal identifiers* här: <http://www.artima.com/pins1ed/functional-objects.html#6.10>.

- a) Förklara vad som händer nedan:

```
scala> val `konstig val` = 42
scala> println(`konstig val`)
```

- b) Scala och Java har olika uppsättningar med reserverade ord. På vilket sätt kan "backticks" vara användbart med anledning av detta?

Uppgift 33. Sök upp dokumentationen för `java.lang.Integer`.

- a) Undersök i REPL hur metoderna `toBinaryString` och `toHexString` fungerar.
 b) Vad betyder literalen `0x2a`?

Uppgift 34. Typannoteringar skapas genom att i ett uttryck placera ett kolon följt av en typ, vid behov omslutet av en parentes. Skapa ett större uttryck med typannoteringar och försök få komplatorn att kontrollera typen på intressanta ställen. Märk att typannoteringar också ibland kan användas för att konvertera mellan numeriska typer.

Uppgift 35. Förklara vad som händer nedan:

```
1 scala> var i = 42
2 scala> i += 1
3 scala> i *= 2
4 scala> i /= 3
```

Uppgift 36. Läs om `BigInt` och `BigDecimal` här: alvinalexander.com/scala/how-to-use-large-integer-decimal-numbers-in-scala-bigint-bigdecimal och prova att skapa riktigt stora tal med hjälp av metoden `pow` på `BigInt` och tal med riktigt många decimaler med `BigDecimal` dess metod `pow`.

Uppgift 37. Sök upp dokumentationen för `java.lang.Math.multiplyExact` och läs om vad den metoden gör.

- a) Vad händer här?

```
scala> Math.multiplyExact(2, 42)
scala> Math.multiplyExact(Int.MaxValue, Int.MaxValue)
```

- b) Varför kan man vilja använda `java.lang.Math.multiplyExact` i stället för "vanlig" multiplikation? 

-  c) Sök med Ctrl+F i webbläsaren och efter förekomster av texten ”overflow” i javadoc för klassen `java.lang.Math` i JDK 8. Vad är ”overflow”? Vilka metoder finns i `java.lang.Math` som hjälper dig att upptäcka om det blir overflow?

Uppgift 38. Använda Scala REPL för att undersöka konstanterna nedan. Vilket av dessa värden är negativt? Vad kan man ha för praktisk nytta av dessa värden i ett program som gör flyttalsberäkningar?

- a) `java.lang.Double.MIN_VALUE`
- b) `scala.Double.MinValue`
- c) `scala.Double.MinPositiveValue`

Uppgift 39. För typerna `Byte`, `Short`, `Char`, `Int`, `Long`, `Float`, `Double`: Undersök hur många bitar som behövs för att representera varje typs omfång?

Tips: Några användbara uttryck:

`Integer.toBinaryString(Int.MaxValue + 1).size`
`Integer.toBinaryString((math.pow(2,16) - 1).toInt).size`
`1 + math.log(Long.MaxValue)/math.log(2)` Se även språkspecifikationen för Scala, kapitlet om heltalsliteraler:
<http://www.scala-lang.org/files/archive/spec/2.11/01-lexical-syntax.html#integer-literals>

- a) Undersök källkoden för paketobjektet `scala.math` här:

<https://github.com/scala/scala/blob/v2.11.7/src/library/scala/math/package.scala>

Hur många olika överlagrade varianter av funktionen `abs` finns det och för vilka parametertyper är den definierad?

Uppgift 40. Läs mer om stränginterpolatorer här:

<docs.scala-lang.org/overviews/core/string-interpolation.html>

Hur kan du använda f-interpolatorn för att göra följande utskrift i REPL? Byt ut ??? mot lämpliga tecken.

```
scala> val g: Double = 1 / 3.0
scala> val s: String = f"Gurkan är ??? meter lång"
scala> println(s)
Gurkan är 0.333 meter lång
```

1.2 Laboration: kojo

Mål

- Kunna kombinera principerna sekvens, alternativ, repetition, och abstraktion i skapandet av egna program om minst 20 rader kod.
- Kunna förklara vad ett program gör i termer av sekvens, alternativ, repetition, och abstraktion.
- Kunna tillämpa principerna sekvens, alternativ, repetition, och abstraktion i enkla algoritmer.
- Kunna formatera egna program så att de blir lätt att läsa och förstå.
- Kunna förklara vad en variabel är och kunna skriva deklarationer och göra tilldelningar.
- Kunna genomföra upprepade varv i cykeln *editera-exekvera-felsöka /förbättra* för att successivt bygga upp allt mer utvecklade program.

Förberedelser

- Gör övning expressions i avsnitt 1.1.
- Bläddra igenom ”Kojo - An Introduction” (25 sidor) som du kan ladda ner i pdf här: <http://www.kogics.net/kojo-ebooks>
- Du behöver en dator med Kojo installerad, se appendix D.2.

1.2.1 Obligatoriska uppgifter

Uppgift 1. Sekvens.

- a) Starta Kojo. Om du inte redan har svenska menyer: välj svenska i språkmbyn och starta om Kojo. Skriv in nedan program och tryck på den *gröna* play-knappen. Du hittar en lista med några fler funktioner på svenska och engelska i appendix D.2.

```
sudda
```

```
fram; höger  
fram; vänster
```

- b) Prova att ändra på ordningen mellan satserna och använd den *gula* play-knappen (programspårning) för att studera vad som händer. Klicka på satser i ditt program och på rutor i programspårningen och se vad som händer.

- c) Prova satser i sekvens på flera rader, respektive på samma rad med semikolon emellan. Hur vill du gruppera dina satser så att de är lätt för en mänsk att läsa?

- d) Vad händer om du *inte* börjar programmet med sudda och kör det upprepade gånger? Varför är det bra att börja programmet med sudda? 

- e) Rita en kvadrat som i bilden nedan.



- f) Rita en trappa som i bilden nedan.



- g) *Rita och mät.*

- Börja ditt program med dessa satser:
sudda; axesOn; gridOn; sakta(0); osynlig
- Rita sedan en kvadrat som har 444 längdenheter i omkrets.
- Ta fram linjalen med höger-klick i ritfönstret och mät så exakt du kan hur lång diagonalen i kvadraten är. Skriv ner resultatet.
Tips: Du kan zooma med mushjulet om du håller nere Ctrl-knappen. Du kan flytta linjalen om du klick-drar på linjalems skalstreck. Du kan vrida linjalen om du klickar på skalstrecken och håller nere Shift-tangenten.
- Kontrollera med hjälp av `math.hypot` och `println` vad det exakta svaret är. Skriv ner svaret med 3 decimalers noggrannhet. Du kan t.e.x. använda REPL i ett terminalfönster bredvid, eller öppna ett nytt extra Kojo-fönster i Arkiv-menyn, eller lägga in utskrifterna sist i ditt befintliga program. Utskrifter med `println` i Kojo sker i utdatafönstret.

- h) Rita en liksidig triangel med sidan 300 längdenheter genom att ge lämpliga argument till `fram` och `höger`. Vinklar anges i grader.

- ✓ i) Visa dina resultat för en handledare och diskutera hur uppgifterna ovan illustrerar principen om sekvens.

Uppgift 2. Repetition.

- a) Rita en kvadrat igen, men nu med hjälp av proceduren `upprepa(n){ ??? }` där du ersätter `n` med antalet repetitioner och `???` med de satser som ska repeteras.
- b) Kör ditt program med den *gula* play-knappen. Studera hur repetitionen påverkar exekveringssekvensen. Vid vilka punkter i programmet sker ett "hopp" i sekvensen i stället för att efterföljande sats exekveras? Använd lämpligt argument till `sakta` för att du ska hinna studera exekveringen.
- c) Anropa proceduren `sakta(???)` med lämplig parameter och gör så att sköldpaddan går totalt 20 varv i kvadraten på ungefär 2 sekunder. *Tips:* Du

kan köra ditt program med *Ctrl+Enter* i stället för att trycka på den gröna play-knappen. Anropa sakta i början av ditt program men *efter* sudda. (Vad händer om du anropar sakta före sudda?)

- d) Prova nedan program. Med sakta(0) blir det ingen fördröjning och utritningen sker så snabbt som möjligt. Ungefär hur många kvadratvarv hinner sköldpaddan rita per sekund om utritningen sker utan fördröjning?

```
sudda; sakta(0)
val t1 = System.currentTimeMillis
upprepa(800*4){fram; höger}
val t2 = System.currentTimeMillis
println("Det tog " + (t2 - t1) + " millisekunder")
```

- e) Rita en kvadrat igen, men nu med hjälp av en **while**-sats och en loopvariabel.

```
var i = 0
while (????) {fram; höger; i = ???}
```

- f) Rita en kvadrat igen, men nu med hjälp av en **for**-sats.

```
for (i <- 1 to ???) {????}
```

- g) Rita en kvadrat igen, men nu med hjälp av **foreach**.

```
(1 to ???).foreach{i => ???}
```

- h) Visa dina resultat för en handledare och diskutera hur uppgifterna ovan ✓  illustrerar principen om repetition.

Uppgift 3. Abstraktion.

- a) Använd en repetition för att abstrahera nedan sekvens, så att programmet blir kortare:

```
sudda

fram; höger; hoppa; fram; vänster; hoppa; fram; höger;
hoppa; fram; vänster; hoppa; fram; höger; hoppa; fram;
vänster; hoppa; fram; höger; hoppa; fram; vänster; hoppa;
fram; höger; hoppa; fram; vänster; hoppa
```

- b) Sök på nätet efter ”DRY principle programming” och beskriv med egna  ord vad DRY betyder och varför det är en viktig princip.
 c) Använd proceduren kvadrat nedan och proceduren hoppa(???) för att rita en stapel med 10 kvadrater enligt bilden.

```
def kvadrat = for (i <- 1 to 4) {fram; höger}
```



d) Kör ditt program med den *gula* play-knappen. Studera hur anrop av proceduren `kvadrat` påverkar exekveringssekvensen av dina satsar. Vid vilka punkter i programmet sker ett ”hopp” i sekvensen i stället för att efterföljande sats exekveras? Använd lämpligt argument till `sakta` för att du ska hinna studera exekveringen.

e) Rita samma bild med 10 staplade kvadrater som ovan, men nu *utan* att använda abstraktionen `kvadrat` – använd i stället en nästlad repetition (alltså en upprepning inuti en upprepning). Vilket av de två sätten (med och utan abstraktionen `kvadrat`) är lättast att lösa?

Tips: Varje gång du trycker på någon av play-knapparna, sparas ditt program. Du kan se dina sparade program om du klickar på *Historik*-fliken. Du kan också stega bakåt och framåt i historiken med de blå pilarna bredvid play-knapparna.

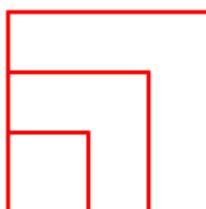
f) Skapa en abstraktion `def` `stapel` = ??? med din kod för att rita en stapel.

g) Du ska nu generalisera din procedur så att den inte bara kan rita exakt 10 kvadrater i en stapel. Ge proceduren `stapel` en parameter `n` som styr hur många kvadrater som ritas.

```
def kvadrat = ???
def stapel(n: Int) = ???

sudda; sakta(100)
stapel(42)
```

h) Ge abstraktionen `kvadrat` en parameter `sida`: Double som anger hur stor kvadraten blir. Rita flera kvadrater i likhet med bilden nedan.



i) Rita nedan bild med hjälp av abstraktionen `stapel`. Det är totalt 100 kvadrater och varje kvadrat har sidan 25. *Tips:* Med ett negativt argument till procedur `hoppa` kan du få sköldpaddan att hoppa baklänges utan att rita, t.ex. `hoppa(-10*25)`



- j) Skapa en abstraktion rutnät med lämpliga parametrar som gör att man kan rita rutnät med olika stora kvadrater och olika många kvadrater i både x- och y-led.
- k) Se över ditt program i föregående uppgift och säkerställ att det är lättläst ✓ och följer en struktur som börjar med alla definitioner i logisk ordning och därefter fortsätter med huvudprogrammet. Diskutera ditt program med en handledare. Vad har du gjort för att programmet ska vara lättläst?

Uppgift 4. Variabel.

- a) Skriv in nedan program *exakt* som det står med blanktecken, indragningar och radbrytningar. Kör programmet och förklara vad som händer.

```
def gurka(x: Double,
           y: Double, namn: String,
           typ: String,
           värde:String) = {
  val bredd = 15
  val h = 30
  hoppaTill(x,y)
  norr
  skriv(namn+": "+typ)
  hoppaTill(x+bredd*(namn.size+typ.size),y)
  skriv(värde); söder; fram(h); vänster
  fram(bredd * värde.size); vänster
  fram(h); vänster
  fram(bredd * värde.size); vänster
}

sudda; färg(svart)
val s = 130
val h = 40
var x = 42; gurka(10, s-h*0, "x","Int", x.toString)
var y = x; gurka(10, s-h*1, "y","Int", y.toString)
x = x + 1; gurka(10, s-h*2, "x","Int", x.toString)
          gurka(10, s-h*3, "y","Int", y.toString)
osynlig
```

- b) Skriv ner namnet på alla variabler som förekommer i programmet ovan.

- c) Vilka av dessa variabler är lokala?
- d) Vilka av dessa variabler kan förändras efter initialisering?
- e) Föreslå tre förändringar av programmet ovan (till exempel namnbyten) som gör att det blir lättare att läsa och förstå.
- f) Gör sök-ersätt av gurka till ett bättre namn. *Tips:* undersök kontextmenyn i editorn i Kojo genom att högerklicka i editorfönstret. Notera kortkommandot för Sök/Ersätt.
- g) Gör automatisk formatering av koden med hjälp av lämpligt kortkommando. Notera skillnaderna. Vilka autoformateringar gör programmet lättare att läsa? Vilka manuella formateringar tycker du bör göras för att öka läsbarheten? Diskutera läsbarheten med en handledare.

Uppgift 5. Alternativ.

- a) Kör programmet nedan. Förklara vad som händer. Använd den gula play-knappen för att studera exekveringen.

```
sudda; sakta(5000)

def move(key: Int): Unit = {
    println("key: " + key)
    if (key == 87) fram(10)
    else if (key == 83) fram(-10)
}

move(87); move('W'); move('W')
move(83); move('S'); move('S');
```

- b) Kör programmet nedan. Notera activateCanvas för att du ska slippa klicka i ritfönstret innan du kan styra paddan. Lägg till kod i move som gör att tangenten A ger en vridning moturs med 5 grader medan tangenten D ger en vridning medurs 5 grader.

```
sudda; sakta(0); activateCanvas

def move(key: Int): Unit = {
    println("key: " + key)
    if (key == 'W') fram(10)
    else if (key == 'S') fram(-10)
}

onKeyPress(move)
```

- c) Bygg vidare på programmet i deluppgift b och lägg till nedan kod i början av programmet. Lägg även till kod som gör så att om man trycker på tangenten G så sätts rutnätet omväxlande på och av.

```
var isGridOn = false
```

```
def toggleGrid =
  if (isGridOn) {
    gridOff
    isGridOn = false
  } else {
    gridOn
    isGridOn = true
  }
```

- d) Visa din lösning för en handledare och förklara vad som händer.



1.2.2 Frivilliga extrauppgifter

Uppgift 6. Gör så att när man trycker på tangenten X så sätter man omväxlande på och av koordinataxlarna. Använd en variabel `isAxesOn` och definiera en abstraktion `toggleAxes` som anropar `axesOn` och `axesOff` på liknande sätt som i föregående uppgift.

Uppgift 7. *Tidmätning.* Hur snabb är din dator?

- a) Skriv in koden nedan i Kojos editor och kör upprepade gånger med den gröna play-knappen. Tar det lika lång tid varje gång? Varför?

```
object timer {
  def now: Long = System.currentTimeMillis
  var saved: Long = now
  def elapsedMillis: Long = now - saved
  def elapsedSeconds: Double = elapsedMillis / 1000.0
  def reset: Unit = { saved = now }
}

// HUVUDPROGRAM:
timer.reset
var i = 0L
while (i < 1e8.toLong) { i += 1 }
val t = timer.elapsedSeconds
println("Räknade till " + i + " på " + t + " sekunder.")
```

- b) Ändra i loopen i uppgift a) så att den räknar till 4.4 miljarder. Hur lång tid tar det för din dator att räkna så långt?¹
- c) Om du kör på en Linux-maskin: Kör nedan Linux-kommando upprepade gånger i ett terminalfönster. Med hur många MHz kör din dators klocka för tillfället? Hur förhåller sig klockfrekvensen till antalet runder i while-loopen i

¹Det går att göra ungefär en heltalsaddition per klockcykel per kärna. Den första elektro-niska datorn [Eniac](#) hade en klockfrekvens motsvarande 5kHz. Den dator på vilken denna övningsuppgift skapades hade en i7-4790K turboklockad upp till 4.4 GHz.

föregående uppgift? (Det kan hända att din dator kan variera centralprocessors klockfrekvens. Prova både medan du kör tidmätningen i Kojo och då din dator ”vilar”. Vad är det för poäng med att en processor kan variera sin klockfrekvens?)

```
> lscpu | grep MHz
```

- d) Ändra i koden i uppgift a) så att **while**-loopen bara kör 5 gånger. Kör programmet med den *gula* play-knappen. Scrolla i programspårningen och förklara vad som händer. Klicka på CALL-rutorna och se vilken rad som markeras i ditt program.
- e) Lägg till koden nedan i ditt program och försök ta reda på ungefär hur långt din dator hinner räkna till på en sekund för Long- respektive Int-variabler. Använd den gröna play-knappen.

```
def timeLong(n: Long): Double = {
    timer.reset
    var i = 0L
    while (i < n) { i += 1 }
    timer.elapsedSeconds
}

def timeInt(n: Int): Double = {
    timer.reset
    var i = 0
    while (i < n) { i += 1 }
    timer.elapsedSeconds
}

def show(msg: String, sec: Double): Unit = {
    print(msg + ": ")
    println(sec + " seconds")
}

def report(n: Long): Unit = {
    show("Long " + n, timeLong(n))
    if (n <= Int.MaxValue) show("Int " + n, timeInt(n.toInt))
}

// HUVUDPROGRAM, mätningar:

report(Int.MaxValue)

for (i <- 1 to 10) {
    report(4.26e9.toLong)
}
```

- f) Hur mycket snabbare går det att räkna med Int-variabler jämfört med ✓  Long-variabler? Visa svaret för en handledare.

Uppgift 8. Lek med färg i Kojo. Sök på internet efter dokumentationen för klassen java.awt.Color och studera vilka heltalsparametrar den sista konstruktorn i listan med konstruktorer tar för att skapa sRGB-färger. Om du högerklickar i editorn i Kojo och väljer ”Välj färg...” får du fram färgväljaren och med den kan du välja fördefinierade färger eller blanda egna färger. När du har valt färg får du se vilka parametrar till java.awt.Color som skapar färgen.

```
1 scala> val c = new java.awt.Color(124,10,78,100)
2 c: java.awt.Color = java.awt.Color[r=124,g=10,b=78]
3
4 scala> c. // tryck på TAB
5 asInstanceOf   getColorComponents    getRGBComponents
6 brighter       getColorSpace        getRed
7 createContext  getComponents      getTransparency
8 darker        getGreen           isInstanceOf
9 getAlpha       getRGB            toString
10 getBlue        getRGBColorComponents
11
12 scala> c.getAlpha
13 res3: Int = 100
```

Skriv ett program som ritar många figurer med olika färger, till exempel cirklar som nedan. Om du använder alfakanalen blir färgerna genomskinliga.



Uppgift 9. Ladda ner dessa pdf-kompendier och gör några uppgifter som du tycker verkar intressanta:

- a) "Uppdrag med Kojo" som kan laddas ner här: lth.se/programmera/uppdrag
- b) "Programming Fundamentals with Kojo" som kan laddas ner här: wiki.kogics.net/kojo-codeactive-books

Uppgift 10. Om du vill jobba med att hjälpa skolbarn att lära sig programmera med Kojo, kontakta <http://www.vattenhallen.lth.se> och anmäl ditt intresse att vara handledare.

Kapitel 2

Kodstrukturer

Begrepp som ingår i denna veckas studier:

- iterering
- for-uttryck
- map
- foreach
- Range
- Array
- Vector
- algoritm vs implementation
- pseudokod
- algoritm: SWAP
- algoritm: SUM
- algoritm: MIN/MAX
- algoritm: MININDEX
- block
- namnsynlighet
- namnöverskuggning
- lokala variabler
- paket
- import
- filstruktur
- jar
- dokumentation
- programlayout
- JDK
- main i Java vs Scala
- java.lang.System.out.println

2.1 Övning: programs

Mål

- Kunna skapa samlingarna Range, Array och Vector med heltals- och strängvärden.
- Kunna indexera i en indexerbar samling, t.ex. Array och Vector.
- Kunna anropa operationerna size, mkString, sum, min, max på samlingar som innehåller heltal.
- Känna till grundläggande skillnader och likheter mellan samlingarna Range, Array och Vector.
- Förstå skillnaden mellan en for-sats och ett for-uttryck.
- Kunna skapa samlingar med heltalsvärdet som resultat av enkla for-uttryck.
- Förstå skillnaden mellan en algoritm i pseudo-kod och dess implementering.
- Kunna implementera algoritmerna SUM, MIN/MAX på en indexerbar samling med en **while**-sats.
- Kunna köra igång enkel Scala-kod i REPL, som skript och som applikation.
- Kunna skriva och köra igång ett enkelt Java-program.
- Känna till några grundläggande syntaxskillnader mellan Scala och Java, speciellt variabeldeklarationer och indexering i Array.
- Förstå vad ett block och en lokal variabel är.
- Förstå hur nästlade block påverkar namnsynlighet och namnöverskuggning.
- Förstå kopplingen mellan paketstruktur och kodfilstruktur.
- Kunna skapa en jar-fil.
- Kunna skapa dokumentation med scaladoc.

Förberedelser

- Studera begreppen i kapitel 2.
- Bekanta dig med grundläggande terminalkommandon, se appendix A.
- Bekanta dig med den editor du vill använda, se appendix B.

2.1.1 Grunduppgifter

Uppgift 1. *Datastrukturen Range.* Evaluera nedan uttryck i Scala REPL. Vad har respektive uttryck för värde och typ?

- a) Range(1, 10)
- b) Range(1, 10).inclusive
- c) Range(0, 50, 5)
- d) Range(0, 50, 5).size
- e) Range(0, 50, 5).inclusive

- f) Range(0, 50, 5).inclusive.size
- g) 0.until(10)
- h) 0 until (10)
- i) 0 until 10
- j) 0.to(10)
- k) 0 to 10
- l) 0.until(50).by(5)
- m) 0 to 50 by 5
- n) (0 to 50 by 5).size
- o) (1 to 1000).sum

Uppgift 2. *Datastrukturen Array.* Kör nedan kodrader i Scala REPL. Beskriv vad som händer.

- a) **val** xs = Array("hej", "på", "dej", "!")
- b) xs(0)
- c) xs(3)
- d) xs(4)
- e) xs(1) + " " + xs(2)
- f) xs.mkString
- g) xs.mkString(" ")
- h) xs.mkString("(, , , ,)")
- i) xs.mkString("Array(, , , ,)")
- j) xs(0) = 42
- k) xs(0) = "42"; println(xs(0))
- l) **val** ys = Array(42, 7, 3, 8)
- m) ys.sum
- n) ys.min
- o) ys.max
- p) **val** zs = Array.fill(10)(42)
- q) zs.sum

 r) Datastrukturen Range håller reda på start- och slutvärde, samt stegstorleken för en uppräkning, men alla talen i uppräkningen genereras inte förrän så behövs. En Int tar 4 bytes i minnet. Ungefär hur mycket plats i minnet tar de objekt som variablerna r respektive a refererar till nedan?

```
1 scala> val r = (1 to Int.MaxValue by 2)
2 scala> val a = r.toArray
```

Tips: Använd uttrycket `BigInt(Int.MaxValue) * 2` i dina beräkningar.

Uppgift 3. *Datastrukturen Vector.* Kör nedan kodrader i Scala REPL. Beskriv vad som händer.

- a) `val words = Vector("hej", "på", "dej", "!")`
- b) `words(0)`
- c) `words(3)`
- d) `words.mkString`
- e) `words.mkString(" ")`
- f) `words.mkString("(, , ,)")`
- g) `words.mkString("Ord(, , , ,)")`
- h) `words(0) = "42"`
- i) `val numbers = Vector(42, 7, 3, 8)`
- j) `numbers.sum`
- k) `numbers.min`
- l) `numbers.max`
- m) `val moreNumbers = Vector.fill(10000)(42)`
- n) `moreNumbers.sum`
- o) Jämför med uppgift 2. Vad kan man göra med en Array som man inte kan  göra med en Vector?

Uppgift 4. *for*-uttryck. Evaluera nedan uttryck i Scala REPL. Vad har respektive uttryck för värde och typ?

- a) `for (i <- Range(1,10)) yield i`
- b) `for (i <- 1 until 10) yield i`
- c) `for (i <- 1 until 10) yield i + 1`
- d) `for (i <- Range(1,10).inclusive) yield i`
- e) `for (i <- 1 to 10) yield i`
- f) `for (i <- 1 to 10) yield i + 1`
- g) `(for (i <- 1 to 10) yield i + 1).sum`
- h) `for (x <- 0.0 to 2 * math.Pi by math.Pi/4) yield math.sin(x)`

Uppgift 5. Metoden *map* på en samling. Evaluera nedan uttryck i Scala REPL. Vad har respektive uttryck för värde och typ?

- a) `Range(0,10).map(i => i + 1)`
- b) `(0 until 10).map(i => i + 1)`
- c) `(1 to 10).map(i => i * 2)`
- d) `(1 to 10).map(_ * 2)`
- e) `Vector.fill(10000)(42).map(_ + 43)`

Uppgift 6. Metoden *foreach* på en samling. Kör nedan satser i Scala REPL. Vad händer?

- a) `Range(0,10).foreach(i => println(i))`
- b) `(0 until 10).foreach(i => println(i))`

- c) `(1 to 10).foreach{i => print("hej"); println(i * 2)}`
- d) `(1 to 10).foreach(println)`
- e) `Vector.fill(10000)(math.random).foreach(r =>`
`if (r > 0.99) print("pling!"))`

Uppgift 7. Algoritm: SWAP.

- a) Skriv med *pseudo-kod* algoritmen SWAP. Beskriv på vanlig svenska, steg för steg, hur en variabel *temp* används för mellanlagring vid värdbytet:

Indata: två heltalsvariabler *x* och *y*

???

Utdata: variablerna *x* och *y* vars värden har bytt plats.

- b) Implementerar algoritmen SWAP. Ersätt ??? nedan med satser separerade av semikolon:

```

1 scala> var (x, y) = (42, 43)
2 scala> ???
3 scala> println("x är " + x + ", y är " + y)
4 x är 43, y är 42

```

Uppgift 8. Skript. Skapa en fil med namn hello-script.scala med hjälp av en editor som innehåller denna enda rad:

```
println("hej skript")
```

Spara filen och kör kommandot `scala hello-script.scala` i terminalen:

```
> scala hello-script.scala
```

- a) Vad händer?
- b) Ändra i filen så att högerparentesen saknas. Spara och kör skriptfilen igen. Vad händer?
- c) Lägg till en sats sist i skriptet som skriver ut summan av de ett tusen stycken heltalen från och med 2 till och med 1001, så som visas nedan.

```

1 > scala hello-script.scala
2 hej skript
3 501500

```

- d) Ändra i hello-script.scala genom att införa **val** *n* = `args(0).toInt` och använd *n* som övre gräns för summeringen av de *n* första heltalen.

```

1 > scala hello-script.scala 5001
2 hej skript
3 12507501

```

- e) Vad blir det för felmeddelande om du glömmer ge programmet ett argument?

Uppgift 9. *Applikation med main-metod.* Skapa med hjälp av en editor en fil med namn hello-app.scala.

```
> gedit hello-app.scala
```

Skriv dessa rader i filen:

```
object Hello {
    def main(args: Array[String]): Unit = {
        println("Hej scala-app!")
    }
}
```

- a) Kompilera med scalac hello-app.scala och kör koden med scala Hello.

```
> scalac hello-app.scala
> ls
> scala Hello
```

Vad heter filerna som kompilatorn skapar?

- b) Ändra i din kod så att kompilatorn ger följande felmeddelande:

Missing closing brace

- c) Varför behövs main-metoden?



- d) Vilket alternativ går snabbast att köra igång, ett skript eller en kompilerad applikation? Varför? Vilket alternativ kör snabbast när väl exekveringen är igång?



Uppgift 10. *Java-applikation.* Skapa med hjälp av en editor en fil med namn Hi.java.

```
> gedit Hi.java
```

Skriv dessa rader i filen:

```
public class Hi {
    public static void main(String[] args) {
        System.out.println("Hej Java-app!");
    }
}
```

Kompilera med javac Hi.java och kör koden med java Hi.

```
> javac Hi.java
> ls
> java Hi
```

- a) Vad heter filen som kompilatorn skapat?



- b) Jämför signaturen för Java-programmets main-metod med signaturen för Scala-programmets main-metod. De betyder samma sak men syntaxen är olika. Beskriv skillnader och likheter i syntaxen.



-  c) Vad blir det för felmeddelande om källkodsfilen och klassnamnet inte överensstämmer i ett Java-program?

Uppgift 11. *Algoritm: SUMBUG.* Nedan återfinns pseudo-koden för SUMBUG.

Indata : heltalet n
Resultat: utskrift av summan av de första n heltalen

```

1 sum ← 0
2 i ← 1
3 while  $i \leq n$  do
4   | sum ← sum + 1
5 end
6 skriv ut sum
```

-  a) Kör algoritmen steg för steg med penna och papper, där du skriver upp hur värdena för respektive variabel ändras. Det finns två buggar i algoritmen. Vilka? Rätta buggarna och test igen genom att ”köra” algoritmen med penna på papper och kontrollera så att algoritmen fungerar för $n = 0$, $n = 1$, och $n = 5$. Vad händer om $n = -1$?
b) Skapa med hjälp av en editor filen `sumn.scala`. Implementera algoritmen SUM enligt den rättade pseudokoden och placera implementationen i en main-metod i ett objekt med namnet `sumn`. Du kan skapa indata `n` till algoritmen med denna deklaration i början av din main-metod:

`val n = args(0).toInt`

Vad ger applikationen för utskrift om du kör den med argumentet 8888?

```
> scalac sumn.scala
> scala sumn 8888
```

- c) Kontrollera att din implementation räknar rätt genom att jämföra svaret med detta uttrycks värde, evaluerat i Scala REPL:

```
scala> (1 to 8888).sum
```

- d) Implementera algoritmen SUM enligt pseudokoden ovan, men nu i Java. Skapa filen `SumN.java` och använd koden från uppgift 10 som mall för att deklarera den publika klassen `SumN` med en main-metod. Några tips om Java-syntax och standarfunktioner i Java:

- Alla satser i Java måste avslutas med semikolon.
- Heltalsvariabler deklareras med nyckelordet `int` (litet i).
- Typnamnet ska stå *före* namnet på variabeln. Exempel:
`int sum = 0;`
- Indexering i en array görs i Java med hakparenteser: `args[0]`
- I stället för Scala-uttrycket `args(0).toInt`, använd Java-uttrycket:
`Integer.parseInt(args[0])`
- `while`-satser i Scala och Java har samma syntax.
- Utskrift i Java görs med `System.out.println`

Uppgift 12. *Algoritm: MAXBUG.* Nedan återfinns pseudo-koden för MAXBUG.

Indata : Array *args* med strängar som alla innehåller heltal

Resultat: utskrift av största heltalet

```

1 max ← det minsta heltalet som kan uppkomma
2 n ← antalet heltalet
3 i ← 0
4 while i < n do
5   | x ← args(i).toInt
6   | if (x > max) then
7   |   | max ← x
8   | end
9 end
10 skriv ut max

```

- a) Kör med penna och papper. Det finns en bugg i algoritmen ovan. Vilken? 
Rätta buggen.

- b) Implementera algoritmen MAX (utan bugg) som en Scala-applikation.

Tips:

- Det minsta Int-värdet som någonsin kan uppkomma: Int.MinValue
- Antalet element i *args* ges av: *args.size*

```

1 > gedit maxn.scala
2 > scalac maxn.scala
3 > scala maxn 7 42 1 -5 9
4 42

```

- c) Skriv om algoritmen så att variabeln *max* initialiseras med det första talet i sekvensen. 

- d) Implementera den nya algoritmvarianten från uppgift c och prova programmet. Vad händer om *args* är tom?

Uppgift 13. *Block, namnsynlighet, namnöverskuggning.* Kör nedan kod i Scala REPL eller i Kojo. Vad händer nedan? Varför?

- `val a = {1 + 1; 2 + 2; 3 + 3; 4 + 4}; println(a)`
- `val b = {1; 2; 3; {val b = 4; b + b; b + 1}}; println(b)`
- `{val a = 42; println(a)}`
- `{val a = 42}; println(a)`
- `{val a = 42; {val a = 43; println(a)}; println(a)}`
- `{var a = 42; {a = a + 1}; var a = 43}`
- `{var a = 42; {a = a + b; var b = 43}; println(a)}`
- `{var a = 42; {var b = 43; a = a + b}; println(a)}`
- `{var a = 42; {a = a + b; def b = 43}; println(a)}`
- `{object a{var b=42;object a{var a=43}};println(a.b+a.a.a)}`
- k)

```
{
  object a {
    var b = 42
    object a {
      var a = 43
    }
  }
  println(a.b + a.a.a)
}
```

- l) Vad är fördelen med att namn deklarerade inne i ett block är lokala i stället för globala?

Uppgift 14. *Paket, import och klassfilstrukturer.* Med Java-8-plattformen kommer 4240 färdiga klasser, som är organiserade i 217 olika paket.¹

- a) Vilka paket finns i paketet javax som börjar på s?

```
scala> javax.s //tryck på TAB-tangenten
```

- b) Kör raderna nedan i REPL. Beskriv vad som händer för varje rad.

```
1 scala> import javax.swing.JOptionPane
2 scala> def msg(s: String) = JOptionPane.showMessageDialog(null, s)
3 scala> msg("Hej på dej!")
4 scala> def input(msg: String) = JOptionPane.showInputDialog(null, msg)
5 scala> input("Vad heter du?")
6 scala> import JOptionPane.{showOptionDialog => optDlg}
7 scala> def inputOption(msg: String, opt: Array[Object]) =
8   optDlg(null, msg, "Option", 0, 0, null, opt, opt(0))
9 scala> inputOption("Vad väljer du?", Array("Sten", "Sax", "Påse"))
```

- c) Vad hade du behövt ändra på efterföljande rader om import-satsen på rad 1 ovan ej hade gjorts?
d) Skapa med en editor filen paket.scala och kompilera. Rita en bild av hur katalogstrukturen ser ut.

```
package gurka.tomat.banan

package p1 {
  package p11 {
    object hello {
      def hello = println("Hej paket p1.p11!")
    }
  }
  package p12 {
    object hello {
      def hello = println("Hej paket p1.p12!")
    }
}
```

¹Se Stackoverflow: [how-many-classes-are-there-in-javas-standard-edition](https://stackoverflow.com/questions/1151034/how-many-classes-are-there-in-javas-standard-edition)

```

        }
    }
}

package p2 {
    package p21 {
        object hello {
            def hello = println("Hej paket p2.p21!")
        }
    }
}

object Main {
    def main(args: Array[String]): Unit = {
        import p1._

        p11.hello.hello
        p12.hello.hello
        import p2.{p21 => apelsin}
        apelsin.hello.hello
    }
}

```

```

1 > gedit paket.scala
2 > scalac paket.scala
3 > scala gurka.tomat.banan.Main
4 > ls -R

```

Uppgift 15. Skapa jar-filer och använda classpath

- Skriv kommandot jar i terminalen och undersök vad som finns för optioner. Se speciellt ”Example 1.” i hjälputskriften. Vilket kommando ska du använda för att packa ihop flera filer i en enda jar-fil?
- Som en fortsättning på uppgift 14, packa ihop biblioteket gurka i en jar-fil med nedan kommando, samt kör igång REPL med jar-filen på classpath.

```

1 > jar cvf mittpaket.jar gurka
2 > scala -cp mittpaket.jar
3 scala> gurka.tomat.banan.Main.main(Array())

```

Uppgift 16. Skapa dokumentation med scaladoc-kommandot

- Som en fortsättning på uppgift 14, kör nedan kommando i terminalen:

```

1 > scaladoc paket.scala
2 > ls
3 > firefox index.html # eller öppna index.html i valfri webbläsare

```

Vad händer?

- b) Lägg till några fler metoder i något av objekten i filen paket.scala och lägg även till några dokumentationskommentarer. Kompilera om och kör. Generera om dokumentationen.

```
//... ändra i filen paket.scala

/** min paketdokumentationskommentar p2 */
package p2 {
    /** min paketdokumentationskommentar p21 */
    package p21 {
        /** ett hälsningsobjekt */
        object hello {
            /** en hälsningsmetod i p2.p21 */
            def hello = println("Hej paket p2.p21!")

            /** en metod som skriver ut tiden */
            def date = println(new java.util.Date)
        }
    }
}
```

```
1 > gedit paket.scala
2 > scalac paket.scala
3 > jar cvf mittpaket.jar gurka
4 > scala -cp mittpaket.jar
5 scala> gurka.tomat.banan.p2.p21.hello.date
6 scala> :q
7 > scaladoc paket.scala
8 > firefox index.html
```

2.1.2 Extrauppgifter

Uppgift 17. Implementera algoritmen MININDEX som söker index för minsta heltalet i en sekvens. Pseudokod för algoritmen MININDEX:

Indata : Sekvens xs med n st heltalet.
Utdata : Index för det minsta talet eller -1 om xs är tom.

```

1 minPos  $\leftarrow 0$ 
2  $i \leftarrow 1$ 
3 while  $i < n$  do
4   | if  $xs(i) < xs(minPos)$  then
5     |   |  $minPos \leftarrow i$ 
6     |   end
7     |    $i \leftarrow i + 1$ 
8 end
9 if  $n > 0$  then
10  |   return  $minPos$ 
11 else
12  |   return  $-1$ 
13 end
```

- Prova algoritmen med penna och papper på sekvensen $(1, 2, -1, 4)$ och rita minnessituationen efter varje runda i loopen. Vad blir skillnaden i exekveringsförfloppet om loopvariablen i initialiseras till 0 i stället för 1?
- Implementera algoritmen MININDEX i Scala i en funktion med denna signatur:

```
def indexOfMin(xs: Array[Int]): Int = ???
```

Testa för olika fall: tom sekvens; sekvens med endast ett tal; lång sekvens med det minsta talet först, någonstans mitt i, samt sist.

```
// kod till facit
def indexOfMin(xs: Array[Int]): Int = {
  var minPos = 0
  var i = 1
  while (i < xs.size) {
    if (xs(i) < xs(minPos)) minPos = i
    i += 1
  }
  if (xs.size > 0) minPos else -1
}
```

2.1.3 Fördjupningsuppgifter

Uppgift 18. Läs om krullparenteser och vanliga parenteser på stack overflow: stackoverflow.com/questions/4386127/what-is-the-formal-difference-in-scala-between-braces-and-parentheses-and-when och prova själv i REPL hur du kan blanda dessa olika slags parenteser på olika vis.

Uppgift 19. Gör jämförande studier av Scalas api-dokumentation för ArrayBuffer, Array och Vector. Ge exempel på metoder som finns på objekt av typen Array och ArrayBuffer men inte på objekt av typen Vector. *Tips:* Kolla efter metoder som returnerar Unit. Prova några muterande metoder på Array och ArrayBuffer i REPL.

Uppgift 20. Bygg vidare på koden nedan och gör ett Sten-Sax-Påse-spel² som även meddelar vem som vinner. Koden fungerar att köra som den är, men funktionen winnerMsg är ej klar. *Tips:* Du kan använda modulo-räkning med %-operatorn för att avgöra vem som vinner.

```
object Rock {
    import javax.swing.JOptionPane
    import JOptionPane.{showOptionDialog => optDlg}

    def inputOption(msg: String, opt: Vector[String]) =
        optDlg(null, msg, "Option", 0, 0, null, opt.toArray[Object], opt(0))

    def msg(s: String) = JOptionPane.showMessageDialog(null, s)

    val opt = Vector("Sten", "Sax", "Påse")

    def userChoice = inputOption("Vad väljer du?", opt)

    def computerChoice = (math.random * 3).toInt

    def winnerMsg(user: Int, computer: Int) = "??? vann!"

    def main(args: Array[String]): Unit = {
        var keepPlaying = true
        while (keepPlaying) {
            val u = userChoice
            val c = computerChoice
            msg("Du valde " + opt(u) + "\n" +
                "Datorn valde " + opt(c) + "\n" +
                winnerMsg(u, c))
            if (u != c) keepPlaying = false
        }
    }
}
```

²sv.wikipedia.org/wiki/Sten,_sax,_p%C3%A5se

Kapitel 3

Funktioner, objekt

Begrepp som ingår i denna veckas studier:

- definera funktion
- anropa funktion
- parameter
- returtyp
- värdeandrop
- namnanrop
- default-argument
- namngivna argument
- applicera funktion på alla element i en samling
- procedur
- värdeanrop vs namnanrop
- uppdelad parameterlista
- skapa egen kontrollstruktur
- objekt
- modul
- punktnotation
- tillstånd
- metod
- medlem
- funktionsvärde
- funktionstyp
- äkta funktion
- stegad funktion
- apply
- lazy val
- lokala funktioner
- anonyma funktioner
- lambda
- aktiveringspost
- anropsstacken
- objektheopen
- rekursion cslib.window.SimpleWindow

3.1 Övning: functions

Mål

- Kunna skapa och använda funktioner med en eller flera parametrar, default-argument, namngivna argument, och uppdelad parameterlista.
- Kunna använda funktioner som äkta värden.
- Kunna skapa och använda anonyma funktioner (s.k. lambda-funktioner).
- Kunna applicera en funktion på element i en samling.
- Förstå skillnader och likheter mellan en funktion och en procedur.
- Förstå skillnader och likheter mellan en värde-anrop och namnanrop.
- Kunna skapa en procedur i form av en enkel kontrollstruktur med fördröjd evaluering av ett block.
- Kunna skapa och använda objekt som moduler.
- Förstå skillnaden mellan äkta funktioner och funktioner med sidoeffekter.
- Kunna skapa och använda variabler med fördröjd initialisering och förstå när de är användbara.
- Kunna förklara hur nästlade funktionsanrop fungerar med hjälp av begreppet aktiveringspost.
- Kunna skapa och använda lokala funktioner, samt förstå nyttan med lokala funktioner.
- Känna till att funktioner är objekt med en apply-metod.
- Känna till stegade funktioner och kunna använda partiellt applicerade argument.
- Känna till rekursion och kunna förklara hur rekursiva funktioner fungerar.

Förberedelser

- Studera begreppen i kapitel 3.

3.1.1 Grunduppgifter

Uppgift 1. *Definiera och anropa funktioner.* En funktion med två parametrar definieras med följande syntax i Scala:

def namn(parameter1: Typ1, parameter2: Typ2): Returtyp = returvärde

- a) Definiera en funktion med namnet öka som har en heltalsparameter x och som returnerar x + 1. Ange returtypen explicit. Testa funktionen i REPL med argumentet 42.

```
1 scala> ??? // definiera funktionen öka
2 scala> öka(42)
3 43
```

- b) Vad har funktionen öka i föregående uppgift för returtyp?



- c) Vad gör kompilatorn om du utelämnar returtypen?
- d) Varför kan det vara bra att ange returtypen explicit?
- e) Vad är det för skillnad mellan parameter och argument?
- f) Vad har uttrycket `öka(öka(öka(öka(42))))` för värde?
- g) Definiera funktionen `minska(x: Int): Int` med returnvärdet `x - 1`.
- h) Vad är värdet av uttrycket `öka(minska(öka(minska(öka(minska(42))))))`

Uppgift 2. Funktion med flera parametrar. Definiera i REPL två funktioner `sum` och `diff` med två heltalsparametrar som returnerar summan respektive differensen av argumenten:

```
def sum(x: Int, y: Int): Int = x + y
def diff(x: Int, y: Int): Int = x - y
```

Vad har nedan uttryck för värden? Föklara vad som händer.

- a) `diff(0, 100)`
- b) `diff(100, sum(42, 43))`
- c) `sum(sum(42, 43), diff(100, sum(0, 0)))`
- d) `sum(diff(Byte.MaxValue, Byte.MinValue), 1)`

Uppgift 3. Funktion med default-argument. Föklara vad som händer här?

```
1 scala> def inc(i: Int, j: Int = 1) = i + j
2 scala> inc(42, 2)
3 scala> inc(42, 1)
4 scala> inc(42)
```

Uppgift 4. Funktionsanrop med namngivna argument.

```
1 scala> def skrivNamn(förnamn: String, efternamn: String) =
2           println("Namn: " + efternamn + ", " + förnamn)
3 scala> skrivNamn("Kim", "Finkodare")
4 scala> skrivNamn(förnamn = "Viktor", efternamn = "Oval")
5 scala> skrivNamn(erternamn = "Triangelsson", förnamn = "Stina")
```

- a) Föklara vad som händer ovan?
- b) Vad är fördelen med namngivna argument?

Uppgift 5. Applicera en funktion på elementen i en samling. Använd dina funktioner `öka` och `minska` från uppgift 1. Vad har nedan uttryck för värde? Föklara vad som händer.

- a) `for (i <- 0 to 4) yield öka(i)`
- b) `for (i <- 1 to 5) yield minska(i)`
- c) `(0 to 4).map(i => öka(i))`
- d) `(1 to 5).map(i => minska(i))`
- e) `(0 to 4).map(öka)`
- f) `(1 to 5).map(minska)`

- g) `Vector(12, 3, 41, -8).map(öka)`
- h) `Vector(12, 3, 41, -8).map(öka).map(minska).map(minska)`

Uppgift 6. En funktion som inte returnerar något intressant värde, men som anropas för det den gör kallas **procedur**. Definiera följande procedur i REPL:

```
def tUvirks(msg: String) = println(msg.reverse)
```

Vad skriver nedan satser ut? Förklara vad som händer.

- a) `println("sallad".reverse)`
- b) `tUvirks("sallad")`
- c) `val x = tUvirks("sallad"); println(x)`
- d) `def enhetsvärdet = (); println(enhetvärdet)`
- e) `def bortkastad: Unit = 1 + 1; println(bortkastad)`
- f) `def bortkastad2 = {val x = 1 + 1}; println(bortkastad2)`
- g) Varför är det bra att explicit ange `Unit` som returtyp för procedurer?



Uppgift 7. Värdeanrop och namnanrop (fördröjd evaluering, "lata" argument). Deklarera nedan funktioner i REPL eller Kojo.

```
def snark: Int = {print("snark "); Thread.sleep(1000); 42}
def callByValue(x: Int) = x + x
def callByName(x: => Int) = x + x
```

Evaluera nedan uttryck. Förklara vad som händer.

- a) `snark`
- b) `snark; snark; snark`
- c) `callByValue(1)`
- d) `callByName(1)`
- e) `callByValue(snark)`
- f) `callByName(snark)`
- g) Förklara vad som händer här:

```
1 scala> def görDetta(block: => Unit) = block
2 scala> görDetta(println("hej"))
3 scala> görDetta(println("goddag"))
4 scala> görDetta(println("hej"); println("svejs"))
5 scala> def görDettaTvåGånger(block: => Unit) = {block; block}
6 scala> görDettaTvåGånger(println("goddag"))
```

Uppgift 8. Uppdelad parameterlista. Man kan dela upp parametrarna till en funktion i flera parameterlistor. Förklara vad som händer här:

```
1 scala> def add(a: Int)(b: Int) = a + b
2 scala> add(22)(20)
3 scala> add(22)(add(1)(19))
```

Uppgift 9. Skapa din egen kontrollstruktur.

- a) Använd födröjd evaluering i kombination med en uppdelad parameterlista och skapa din egen kontrollstruktur enligt nedan. (Det är så här som loopen upprepa i Kojo är definierad.)

```
1 scala> def upprepa(n: Int)(block: => Unit): Unit = {
2     var i = 0
3     while (i < n) {block; i = i + 1}
4 }
```

- b) Använd din nya loop-procedur och förklara vad som händer nedan.

```
1 scala> upprepa(10)(println("hej"))
2 scala> upprepa(1000){
3     val tärning = (math.random * 6 + 1).toInt
4     print(tärning + " ")
5 }
```

Uppgift 10. Funktion som värde. Funktioner är äkta värden i Scala.

- a) Förklara vad som händer nedan. Notera understrecket på rad 4:

```
1 scala> def inc(x: Int): Int = x + 1
2 scala> inc(42)
3 scala> Vector(12, 3, 41, -8).map(inc)
4 scala> val f = inc _
5 scala> Vector(12, 3, 41, -8).map(f)
```

- b) Vad händer om du bara skriver **val** *f* = *inc* utan understreck?
 c) På liknande sätt som i uppgift a: definiera en funktion dec som i stället *minskar* med 1. Deklarera ett funktionsvärdet g som tilldelas funktionen dec och kör sedan g på varje element i Vector(12, 3, 41, -8) med metoden map.
 d) Vad har variablerna *f* och *g* ovan för typ?
 e) Förklara vad som händer nedan. Vad får *d* och *h* för värde?

```
1 scala> def applicera(x: Int, f: Int => Int) = f(x)
2 scala> def dubbla(x: Int) = 2 * x
3 scala> def halva(x: Int) = x / 2
4 scala> val d = applicera(42, dubbla)
5 scala> val h = applicera(42, halva)
```

Uppgift 11. Stegade funktioner ("Curry-funktioner"). Förklara vad som händer nedan.

```
1 scala> def sum(a: Int)(b: Int) = a + b
2 scala> sum(1)(2)
3 scala> val f = sum(42) _
4 scala> f(1)
5 scala> val inc = sum(1) _
6 scala> val dec = sum(-1) _
7 scala> inc(42)
8 scala> dec(42)
```

Uppgift 12. Objekt som moduler.

a) Lär dig följande terminologi utantill:

- Ett objekt som samlar funktioner och variabler kallas även en **modul**.
- Funktioner i objekt kallas även **metoder**.
- Variabler och metoder i objekt kallas **medlemmar**.
- Moduler kan i sin tur innehålla moduler, i godtyckligt **nästlingsdjup**.
- Man kommer åt innehållet i en modul med **punktnotation**.
- Med **import** slipper man punktnotation.
- Ett objekt med variabler sägs ha ett **tillstånd**.

b) Deklarera modulerna stringstat och Test nedan i REPL eller i Kojo.

```

object stringstat {
    object stringfun {
        def sentences(s: String): Array[String] = s.split('.')
        def words(s: String): Array[String] = s.split(' ')
        def countWords(s: String): Int = words(s).size
        def countSentences(s: String): Int = sentences(s).size
    }

    object statistics {
        var history = ""
        def printFreq(s: String): Unit = {
            println("\n---- Frekvenser ----")
            println("Antal tecken: " + s.size)
            println("Antal ord: " + stringfun.countWords(s))
            println("Antal meningar: " + stringfun.countSentences(s))
            history = history + " " + s
        }
        def printTotal: Unit = printFreq(history)
    }
}

object Test {
    import stringstat._

    def apply(n: Int = 42): Unit = {
        val s1 = "Fem myror är fler än fyra elefanter. Ät gurka."
        val s2 = "Galaxer i mina braxter. Tomat är gott. Hejsan."
        statistics.printFreq(s1 * n)
        statistics.printFreq(s2 * n)
        statistics.printTotal
    }
}

```

c) Anropa Test() och förklara vad som händer. Vad skrivs ut?

- d) Vilket av objekten i modulen `stringstat` har tillstånd och vilket av objekten är tillståndslöst? Vad består tillståndet av?

Uppgift 13. *Äkta funktioner.* En **äkta funktion** ger alltid samma resultat med samma argument (så som vi är vana vid inom matematiken).¹

```
object inSearchOfPurity {
    var x = 0
    val y = x
    def inc(i: Int) = i + 1
    def oink(i: Int) = {x = x + i; "Pig says " + "oink " * x}
    def addX(i: Int): Int = x + i
    def addY(i: Int): Int = y + i
    def isPalindrome(s: String): Boolean = s == s.reverse
    def rnd(min: Int, max: Int) = math.random * max + min
}
```

- 📎 a) Vilka funktioner i objektet `inSearchOfPurity` är äkta funktioner?
- b) Anropa de funktioner som inte är äkta i REPL och demonstrera med exempel att de kan ge olika resultat för **samma argument**.
- c) Vad är objektets tillstånd efter dina körningar i uppgift b?
- d) Vilken del av tillståndet i objektet är oföränderligt?

Uppgift 14. Funktioner är objekt med en `apply`-metod.

- a) Föklara vad som händer här:

```
1 scala> object plus { def apply(x: Int, y: Int) = x + y }
2 scala> plus.apply(42,43)
3 scala> plus(42, 43)
4 scala> val add: (Int, Int) => Int = (x, y) => x + y
5 scala> add(42, 42)
6 scala> add. // tryck på TAB
7 scala> add.apply(42, 42)
8 scala> val inc = add.curried(1)
9 scala> inc(42)
```

- b) Definiera i REPL ett objekt som heter `slumptal` som har en `apply`-metod som tar två heltalsparametrar `a` och `b` och som med hjälp av `math.random` returnerar ett slumpmässigt heltal i intervallet $[a, b]$. Anropa objektets `apply`-metod med `(1 to 100).foreach(i => print(??? + " "))` för att skriva ut 100 slumptal mellan 1 och 6. Prova både att explicit anropa `apply` med punktnotation och att använda funktionsappliceringssyntax.

Uppgift 15. *Fördröjd initialisering ("lata" variabler).*

- a) Föklara vad som händer här:

¹Äkta funktioner uppfyller per definition *referentiell transparens* (eng. *referential transparency*) som du kan läsa mer om här: en.wikipedia.org/wiki/Referential_transparency

```

1  scala> val olat = 42
2  scala> lazy val lat = 42
3  scala> println(lat)
4  scala> val nu = {Thread.sleep(1000); println("nu"); 42}
5  scala> lazy val sen = {Thread.sleep(1000); println("sen"); 42}
6  scala> def igen = {Thread.sleep(1000); println("hver gang"); 42}
7  scala> println(nu)
8  scala> println(sen)
9  scala> println(igen)
10 scala> println(nu)
11 scala> println(sen)
12 scala> println(igen)
13 scala> object m {lazy val stor = Array.fill(1e9.toInt)(liten); val liten = 42}
14 scala> m.liten
15 scala> m.stor

```

b) Vad är skillnaden mellan **val**, **lazy val** och **def**, vad gäller *när* evalueringen sker? 

c) Förklara vad som händer här:

```

1  scala> object objektÄrLata { val sen = { println("nu!"); 42 } }
2  scala> objektÄrLata
3  scala> objektÄrLata.sen
4  scala> {val x = y; val y = 42}
5  scala> object buggig {val a = b; val b = 42}
6  scala> buggig.a
7  scala> object funkar {lazy val a = b; val b = 42}
8  scala> funkar.a
9  scala> object nowarning {val many = Array.fill(10)(one); val one = 1}
10 scala> nowarning.many

```

d) Med ledning av uppgift a och uppgift c, beskriv två olika situationer när kan man ha nytta av **lazy val**? 

Uppgift 16. *Aktiveringspost.* Antag att vi bara kan addera eller subtrahera med ett. Då kan man ändå skapa en additionsfunktion på nedan (ganska omständliga) sätt. Skriv nedan program i en editor, kompilera och exekvera.

```

object Count {
  def inc(x: Int) = {println("inc[x = " + x + "]"); x + 1}
  def dec(x: Int) = {println("dec[x = " + x + "]"); x - 1}

  def add(x: Int, y: Int) = {
    println("add[x = " + x + ", y = " + y + "]")
    var result = x
    var i = 0
    while (i < math.abs(y)){
      result = if (y >= 0) inc(result) else dec(result)
      i = i + 1
    }
    result
  }
}

```

```

}

def main(args: Array[String]): Unit = {
    val x = inc(dec(inc(0)))
    println(x)
    val y = add(1, add(1, add(1, -2)))
    println(y)
}
}

```

- a) Vad skrivs ut? Föklara vad som händer.
- b) Rita hur anropsstacken förändras under exekveringen av main-metoden.

Uppgift 17. *Lokala funktioner.* Skapa nedan program i en editor, kompilerar och exekvera. I programmet nedan har metoden add två lokala funktioner som skiljer sig från metoderna med samma namn.

```

object Count {
    def inc(x: Int) = x + 1
    def dec(x: Int) = x - 1

    def add(x: Int, y: Int) = {
        def inc(x: Int) = {println("inc[x = " + x + "]"); x + 1}
        def dec(x: Int) = {println("dec[x = " + x + "]"); x - 1}
        println("add[x = " + x + ", y = " + y + "]")
        var result = x
        var i = 0
        while (i < math.abs(y)){
            result = if (y >= 0) inc(result) else dec(result)
            i = i + 1
        }
        result
    }

    def main(args: Array[String]): Unit = {
        val x = inc(dec(inc(0)))
        println(x)
        val y = add(1, add(1, add(1, -2)))
        println(y)
    }
}

```

- a) Vad skrivs ut? Föklara vad som händer.
- b) Vilka fördelar finns med lokala funktioner?

Uppgift 18. *Anonyma funktioner.* Vi har flera gånger sett syntaxen `i => i + 1`, till exempel i en loop `(1 to 10).map(i => i + 1)` där funktionen `i => i + 1`

appliceras på alla heltal från 1 till och med 10. Funktionen `i => i + 1` kallas en **anonym** funktion, eftersom den inte har något namn, till skillnad från `öka(i: Int) = i + 1`, som har namnet `öka`.

Anonyma funktioner kallas även för *funktionsliteraler* eller *lambda*.

Det finns ett ännu kortare sätt att skriva en anonym funktion om den bara använder sin parameter en enda gång, med understreck `_ + 1` som expanderas av kompilatorn till `ngtnamn => ngtnamn + 1` (namnet på parametern spelar ingen roll; kompilatorn väljer något eget, internt namn).

- a) Förklara vad som händer nedan. Vad blir resultatet av varje uttryck?

```
1 scala> (1 to 4).map(i => i + 1)
2 scala> (1 to 4).map(_ + 1)
3 scala> (1 to 4).map(math.pow(2, _))
4 scala> (1 to 4).map(math.pow(_, 2))
5 scala> (1 to 4).map(i => i.toString)
6 scala> (1 to 4).map(_.toString)
```

- b) Vilken typ kommer kompilatorn att härleda för de anonyma funktionerna i argumenten till metoden `map` på rad 1–6 i uppgiften ovan? Vad använder kompilatorn för information i dessa exempel för att härleda funktionstyperna? 

- c) Vilka felmeddelande ger kompilatorn när den inte kan lista ut att funktionsliteralerna nedan har typen `Int => Int`? 

```
1 scala> val inc = i => i + 1
2 scala> val inc = (i: Int) => i + 1
3 scala> (1 to 10).map(inc)
4 scala> val dec = _ - 1
5 scala> val dec: Int => Int = _ - 1
6 scala> (1 to 10).map(dec)
```

Uppgift 19. Rekursion.

En rekursiv funktion anropar sig själv.

- a) Förklara vad som händer nedan.

```
1 scala> def countdown(x: Int): Unit = if (x > 0) {println(x); countdown(x - 1)}
2 scala> countdown(10)
3 scala> countdown(-1)
4 scala> def finalCountdown(x: Byte): Unit =
5         {println(x); Thread.sleep(100); finalCountdown((x-1).toByte); 1 / x}
6 scala> finalCountdown(Byte.MaxValue)
```

- b) Vad händer om du gör satsen som riskerar division med noll *före* det rekursiva anropet i funktionen `finalCountdown` ovan?

- c) Förklara vad som händer nedan. Varför tar sista raden längre tid än näst sista raden?

```
1 scala> def signum(a: Int): Int = if (a >= 0) 1 else -1
2 scala> def add(x: Int, y: Int): Int =
3         if (y == 0) x else add(x + 1, y - signum(y))
4 scala> add(100,100)
5 scala> add(Int.MaxValue, 0)
6 scala> add(0, Int.MaxValue)
```

3.1.2 Extrauppgifter

Uppgift 20. Skriv och testa en funktion avg som räknar ut medelvärdet mellan två heltal och returnerar en Double.

Uppgift 21. Skriv och testa nedan två varianter av beräkning av avståndet mellan två punkter:

```
1 scala> def dist(x1: Int, y1: Int, x2: Int, y2: Int): Double = ???  
2 scala> def dist(p1: (Int, Int), p2: (Int, Int)): Double = ???
```

TODO!!! Fler förslag på extrauppgifter om funktioner välkomna, speciellt på det som man kan behöva öva mer på för att lättare förstå!

3.1.3 Fördjupningsuppgifter

Uppgift 22. Undersök den genererade byte-koden. Kompilatorn genererar **byte-kod**, uttalas ”bajtkod” (eng. *byte code*), som den virtuella maskinen tolkar och översätter till maskinkod medan programmet kör. Med kommandot :javap i REPL kan du undersöka byte-koden.

```
1 scala> def plusxy(x: Int, y: Int) = x + y  
2 scala> :javap plusxy
```

a) Leta upp raden `public int plusxy(int, int);` och studera koden efter `Code:` och försök gissa vilken instruktion som utför själva additionen.

b) Lägg till en parameter till:

`def plusxyz(x: Int, y: Int, z: Int) = x + y + z`

och studera byte-koden med :javap plusxyz. Vad skiljer byte-koden mellan plusxy och plusxyz?

c) Läs om byte-kod här: [en.wikipedia.org/wiki/Java_bytecode](https://en.wikipedia.org/wiki/Java bytecode). Vad betyder den inledande bokstaven i additionsinstruktionen?

Uppgift 23. Undersök svansrekursion genom att kasta undantag. Förklara vad som händer. Kan du hitta bevis för att kompilatorn kan optimera rekursionen till en vanlig loop?

```
1 scala> def explode = throw new Exception("BANG!!!!")  
2 scala> explode  
3 scala> lastException.printStackTrace  
4 scala> def countdown(n: Int): Unit =  
5     if (n == 0) explode else countdown(n-1)  
6 scala> countdown(10)  
7 scala> lastException.printStackTrace  
8 scala> def countdown2(n: Int): Unit =  
9     if (n == 0) explode else {countdown2(n-1); print("no tailrec")}  
10 scala> countdown2(10)  
11 scala> countdown2(1000)  
12 scala> lastException  
13 scala> lastException.getStackTrace.size
```

```
14 scala> :javap countdown
15 scala> :javap countdown2
```

Uppgift 24. *@tailrec-annotering.* Du kan be kompilatorn att ge felmeddelande om den inte kan optimera koden till motsvarande en while-loop. Om den inte kan det hämmas prestanda och det finns risk för en överfull anropssstack (eng. *stack overflow*). Prova nedan rader i REPL och förklara vad som händer.

```
1  scala> def countNoTailrec(n: Long): Unit =
2      if (n <= 0L) println("Klar! " + n) else {countNoTailrec(n-1L); ()}
3  scala> countNoTailrec(1000L)
4  scala> countNoTailrec(100000L)
5  scala> import scala.annotation.tailrec
6  scala> @tailrec def countNoTailrec(n: Long): Unit =
7      if (n <= 0L) println("Klar! " + n) else {countNoTailrec(n-1L); ()}
8  scala> @tailrec def countTailrec(n: Long): Unit =
9      if (n <= 0L) println("Klar! " + n) else countTailrec(n-1L)
10 scala> countTailrec(1000L)
11 scala> countTailrec(100000L)
12 scala> countTailrec(Int.MaxValue.toLong * 2L)
```

3.2 Laboration: blockmole

Mål

- Kunna kompilera Scalaprogram med `scalac`.
- Kunna köra Scalaprogram med `scala`.
- Kunna definiera och anropa funktioner.
- Kunna använda och förstå default-argument.
- Kunna ange argument med parameternamn.
- Kunna definiera objekt med medlemmar.
- Förstå kvalificerade namn och import.
- Förstå synlighet och skuggning.

Förberedelser

- Gör övning `programs` i avsnitt [2.1](#).
- Gör övning `functions` i avsnitt [3.1](#).

3.2.1 Obligatoriska uppgifter

Blockmullvad (*Talpa laterculus*) är ett fantasidjur i familjen mullvadsdjur. Den är känd för sitt karaktäristiska kvadratiska utseende. Den lever mest ensam i sina underjordiska gångar som till skillnad från mullvadens (*Talpa europaea*) har helt raka väggar.

Uppgift 1. Du ska skriva ett Scala-program med en vanlig texteditor och kompilera ditt program med kommandot `scalac` och sedan köra programmet med kommandot `scala`.

- a) Öppna en texteditor, till exempel gedit eller Atom (se appendix [B](#) för hjälp). Skapa en ny fil med namnet `Mole.scala` och spara den i en ny katalog i din hemkatalog, till exempel `~/pgk/mole/Mole.scala`, där `~` är din hemkatalog.
- b) Öppna ett terminalfönster (se appendix [A](#) för hjälp). Navigera till din nya katalog med `cd`-kommandot (eng. *change directory*) och kontrollera med `ls`-kommandot (eng. *list*) att din nya fil finns där.

```
> cd ~/pgk/mole  
> ls
```

Om allt går bra ska `ls`-kommandot skriva ut `Mole.scala`.

- c) Gå tillbaka till din texteditor och skriv in ett objekt med namnet `Mole` i din fil. Lägg till en `main`-funktion i objektet som skriver ut texten *Keep on digging!* med hjälp av funktionen `println`. Behöver du hjälp kan du gå tillbaka till övningarna i kapitel [3.1](#).
- d) Kör kommandot `scalac Mole.scala` i terminalfönstret för att kompilera ditt program. Om kompilatorn rapporterar några fel rättar du till det i din

texteditor kompilerar igen. Kontrollera sedan med `ls`-kommandot att några filer som slutar på `class` har skapats.

- e) Kör kommandot `scala Mole` för att köra ditt program. Om att går bra ska texten du angivit skrivas ut i terminalfönstret.

Uppgift 2. Nu har du skrivit ett Scala-program som skriver ut en uppmaning till en mullvad att fortsätta gräva. Det programmet är inte så användbart, eftersom mullvadar inte kan inte läsa. Nästa steg är att skriva ett grafiskt program, snarare än ett textbaserat.

Funktionen `println` som anropas i `main`-funktionen ingår i Scalas standardbibliotek. Ett programbibliotek innehåller kod eller kompilerade programsnuttar som kan användas av andra program, och för de flesta programspråk ingår ett standardbibliotek som alla program kan nyttja. Till grafiken i denna uppgift ska du använda ett bibliotek som kallas `cslib` och som kommer att användas även i senare labbar.

a)

Ladda ner `cslib.jar` via länken <http://cs.lth.se/pgk/cslib> och lägg jar-filen i samma katalog som ditt Scala program. En jar-fil används för att paketera färdigkompilerade program, kod, dokumentation, resursfiler, etc, och är komprimerad på samma sätt som en zip-fil.

- b) Byt ut `main`-funktionens kropp mot följande block:

```
{
  val w = new cslib.window.SimpleWindow(300, 500, "Digging")
  w.moveTo(10, 10)
  w.lineTo(10, 20)
  w.lineTo(20, 20)
  w.lineTo(20, 10)
  w.lineTo(10, 10)
}
```

Den första raden skapar ett nytt `SimpleWindow` som ritar upp ett fönster som är 300 bildpunkter bredd och 500 bildpunkter högt med titeln *Digging*. `SimpleWindow` har en `penna` som kan flyttas runt och rita linjer. Anropet `w.moveTo(10, 10)` flyttar pennan för fönstret `w` till position (10,10) utan att rita något, och anropet `w.lineTo(10, 20)` ritar en linje därifrån till position (10,20).

- c) Nu ska du kompilera ditt program, men eftersom `SimpleWindow` inte finns i Scalas standardbibliotek utan i `cslib.jar` behöver du visa kompilatorn var den ska leta. Det gör du genom att ange en `classpath`, dvs. en sökväg till `class`-filer, när du kompilar. Använd flaggan `-cp cslib.jar` för att ange `cslib.jar` som classpath och kompilera ditt Scala-program igen:

```
> scalac -cp cslib.jar Mole.scala
```

- d) Nu ska du köra ditt program, och då behöver du också ange var `class`-filerna ligger. Du ska ange den katalog där `class`-filerna för `Mole` ligger, som

du just kompilerat, men du ska också ange `cslib.jar`, och det gör du med en kolon-separerad lista², till exempel "sökväg1:sökväg2:sökväg3". Katalogen du står i, där dina `class`-filer ligger, kan anges med en punkt (.). Kör programmet med följande kommando (om Windows använd semikolon):

```
> scala -cp ".:cslib.jar" Mole
```

Du ska nu få upp ett fönster med en liten kvadrat utritad i övre vänstra hörnet.

Uppgift 3. Hela ditt program är för tillfället samlat i en och samma funktion, vilket fungerar bra för väldigt små program. Nu ska vi strukturera programmet så det blir lättare att återanvända samma kodsnuttar.

- a) Lägg till ett objekt med namnet `Graphics` i `Mole.scala` och flytta dit deklarationen av fönstret `w`. Skapa en ny funktion med namnet `square` i det nya objektet och flytta dit koden som ritar kvadraten. Anropa `square` i din `main`-funktion. Filen `Mole.scala` ska se ut såhär (förutom ???):

```
object Graphics {
    val w = new cslib.window.SimpleWindow(300, 500, "Digging")
    def square(): Unit = ???
}
object Mole {
    def main(args: Array[String]): Unit = {
        Graphics.square()
    }
}
```

Observera att du inte kan anropa `square` direkt i funktionen `main`, utan måste ange att det är `square` funktionen inuti `Graphics` du vill anropa.

- b) Kompilera `Mole.scala` med `scalac`. Glöm inte att ange korrekt classpath. (*Tips:* Du kan trycka uppåtpil för att komma till tidigare kommandon i terminalen.) Kontrollera med `ls` att det nu också finns `class`-filer för `Graphics`-objektet.
- c) Kör programmet `Mole` med `scala`. Glöm inte att ange korrekt classpath. Om allt fungerar ska programmet göra samma sak som innan.

Uppgift 4. Nu har du gjort ett grafiskt program, men ännu syns ingen mullvad. Det är dags att ta reda på hur koordinatsystemet fungerar i denna grafiska miljö, så vi kan få mullvaden att hitta rätt.

- a) Ändra i `Graphics.square` så att kvadraten ritas upp i *övre högra* hörnet istället. Prova dig fram för att ta reda på hur koordinatsystemet fungerar genom att ändra i koden, kompilera och köra programmet tills du får rätt på det.
- ✓ b) Visa kvadraten för din labbhandledare och förklara vad de två parametrarna gör genom att peka ut ungefärliga positionerna $(0,0)$, $(300,0)$, $(0,300)$

²kolon används i Linux och Mac OSX, medan windows använder semikolon.

och (300,300) ligger.

- c) Ta bort anropet till funktionen `square` när du har visat den för din labbhandledare.

Uppgift 5. Nu ska du skapa ett nytt koordinatsystem för `Graphics` som har *stora* bildpunkter. Vi kallar `Graphics` stora bildpunkter för *block* för att lättare skilja dem från `SimpleWindows` bildpunkter. Om blockstorleken är *b*, så ligger koordinaten (x, y) i `Graphics` på koordinaten (bx, by) i `SimpleWindow`.

- a) Lägg till följande deklarationer överst i objektet `Graphics`.

```
val width = 30
val height = 50
val blockSize = 10
```

Ändra bredden på ditt `SimpleWindow` till `width * blockSize` och ändra höjden till `height * blockSize`.

- b) Skapa en ny funktion i `Graphics` med namnet `block` och två parametrar *x* och *y* av typen `Int` och returtypen `Unit`. Metodens *kropp* ska se ut så här:

```
{
    val left = x * blockSize
    val right = left + blockSize - 1
    val top = y * blockSize
    val bottom = top + blockSize - 1

    for (row <- top to bottom) {
        w.moveTo(left, row)
        w.lineTo(right, row)
    }
}
```

- c) Metoden `block` ritar ett antal linjer. Hur många linjer ritas ut? I vilken ordning ritas linjerna? 
- d) Anropa funktionen `Graphics.block` några gånger i `Mole.main` så att några `block` ritas upp i fönstret när programmet körs. Komplilera och kör ditt program.

Uppgift 6. Det finns många sätt att beskriva färger. I naturligt språk har vi olika namn på färgerna, till exempel *vitt*, *rosa* och *magenta*. I datorn är det vanligt att beskriva färgerna som en blandning av *rött*, *grönt* och *blått* i det så kallade RGB-systemet. `SimpleWindow` använder typen `java.awt.Color` för att beskriva färger och `java.awt.Color` bygger på `RGB`. Det finns några fördefinierade färger i `java.awt.Color`, till exempel `java.awt.Color.black` för svart och `java.awt.Color.green` för grönt. Andra färger kan skapas genom att ange mängden rött, grönt och blått.

- a) Skapa ett nytt objekt i `Mole.scala` med namnet `Colors` och lägg in följande definitioner:

```
val mole = new java.awt.Color(51, 51, 0)
val soil = new java.awt.Color(153, 102, 51)
val tunnel = new java.awt.Color(204, 153, 102)
```

Den tre parametrarna till `new java.awt.Color(r, g, b)` anger hur mycket rött, grönt respektive blått som färgen ska innehålla, och mängderna ska vara i intervallet 0–255. Färgen (153,102,51) innehåller ganska mycket rött, lite mindre grönt och ännu mindre blått och det upplevs som brunt. Objektet Colors är en färgpallegg, men vi har inte ritat något med färg ännu. Kompilera och kör ditt program ändå, för att se så programmet fungerar likadant som sist.

b) Lägg till en parameter till `Graphics.block` sist i parameterlistan med namnet `color` och typen `java.awt.Color`. Låt *default-argumentet* för den nya parametern vara `java.awt.Color.black`. (Kommer du inte ihåg hur man gör default-argument kan du titta på övningarna i kapitel 3.1.) För att ändra färgen på blocket kan du byta linjefärg innan du ritar. Lägg till följande rad i början på `Graphics.block`:

```
w.setLineColor(color)
```

Kompilera och kör ditt program igen för att se om det fortfarande fungerar.

- ☞ c) Funktionen `Graphics.block` har tre parametrar, men den anropas bara med två parametrar i `Mole.main`. Varför är det tillåtet? Vilket värde har den tredje parametern om ingen anges?
- d) Ändra i `Mole.main` och lägg till en av definitionerna från objektet `Colors` som tredje parameter till `Graphics.block`. Kompilera och kör ditt program och upplev världen i färg.

Uppgift 7. I programmet används många långa namn med punkter, som till exempel `java.awt.Color` och `Graphics.block`. Dessa punkt-separerade namn kallas *kvalificerade* namn. För att slippa skriva dessa långa namn hela tiden kan man *importera* en definition och sen använda bara den sista delen av namnet.

- a) Importera namnet `java.awt.Color` i objektet `Colors`. Ändra sen alla `new java.awt.Color(...)` i objektet till `new Color(...)`. (Har du glömt hur man importrar ett namn kan du gå tillbaka till övningarna i kapitel 2.1.)
- ☞ b) I vilka av objekten `Mole`, `Colors` och `Graphics` kan du använda det korta respektive det kvalificerade namnet av `java.awt.Color`?
- c) Importera namnet `java.awt.Color` så att det korta namnet `Color` kan användas i objekten `Colors` och `Graphics` men inte i `Mole`. Byt sedan ut de långa namnen mot de korta i `Graphics`.

Uppgift 8. Nu ska du skriva en funktion för att rita en rektangel och rektangeln ska ritas med hjälp av funktionen `block`. Sen ska du rita upp mullvadens underjordiska värld med hjälp av denna funktion.

- a) Lägg till en funktion i objektet `Graphics` med namnet `rectangle` som tar fem parametrar `x`, `y`, `width` och `height` av typen `Int` och `color` av typen `Color`. Parametrarna `x` och `y` anger `Graphics`-koordinaten för rektangelns övre vänstra hörn och `width` och `height` anger bredden respektive höjden. Använd följande `for`-satser för att rita ut rektangeln.

```
for (yy <- y until (y + height)) {
    for (xx <- x until (x + width)) {
        block(xx, yy, color)
    }
}
```

- b) I vilken ordning ritas blocken ut?
- c) Skriv en funktion i objektet `Mole` med namnet `drawWorld` som ritar ut mullvadens värld, det vill säga en massa jord där den kan gräva sina tunnlar. `Mole.drawWorld` ska inte ha några parametrar och returntypen ska vara `Unit` och den ska anropa `Graphics.rectangle` för att rita en rektangel med färgen `Colors.soil` som precis täcker fönstret. Eftersom funktionen har många parametrar som lätt kan blandas ihop ska du använda namngivna argument vid anropet. (Om du har glömt hur man använder namngivna argument kan du titta på övningarna i kapitel 3.1.)
- d) Anropa `Mole.drawWorld` i `Mole.main` och testa så att det fungerar genom att kompilera och köra.



Uppgift 9. I `SimpleWindow` finns funktioner för att känna av tangenttryckningar och musklick. Du ska använda de funktionerna för att styra en liten blockmullvad.

- a) Importera `cslib.window.SimpleWindow` i objektet `Graphics` och lägg till följande funktion:

```
def waitForKey(): Char = {
    do {
        w.waitForEvent()
    } while (w.getEventType() != SimpleWindow.KEY_EVENT)
    w.getKey()
}
```

Det finns olika sorters händelser som ett `SimpleWindow` kan reagera på, till exempel tangenttryckningar och musklick. Funktionen som du precis lagt in väntar på en händelse i ditt `SimpleWindow` (`w.waitForEvent`) ända tills det kommer en tangenttryckning (`KEY_EVENT`). När det kommit en tangenttryckning anropas `w.getKey` för att ta reda på vilken bokstav eller vilket tecken det blev, och det resultatet blir också resultatet av `waitForKey`, eftersom det ligger sist i det yttre `{}`-blocket.

- b) Lägg till en funktion i objektet `Mole` med namnet `dig`, utan parametrar och med returntypen `Unit`. Funktionens kropp ska se ut så här (fast utan ???):

```
{
    var x = Graphics.width / 2
    var y = Graphics.height / 2
    while (true) {
        Graphics.block(x, y, Colors.mole)
        val key = Graphics.waitForKey()
        if (key == 'w') ???
        else if (key == 'a') ???
        else if (key == 's') ???
        else if (key == 'd') ???
    }
}
```

Fyll i alla ??? så att 'w' styr mullvaden ett steg uppåt, 'a' ett steg åt vänster, 's' ett steg nedåt och 'd' ett steg åt höger.

- c) Ändra Mole.main så att den bara innehåller två anrop: ett till drawWorld och ett till dig. Kompilera och kör ditt program för att se om programmer reagerar på w, a, s och d.
- d) Om programmet fungerar kommer det bli många mullvadar som tillsammans bildar en lång mask, och det är ju lite underligt. Lägg till ett anrop i Mole.dig som ritar ut en bit tunnel på position (x, y) efter anropet till Graphics.waitForKey men innan **if**-satserna. Kompilera och kör ditt program för att gräva tunnlar med din blockmullvad.

3.2.2 Frivilliga extrauppgifter

Uppgift 10. Mullvaden kan för tillfället gräva sig utanför fönstret. Lägg till några **if**-satser i början av **while**-satsen som upptäcker om x eller y ligger utanför fönstrets kant och flyttar i så fall tillbaka mullvaden precis innanför kanten.

Uppgift 11. Mullvadar är inte så intresserade av livet ovanför jord, men det kan vara trevligt att se hur långt ner mullvaden grävt sig. Lägg till en himmelsfärg och en gräsfärg i objektet Colors och rita ut himmel och gräs i Mole.drawWorld. Justera också det du gjorde i föregående uppgift, så mullvaden håller sig under jord. (*Tips:* Den andra parametern till Color reglerar mängden grönt och den tredje parametern reglerar mängden blått.)

Uppgift 12. Ändra så att mullvaden kan springa uppe på gräset också, men se till så att ingen tunnel ritas ut där.

Kapitel 4

Datastrukturer

Begrepp som ingår i denna veckas studier:

- attribut (fält)
- medlem
- metod
- tupel
- klass
- Any
- instanceof
- toString
- case-klass
- samling
- scala.collection
- föränderlighet vs oföränderlighet
- List
- Vector
- Set
- Map
- typparameter
- generisk samling som parameter
- översikt samlingsmetoder
- översikt strängmetoder
- läsa/skriva textfiler
- Source.fromFile
- java.nio.file

Olika sätt att skapa datastrukturer

- **Tupler**
 - samla n st datavärden i element **_1, _2, ... _n**
 - elementen kan vara av **olika** typ
- **Klasser**
 - samlar data i **attribut** med (väl valda!) namn
 - attributen kan vara av **olika** typ
 - definierar även **metoder** som använder attributen
(kallas även **operationer** på data)
- **Färdiga samlingar**
 - speciella klasser som samlar data i element av **samma** typ
 - exempel: `scala.collection.immutable.Vector`
 - har ofta *många* färdiga **bra-att-ha-metoder**,
se snabbreferensen <http://cs.lth.se/pgk/quickref>
- **Egenimplementerade samlingar**
 - → fördjupningskurs

Hierarki av samlingar i `scala.collection`



Läs mer om Scalas samlingar här:

<http://docs.scala-lang.org/overviews/collections/overview>

4.1 Övning: data

Mål

- Kunna skapa och använda tupler, som variabelvärdet, parametrar och returnvärdet.
- Förstå skillnaden mellan ett objekt och en klass och kunna förklara betydelsen av begreppet instans.
- Kunna skapa och använda attribut som medlemmar i objekt och klasser och som som klassparametrar.
- Beskriva innebördens av och syftet med att ett attribut är privat.
- Kunna byta ut implementationen av metoden `toString`.
- Kunna skapa och använda en objektfabrik med metoden `apply`.
- Kunna skapa och använda en enkel case-klass.
- Kunna använda operatornotation och förklara relationen till punktnotation.
- Förstå konsekvensen av uppdatering av föränderlig data i samband med multipla referenser.
- Känna till och kunna använda några grundläggande metoder på samlingar.
- Känna till den principiella skillnaden mellan `List` och `Vector`.
- Kunna skapa och använda en oföränderlig mängd med klassen `Set`.
- Förstå skillnaden mellan en mängd och en sekvens.
- Kunna skapa och använda en nyckel-värde-tabell, `Map`.
- Förstå likheter och skillnader mellan en `Map` och en `Vektor`.

Förberedelser

- Studera begreppen i kapitel 4.

4.1.1 Grunduppgifter

Uppgift 1. En enkel datastruktur: tupel. Du kan samla olika data i en tupel. Du kommer åt värdena med en metod som har namnet understreck följt av ordningsnumret.

```
1 scala> val namn = ("Pippi", "Långstrump")
2 scala> namn._1
3 scala> namn._2
4 scala> println("Förnamn: " + namn._1 + "\nEfternamn: " + namn._2)
```

- a) Definiera en oföränderlig variabel med namnet `pt` som representerar en punkt med x-koordinaten 15.9 och y-koordinaten 28.9. Använd sedan `math.hypot` för att ta reda på avståndet från origo till punkten. Vad blir svaret?

- b) Du kan dela upp en tupel i sina beståndsdelar så här:

```
scala> val (förnamn, efternamn) = ("Ronja", "Rövardotter")
```

Dela upp din punkt pt i sina beståndsdelar och kalla delarna x och y

- c) Värdena i en tupel kan ha olika typ.

```
scala> val creature = ("Doktor", "Krokodil", 65.0, false)
scala> val (title, name, weight, isHuman) = creature
```

Vilken typ har 4-tupeln creature ovan?

- d) Tupler kan ingå i samlingar.

```
scala> val pts = Vector((0.0, 0.0), (1.0, 0.0), (1.0, 1.0), (0.0, 1.0))
scala> pts.foreach(println)
```

Vilken typ har vektorn pts ovan?

- e) För 2-tupler finns ett kortare skrivsätt:

```
scala> ("Skåne", "Malmö")
scala> "Skåne" -> "Malmö"
scala> val huvudstäder = Vector("Sverige" -> "Stockholm", "Norge" -> "Oslo")
```

Lägg till fler huvudstäder i vektorn ovan.

- f) Funktioner kan ta tupler som parametrar.

```
1 scala> def length(pt: (Double, Double)) = math.hypot(pt._1, pt._2)
2 scala> length((3.0, 4.0))
3 scala> length(3.0, 4.0) //kompilatorn lägger till parenteserna innan anrop
```

Applicera funktionen length ovan på alla tupler i samlingen pts från uppgift d med map. Vad får resultatet för värde och typ?

- g) Funktioner kan ge tupler som resultat.

```
1 scala> def div(a: Int, b: Int) = (a / b, a % b)
2 scala> div(10, 3)
3 scala> (div(9,2), div(10,2))
4 scala> (div(9,2)._2, div(10,2)._2)
5 scala> val n0Odd = (1 to 10).map(i => div(i, 2)._2).sum
```

Förklara vad som händer ovan. Använd div ovan för att ta reda på hur många udda tal finns det i intervallet [1234,3456].

- h) En tupel med n värden kallas n -tupel. Om man betraktar enhetsvärdet () som en tupel, vad kan man då kalla detta värde?

Uppgift 2. *Objekt med attribut (fält).* Ett objekt kan samla data som hör ihop och på så sätt skapa en datastruktur. Data i ett objekt kallas *attribut* eller *fält*, (eng. *field*). Objekt som samlar enbart data kallas även *post* (eng. *record*).

```
scala> object mittKonto { var saldo = 0; val nummer = 12345L }
```

- a) Skriv en sats som sätter in ett slumpmässigt belopp mellan 0 och en miljon på mittKonto ovan med hjälp av punktnotation och tilldelning.

- b) Vad händer om du försöker ändra attributet nummer?

Uppgift 3. *Klass med attribut.* Om du vill ha många objekt av samma typ, kan du använda en **klass**. På så sätt kan man skapa många datastrukturer av samma typ men med olika innehåll. Man skapar nya objekt med nyckelordet **new** följt av klassens namn. Klassen utgör en ”mall” för objektet som skapas. Ett objekt som skapas med **new** Klassnamn kallas även en **instans** av klassen Klassnamn. Nedan skapas en datastruktur Konto som samlar data om ett bankkonto. Instanser av typen Konto håller reda på hur mycket pengar det finns på kontot och vilket kontonumret är. Datavärden som sparas i varje objektinstans, så som saldo och nummer, kallas **attribut** (eng. *attribute*) eller **fält** (eng. *field*).

```

1  scala> class Konto {
2      var saldo = 0
3      var nummer = 0L
4  }
5  scala> val k1 = new Konto
6  scala> val k2 = new Konto
7  scala> k1.saldo = 1000
8  scala> k1.nummer = 12345L
9  scala> k2.saldo = 2000
10 scala> k2.nummer = 67890L
11 scala> println("Konto: " + k1.nummer + " Saldo:" + k1.saldo)
12 scala> println("Konto: " + k2.nummer + " Saldo:" + k2.saldo)
```

- ☞ a) Rita hur minnessituationen ser ut efter att ovan rader har exekverats.
- ☞ b) Vad hade det fått för konsekvenser om attributet nummer vore oföränderligt i klassen ovan? (Jämför med objektet mittKonto.)

Uppgift 4. *Klass med attribut som parametrar.* Om man vill ge attributen initialvärdet när objektet skapas med **new**, kan man placera attributten i en parameterlista till klassen. Koden som körs när objektet skapas och attributten tilldelas sina initialvärden, kallas **konstruktör** (eng. *constructor*).

```

1  scala> class Konto(var saldo: Int, val nummer: Long)
2  scala> val k = new Konto(0, 12345L)
3  scala> println("Konto: " + k.nummer + " Saldo:" + k.saldo)
4  scala> println(k)
5  scala> k.toString
```

- a) Den två sista raderna ovan skriver ut den identifierare som JVM använder för att hålla reda på objektet i sina interna datastrukturer. Vad skrivs ut?
- b) Skapa ännu en instans av klassen Konto med samma saldo och nummer som k ovan och spara den i **val** k2 och undersök dess objektidentifierare. Får objekten k och k2 olika objektidentifierare?
- c) Sätt in olika belopp på respektive konto.
- d) Vad händer om du försöker ändra attributet nummer?
- ☞ e) Ibland räcker det fint med en tupel, men ofta vill man ha en klass istället. Beskriv några fördelar med en Konto-klassen ovan jämfört med en tupel av typen (Int, Long).

```
scala> var k3 = (0, 12345L)
scala> k3 = (k3._1 + 100, k3._2)
```

Uppgift 5. *Publikt eller privat attribut?* Man kan förhindra att ett attribut syns utanför klassen med hjälp av nyckelordet **private**.

```
1 scala> class Konto1(val nummer: Long){ var saldo = 0 }
2 scala> val k1 = new Konto1(12345678901L)
3 scala> k1.nummer
4 scala> k1.saldo += 1000
5 scala> class Konto2(val nummer: Long){ private var saldo = 0 }
6 scala> val k2 = new Konto2(12345678901L)
7 scala> k2.nummer
8 scala> k2.saldo += 1000
```

- a) Vad händer ovan?
- b) Gör en ny version av klassen Konto enligt nedan:

```
class Konto(val nummer: Long){
    private var saldo = 0
    def in(belopp: Int): Unit = {saldo += belopp}
    def ut(belopp: Int): Unit = {saldo -= belopp}
    def show: Unit =
        println("Konto Nr: " + nummer + " saldo: " + saldo)
}

object Main {
    def main(args: Array[String]): Unit = {
        val k = new Konto(1234L)
        k.show
        k.in(1000)
        println("Uttag: " + k.ut(500))
        println("Uttag: " + k.ut(1000))
        k.show
    }
}
```

- c) Spara koden i en fil, kompilera med `scalac` och kör. Testa även vad som händer om du försöker komma åt attributet `saldo` i `main`-metoden med t.ex. `println(k.saldo)` eller `k.saldo += 1000`.
- d) Vi ska nu förhindra överuttag. Ändra i metoden `ut` så att den får signaturen `ut(belopp: Int): (Int, Int) = ???` och implementera `ut` så att den returnerar både beloppet man verkligen kan ta ut och kvarvarande saldo. Om man försöker ta ut mer än det finns på kontot så ska saldot bli 0 och man får bara ut det som finns kvar. Spara, kompilera, kör.
- e) Förbättra metoderna `in` och `ut` så att man inte kan sätta in eller ta ut negativa belopp.

- f) Vad är fördelen med att göra föränderliga attribut privata och bara påverka deras värden indirekt via metoder?

Uppgift 6. Vilken typ har ett objekt? Objektets typ bestäms av klassen. Vid tilldelning måste typerna passa ihop.

- a) Vilka rader nedan ger felmeddelande? Hur lyder felmeddelandet?

```

1 scala> class Punkt(val x: Double, val y: Double)
2 scala> val pt: Punkt = new Punkt(10.0, 10.0)
3 scala> val i: Int = pt.x
4 scala> val (x: Double, y: Double) = (pt.x, pt.y)
5 scala> val p: Double = new Punkt(5.0, 5.0)
6 scala> val p = new Punkt(5.0, 5.0): Double
7 scala> val p = new Punkt(5.0, 5.0): Punkt
8 scala> pt: Punkt

```

- b) Man kan undersöka om ett objekt är av en viss typ med metoden `isInstanceOf[Typnamn]`. Vad ger nedan anrop av metoden `isInstanceOf` för värde?

```

1 scala> class Punkt(val x: Double, val y: Double)
2 scala> val pt: Punkt = new Punkt(1.0, 2.0)
3 scala> pt.isInstanceOf[Punkt]
4 scala> pt.isInstanceOf[Double]
5 scala> pt.x.isInstanceOf[Punkt]
6 scala> pt.x.isInstanceOf[Double]
7 scala> pt.x.isInstanceOf[Int]

```

Uppgift 7. Any. Alla klasser är också av typen Any. Alla klasser får därför med sig några gemensamma metoder som finns i den fördefinierade klassen Any, däribland metoderna `isInstanceOf` och `toString`. Vad blir resultatet av respektive rad nedan? Vilken rad ger ett felmeddelande?

```

1 scala> class Punkt(val x: Double, val y: Double)
2 scala> val pt: Punkt = new Punkt(1.0, 2.0)
3 scala> pt.isInstanceOf[Punkt]
4 scala> pt.isInstanceOf[Any]
5 scala> pt.x.toString
6 scala> println(pt.x)
7 scala> val a: Any = pt
8 scala> println(a.x)
9 scala> a.toString
10 scala> pt.y.toString
11 scala> a.y.toString

```

Uppgift 8. Byta ut metoden `toString`. I klassen Any finns metoden `toString` som skapar en strängrepresentation av objektet. Du kan byta ut metoden `toString` i klassen Any mot din egen implementation. Man använder nyckelordet **override** när man vill byta ut en metodimplementation.

```

1 scala> class Punkt(val x: Double, val y: Double) {
2   override def toString: String = "[x=" + x + ",y=" + y + "]"

```

```

3     }
4 scala> val pt = new Punkt(1.0, 42.0)
5 scala> pt.toString
6 scala> println(pt)

```

- a) Vad händer egentligen på sista raden ovan?
- b) Omdefiniera `toString` så att den ger en sträng på formen `Punkt(1.0, 42.0)`.
- c) Vad händer om du utelämnar nyckelordet **override** vid omdefiniering?

Uppgift 9. *Objektfabrik med apply-metod.* Man kan ordna så att man slipper skriva `new` med ett s.k. *fabriksobjekt* (eng. *factory object*).

```

class Pt(val x: Double, y: Double) {
  override def toString: String = "Pt(x=" + x + ",y=" + y + ")"
}
object Pt {
  def apply(x: Double, y: Double): Pt = new Pt(x, y)
}

```

- a) Skriv satser som använder metoden `apply` i fabriksobjektet **object** `Pt` för att skapa flera olika punkter.
- b) Ge `apply`-metoden default-argument 0.0 för både `x` och `y` så att `Pt()` skapar en punkt i origo.
- c) Skapa en klass `Rational` som representerar rationellt tal som en kvot mellan två heltal. Ge klassen två oföränderliga, publika klassparameterattribut med namnen `nom` för täljaren och `denom` för nämnaren.
- d) Skapa ett fabriksobjekt med en `apply`-metod som tar två heltalsparametrar och skapar en instans av klassen `Rational`.
- e) Skapa olika instanser av din klass `Rational` ovan med hjälp av fabriksobjektet.

Uppgift 10. *Skapa en case-klass.* Med en case-klass får man `toString` och fabriksobjekt på köpet. Man behöver inte skriva **val** framför klassparametrar i case-klasser; klassparametrar blir publika, oföränderliga attribut automatiskt när man deklarerar en case-klass.

```

1 scala> case class Pt(x: Double, y: Double)
2 scala> val p = Pt(1.0, 42.0)
3 scala> p.toString
4 scala> println(p)
5 scala> println(Pt(5,6))

```

- a) Implementera din klass `Rational` från föregående uppgift, men nu som en case-klass.

Uppgift 11. *Metoder på datastrukturer.* En datastruktur blir mer användbar om det finns metoder som kan användas på datastrukturen. Metoder i Scala kan även ha (vissa) specialtecken som namn, t.ex. + enligt nedan.

```

1  scala> case class Point(x: Double, y: Double) {
2      def distToOrigin: Double = math.hypot(x, y)
3      def add(p: Point): Point = Point(x + p.x, y + p.y)
4      def +(p: Point): Point = add(p)
5  }

```

- a) Använd metoden `distToOrigin` för att ta reda på vad punkten med koordinaterna (3, 4) har för avstånd till origo?
- b) Skriv satser som skapar två punkter (3,4) och (5, 6) och låt variablerna `p1` och `p2` referera till respektive punkt. Låt variabeln `p3` bli summan av `p1` och `p2` med hjälp av metoden `add`. Vad får uttryckene `p3.x` resp. `p3.y` för värdet?

Uppgift 12. *Operatornotation.* Vid punktnotation på formen:

`objekt.metod(argument)`

kan man skippa punkten och parenteserna och skriva:

`objekt metod argument`

Detta förenklade skrivsätt kallas **operatornotation**.

- a) Använd klassen `Point` från uppgift 11 och prova nedan satser. Vilka rader använder operatornotation och vilka rader använder punktnotation? Vilka rader ger felmeddelande?

```

1  scala> val p1 = Point(3,4)
2  scala> val p2 = Point(3,4)
3  scala> p1.add(p2)
4  scala> p1 add p2
5  scala> p1.+(p2)
6  scala> p1 + p2
7  scala> 42 + 1
8  scala> 42.+(1)
9  scala> 42.+ 1
10 scala> 42 +(1)
11 scala> 1.to(42)
12 scala> 1 to 42
13 scala> 1.to(42)

```

- b) Implementera metoderna `sub` och `-` i klassen `Point` och skriv uttryck som kombinerar `add` och `sub`, samt `+` och `-` i både punktnotation och operatornotation.
- c) Operatornotation fungerar även med flera argument. Man använder då parenteser om listan med argumenten: `objekt metod (arg1, arg2)`
Definiera en metod
`def scale(a: Double, b: Double) = Point(x * a, y * b)`
 i klassen `Point` och skriv satser som använder metoden med punktnotation och operatornotation.

Uppgift 13. *Föränderlighet och oföränderlighet.* Oföränderliga och föränderliga objekt beter sig olika vid tilldelning.

- a) Innan du kör nedan kod: Försök lista ut vad som kommer att skrivas ut. Rita minnessituationen efter varje tilldelning.

```
println("\n--- Example 1: mutable value assignment")
var x1 = 42
var y1 = x1
x1 = x1 + 42
println(x1)
println(y1)
```

- b) Innan du kör nedan kod: Försök lista ut vad som kommer att skrivas ut. Rita minnessituationen efter varje tilldelning.

```
println("\n--- Example 2: mutable object reference assignment")
class MutableInt(private var i: Int) {
  def +(a: Int): MutableInt = { i = i + a; this }
  override def toString: String = i.toString
}
var x2 = new MutableInt(42)
var y2 = x2
x2 = x2 + 42
println(x2)
println(y2)
```

- c) Innan du kör nedan kod: Försök lista ut vad som kommer att skrivas ut. Rita minnessituationen efter varje tilldelning.

```
println("\n--- Example 3: immutable object reference assignment")
class ImmutableInt(val i: Int) {
  def +(a: Int): ImmutableInt = new ImmutableInt(i + a)
  override def toString: String = i.toString
}
var x3 = new ImmutableInt(42)
var y3 = x3
x3 = x3 + 42
println(x3)
println(y3)
```

- d) Vad finns det för fördelar med oföränderliga datastrukturer?

Uppgift 14. Några användbara samlingar. En **samling** (eng. *collection*) är en datastruktur som samlar många objekt av samma typ. I `scala.collection` och `java.util` finns många olika samlingar med en uppsjö användbara metoder. De olika samlingarna i `scala.collection` är ordnade i en gemensam hierarki med många gemensamma metoder; därför har man nytta av det man lär sig om metoderna i en Scala-samling när man använder en annan samling. Vi har redan tidigare sett samlingen `Vector`:

```

1  scala> val tärningskast = Vector.fill(10000)((math.random * 6 + 1).toInt)
2  scala> tä    // tryck TAB
3  scala> tärningskast. // tryck TAB

```

- a) Ungefär hur många metoder finns det som man kan göra på objekt av typen `Vector`? Det är svårt att lära sig alla dessa på en gång, så vi väljer ut några få i kommande uppgifter.
- b) Jämför överlappet mellan metoderna i `Vector` och `List` och uppskatta hur stor andel av metoderna som är gemensamma:

```

1  scala> val myntkast =
2      List.fill(10000)(if (math.random < 0.5) "krona" else "klave")
3  scala> my    // tryck TAB
4  scala> myntkast. // tryck TAB

```

Uppgift 15. *Typparameter.* Vissa funktioner är generella för många typer och tar en så kallad **typparameter** inom hakparenteser. Ofta slipper man skriva typparametrar, då komplatorn kan härleda typen utifrån argumenten. Om man anger typparametrar explicit så hjälper komplatorn dig med att kolla att det verkligen är rätt typ i samlingen.

- a) Vad händer nedan?

```

1  scala> var xs = Vector.empty[Int]
2  scala> xs = xs :+ "42"
3  scala> xs = xs :+ 43 :+ 64 :+ 46
4  scala> xs
5  scala> xs := "42".toInt
6  scala> var ys = Vector[Int]("ett", "två", "tre")
7  scala> var ingenting = Vector.empty
8  scala> ingenting = Vector(1,2,3)

```

- b) Samlingar är mer användbara om de är *generiska*, vilket innebär att elementens typ avgörs av en typparameter och därför kan vara av vilken typ som helst. Man kan definiera egna funktioner som tar generiska samlingar som parametrar. Förlara vad som händer här:

```

1  scala> val vego = Vector("gurka", "tomat", "apelsin", "banan")
2  scala> val prim = Vector(2, 3, 5, 7, 11, 13)
3  scala> def först[T](xs: Vector[T]): T = xs.head
4  scala> def sist[T](xs: Vector[T]) = xs.last
5  scala> def förstOchSist[T](xs: Vector[T]): (T, T) = (xs.head, xs.last)
6  scala> först(vego)
7  scala> sist(prim)
8  scala> förstOchSist(vego)
9  scala> förstOchSist(prim)
10 scala> def wrap[T](pair: (T, T))(xs: Vector[T]) = pair._1 +: xs :+: pair._2
11 scala> wrap("Odla", "och ät!")(vego)
12 scala> wrap("Odla", "och ät!")(vego).mkString(" ")

```

Uppgift 16. *Några viktiga samlingsmetoder.* Deklarera följande vektorer i REPL.

```

1 scala> val xs = (1 to 10).toVector
2 scala> val a = Vector("abra", "ka", "dabra")
3 scala> val b = Vector( "sim", "sala", "bim", "sala", "bim")
4 scala> val stor = Vector.fill(100000)(math.random)

```

Undersök i REPL vad som händer nedan. Alla dessa metoder fungerar på alla samlingar som är indexerbara sekvenser. Givet deklarationerna ovan: vad har uttrycken nedan för värde och typ? Förklara vad som händer hjälп av denna översikt: docs.scala-lang.org/overviews/collections/seqs

- a) a(1) + xs(1)
- b) a apply 0
- c) a.isDefinedAt(3)
- d) a.isDefinedAt(100)
- e) stor.length
- f) stor.size
- g) stor.min
- h) stor.max
- i) a indexOf "ka"
- j) b.lastIndexOf("sala")
- k) "först" +: b //minnesregel: colon on the collection side
- l) a :+ "sist" //minnesregel: colon on the collection side
- m) xs.updated(2,42)
- n) a.padTo(10, "!")
- o) b.sorted
- p) b.reverse
- q) a.startsWith(Vector("abra", "ka"))
- r) "hejsan".endsWith("san")
- s) b.distinct

Uppgift 17. *Några generella samlingsmetoder:* Det finns metoder som går att köra på *alla* samlingar även om de inte är indexerbara. Givet deklarationerna i föregående uppgift: vad har uttrycken nedan för värde och typ? Förklara vad som händer med hjälп av dessa översikter:

docs.scala-lang.org/overviews/collections/trait-traversable

docs.scala-lang.org/overviews/collections/trait-iterable

- a) a ++ b
- b) a ++ stor
- c) **val** ys = xs.map(_ * 5)
- d) b.toSet // En mängd har inga dubletter
- e) a.head + b.last
- f) a.tail

- g) `a.head +: a.tail == a`
- h) `Vector(a.head) ++ Vector(b.last)`
- i) `a.take(1) ++ b.takeRight(1)`
- j) `a.drop(2) ++ b.drop(1).dropRight(2)`
- k) `a.drop(100)`
- l) `val e = Vector.empty[String]; e.take(100)`
- m) `Vector(e.isEmpty, e.nonEmpty)`
- n) `a.contains("ka")`
- o) `"ka" contains "a"`
- p) `a.filter(s => s.contains("k"))`
- q) `a.filter(_.contains("k"))`
- r) `a.map(_.toUpperCase).filterNot(_.contains("K"))`
- s) `xs.filter(x => x % 2 == 0)`
- t) `xs.filter(_ % 2 == 0)`

Uppgift 18. De olika samlingarna i `scala.collection` används flitigt i andra paket, exempelvis `scala.util` och `scala.io`.

- a) Vad händer här? (Metoden `shuffle` skapar en ny samling med elementen i slumpvis ordning.)

```

1 val xs = Vector(1,2,3)
2 def blandat = scala.util.Random.shuffle(xs)
3 def test = if (xs == blandat) "lika" else "olika"
4 (for(i <- 1 to 100) yield test).count(_ == "lika")

```

- b) Skapa en textfil med namnet `fil.txt` som innehåller lite text och läs in den med:

```
scala.io.Source.fromFile("fil.txt", "UTF-8").getLines.toVector
```

```

1 > cat > fil.txt
2 hejsan
3 svejsan
4 > scala
5 scala> val xs = scala.io.Source.fromFile("fil.txt", "UTF-8").getLines.toVector
6 scala> xs.foreach(println)

```

- c) Vad händer här? (Metoden `trim` på värden av typen `String` ger en ny sträng med blanktecken i början och slutet borttagna.)

```

1 scala> val pgk =
2   scala.io.Source.fromURL("http://cs.lth.se/pgk/", "UTF-8").getLines.toVector
3 scala> pgk.foreach(println)
4 scala> pgk.map(_.trim).
5       filterNot(_.startsWith("<")).
6       filterNot(_.isEmpty).
7       foreach(println)

```

Uppgift 19. *Jämför List och Vector.* En indexerbar sekvens av värden kallas vektor eller lista. I Scala finns flera klasser som kan indexeras, däribland klasserna Vector och List.

- a) *Likheter mellan Vector och List.* Kör nedan rader i REPL. Prova indexera i båda och studera hur stor andel av metoderna som är gemensamma.

```
1 scala> val sv = Vector("en", "två", "tre", "fyra")
2 scala> val en = List("one", "two", "three", "four")
3 scala> sv(0) + sv(3)
4 scala> en(0) + en(3)
5 scala> sv. //tryck TAB
6 scala> en. //tryck TAB
```

- b) *Skillnader mellan Vector och List.* Klassen Vector i Scala har ”under huven” en avancerad datastruktur i form av ett s.k. självbalanserande träd, vilket gör att Vector är snabbare än List på nästan allt, *utom* att bearbeta elementen *i början* av sekvensen; vill man lägga till och ta bort i början av en List så kan det ibland gå ungefär dubbelt så fort jämfört med Vector, medan alla andra operationer är lika snabba eller snabbare med Vector. Det finns ett fåtal speciella metoder, som bara finns i List, för att skapa en lista och lägga till i början av en lista. Vad händer nedan?

```
1 scala> var xs = "one" :: "two" :: "three" :: "four" :: Nil
2 scala> xs = "zero" :: xs
3 scala> val ys = xs.reverse ::: xs
```

Uppgift 20. *Mängd.* En mängd är en samling som garanterar att det inte finns några dubblettar. Det går dessutom väldigt snabbt, även i stora mängder, att kolla om ett element finns eller inte i mängden. Elementen i samlingen Set hamnar ibland, av effektivitetsskäl, i en förväntande ordning.

```
1 scala> val s = Set("Malmö", "Stockholm", "Göteborg", "Köpenhamn", "Oslo")
2 s: scala.collection.immutable.Set[String] =
3   Set(Oslo, Malmö, Köpenhamn, Stockholm, Göteborg)
4
5 scala> val t = Set("Sverige", "Sverige", "Sverige", "Danmark", "Norge")
6 t: scala.collection.immutable.Set[String] = Set(Sverige, Danmark, Norge)
```

Givet ovan deklarationer: vad blir värde och typ av nedan uttryck?

- a) `s + "Malmö" == s`
- b) `s ++ t`
- c) `Set("Malmö", "Oslo").subsetOf(s)`
- d) `s subsetOf Set("Malmö", "Oslo")`
- e) `s contains "Lund"`
- f) `s apply "Lund"`
- g) `s("Malmö")`
- h) `s - "Stockholm"`

- i) $t - ("Norge", "Danmark", "Tyskland")$
- j) $s -- t$
- k) $s -- Set("Malmö", "Oslo")$
- l) $Set(1,2,3) \text{ intersect } Set(2,3,4)$
- m) $Set(1,2,3) \& Set(2,3,4)$
- n) $Set(1,2,3) \text{ union } Set(2,3,4)$
- o) $Set(1,2,3) | Set(2,3,4)$

Uppgift 21. Slå upp värden från nycklar med Map. Samlingen Map är mycket användbar. Med den kan man snabbt leta upp ett värde om man har en nyckel. Samlingen Map är en generalisering av en vektor, där man kan ”indexera”, inte bara med ett heltal, utan med vilken typ av värde som helst, t.ex. en sträng. Datastrukturen Map är en s.k. *associativ array*¹, implementerad som en s.k. *hashtabell*².

```
1  scala> var huvudstad =
2    Map("Sverige" -> "Stockholm", "Norge" -> "Oslo", "Skåne" -> "Malmö")
```

Givet ovan variabel huvudstad, förklara vad som händer nedan?

- a) huvudstad apply "Skåne"
- b) huvudstad("Sverige")
- c) huvudstad.contains("Skåne")
- d) huvudstad.contains("Malmö")
- e) huvudstad += "Danmark" -> "Köpenhamn"
- f) huvudstad.foreach(println)
- g) huvudstad getOrElse ("Norge", "??")
- h) huvudstad getOrElse ("Finland", "??")
- i) huvudstad.keys.toVector.sorted
- j) huvudstad.values.toVector.sorted
- k) huvudstad - "Skåne"
- l) huvudstad - "Jylland"
- m) huvudstad = huvudstad.updated("Skåne", "Lund")

Uppgift 22. Skapa Map från en samling.

- a) Definiera denna vektor och undersök dess typ:

```
val pairs = Vector(
  ("Björn", 46462229009L),
  ("Maj", 46462221667L),
  ("Gustav", 46462224906L))
```

¹https://en.wikipedia.org/wiki/Associative_array

²https://en.wikipedia.org/wiki/Hash_table

- b) Vad har variablen `telnr` nedan för typ:

```
var telnr = pairs.toMap
```

- c) Använd `telnr` för att slå upp telefonnummer för Maj och Kim med hjälp av metoderna `apply` och `get`.

- d) Använd metoden `getOrElse` vid upplagningar av `telnr` och ge `-1L` som telefonnummer i händelse av att ett nummer inte finns.

- e) Lägg till ("Fröken Ur", 464690510L) i `telnr`-mappen.

- f) Skapa en `Vector[(String, String)]` enligt nedan, så att telefonnumret blir en sträng utan inledande landsnummer men med en nolla i riktnumret. Byt ut `???` mot lämpligt uttryck.

```
1 scala> telnr.toVector.map(p => ???)
2 res85: Vector[(String, String)] = Vector(("Björn", "0462229009"), ("Maj",
3 "0462221667"), ("Gustav", "0462224906"), ("Fröken Ur", 04690510"))
```

- g) Använd vektorn i resultatet ovan för att skapa en ny `Map[String, String]` med nationella telefonnummer. Slå upp numret till Fröken Ur.

Uppgift 23. *Samlingsmetoden maxBy.* Med samlingsmetoden `maxBy` kan man själv definiera vad som ska maximeras. (Denna metod kommer du att behöva i veckans laboration.)

- a) Förklara vad som händer nedan.

```
1 scala> val xs = Vector((2,3), (1,5), (-1, 1), (7, 2))
2 scala> xs.maxBy(x => x._1)
3 scala> xs.maxBy(x => x._2)
```

- b) Om man bara använder en parameter i en anonym funktion, till exempel parametern `x` i lambdauttrycket `x => x + 1` en enda gång, och kompilatorn kan gissa alla typer, kan man använda understreck som "platshållare" för att förkorta lambdauttrycket så här: `_ + 1`

Skriv uttrycken på raderna 2 och 3 i föregående deluppgift på ett kortare sätt med hjälp platshållarsyntax (eng. *place holder syntax*).

- c) På motsvarande sätt kan man använda `minBy` för att välja vilket funktion som definierar minimum. Prova `minBy` på motsvarande sätt som i föregående deluppgifter.

Uppgift 24. *Samlingsmetoden sliding.* I veckans labb kommer du att ha nytta av samlingsmetoden `sliding`, som ger en iterator för speciella delsekvenser av en sekvens, vilka kan liknas vid "utsikten" i ett "glidande fönster". Kör nedan i REPL och beskriv vad som händer.

```
1 scala> val xs = Vector("fem", "gurkor", "är", "fler", "än", "fyra", "tomater")
2 scala> xs.sliding(2).toVector
3 scala> xs.sliding(3).toVector
4 scala> xs.sliding(4).toVector
5 scala> xs.sliding(7).toVector
6 scala> xs.sliding(10).toVector
```

4.1.2 Extrauppgifter

Uppgift 25. Skriv nedan program med en editor och kompilera från terminalen. Lägg till kod i huvudprogrammet som testar klassen Account och kompilera och kör. Utvidga sedan klassen Account med fler attribut och funktioner som du väljer själv.

```
class Account(val number: Long, val maxCredit: Int){
    private var balance = 0

    def deposit(amount: Int): Int = {
        if (amount > 0) {balance += amount}
        balance
    }

    def withdraw(amount: Int): (Int, Int) = if (amount > 0) {
        val allowedWithdrawal =
            if (amount < balance + maxCredit) amount
            else balance + maxCredit
        balance = balance - allowedWithdrawal
        (allowedWithdrawal, balance)
    } else (0, balance)

    def show: Unit =
        println("Account Nbr: " + number + " balance: " + balance)
    }

    object Main {
        def main(args: Array[String]): Unit = {
            ???
        }
    }
}
```

Uppgift 26. Läs om reglerna för spelet Keno här:

<https://sv.wikipedia.org/wiki/Keno> och gör deluppgifterna nedan.

- Skapa en klass Keno som kan användas för att genomföra en Kenodragning. Låt klassen ha ett privat attribut balls som är en föränderlig mängd med heltal och som från början är tom. Implementera lämpliga metoder i klassen för att användaren av klassen ska kunna dra nya slumpmässiga bollar som inte redan är dragna.
- Skapa en **case class** KenoBet(bet: Set[Int]) för att hålla reda vilka 11 bollar en viss person satsar på. Definiera en metod
def number0fHits(keno: Keno): Int = ???
i case-klassen KenoBet som givet en kenodragning räknar ut hur många bollar som satsats rätt.

- c) Skriv ett huvudprogram som simulerar en enkel Kenodragning. Låt två personer satsa på 11 slumpmässiga dollar, genomför en dragning av 20 dollar ur 70 möjliga och kontrollera sedan hur många dollar som personerna har prickat rätt.

4.1.3 Fördjupningsuppgifter

Uppgift 27. *Dokumentationen för Any.* Undersök vilka metoder som finns i klassen Any här: <http://www.scala-lang.org/api/current/#scala.Any>. Prova några av metoderna i REPL.

Uppgift 28. *Dokumentationen för samlingar.* Leta upp metoden tabulate i dokumentationen för objektet Vector nästan längst ner i listan här:

[http://www.scala-lang.org/api/current/#scala.collection.immutable.Vector\\$](http://www.scala-lang.org/api/current/#scala.collection.immutable.Vector$)

Leta upp den variant av tabulate som har signaturen:

def tabulate[A](n: Int)(f: (Int) => A): Vector[A]

Klicka på den gråfyllda trekanten till vänster om signaturen som fäller ut beskrivningen

- a) Förklara vad som händer här:

```
scala> Vector.tabulate(10)(i => i % 3)
```

- b) Klicka på det blåa stora o-et överst på sidan, för att växla till klass-vyn och studera listan med alla metoder i klassen Vector.

Uppgift 29. *Fler metoder på indexerbara sekvenser.* Deklarera följande vektorer i REPL.

```
1 scala> val xs = (1 to 10).toVector
2 scala> val a = Vector("abra", "ka", "dabra")
3 scala> val b = Vector("sim", "sala", "bim", "sala", "bim")
```

Undersök i REPL vad som händer nedan. Alla dessa metoder fungerar på alla samlingar som är indexerbara sekvenser. Vad har uttrycken för värde och typ? Förklara vad metoden gör. Studera även denna översikt: docs.scala-lang.org/overviews/collections/seqs

- a) b.indexWhere(s => s.startsWith("b"))
- b) a.indices
- c) xs.patch(1, Vector(42,43,44), 7)
- d) xs.segmentLength(_ < 8, 2)
- e) b.sortBy(_.reverse)
- f) b.sortWith((s1, s2) => s1.size < s2.size)
- g) a.reverseMap(_.size)
- h) a intersect Vector("ka", "boom", "pow")
- i) a diff Vector("ka")

j) a union Vector("ka", "boom", "pow")

Uppgift 30. Jämför tidsprestanda mellan List och Vector vid hantering i början och i slutet.

a) Hur snabbt går nedan på din dator? (Exemplet nedan är exekverat på en Intel i7-4790K CPU @ 4.00GHz.)

```
$scala
Welcome to Scala 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_66).
Type in expressions for evaluation. Or try :help.

scala> :pa
// Entering paste mode (ctrl-D to finish)

def time(n: Int)(block: => Unit): Double =  {
  def now = System.nanoTime
  var timestamp = now
  var sum = 0L
  var i = 0
  while (i < n) {
    block
    sum = sum + (now - timestamp)
    timestamp = now
    i = i + 1
  }
  val average = sum.toDouble / n
  println("Average time: " + average + " ns")
  average
}

// Exiting paste mode, now interpreting.

time: (n: Int)(block: => Unit)Double

scala> val n = 100000
scala> val l = List.fill(n)(math.random)
scala> val v = Vector.fill(n)(math.random)

scala> (for(i <- 1 to 20) yield time(n){l.take(10)}).min
Average time: 476.85004 ns
Average time: 52.29291 ns
Average time: 221.50289 ns
Average time: 218.60302 ns
Average time: 45.01888 ns
Average time: 243.7818 ns
Average time: 45.02228 ns
Average time: 2132.03995 ns
Average time: 52.83995 ns
Average time: 46.7478 ns
Average time: 51.8753 ns
Average time: 70.57658 ns
Average time: 45.26142 ns
Average time: 95.16307 ns
Average time: 43.84092 ns
Average time: 55.24695 ns
```

```
Average time: 84.06113 ns
Average time: 42.04872 ns
Average time: 50.9871 ns
Average time: 122.80649 ns
res0: Double = 42.04872

scala> (for(i <- 1 to 20) yield time(n){v.take(10)}).min
Average time: 429.23201 ns
Average time: 257.70543 ns
Average time: 271.02261 ns
Average time: 198.8826 ns
Average time: 161.67466 ns
Average time: 190.5253 ns
Average time: 112.82044 ns
Average time: 82.45798 ns
Average time: 81.17192 ns
Average time: 129.28968 ns
Average time: 104.86973 ns
Average time: 80.33942 ns
Average time: 81.64533 ns
Average time: 92.22053 ns
Average time: 78.5791 ns
Average time: 84.55555 ns
Average time: 78.88382 ns
Average time: 77.25284 ns
Average time: 83.62473 ns
Average time: 72.39703 ns
res1: Double = 72.39703

scala> (for(i <- 1 to 20) yield time(1000){l.takeRight(10)}).min
Average time: 264902.261 ns
Average time: 225706.676 ns
Average time: 228625.873 ns
Average time: 230358.379 ns
Average time: 229971.679 ns
Average time: 237404.948 ns
Average time: 242580.96 ns
Average time: 242455.325 ns
Average time: 242316.002 ns
Average time: 242046.311 ns
Average time: 242378.896 ns
Average time: 242740.221 ns
Average time: 242131.301 ns
Average time: 242466.169 ns
Average time: 242075.599 ns
Average time: 242247.534 ns
Average time: 242739.886 ns
Average time: 241982.93 ns
Average time: 242118.373 ns
Average time: 241941.998 ns
res2: Double = 225706.676

scala> (for(i <- 1 to 20) yield time(1000){v.takeRight(10)}).min
Average time: 661.737 ns
Average time: 420.765 ns
```

```
Average time: 225.867 ns
Average time: 256.524 ns
Average time: 235.596 ns
Average time: 231.764 ns
Average time: 154.279 ns
Average time: 139.37 ns
Average time: 139.183 ns
Average time: 153.957 ns
Average time: 142.883 ns
Average time: 140.837 ns
Average time: 154.178 ns
Average time: 138.72 ns
Average time: 202.93 ns
Average time: 174.179 ns
Average time: 175.98 ns
Average time: 171.658 ns
Average time: 177.097 ns
Average time: 173.1 ns
res3: Double = 138.72
```

- b) Varför går det olika snabbt olika körningar?

Uppgift 31. Studera skillnader i prestanda mellan olika samlingar här:
docs.scala-lang.org/overviews/collections/performance-characteristics.html
(Mer om detta i kommande kurser.)

Uppgift 32. För samlingen List finns en alternativ metod till +: som heter :: och kallas ”cons” och som i kombination med objektet Nil kan användas för att med alternativ syntax bygga listor. Läs om detta här:
alvinalexander.com/scala/how-create-scala-list-range-fill-tabulate-constructors
och hitta på några egna övningar för att undersöka hur cons och Nil fungerar.
Metoder som slutar med kolon är högerassociativa. Läs mer om detta här:
<http://www.artima.com/pins1ed/basic-types-and-operations.html#5.8>

4.2 Laboration: pirates

Mål

- Kunna använda en integrerad utvecklingsmiljö (IDE).
- Kunna använda färdiga funktioner för att läsa till, och skriva från, textfil.
- Kunna använda enkla case-klasser.
- Kunna skapa och använda enkla klasser med föränderlig data.
- Kunna använda samlingstyperna Vector och Map.
- Kunna skapa en ny samling från en befintlig samling.
- Förstå skillnaden mellan kompileringsfel och exekveringsfel.
- Kunna felsöka i små program med hjälp av utskrifter.
- Kunna felsöka i små program med hjälp av en debugger i en IDE.

Förberedelser

- Gör övning data i avsnitt 4.1. och repetera programs, speciellt **for yield**.
Det är livsnödvändig kunskap för en pirat!
- Läs om integrerade utvecklingsmiljöer i appendix D.
- Välj vilken IDE du vill använda på denna lab. Om du inte vet vilken, välj **Eclipse** med ScalalIDE, som flest handledare känner väl till.
- Bekanta dig med utvecklingsmiljön genom att skapa ett nytt projekt och gör ett "Hello World"-program.
- Ladda hem kursens *workspace* enligt instruktioner i appendix D.3.3 och kontrollera så att du med *Run* kan köra igång de båda ofärdiga main-metoderna i projektet w04_pirates inifrån din IDE. Om du inte får rätt på *Run Configuration...* etc. så fråga någon om hjälp.
- Läs igenom hela laborationen.

4.2.1 Bakgrund

Efter en rad olyckliga omständigheter har du blivit en tidsresande pirat och hamnat i 1700-talets Karibien. Du ska försöka överleva med hjälp av dina hemliga programmeringskunskaper från 2000-talet. För att klara det behöver du datastrukturerna Vector och Map, samt använda en integrerad utvecklingsmiljö. Ditt uppdrag är tvådelat:

1. Du ska skriva till, och sedan läsa från, en hemlig (nåja) textfil för att spara undan namnen på dina skeppskamrater så att de kan benådas. Du ska även undersöka vad som händer med ditt program om någon illvillig typ får tag i din textfil och försöker inplantera korrupta tecken.
2. Du ska förutsäga vilka ord som kommer ur piraters mun med hjälp av smart ordstatistik och så kallade *bigram*³. Du ska med hjälp av bigram räkna ut det vanligaste ordet som följer på ett annat.

³en.wikipedia.org/wiki/Bigram

Till första delen har du det färdiga objektet `FileUtils` till din hjälp, som visas i sin helhet nedan. Begrunda dessa hjälpfunktioner, ovärderliga för en pirat i nöd, och försök lista ut hur de fungerar, gärna genom experiment i REPL.

```

1 package pirates
2
3 object FileUtils {
4     import java.nio.file.{ Paths, Files }
5
6     def save(s: String, fileName: String): Unit =
7         Files.write(Paths.get(fileName), s.getBytes("UTF-8"))
8
9     def readLines(fileName: String): Vector[String] =
10        scala.io.Source.fromFile(fileName)("UTF-8").getLines.toVector
11
12    def readWords(fileName: String): Vector[String] = {
13        val data: String          = readLines(fileName).mkString(" ")
14        def convertSpace(c: Char) = if (c.isWhitespace) ' ' else c
15        val spaceConverted       = data.map(convertSpace)
16        def isLetterOrSpace(c: Char) = c.isLetter || c.isSpaceChar
17        val lettersAndSpaces     = spaceConverted.filter(isLetterOrSpace)
18        val words: Array[String]   = lettersAndSpaces.map(_.toLowerCase).split(' ')
19        words.filterNot(_.isEmpty).toVector
20    }
21 }
```

4.2.2 Obligatoriska uppgifter

Del A: Rädda din besättning

I denna del ska du först skapa en textfil med namnet på skeppskamraterna. Sedan ska du läsa in filen och skapa en vektor med dina kamrater representerade som objekt med hjälp av en case-klass. Du ska också undersöka vad som händer om filen visar sig vara manipulerad.

Uppgift 1. *Save your crew.* I följande deluppgifter ska vi steg-för-steg fylla i kodskelettet i filen `SaveMyCrew.scala`:

<i>Specification SaveMyCrew</i>
<pre> package pirates object SaveMyCrew { def main(arhg: Array[String]): Unit = { // change this code to your tests! val filename = if (!argh.isEmpty) argh(0) else "crew.txt" } def saveCrew(fileName: String): Unit = ??? // add code for asking the user for crew members and save them to file }</pre>

```

def readCrew(fileName: String): Unit = ???  

  // add code for reading the crew from the file  

}  
  

// Add your case class here!

```

- a) Kung George är villig att benåda **fem** personer ur din besättning!
 Vi ska skapa en lista (nåja, vektor) där personerna sparar med förnamn, efternamn och befattning genom att läsa in dem från konsolen.
 Börja med att implementera en **case class** CrewMember med förnamn, efternamn och befattningen.
- b) Inläsning från konsolen görs med `scala.io` med kodraden:

```
val first = scala.io.StdIn.readLine("Förnamn: ") .
```

Skriv funktionen `saveCrew` som läser in namn och befattning på dina fem besättningsmedlemmar och sparar dem i en **vektor**. *Hint:* du kan spara undan värdet från en `for` loop till en vektor direkt med hjälp av **yield** så här

```

val crew = for(...) yield {  

... // lines to read input from user  

  CrewMember(...) // this will be an element in the vector!  

}

```

- c) Vi vill skriva namnen till en fil så att den faktiskt sparas. Till din hjälp får hjälpprojektet `FileUtils` med funktionen `save` som tar en sträng `s` och sparar den till en fil `fileName`. Lägg till kodrader i `saveCrew` så att vektoron skrivs till en sträng och sparas till filen `crew.txt`. *Hint:* samlingsklasser har metoden `mkString("\n")` som skapar en sträng av innehållet där varje element separeras med en radbrytning. Använd F5 för att uppdatera projektet så att filen dyker upp i Eclipse.
- d) Det går att överskugga `toString()` i `CrewMember` och på så sätt ändra utskriften genom att lägga till följande kodrad inne i klassen:

```
override def toString(): String = ??? // add your code here.
```

Ändra utskriften så att den blir *snygg*, t ex med komma

Jack Sparrow, kapten
 Anne Bonny, mordlysten matros
 Ed Kenway, lönnmördare
 ...

Uppgift 2. Avlusa din besättning.

- a) Din moraliska kompass hindrar dig inte från att också jobba för kung George. Hjälp honom att läsa listan!

En fil `fileName` kan läsas rad för rad till en vektor med strängar (en för varje rad) med följande kod som också finns i objektet `FileUtils`:

```
def readLines(fileName: String): Vector[String] =
  scala.io.Source.fromFile(fileName)("UTF-8").getLines.toVector
```

Fyll i koden i `readCrew` som läser in besättningen från filen och skapar en `CrewMember` för varje rad. Skriv ut personen till konsolen för att se om det blev korrekt. *Hint:* strängar har många metoder, t ex kan man ersätta alla komman med mellanslag med funktionen `replaceAllLiterally(", ", "")`, separera med hjälp av `split(' ')` och `drop(2)` som ger en ny vektor utan de två första elementen.

- b) Ändra också i `main` och testa ditt program. Stämmer det med filinnehållet? Använd debuggern för att hitta eventuella fel.
- c) Den konkurrerande kaptenen Charles Vane⁴ betalar dig för att sabotera listan genom att lägga till nonsensrader i filen. Gör det. Vad händer då när du exekverar ditt testprogram?
- d) Lägg till några lämpliga if-satser i `readCrew` så att en korrekt formaterad rad skapar en `CrewMember` medan felaktiga rader skriver ut felmeddelande. Testa ditt nya program och se om det blir som förväntat genom lämpliga brytpunkter och utskrifter. Går det att lura ditt program (det är OK)?

Del B: Hur pratar pirater?

Lögner, förbannade lögner och statistik. För att du inte ska bli överlistad av dina sluga, lögnaktiga skeppskamrater behöver du kunna gissa hur en pirat tänker. Därför förkovrar du dig i Robert Louis Stevensons *Skattkammarön*⁵ som finns i filen *skattkammaron.txt* i workspaceet. Genom att för varje ord spara det mest frekventa nästkommande ordet går det att förutse vad som kommer sägas⁶.

I vårt program ska varje ord få en egen `WordCounter` som i sin tur innehåller en samling, i vårt fall en föränderlig `Map` med ord och antal som nyckel-värde-par. Ditt uppdrag är att steg-för-steg implementera `WordCounter`, räkna bigram i *Skattkammarön* och skriva ett program för att gissa pirat-prat genom att implementera funktionerna i objektet `PirateSpeech`.

Kodskelettet vi ska fylla ut steg-för-steg i uppgifterna nedan finns i klassen `WordCounter` och i objektet `PirateSpeech` och ser ut som följer:

⁴hängd för sjöröveri i Port Royal 1721.

⁵Vars copyright har gått ut så du behöver inte piratkopiera den.

⁶Detta används till exempel i Swiftkey på smarttelefoner.

Specification WordCounter

```
package pirates

/** Sparar olika ord och räknar antalet förekomster. */
class WordCounter {

    /** Lägger till ordet word och räknar upp antalet förekomster. */
    def addWord(word: String): Unit = ???

    /** Ger det vanligaste ordet och antalet förekomster. */
    def mostCommonWord: (String, Int) = ???

    /** Skriver ut det vanligaste ordet och dess antal*/
    override def toString(): String = ???
}
```

Specification PirateSpeech

```
package pirates

object PirateSpeech {
    def main(arhg: Array[String]): Unit = {
        val filename = if (!argh.isEmpty) arhg(0) else "skattkammaron.txt"
        // add your tests here!!!
    }

    def readBook(bookFile: String): Map[String, WordCounter] = {
        val words = ??? // ("herr", "trelawney", "doktor", "livesey", "och", ...)
        val wordSet = ??? //all words only once

        val counterMap = ???

        ??? // go through the book looking at two words at a time

        counterMap // return the map
    }

    /* Optional */
    def readBook(bookFile: String, saveToFile: String): Unit = ???

    /* Optional */
    def testSpeech(file: String): Unit = ???
}
```

WordCounter ska fungera såhär: I texten förekommer ordkombinationerna *och de*, *och jag*, *och återvänder* och lite senare *och jag* igen. Vi skapar en WordCounter för *och* och lägger till *de*, *jag*, *återvänder* och *jag* och sen ska den bästa gissningen för *och vara jag* med antalet 2. WordCounter behöver alltså en *förändrig* tabell i form av en `scala.collection.mutable.Map[String, Int]` för att räkna förekomsterna av varje ord.

- e) Lägg till ett Map-attribut i WordCounter för att spara ord (strängar) och antal. Tänk på att välja ett lämpligt namn på attributet. Hint: Typerna i en Map anges inom hakparenteser `Map[String, Int]()`.

- f) När vårt Map-attribut skapas är den tom. Implementera funktionen `addWord` i `WordCounter` som räknar upp antalet förekomster av ett visst ord. Om det är första gången ordet förekommer behöver det läggas in i samlingen med antalet 1. *Hint:* Samlingar har funktionen `contains` för att se om ett element finns i den och värden både uppdateras och returneras genom att använda nyckeln inom parentes.
- g) Implementera funktionen `mostCommonWord` som returnerar en tuppel med bästa gissning och hur ofta den förekommer. Detta kan vi göra på olika sätt, t ex genom att gå igenom alla tuppler i vår Map och spara undan den med högst *värde* (se uppgift 12 i övning 2), men en lat pirat använder den färdiga samlingsfunktionen `maxBy` för att jämföra tupplernas värden. I en Map `wordCounter` används det så här: `wordCounter.maxBy(pair => pair._2)` som för varje nyckel-värde-par *pair* bara jämför *värdet* som hämtas med `_2`. Det går att skriva samma sak superkort `wordCounter.maxBy(_._2)`.
- h) Implementera `toString` som returnerar ett par med den bästa gissningen och hur ofta den förekom.
- i) Testa din klass genom att lägga till kodrader i `main` för att skapa en `WordCounter` för ett ord, lägg till data med `addWord` och avsluta med att skriva ut resultatet. Använd t ex *och*-exemplet ovan.
- j) Nu är det dags att implementera funktionen `readBook`. Konceptuellt vill vi göra följande: 1) läsa in all data från textfilen med boken till en vektor med ord, 2) få ut en samling unika ord så att vi kan 3) skapa en `WordCounter` för varje ord och slutligen, 4) gå igenom hela boken och räkna ordpar (bigrammen). Du får följande ledtrådar:
1. `FileUtils.readWords` är en funktion för att läsa in ord från en fil med hjälp av `scala.io.Source.fromFile`. Funktionen tar bort allt som *inte* är svenska bokstäver och separerar orden med hjälp av `split` på mellanslag samt gör om alla tecken till gemener.
 2. Samlingsklassen `Set` tillåter bara unika element, så om man skapar ett tomt `Set(... = Set())` och sen lägger till sina ord med `++` filtreras dubbletter bort.
 3. Spara räknaren för varje ord i en `Map[String, WordCounter]` där ordsträngen och räknaren är sparade som nyckel-värde-par. Använd funktionen `map` på setet för att för varje ord `word` skapa ett par (en 2-tuppel) med en ny räknare (`word -> new WordCounter`). Resultatet från funktionen `map` ger en ny samling av samma typ som ursprungssamlingen, den kan omvandlas till en Map genom att anropa `toMap`!
 4. I Övn 2.1 lärde vi oss att iterera genom en vektor med `for(word <- words)` men nu vill vi få två ord i taget! Vektor-funktionen `sliding(2)` ger en minivektor av storlek 2 som vi kan använda för att iterera över ordpar:

```
for(pair <- words.sliding(2)) { ... }
```

5. När vi har ett ordpar med två ord, first och second, behöver vi placera fram räknaren för first från vår counterMap som i sin tur sköter uppräkningen av ordet second i addWord.
- k) Testa att ditt program genom att lägga till kodrader i main som skriver ut några gissningar, t ex vad kommer troligtvis efter ”och” respektive ”jim”?

Du blir så bra på tankeläsning att dina skeppskamrater anklagar dig för svart magi och din sjörövarhistoria slutar en stormig natt med att du sveps över bord under mystiska omständigheter.

4.2.3 Frivilliga extrauppgifter

Det är onödigt att läsa in hela boken och räkna alla frekvenser varje gång programmet körs. Eftersom boken inte ändras kan du spara resultatet från uträkningen till en fil och bara läsa in *resultatfilen*, dvs, efter att du skrivit readBook kan du spara varje ord tillsammans med den bästa gissningen till en fil. Sen kan du testa programmet genom att läsa in dessa par ...

- l) Fyll i funktionen

```
def readBook(bookFile: String, saveToFile: String)}
```

så att den, utöver det som din gamla funktion gör, också skapar en sträng där varje rad innehåller ett ord samt dess bästa gissning och sen sparar strängen till filen saveToFile. Kontrollera att utskriften till filen ser OK ut.

- m) Skriv funktionen testSpeech(savedFile: String) så att den läser in raderna från din sparade fil med Utils.readLines. Ordet och gissningen ska sparas i en Map[String, String]. Testa slutligen ditt program med några väl valda ord.

Hint: dela upp strängen på ord med hjälp av split(...). Kom ihåg att det går att generera samlingar direkt med hjälp av en forloop

```
val s = for(...) yield{
  ...
  // element
}
```

Vilken samlingstyp blir resultatet? I labben lärde du dig två sätt att konvertera mellan olika typer av samlingar, antingen kan du skapa en tom samling och lägga in elementen med ++ eller anropa konverteringsfunktioner som toMap.

Kapitel 5

Sekvensalgoritmer

Begrepp som ingår i denna veckas studier:

- sekvensalgoritm
- algoritm: SEQ-COPY
- in-place vs copy
- algoritm: SEQ-REVERSE
- algoritm: SEQ-REGISTER
- sekvenser i Java vs Scala
- for-sats i Java
- java.util.Scanner
- scala.collection.mutable.ArrayBuffer
- StringBuilder
- java.util.Random
- slumptalsfrö

Vad är en sekvensalgoritm?

- En algoritm är en stegvis beskrivning av hur man löser ett problem.
- En sekvensalgoritm är en algoritm där dataelement i sekvens utgör en viktig del av problembeskrivningen och/eller lösningen.
- Exempel: sortera en sekvens av personer efter deras ålder.
- Två olika principer:
 - Skapa **ny sekvens** utan att förändra indatasekvensen
 - Ändra **på plats** (eng. *in place*) i den **förändringsbara** indatasekvensen

Skapa ny sekvenssamling eller ändra på plats?

- Ofta är det **lättast att skapa ny samling** och kopiera över elementen medan man loopar.
- Om man har mycket stora samlingar kan man behöva ändra på plats för att spara tid/minne.
- Det är bra att själv kunna implementera sekvensalgoritmer även om många av dem finns färdiga, för att bättre förstå vad som händer ”under huven”, och för att i enstaka fall kunna optimera om det verkligen behövs.
- Vi illustrerar därför hur man kan implementera några sekvensalgoritmer med primitiva arrayer även om man sällan gör så i praktiken (i Scala).

Algoritm: SEQ-COPY

Pseudokod för algoritmen SEQ-COPY som kopierar en sekvens, här en Array med heltal:

Indata : Heltalsarray xs

Resultat: En ny heltalsarray som är en kopia av xs .

```

1 result ← en ny array med plats för  $xs.length$  element
2 i ← 0
3 while  $i < xs.length$  do
4   | result( $i$ ) ←  $xs(i)$ 
5   |  $i$  ←  $i + 1$ 
6 end
7 return result
```

Oföränderlig eller förändringsbar?

- **Oföränderlig:** Kan ej ändra elementreferenserna, men effektiv på att skapa kopia som är (delvis) förändrad
(vanliga i Scala, men inte i Java): **Vector** eller **List**
- **Förändringsbar:** kan ändra elementreferenserna
 - Kan **ej ändra storlek** efter allokerings:
Scala+Java: **Array**: indexera och uppdatera varsomhelst
 - Kan ändra storlek efter allokerings:
Scala: **ArrayBuffer** eller **ListBuffer**
Java: **ArrayList** eller **LinkedList**
- Ofta funkar oföränderlig sekvenssamling utmärkt, men om man efter prestandamätning upptäcker en flaskhals kan man ändra från **Vector** till t.ex. **ArrayBuffer**.

Egenskaper hos några sekvenssamlingar

- Vector
 - **Oföränderlig.** Snabb på att skapa kopior med små förändringar.
 - Allsidig prestanda: **bra till det mesta.**
- List
 - **Oföränderlig.** Snabb på att skapa kopior med små förändringar.
 - Snabb vid bearbetning **i början**.
 - Smidig & snabb vid **rekursiva** algoritmer.
 - Långsam vid upprepad **indexering** på godtyckliga ställen.
- Array
 - **Föränderlig: snabb indexering & uppdatering.**
 - Kan **ej ändra storlek**; storlek anges vid allokerings.
 - Har särställning i JVM: ger snabbaste minnesaccessen.
- ArrayBuffer
 - **Föränderlig: snabb indexering & uppdatering.**
 - Kan **ändra storlek** efter allokerings. Snabb att indexera överallt.
- ListBuffer
 - **Föränderlig:** snabb indexering & uppdatering **i början**.
 - Snabb om du bygger upp sekvensen genom många tillägg i början.

Vilken sekvenssamling ska jag välja?

- Vector
 - Om du vill ha oföränderlighet: `val xs = Vector[Int](1,2,3)`
 - Om du behöver ändra (men ej prestandakritiskt):
`var xs = Vector.empty[Int]`
 - Om du ännu inte vet vilken sekvenssamling som är bäst; du kan alltid ändra efter att du mätt prestanda och kollat flaskhalsar.
- List
 - Om du har en rekursiv sekvensalgoritm och/eller bara lägger till i början.
- Array
 - Om det behövs av prestandaskäl och du **vet** storlek vid allokering:
`val xs = Array.fill(initSize)(initValue)`
- ArrayBuffer
 - Om det behövs av prestandaskäl och du **inte** vet storlek vid allokkering:
`val xs = scala.collection.mutable.empty[Int]`
- ListBuffer
 - om det behövs av prestandaskäl och du bara behöver lägga till i början:
`val xs = scala.collection.mutable.ListBuffer.empty[Int]`

5.1 Övning: sequences

Mål

- Kunna implementera funktioner som tar argumentsekvenser av godtycklig längd.
- Kunna tolka enkla sekvensalgoritmer i pseudokod och implementera dem i programkod, t.ex. tillägg i slutet, insättning, borttagning, omvärdning, etc., både genom kopiering till ny sekvens och genom förändring på plats i befintlig sekvens.
- Kunna använda föränderliga och oföränderliga sekvenser.
- Förstå skillnaden mellan om sekvenser är föränderliga och om innehållet i sekvenser är föränderligt.
- Kunna välja när det är lämpligt att använda `Vector`, `Array` och `ArrayBuffer`.
- Känna till att klassen `Array` har färdiga metoder för kopiering.
- Kunna implementera algoritmer som registrerar antalet förekomster av något utfall i en sekvens som indexeras med utfallet.
- Kunna generera sekvenser av pseudoslumptal med specificerat slumptalsfrö.
- Kunna implementera sekvensalgoritmer i Java med `for`-sats och primitiva arrayer.
- Kunna beskriva skillnaden i syntax mellan arrayer i Scala och Java.
- Kunna använda klassen `java.util.Scanner` i Scala och Java för att läsa in heltalssekvenser från `System.in`.

Förberedelser

- Studera begreppen i kapitel 5.

5.1.1 Grunduppgifter

Uppgift 1. *Variabelt antal argument.* Det går fint att deklarera en funktion som tar en argumentsekvens av godtycklig längd. Syntaxen består av en asterisk * efter typen.

- a) Vad händer nedan?

```

1 scala> def printAll(xs: Int*) = xs.foreach(println)
2 scala> printAll(42)
3 scala> printAll(1, 2, 7, 42)
4 scala> def printStrings(wa: String*) = println(wa)
5 scala> printStrings("hej", "på", "dej")

```

- b) Vad har parametern `wa` i `printStrings` ovan för typ?
 c) Ändra i `printAll` så att även längden på `xs` skrivas ut före utskriften av alla element. Testa att anropa `printAll` med olika antal parametrar.
 d) Vad händer om du anropar `printAll` med noll parametrar?

Uppgift 2. Oföränderliga sekvenser med föränderliga objekt.

- a) Vad får xs för värde efter att attributet i objektet som c2 refererar till ändras på rad 4 nedan? Föklara vad som händer.

```

1 scala> class IntCell(var x: Int){override def toString = "[Int](" + x + ")"}
2 scala> val (c1, c2, c3) = (new IntCell(7), new IntCell(8), new IntCell(9))
3 scala> val xs = Vector(c1, c2, c3)
4 scala> c2.x = 42
5 scala> xs

```

- b) Rita en bild av minnessituationen efter rad 4 ovan.



- c) Vad krävs för att allt innehåll i en oföränderlig samling garanterat ska förbli oförändrat?

**Uppgift 3.** Föränderliga, indexerbara sekvenser: Array och ArrayBuffer

- a) Samlingen `scala.Array` har speciellt stöd i JVM och är extra snabb att alloker och indexera i. Dock kan man inte ändra storleken efter att en `Array` allokerats. Behöver man mer plats kan man kopiera den till en ny, större array. Koden nedan visar hur det kan gå till.

```

1 scala> val xs = Array(42, 43, 44)
2 scala> val ys = new Array[Int](4) //plats för 4 heltal, från början nollor
3 scala> for (i <- 0 until xs.size){ys(i) = xs(i)}
4 scala> ys(3) = 45

```

Definiera funktionen `def copyAppend(xs: Array[Int], x: Int): Array[Int]` som implementerar nedan algoritm, *etter* att du rätta de **två buggarna** i algoritmens while-loop:

Indata : Heltalsarray `xs` och heltalet `x`

Resultat: En ny array som är en kopia av `xs` men med `x` tillagt på slutet som extra element.

```

1 n ← antalet element i xs
2 ys ← en ny array med plats för n + 1 element
3 i ← 0
4 while i ≤ n do
5   | ys(i) ← xs(i)
6 end
7 ys(n) ← x

```

- b) Samlingen `scala.collection.mutable.ArrayBuffer` är inte riktigt lika snabb i alla lägen som `scala.Array` men storleksändring hanteras automatiskt, vilket är en stor fördel då man slipper att själv implementera algoritmer liknande `copyAppend` ovan. Speciellt använder man ofta `ArrayBuffer` om man stegvis vill bygga upp en sekvens. Vad händer nedan?

```

1 scala> val xs = scala.collection.mutable.ArrayBuffer.empty[Int]
2 scala> xs.append(1, 1)
3 scala> while (xs.last < 100) {xs.append(xs.takeRight(2).sum); println(xs)}
4 scala> xs.last
5 scala> xs.length

```

- c) Talen i sekvensen som produceras ovan kallas Fibonaccitäl¹. Hur lång ska en Fibonacci-sekvens vara för att det sista elementet ska komma så nära (men inte över) `Int.MaxValue` som möjligt?

Uppgift 4. *Kopiering och uppdatering.* Metoder på oföränderliga samlingar skapar nya samlingar istället för att ändra. Därför behöver man inte själv skapa kopior. När en *föränderlig* samling uppdateras på plats, syns denna förändring via alla referenser till samlingen.

```

1 scala> val xs = Vector(1, 2, 3)
2 scala> val ys = xs.toArray
3 scala> ys(1) = 42
4 scala> xs
5 scala> ys
6 scala> val zs = ys.toArray
7 scala> zs(1) = 84
8 scala> xs
9 scala> ys
10 scala> zs

```

- a) Syns uppdateringen av objektet som `ys` refererar till via referensen `xs`? Varför?
- b) Syns uppdateringen av objektet som `zs` refererar till via referensen `ys`? Varför?
- c) Syns uppdateringen av objektet som `zs` refererar till via referensen `xs`? Varför?

Uppgift 5. *Färdig metod för att skapa kopia av array.* Om man inte vill att en uppdatering av en föränderlig samling ska få oönskad påverkan på andra kodelar som refererar till samlingen, behöver man göra kopior av samlingen före uppdatering. Det finns färdiga metoder för kopiering av objekt av typen `Array` i paketet `java.util.Arrays`.

- 💡 a) Studera dokumentationen för metoden `java.util.Arrays.copyOf` här: docs.oracle.com/javase/8/docs/api/java/util/Arrays.html#copyOf-int:A-int-
Notera att syntaxen för arrayer i Java är annorlunda: När det står `int[]` i Java så motsvarar det `Array[Int]` i Scala. Vad används den andra parametern till?
- 💡 b) Rita en bild av hur minnet ser ut efter varje tilldelning nedan. Vad har `xs`, `ys` och `zs` för värden efter exekveringen av raderna 1–5 nedan? Varför?

```

1 scala> val xs = Array(1, 2, 3, 4)
2 scala> val ys = xs
3 scala> val zs = java.util.Arrays.copyOf(xs, xs.size - 1)
4 scala> xs(0) = 42
5 scala> zs(0) = 84
6 scala> xs
7 scala> ys
8 scala> zs

```

¹sv.wikipedia.org/wiki/Fibonaccital

Uppgift 6. Algoritmen: SEQ-REVERSE-COPY. Implementera nedan algoritmen:

Indata : Heltalsarray xs Resultat : En ny heltalsarray med elementen i xs i omvänt ordning. 1 $n \leftarrow$ antalet element i xs 2 $ys \leftarrow$ en ny heltalsarray med plats för n element 3 $i \leftarrow 0$ 4 while $i < n$ do 5 $ys(n - i - 1) \leftarrow xs(i)$ 6 $i \leftarrow i + 1$ 7 end 8 return ys

- a) Skriv implementation med penna och papper. Använd en **while**-sats på samma sätt som i algoritmen. Prova sedan din implementation på dator och kolla så att den fungerar. 
- b) Skriv implementationen med penna och papper igen, men använd nu istället en **for**-sats som räknar baklänges. Prova sedan din implementation på dator och kolla så att den fungerar. 
- c) Definiera en funktion i REPL med namnet `reverseCopy` med din implementation i uppgift b.

Uppgift 7. Algoritmen: SEQ-REVERSE. Strängar av typen `String` är oföränderliga. Vill man ändra i en sträng utan att skapa en ny kopia kan man använda en `StringBuilder` enligt nedan algoritmen som vänder bak-och-fram på en sträng.

Indata : En sträng s av typen <code>String</code> Resultat : En ny sträng av typen <code>String</code> 1 $sb \leftarrow$ en ny <code>StringBuilder</code> som innehåller s 2 $n \leftarrow$ antalet tecken i s 3 for $i \leftarrow 0$ to $\frac{n}{2} - 1$ do 4 $temp \leftarrow sb(i)$ 5 $sb(i) \leftarrow sb(n - i - 1)$ 6 $sb(n - i - 1) \leftarrow temp$ 7 end 8 return sb omvandlad till en <code>String</code>
--

- a) Implementera algoritmen ovan i en funktion med signaturen:
`def reverseString(s: String): String`
- b) Använd din funktion `reverseString` från föregående deluppgift i en ny funktion med signaturen:
`def isPalindrome(s: String): Boolean`
 som avgör om en sträng är en palindrom.²
- c) Man kan med en **while**-sats och indexering direkt i en `String` avgöra om en sträng är en palindrom utan att kopiera den till en `StringBuilder`. 

²sv.wikipedia.org/wiki/Palindrom

Implementera en ny variant av `isPalindrome` som använder denna metod. Skriv först algoritmen på papper i pseudo-kod.

Uppgift 8. *Algoritm: SEQ-REGISTER.* Algoritmer för registrering löser problemet att räkna förekomst av olika saker, till exempel antalet tärningskast som gav en sexa. Antag att vi har följande vektor `xs` som representerar 13 st tärningskast:

```
1  scala> val xs = Vector(5, 3, 1, 6, 1, 3, 5, 1, 1, 6, 3, 2, 6)
```

- a) Använd metoderna `filter` och `size` på `xs` för att filtrera ut alla 6:or och räkna hur många de är.
- b) Använd metoderna `filter` och `size` på `xs` för att filtrera ut alla jämnä kast och räkna hur många de är.
- c) Metoden `groupBy` på en samling tar en funktion `f` som parameter och skapar en ny `Map` med nycklar `k` som är associerade till samlingar som utgör grupper av värden där `f(x) == k`. Vad händer här:

```
1  scala> xs.groupBy(x => x % 2)
2  scala> xs.groupBy(_ % 2)
3  scala> xs.groupBy(_ % 3)
4  scala> xs.groupBy(_ % 3).foreach(println)
5  scala> val freqEvenOdd = xs.groupBy(_ % 2).map(p => (p._1, p._2.size))
6  scala> val nEven = freqEvenOdd(0)
7  scala> val nOdd = freqEvenOdd(1)
```

- d) Använd metoden `groupBy` på `xs` med den s.k. identitetsfunktionen `i => i` som returnerar sitt eget argument. Vad händer?
- e) Definiera en `val freq: Map[Int, Int]` som räknar antalet olika tärningsutfall i `xs`. Använd metoden `groupBy` på `xs` med identitetsfunktionen följt av en `map` med funktionen `p => (p._1, p._2.size)`.
- f) Du ska nu själv implementera en registreringsalgoritm. Skriv en funktion:

```
def tärningsRegistrering(xs: Array[Int]): Array[Int] = ???
```

som implementerar nedan algoritm (som alltså inte använder `groupBy` eller andra färdiga metoder på samlingar förutom `size` och `apply`).

Indata : En array xs med heltalet mellan 1 och 6 som representerar utfall av många tärningskast.

Resultat: En array f med 7 st element där $f(0)$ innehåller totala antalet kast, $f(1)$ anger antalet ettor, $f(2)$ antalet tvåor, etc. till och med $f(6)$ som anger antalet sexor.

```

1  $f \leftarrow$  en ny array med 7 element där alla element initialiseras till 0.
2  $f(0) \leftarrow$  antalet element i  $xs$ 
3  $i \leftarrow 0$ 
4 while  $i < f(0)$  do
5    $f(xs(i)) \leftarrow f(xs(i)) + 1$ 
6    $i \leftarrow i + 1$ 
7 end
8 return  $f$ 
```

Testa din funktion med nedan funktionsanrop:

```

1 scala> tärningsRegistrering(Array.fill(1000)((math.random * 6).toInt +1))
2 res12: Array[Int] = Array(1000, 174, 174, 167, 171, 145, 169)
```

Uppgift 9. *Algoritm: SEQ-REMOVE-COPY.* Ibland vill man kopiera alla element till en ny `Array` *utom* ett element på en viss plats `pos`.

- a) Skriv algoritmen SEQ-REMOVE-COPY i pseudokod med penna på papper. 
- b) Implementera algoritmen SEQ-REMOVE-COPY i en funktion med denna signatur:

```
def removeCopy(xs: Array[Int], pos: Int): Array[Int]
```

Uppgift 10. *Algoritm: SEQ-REMOVE.* Ibland vill man ta bort ett element på en viss position i en befintlig `Array` utan att kopiera alla element till en ny `Array`. Ett sätt att göra detta är att flytta alla efterföljande element ett steg mot lägre index och låta sista platsen bli 0.

- a) Skriv algoritmen SEQ-REMOVE i pseudokod med penna på papper. 
- b) Implementera algoritmen SEQ-REMOVE i en funktion med denna signatur:

```
def remove(xs: Array[Int], pos: Int): Unit
```

Uppgift 11. *Deterministiska pseudoslumptalssekvenser med `java.util.Random`.*

Klassen `java.util.Random` ger möjlighet att generera en sekvens av tal som verkar slumpmässiga. Genom att välja ett visst s.k. **frö** (eng. *seed*) kan man få samma sekvens av pseudoslumptal varje gång.

- a) Sök upp och studera dokumentationen för `java.util.Random`. Hur skapar man en ny instans av klassen `Random`? Vad gör operationen `nextInt` på `Random`?

objekt.

- b) Föklara vad som händer nedan?

```

1  scala> import java.util.Random
2  scala> val frö = 42L
3  scala> val rnd = new Random(frö)
4  scala> rnd.nextInt(10)
5  scala> (1 to 100).foreach(_ => print(rnd.nextInt(10)))
6  scala> val rnd1 = new Random(frö)
7  scala> val rnd2 = new Random(frö)
8  scala> val rnd3 = new Random(System.nanoTime)
9  scala> val rnd4 = new Random((math.random * Long.MaxValue).toLong)
10 scala> def flip(r: Random) = if (r.nextInt(2) > 0) "krona" else "klave"
11 scala> val xs = (1 to 100).map{i =>
12     (flip(rnd1), flip(rnd2), flip(rnd3), flip(rnd4))}
13 scala> xs foreach println
14 scala> xs.exists(q => q._1 != q._2)
15 scala> xs.exists(q => q._1 != q._3)

```

-  c) Nämn några sammanhang då det är användbart att kunna bestämma fröet till en slumptalssekvens.
- d) Blir det samma sekvens om du använder fröet 42L som argument till konstruktorn vid skapandet av en instans av `java.util.Random` på en *annan* dator?
- e) Sök reda på dokumentationen för `java.math.random` och undersök hur denna sekvens skapas.
- f) Vad blir det för frö till slumptalssekvensen om man skapar ett `Random`-objekt med hjälp av konstruktorn utan parameter?

Uppgift 12. Undersök om tärningskast är rektangelfördelade.³

Skriv en funktion `def testRandom(r: Random, n: Int): Unit = ???` som ger följande utskrift:

```

1  scala> val rnd = new Random(42L)
2  scala> testRandom(rnd, 1000)
3  Antal kast: 1000
4  Antal 1:or: 178
5  Antal 2:or: 187
6  Antal 3:or: 167
7  Antal 4:or: 148
8  Antal 5:or: 155
9  Antal 6:or: 165

```

Tips: Anropa din funktion `tärningsRegistrering` från uppgift 8.

Uppgift 13. Array och `for`-sats i Java.

³För ett rektangelfördelat slumpvärde gäller att om man drar (nästan oändligt många) slumpvärden så blir det (nästan) lika många av varje möjligt värde. Om man ritar en sådan fördelning i ett koordinatsystem med antalet utfall på y-axeln och de olika värdena på x-axeln, så blir bilden rektangelformad. Du får lära dig mer om sannolikhetsfördelningar i kommande kurser i matematisk statistik.

- a) Skriv nedan program i en editor och spara i filen DiceReg.java:

```
// DiceReg.java
import java.util.Random;

public class DiceReg {
    public static void main(String[] args) {
        int[] diceReg = new int[6];
        int n = 100;
        Random rnd = new Random();
        if (args.length > 0) {
            n = Integer.parseInt(args[0]);
        }
        System.out.print("Rolling the dice " + n + " times");
        if (args.length > 1) {
            int seed = Integer.parseInt(args[1]);
            rnd.setSeed(seed);
            System.out.print(" with seed " + seed);
        }
        System.out.println(".");
        for (int i = 0; i < n; i++) {
            int pips = rnd.nextInt(6);
            diceReg[pips]++;
        }
        for (int i = 1; i <= 6; i++) {
            System.out.println("Number of " + i + "'s: " +
                diceReg[i-1]);
        }
    }
}
```

- b) Kompilera med javac DiceReg.java och kör med java DiceReg 10000 42 och förklara vad som händer.
- c) Beskriv skillnaderna mellan Scala och Java, vad gäller syntaxen för array och **for**-sats. Beskriv några andra skillnader mellan språken som syns i programmet ovan. 
- d) Ändra i programmet ovan så att loop-variabeln i skrivas ut i varje runda i varje **for**-sats. Kompilera om och kör.
- e) Skriv om programmet ovan genom att abstrahera huvudprogrammets delar till de statiska metoderna parseArguments, registerPips och printReg enligt nedan skelett. Notera speciellt hur **private** och **public** är angivet. Spara programmet i filen DiceReg2.java och kompilerar med javac DiceReg2.java i terminalen.

```
// DiceReg2.java
import java.util.Random;
```

```

public class DiceReg2 {
    public static int[] diceReg = new int[6];
    private static Random rnd = new Random();

    public static int parseArguments(String[] args) {
        // ???
        return n;
    }

    public static void registerPips(int n){
        // ???
    }

    public static void printReg() {
        // ???
    }

    public static void main(String[] args) {
        int n = parseArguments(args);
        registerPips(n);
        printReg();
    }
}

```

- f) Starta Scala REPL i samma bibliotek som filen `DiceReg2.class` ligger i och kör nedan satser och förklara vad som händer:

```

1 scala> DiceReg2.main(Array("1000", "42"))
2 scala> DiceReg2.diceReg
3 scala> DiceReg2.registerPips(1000)
4 scala> DiceReg2.printReg
5 scala> DiceReg2.registerPips(1000)
6 scala> DiceReg2.printReg
7 scala> DiceReg2.rnd

```

- g) Växla synligheten på attributen mellan `private` och `public`, kompilera om och studera effekten i Scala REPL. Hur lyder felmeddelandet om du försöker komma åt en privat medlem?
 h) Ange en viktig anledning till att man kan vilja göra medlemmar privata.

Uppgift 14. Läsa in tal med `java.util.Scanner`. Med `new Scanner(System.in)` skapas ett objekt som kan läsa in tal som användaren skriver i terminalfönstret.

- a) Sök upp och studera dokumentationen för `java.util.Scanner`. Vad gör metoderna `hasNextInt()` och `nextInt()`?
b) Skriv nedan program i en editor och spara i filen `DiceScanBuggy.java`:

```
// DiceScanBuggy.java
```

```

import java.util.Random;
import java.util.Scanner;

public class DiceScanBuggy {
    public static int[] diceReg = new int[6];
    public static Scanner scan = new Scanner(System.in);

    public static void registerPips(){
        System.out.println("Enter pips separated by blanks.");
        System.out.println("End with -1 and <Enter>.");
        boolean isPips = true;
        while (isPips && scan.hasNextInt()) {
            int pips = scan.nextInt();
            if (pips >= 1 && pips <=6 ) {
                diceReg[pips]++;
            } else {
                isPips = false;
            }
        }
    }

    public static void printReg() {
        for (int i = 0; i < 6; i++) {
            System.out.println("Number of " + i + "'s: " +
                diceReg[i-1]);
        }
    }

    public static void main(String[] args) {
        registerPips();
        printReg();
    }
}

```

c) Kompilera och kör med indatasekvensen 1 2 3 4 -1 och notera hur registreringen sker.

d) Programmet fungerar inte som det ska. Du behöver korrigera 3 saker för att programmet ska göra rätt. Rätta buggarna och spara det rättade programmet som DiceScan.java. Kompilera och testa att det rättade programmer fungerar med olika indata.

Uppgift 15. Välja sekvenssamling. Vilken av Vector, Array och ArrayBuffer hade du valt i dessa situationer? 

a) Ditt program innehåller en sekvens av objekt med data om alla ca 10^7 medborgare i sverige. Efter noggranna mätningar visar det sig att tillägg av objekt på godtyckliga ställen i sekvensen är en flaskhals.

- b) Ditt program innehåller en sekvens av objekt med data om ca 10^2 **residensstäder** i Sverige. Senast det skedde en uppdatering av mängden referensstäder var 1997. Prestandamätningar visar att det är uppdatering av attributvärdet i objekten som tar mest tid. Städerna behöver kunna bearbetas i godtycklig ordning.
- c) Ditt program innehåller en sekvens av ca 10^9 osorterade heltal som ska läsas in från fil och sorteras på plats i minnet. Det första talet i filen anger antalet heltal. Det är viktigt att sorteringen går snabbt. När talen är sorterade ska de skrivas tillbaka till fil i sorterad ordning.
- d) Ditt program innehåller en sekvens av ett känt antal oföränderliga objekt med data om genomförda banktransaktioner. Sekvensen ska bearbetas parallellt i godtycklig ordning med olika algoritmer som kan köras oberoende av varandra.

5.1.2 Extrauppgifter

Uppgift 16. Algoritm: SEQ-INSERT-COPY.

Indata : En sekvens xs av typen `Array[Int]` och heltalet x och pos

Resultat: En ny sekvens av typen `Array[Int]` som är en kopia av xs men där x är infogat på plats pos

```

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

- a) Implementera ovan algoritm i en funktion med denna signatur:

```
def insertCopy(xs: Array[Int], x: Int, pos: Int): Array[Int]
```

- b) Vad måste pos vara för att det ska fungera med en tom array som argument?
- c) Vad händer om din funktion anropas med ett negativt argument för pos ?
- d) Vad händer om din funktion anropas med pos lika med $xs.size$?
- e) Vad händer om din funktion anropas med pos större än $xs.size$?

Uppgift 17. Algoritm: SEQ-INSERT. Man kan implementera algoritmen SEQ-INSERT på plats i en `Array[Int]` så att alla elementen efter pos flyttas fram ett steg och att sista elementet ”försinner”.

-  a) Skriv algoritmen SEQ-INSERT i pseudokod med penna och papper.

- b) Implementera SEQ-INSERT i en funktion med denna signatur:

```
def insert(xs: Array[Int], x: Int, pos: Int): Unit
```

Uppgift 18. Implementera funktionen tärningsRegistrering från uppgift 8 på nytt, men nu med en **for**-sats istället.

Uppgift 19. Bygg vidare på Keno-uppgiften nummer 26 i kapitel 4 på sidan 103 och gör registrering av det slumpmässiga utfallet av 365 Keno-dragningar och skriv ut frekvenserna för förekomsten av varje boll. Öka sedan antalet dragningar och undersök hur många dragningar du behöver göra för att frekvenserna ska bli nästan lika?

5.1.3 Fördjupningsuppgifter

Uppgift 20. Sök reda på dokumentationen för metoden **patch** på klassen **Array**.

- a) Använd metoden **patch** för att implementera SEQ-INSERT-COPY:

```
def insertCopy(xs: Array[Int], x: Int, pos: Int): Array[Int] =  
  xs.patch(???, ???, ???)
```

- b) Använd metoden **patch** för att implementera SEQ-REMOVE-COPY:

```
def removeCopy(xs: Array[Int], pos: Int): Array[Int] =  
  xs.patch(???, ???, ???)
```

Uppgift 21. Studera skillnader och likheter mellan

- a) **Array**
- b) **WrappedArray**
- c) **ArraySeq**

genom att läsa mer om dessa arrayvarianter här:

docs.scala-lang.org/overviews/collections/concrete-mutable-collection-classes

docs.scala-lang.org/overviews/collections/arrays.html

stackoverflow.com/questions/5028551/scala-array-vs-arrayseq

Uppgift 22. Studera vad metoden **java.util.Arrays.deepEquals** gör här:

[Arrays.html#deepEquals-T:A-T:A-](https://docs.scala-lang.org/api/current/java/util/Arrays.html#deepEquals-T:A-T:A-)

Vad skiljer ovan metod från metoden **java.util.Arrays.equals**?

Uppgift 23. Använda *jline* istället för *Scanner* i *REPL*. Om du använder **java.util.Scanner** i Scala REPL så ekas inte de tecken som skrivs, så som sker om du använder scannern med **System.in** i en kompilerad applikation. Om du vill se vad du skriver vid indata i REPL kan du använda *jline*⁴

⁴ github.com/jline/jline2

och klassen `jline.console.ConsoleReader`⁵. Då får du dessutom editeringsfunktioner vid inmatning med t.ex. Ctrl+A och Ctrl+K så som i en vanlig unixterminal. Med pil upp och pil ner kan du bläddra i inmatningshistoriken.

```

1 scala> val scan = new java.util.Scanner(System.in)
2 scala> scan.nextInt
3 scala> scan.nextInt
4 scala> val cr = new jline.console.ConsoleReader
5 scala> cr.readLine
6 scala> cr.readLine("> ")
7 scala> cr.readLine("Ange tal: ").toInt
8 scala> scala.util.Try{cr.readLine("Ange tal: ").toInt}.toOption

```

- a) Prova ovan rader i REPL. Vad händer om du matar in bokstäver i stället för siffror på sista raden ovan? (Mer om Option i kapitel 8).
- b) Skriv ett funktion `readPalindromLoop` som låter användaren mata in strängar och som kollar om de är palindromer så som nedan REPL-körning indikerar. Skriv funktionen i en editor och klippa in den i REPL enligt nedan istället för ???

```

1 scala> val cr = new jline.console.ConsoleReader
2 scala> def isPalindrome(s: String): Boolean = s == s.reverse
3 scala> :paste
4 // Entering paste mode (ctrl-D to finish)
5
6 def readPalindromLoop: Unit = ???
7
8 // Exiting paste mode, now interpreting.
9
10 readPalindromLoop: Unit
11
12 scala> readPalindromLoop
13 Ange sträng följt av <Enter>
14 Programmet avslutas med tom sträng + <Enter>
15 > gurka
16 gurka är ingen palindrom
17 > dallassallad
18 dallassallad är en palindrom!
19 >
20 Tack och hej!
21 scala>

```

- c) Skapa ett objekt med inläsningsstöd enligt nedan specifikation. Objektet ska delegera implementationerna till ett attribut `private val reader` som innehåller en referens till ett `ConsoleReader`-objekt.

Specification `termutil`

```

object termutil {
    /** Reads one line from terminal input. */
    def readLine: String = ???
}

```

⁵ jline.github.io/jline2/apidocs/reference/jline/console/ConsoleReader.html

```
/** Prints prompt and reads one line. */
def readLine(prompt: String): String = ???

/** Reads one line and converts it to an Int.
 * If a non-integer is input, a NumberFormatException is thrown. */
def readInt: Int = ???

/** Prints prompt, reads one line and converts it to an Int.
 * If a non-integer is input, a NumberFormatException is thrown. */
def readInt(prompt: String): Int = ???

/** Reads one line and converts it to an Option[Int]
 * with Some integer or None if the input cannot be converted. */
def readIntOpt: Option[Int] = ???

/** Prints prompt, reads one line and converts it to an Option[Int]
 * with Some integer or None if the input cannot be converted. */
def readIntOpt(prompt: String): Option[Int] = ???
}
```

Biblioteket jline finns inbyggd i REPL men om du vill kompilera din kod separat kan du ladda ner jar-filen här: repo1.maven.org/maven2/jline/jline/2.10/ eller så hittar du den bland dina Scala-installationsfiler och kan kopiera filen till dit du vill ha den. Placera jline-jar-filen i samma bibliotek som din kod, eller lägg den i ett biblioteket där du vill ha den och placera jarfilen på classpath med optionen -cp när du kompilrar ungefär så här:

```
scalac -cp "lib/jline-2.10.jar" termutil.scala
```

5.2 Laboration: shuffle

Mål

- Kunna skapa och använda sekvenssamlingar.
- Kunna använda sekvensalgoritmen SHUFFLE för blandning på plats av innehållet i en array.
- Kunna registrera antalet förekomster av olika värden i en sekvens.

Förberedelser

- Gör övning sequences i avsnitt 5.1.
- Läs igenom hela laborationen och säkerställ att du förstår hur SHUFFLE-algoritmen nedan fungerar.

5.2.1 Bakgrund

Denna labb handlar om kortblandning. Att blanda kort så att varje möjlig permutation (ordning som korten ligger i) är lika sannolik är icke-trivialt; en osystematiskt blandning leder till en skev fördelning.

Givet en bra slumpgenerator går det att blanda en kortlek genom att lägga alla kort i en hög och sedan ta ett slumpvist kort från högen och lägga det överst i leken, tills alla kort ligger i leken. Fisher-Yates-algoritmen⁶ (även kallad Knuth-shuffle), fungerar på det sättet. Här benämner vi denna algoritm SHUFFLE. Den återfinns i pseudokod nedan:

Indata: Array xs som ska blandas

```

1  $len \leftarrow$  antalet element i  $xs$ 
2 for  $i \leftarrow (len - 1)$  to 0 do
3    $r \leftarrow$  slumptal mellan 0 och  $i$ 
4    $temp \leftarrow xs(i)$ 
5    $xs(i) \leftarrow xs(r)$ 
6    $xs(r) \leftarrow temp$ 
7 end
```

Kortspelet poker handlar om att dra kort och få upp vissa kombinationer av kort, s.k. ”händer”⁷. Dessa är ordnade från bättre till sämre; den spelare vinner som fått bäst hand. Det är därför intressant att veta med vilken sannolikhet en viss hand dyker upp vid dragning från en blandad kortlek.

De vanliga pokerhänderna är, i fallande värde, färgstege (straight flush), fyrtal, kåk (full house), färg (flush), stege (straight), triss, tvåpar och par. Dessa finns illustrerade i labbhandledningen. Ofta finns ytterligare en hand, s.k. ”royal flush”, men dess sannolikhet är för låg för att kunna simuleras fram på rimlig tid.

Under laborationen ska du börja med att göra klar den ofärdiga klassen CardDeck som visas nedan, och återfinns i laborationens workspace som du

⁶https://en.wikipedia.org/wiki/Fisher%20%93Yates_shuffle

⁷<https://sv.wikipedia.org/wiki/Pokerhand>

kan ladda ner från <http://cs.lth.se/pgk/ws> och importera i en IDE enligt instruktioner i appendix D.

```

1 package cardSimulation
2
3 class CardDeck {
4     val rand = new util.Random
5
6     lazy val cards: Array[Card] = ???
7
8     def shuffle: Unit = ???
9 }
10
11 object CardDeck {
12     def main(args: Array[String]): Unit = {
13         val d = new CardDeck
14         println(d.cards.mkString(" "))
15         d.shuffle
16         println()
17         println(d.cards.mkString(" "))
18     }
19 }
```

Labbinstruktionerna i avsnitt 5.2.2 ger tips om hur du ska ersätta ??? i givna kodskelett med med dina lösningar.

Efter att du testat så att din implementation av CardDeck fungerar, ska du använda CardDeck tillsammans med andra klasser som beskrivs nedan för att ta reda på sannolikheter för att olika pokerhänder uppkommer när man delar ut 5 kort ur en bra blandad kortlek.

Till din hjälp har du ett antal färdiga kodfiler och en del ofärdig kod som du ska färdigställa. Alla klasserna ligger i ett paket med namnet CardSimulation.

De givna kodfilerna är:⁸

- CardDeck.scala innehåller en klass som har en kortlek som kan blandas. Kompanjonsobjektet har en main-metod som skapar en kortlek, skriver ut, blandar och skriver ut.
- Card.scala innehåller en **case class** Card(suit: Int, value: Int) som representerar ett kort och har en koncis **toString** med färg (eng. *suit*) och valör.
- Hand.scala innehåller en **case class** Hand(cards: Vector[Card]) som representerar en pokerhand och har metoder för att avgöra vilken handen det är. I kompanjonsobjektet finns en fabriksmetod som kan skapa en hand genom att dra fem kort från en kortlek.
- PokerProbability.scala har en main-metod som testar pokersannolikheter, samt ett par metoder som hjälper denna.

⁸Du kan bläddra bland klasserna i paketet cardSimulation här:

https://github.com/lunduniversity/introprog/tree/master/workspace/w05_shuffle/src/main/scala/cardSimulation

- `TestingDeck.scala` är en nedskalad subclass till `CardDeck` som endast innehåller tre kort. Main-metoden i dess kompanjonsobjekt testar implementationen av `shuffle` och redovisar resultatet i ASCII-grafik.
- `AsciiBarGraph.scala` innehåller enbart en metod som skapar ett stapeldiagram åt `TestingDeck`

5.2.2 Obligatoriska uppgifter

Uppgift 1. Implementera algoritmen SHUFFLE.

- Implementera metoden `shuffle` i klassen `CardDeck`. Följ algoritmen noga, och använd `cards.length` för att få fram längden på kortleken.
- Kör `TestingDeck` för att testa att blandningen är jämnt fördelad. `TestingDeck` blandar en kortlek med tre kort och räknar hur ofta olika permutationer dyker upp. Du bör få en utskrift med sex (3!) ungefär lika långa staplar.

Uppgift 2. Fyll i de ofärdiga delarna av klassen `CardDeck`.

- Skriv kod för att skapa en array innehållande en 52-korts standardlek. Använd konstanterna i `Card`. Tänk på att en `for/yield`-sats inte nödvändigtvis ger en `Array`, men att alla samlingar kan omvandlas till en sådan med `toArray`.
- Kör `CardDeck` och kontrollera så att du får kort av alla fyra färger, och både ess och kungar.

Uppgift 3. Använd den färdiga `CardDeck`-klassen för att ta fram sannolikheterna för att "straight flush", "straight" eller "flush" dyker upp.

- Implementera funktionen `test` i `PokerProbability`. Använd de färdiga funktionerna `Hand.drawFrom` och `testHand` för att dra och klassificera en hand från en kortlek. Lagra frekvenserna i en muterbar `Map` (`collection.mutable.Map` finns redan importerad).
- Kör `PokerProbability`, förslagsvis med 1000 000 iterationer. Du bör få ungefär dessa sannolikheter:

<i>hand</i>	<i>sannolikhet</i>
Straight flush	0.00154%
Flush	0.197%
Straight	0.39%
High card	99.41%

5.2.3 Frivilliga extrauppgifter

Uppgift 4. Implementera metoden `tally` i klassen `Hand` så att simuleringen även kan registrera kortkombinationerna fyrtal, kåk, triss, tvåpar och par. Kör sedan `PokerProbability` igen.

Uppgift 5. Implementera ett interaktivt kortspel, t.ex. någon enkel pokervariant. Börja med något mycket enkelt och bygg vidare med sådant som du tycker verkar roligt.

5.2.4 Bilder med exempel på olika pokerhänder

Figurerna 5.1 – 5.9 visar bilder på olika korthänder i poker.



Figur 5.1: Par - två kort har samma valör



Figur 5.2: Tvåpar - två olika par



Figur 5.3: Triss - tre kort har samma valör



Figur 5.4: Stege - kortens valörer bildar en följd, ess kan vara antingen 1 eller 14



Figur 5.5: Färg - alla kort har samma färg



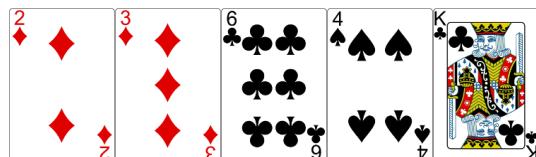
Figur 5.6: Kåk - både triss och par



Figur 5.7: Fyrtal - fyra kort har samma valör



Figur 5.8: Färgstege - både stege och färg



Figur 5.9: Högt kort - inget mönster finns

Kapitel 6

Klasser

Begrepp som ingår i denna veckas studier:

- objektorientering
- klass
- Point
- Square
- Complex
- new
- null
- this
- inkapsling
- accessregler
- private
- private[this]
- kompanjonsobjekt
- getters och setters
- klassparameter
- primär konstruktör
- objektfabriksmetod
- överlägning av metoder
- referenslikhet vs strukturlikhet
- eq vs ==

Vad är en klass?

- En klass är en mall för att skapa objekt.
- Objekt skapas med **new** Klassnamn och kallas för **instanser** av klassen Klassnamn.
- En klass innehåller medlemmar (eng. *members*):
 - **attribut**, kallas även fält (eng. *field*): **val**, **lazy val**, **var**
 - **metoder**, kallas även operationer: **def**
- Varje instans har sin uppsättning värden på attributen (fälten).

Implementation saknas: ???

- Ofta vill man bygga kod iterativt och steg för steg lägga till olika funktionalitet.
- Standardfunktionen **???** ger vid anrop undantaget **NotImplementedError** och kan användas på platser i kodens där man ännu inte är färdig.
- **???** tillåter **kompilering av ofärdig kod**.
- Undantag har bottentypen **Nothing** som är subtyp till *alla* typer och kan därmed tilldelas referenser av godtycklig typ.

```
scala> lazy val sprängsSnart: Int = ???

scala> sprängsSnart + 42
scala.NotImplementedError: an implementation is missing
  at scala.Predef$.qmark$qmark$qmark(Predef.scala:230)
  at .sprängsSnart$lzycompute(<console>:11)
  at .sprängsSnart(<console>:11)
```

Exempel: ofärdig kod

```
case class Person(name: String, age: Int){
    def ärGammal: Boolean = ??? //def ännu ej bestämd
    def ärUng = !ärGammal
    def ärTonåring = age >= 13 && age <= 19
}
```

```
scala> Person("Björn", 49).ärTonåring
res23: Boolean = false

scala> Person("Sandra", 35).ärUng
scala.NotImplementedError: an implementation is missing
  at scala.Predef$.qmark$qmark$qmark(Predef.scala:230)
  at Person.ärGammal(<console>:12)
  at Person.ärUng(<console>:13)
```

Specifikationer av klasser i Scala

- Specifikationer av klasser innehåller information som *den som ska implementera* klassen behöver veta.
- Specifikationer innehåller liknande information som dokumentationen av klassen (scaladoc), som beskriver vad *användaren* av klassen behöver veta.

Specification Person

```
/** Encapsulate immutable data about a Person: name and age. */
case class Person(name: String, age: Int = 0){
    /** Tests whether this Person is more than 17 years old. */
    def isAdult: Boolean = ???
}
```

- Specifikationer av Scala-klasser utgör i denna kurs ofullständig kod som kan kompileras utan fel.
- Saknade implementationer markeras med ???
- **Dokumentationskommentarer** utgör **krav** på implementationen.

Specifikationer av klasser och objekt

Specification MutablePerson

```
/** Encapsulates mutable data about a person. */
class MutablePerson(initName: String, initAge: Int){
    /** The name of the person. */
    def getName: String = ???

    /** Update the name of the Person */
    def setName(name: String): Unit = ???

    /** The age of this person. */
    def getAge: Int = ???

    /** Update the age of this Person */
    def setAge(age: Int): Unit = ???

    /** Tests whether this Person is more than 17 years old. */
    def isAdult: Boolean = ???

    /** A string representation of this Person, e.g.: Person(Robin, 25) */
    override def toString: String = ???
}

object MutablePerson {
    /** Creates a new MutablePerson with default age. */
    def apply(name: String): MutablePerson = ???
}
```

Man brukar inte använda get och set i metodnamn i Scala. Mer senare om principen om enhetlig access (eng. *uniform access principle*) och hur man gör "setters" som möjliggör tilldelningssyntax.

Specifikationer av Java-klasser

- Specificerar signaturer för konstruktörer och metoder.
- Kommentarerna utgör krav på implementationen.
- Används flitigt på extentor i EDA016, EDA011, EDA017...
- Javaklass-specifikationerna **saknar implementationer** och behöver kompletteras med metodkroppar och klassrubriker innan de kan kompileras.

```
class Person

/** Skapar en person med namnet name och åldern age. */
Person(String name, int age);

/** Ger en sträng med denna persons namn. */
String getName();

/** Ändrar denna persons ålder. */
void setAge(int age);

/** Anger åldersgränsen för när man blir myndig. */
static int adultLimit = 18;
```

6.1 Övning: classes

Mål

- Kunna deklarera klasser med klassparametrar.
- Kunna skapa objekt med **new** och konstruktorgument.
- Förstå innehördet av referensvariabler och värdet **null**.
- Förstå innehördet av begreppen instans och referenslikhet.
- Kunna använda nyckelordet **private** för att styra synlighet i primärkonstruktör.
- Förstå i vilka sammanhang man kan ha nytta av en privat konstruktur.
- Kunna implementera en klass utifrån en specifikation.
- Förstå skillnaden mellan referenslikhet och strukturlikhet.
- Känna till hur case-klasser hanterar likhet.
- Förstå nyttan med att möjliggöra framtida förändring av attributrepresentation.
- Känna till begreppen getters och setters.
- Känna till accessregler för kompanjonsobjekt.
- Känna till skillnaden mellan == och eq, samt != versus ne.

Förberedelser

- Studera begreppen i kapitel 6.

6.1.1 Grunduppgifter

Uppgift 1. Instansiering med **new och värdet **null**.** Man skapar instanser av klasser med **new**. Då anropas konstruktorn och plats reserveras i datorns minne för objektet. Variabler av referenstyp som inte refererar till något objekt har värdet **null**.

- a) Vad händer nedan? Vilka rader ger felmeddelande och i så fall hur lyder felmeddelandet?

```
1 scala> class Gurka(val vikt: Int)
2 scala> var g: Gurka = null
3 scala> g.vikt
4 scala> g = new Gurka(42)
5 scala> g.vikt
6 scala> g = null
7 scala> g.vikt
```

- b) Rita minnessituationen efter raderna 2, 4, 6.



Uppgift 2. Klasser och instanser.

- a) Vad händer nedan?

```
1 scala> :pa
2 class Arm(val ärTillVänster: Boolean)
3 class Ben(val ärTillVänster: Boolean)
```

```

4  class Huvud(val harHår: Boolean)
5  class Rymdvarelse {
6      var arm1 = new Arm(true)
7      var arm2 = new Arm(false)
8      var ben1 = new Ben(true)
9      var ben2 = new Ben(false)
10     var huvud1 = new Huvud(false)
11     var huvud2 = new Huvud(true)
12     def ärSkallig = !huvud1.harHår && !huvud2.harHår
13 }
14 scala> val alien = new Rymdvarelse
15 scala> alien.ärSkallig
16 scala> val predator = new Rymdvarelse
17 scala> predator.ärSkallig
18 scala> predator.huvud2 = alien.huvud1
19 scala> predator.ärSkallig

```

- b) Rita minnessituationen efter rad 18.
- c) Vad händer så småningom med det ursprungliga huvud2-objektet i predator efter tilldelningen på rad 18? Går det att referera till detta objekt på något sätt?

Uppgift 3. *Synlighet i primärkonstruktörer.* Undersök nedan vad nyckelorden **val** och **private** får för konsekvenser. Förklara vad som händer. Vilka rader ger vilka felmeddelanden?

```

1  scala> class Gurka1(vikt: Int)
2  scala> new Gurka1(42).vikt
3  scala> class Gurka2(val vikt: Int)
4  scala> new Gurka2(42).vikt
5  scala> class Gurka3(private val vikt: Int)
6  scala> new Gurka3(42).vikt
7  scala> class Gurka4(private val vikt: Int, kompis: Gurka4){
8      def kompisVikt = kompis.vikt
9  }
10 scala> val ingenGurka: Gurka4 = null
11 scala> new Gurka4(42, ingenGurka).kompisVikt
12 scala> new Gurka4(42, new Gurka4(84, null)).kompisVikt
13 scala> class Gurka5(private[this] val vikt: Int, kompis: Gurka5){
14     def kompisVikt = kompis.vikt
15 }
16 scala> class Gurka6 private (vikt: Int)
17 scala> new Gurka6(42)
18 scala> :pa
19 class Gurka7 private (var vikt: Int)
20 object Gurka7 {
21     def apply(vikt: Int) = {
22         require(vikt >= 0, s"negativ vikt: $vikt")
23         new Gurka7(vikt)
24     }
25 }
26 scala> new Gurka7(-42)
27 scala> Gurka7(-42)
28 scala> val g = Gurka7(42)

```

```
29 scala> g.vikt
30 scala> g.vikt = -1
31 scala> g.vikt
```

Uppgift 4. Egendefinierad setter kombinerat med privat konstruktör.

- a) Förklara vad som händer nedan. Vilka rader ger vilka felmeddelanden?

```
1  scala> :pa
2  class Gurka8 private (private var _vikt: Int) {
3      def vikt = _vikt
4      def vikt_=(v: Int): Unit = {
5          require(v >= 0, s"negativ vikt: $v")
6          _vikt = v
7      }
8  }
9
10 object Gurka8 {
11     def apply(vikt: Int) = {
12         require(vikt >= 0, s"negativ vikt: $vikt")
13         new Gurka8(vikt)
14     }
15 }
16 scala> val g = Gurka8(-42)
17 scala> val g = Gurka8(42)
18 scala> g.vikt
19 scala> g.vikt = 0
20 scala> g.vikt = -1
21 scala> g.vikt += 42
22 scala> g.vikt -= 1000
```

- b) Vad är fördelen med möjligheten att skapa egendefienerade setters? 

Uppgift 5. En oföränderlig kvadrat med alternativ fabriksmetod.

- a) Implementera klassen Square enligt nedan specifikation. Gör implementationen i en kodeditor, så som gedit, och klistra in klassen i Scala REPL efter kommandot :pa (förkortning av :paste). På så sätt blir **object** Square ett kompanjonsobjekt till **class** Square.

Specification Square

```
/** A class representing a square object with position and side. */
class Square(val x: Int, val y: Int, val side: Int) {
    /** The area of this Square */
    val area: Int = ???

    /** Creates a new Square moved to position (x + dx, y + dy) */
    def move(dx: Int, dy: Int): Square = ???

    /** Tests if this Square has equal size as that Square */
    def isEqualSizeAs(that: Square): Boolean = ???

    /** Multiplies the side with factor and rounded to nearest integer */
    def scale(factor: Double): Square = ???
```

```

    /** A string representation of this Square */
    override def toString: String = ???
}

object Square {
    /** A square placed in origin with size 1 */
    val unit: Square = ???

    /** Constructs a new Square object at (x, y) with size side */
    def apply(x: Int, y: Int, side: Int): Square = ???

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

```

- b) Testa din kvadrat enligt nedan. Förlära vad som händer.

```

1 scala> val (s1, s2) = (Square(), Square(1, 10, 1))
2 scala> val s3 = s1.move(1, -5)
3 scala> s1 isEqualSizeAs s3
4 scala> s2 isEqualSizeAs s1
5 scala> s1 isEqualSizeAs Square.unit
6 scala> s2.scale(math.Pi) isEqualSizeAs s2
7 scala> s2.scale(math.Pi) isEqualSizeAs s2.scale(math.Pi)

```

Uppgift 6. *Referenslikhet versus strukturlikhet.* Metoden == på case-klasser ger **strukturlikhet** (även kallad innehållslikhet) så att *innehållet* i klassens klassparametrar jämförs om de har lika värde, medan för vanliga klasser ger metoden == **referenslikhet** där olika objekt är olika även om de har samma innehåll (om man inte överskuggar metoden equals som anropas av == vilket vi ska titta närmare på i kapitel 8).

```

1 scala> class GurkaRef(val vikt: Int)
2 scala> case class GurkaStrukt(val vikt: Int)
3 scala> val a = new GurkaRef(42)
4 scala> val b = new GurkaRef(42)
5 scala> val c = new GurkaStrukt(42)
6 scala> val d = new GurkaStrukt(42)
7 scala> a == b
8 scala> c == d

```

- a) Förlära vad som händer ovan.
 b) Istället för ==, prova metoden eq på objekten ovan. Metoden eq ger alltid referenslikhet (även om byter ut metoden equals).

Uppgift 7. *Klassen Point med case-klass.*

- a) Implementera klassen Point som en oföränderlig case-klass med heltalsattributten x och y.
 b) Lägg till metoden distanceTo(that: Point): Double som räknar ut avståndet till en annan punkt med hjälp av math.hypot.

- c) Lägg till metoden `distanceTo(x: Int, y: Int): Double` som räknar ut avståndet till koordinaterna x och y med hjälpa av metoden i föregående deluppgift.
- d) Lägg till metoden `move(dx: Int, dy: Int): Point` som skapar en ny punkt på translaterad position enligt delta-koordinaterna dx och dy.
- e) Lägg till ett kompanjonsobjekt med medlemmen `val origin` som ger en punkt i origo.
- f) Undersök metoderna `==`, `!=`, `eq` och `ne` och förklara vad som händer nedan:

```

1  scala> Point(1, 2) == Point(1, 3)
2  scala> Point(1, 2) != Point(1, 3)
3  scala> Point(1, 2) == Point(1, 2)
4  scala> Point(1, 2) != Point(1, 2)
5  scala> Point.origin.move(1, 1) == Point.origin.move(1, 1)
6  scala> Point.origin.move(1, 1).move(1, 1) != Point(2, 2)
7  scala> Point(0, 0) eq Point(0, 0)
8  scala> Point(0, 0) ne Point(0, 0)
9  scala> Point.origin eq Point.origin
10 scala> Point.origin ne Point.origin
11 scala> val p1 = Point(0, 0)
12 scala> val p2 = p1
13 scala> p1 eq p2

```

- g) Vad ger `Point.origin eq Point.origin` för resultat om `origin` istället implementeras som `def origin: Point = Point(0, 0)`
- h) Vad är det för skillnad på strukturlikhet och referenslikhet?



Uppgift 8. Ändra representationen av positionen i klassen `Square` från deluppgift 5 till att vara en `Point` från deluppgift 7.

Uppgift 9. *Case-klassen Point med 2-tupel.* I ett utvecklingsprojekt vill man ändra representationen av positionen i den gamla klassen

`case class Point(x: Int, y: Int)` så att positionen istället i den uppdaterade klassen representeras av en 2-tupel. Man kan då vid konstruktion utnyttja att n-tupler som parameter även kan skrivas som en parameterlista med n argument, varför både `Point(1,2)` och `Point((1,2))` fungerar fint. Samtidigt vill man att befintlig kod som fortfarande använder x och y ska fungera utan ändringar. Implementera den nya `Point` enligt specifikationen nedan.

Specification Point

```

/** A 2-dimensional immutable position p in an integer coordinate system */
case class Point(p:(Int, Int)) {
    /** The x-axis position of this Point */
    val x: Int = ???

    /** The y-axis position of this Point */
    val y: Int = ???

    /** The distance to another Point that */
}

```

```

def distanceTo(that: Point): Double = ???

/** The distance to another 2-tuple that representing (x, y). */
def distanceTo(that: (Int, Int)): Double = ???

/** A new Point that is moved (dx, dy) */
def move(dxdy: (Int, Int)): Point = ???
}

object Point {
    /** A Point object at position (0, 0) */
    val origin: Point = ???
}

```



Uppgift 10. Vad behöver du ändra i klassen Square från uppgift 8 för att den ska fungera med en Point med 2-tupel från uppgift 9?

Uppgift 11. *Objekt med föränderligt tillstånd (eng. mutable state).* Du ska implementera en modell av en hoppande groda som uppfyller följande krav:

1. Varje grodobjekt ska hålla reda på var den är.
2. Varje grodobjekt ska hålla reda på hur långt grodan hoppat totalt.
3. Varje grodobjekt ska kunna beräkna hur långt det är mellan grodans nuvarande position och utgångsläget.
4. Alla grodor börjar sitt hoppande i origo.
5. En groda kan hoppa enligt två metoder:
 - relativ förflyttning enligt parametrarna dx och dy,
 - slumpmässig förflyttning [1,10] i x-led och [1,10] i y-led.

a) Implementera klassen Frog enligt nedan specifikation och ovan krav.

Tips:

- Om namnet man vill ge ett privat föränderligt attribut "krockar" med ett metodnamn, är det vanligt att man börjar attributets namn med understreck, t.ex. **private var** _x för att på så sätt undkomma namnkonflikten.
- Inför en metod i taget och klistra in den nya grodan i REPL efter varje utvidgning och testa.

Specification Frog

```

class Frog private (initX: Int = 0, initY: Int = 0) {
    def jump(dx: Int, dy: Int): Unit = ???
    def x: Int = ???
    def y: Int = ???
    def randomJump: Unit = ???
    def distanceToStart: Double = ???
    def distanceJumped: Double = ???
    def distanceTo(that: Frog): Double = ???
}
object Frog {
    def spawn(): Frog = ???
}

```

- b) Skriv ett testhuvudprogram som kontrollerar så att alla krav är uppfyllda och att alla metoder fungerar som de ska.
- c) Vad kallas en metod som enbart returnerar värdet av ett privat attribut? 
- d) Hur kan man från en metods signatur få en ledtråd om att ett objekt har föränderligt tillstånd (eng. *mutable state*)? 
- e) Inför setters för attributen som håller reda på x- och y-positionen. Försändringar av positionen i x- eller y-led ska räknas som ett hopp och alltså registreras i det attribut som håller reda på det ackumulerade hoppavståndet.
- f) Simulera ett massivt grodhoppande med krockdetektering genom att skapa 100 grodor som till att börja med är placerade på x-axeln med avståndet 8 längdenheter mellan sig. Låt grodorna i en **while**-sats hoppa slumpmässigt tills någon groda befinner sig närmare än 0.5 längdenheter som är definitionen på att de har krockat. Räkna hur många looprundor som behövs innan något grodpar krockar och skriv ut antalet.

Tips: Börja med pseudokod på papper. Använd en grodvektor.

6.1.2 Extrauppgifter

Uppgift 12. En kvadratklass med föränderligt tillstånd (eng. *mutable state*).

Webbshoppen UberSquare säljer flyttbara kvadrater. I affärsmodellen ingår att ta betalt per förflyttning. Du ska hjälpa UberSquare med att utveckla en enkel systemprototyp.

- a) Implementera Square enligt nedan specifikation, under uppfyllandet av följande krav:

1. Till skillnad från uppgift 5 ska du nu göra en kvadrat med föränderligt tillstånd (eng. *mutable state*). I stället för att vid förflyttning returnera ett nytt kvadratobjekt, returneras Unit i samband med att privata attribut uppdateras.
2. Du ska införa funktionalitet som räknar antalet förflyttningar som gjorts för varje kvadrat som skapats och även räkna ut det totala antalet förflyttningar som någonsin gjorts.
3. Varje gång förflyttning sker adderas en kostnad till den ackumulerade kostnaden för respektive kvadrat. Kostnaden för varje förflyttning är avståndet till ursprungsläget multiplicerat med storleken på kvadraten.

Specification Square

```
/** 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 += ???
    }

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

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

    /** The size of this Square */
    def side = ???

    /** Scales the side of this square and rounds it to nearest integer */
    def scale(factor: Double): Unit = ???

    /** Moves this square to position (x + xd, y + dy) */
    def move(dx: Int, dy: Int): Unit = ???

    /** Moves this square to position (x, y) */
    def moveTo(x: Int, y: Int): Unit = ???
}
```

```

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

/** Reset the cost of this Square */
def pay: Unit = ???

/** 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")
    ???
  }

  /** 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 = ???

  /** The total cost of all squares. */
  def totalCost: Double = ???
}

```

b) Testa din kvadratprototyp i REPL enligt nedan:

```

1 scala> val xs = Vector.fill(10)(Square())
2 scala> xs.foreach(_.move(2,3))
3 scala> xs.foreach(_.scale(2.9))
4 scala> val (m, c) = (Square.totalNumberOfMoves, Square.totalCost)
5 m: Int = 10
6 c: Double = 36.055512754639885

```

6.1.3 Fördjupningsuppgifter

Uppgift 13. Hjälpkonstruktör. I uppgift 5 erbjöds ett alternativt sätt att skapa Square med en extra fabriksmetod med namnet `apply` i kompanjonobjektet. Ett annat sätt att göras detta på, som i Scala är mindre vanligt (men i Java är desto vanligare), är att definiera flera konstruktörer innuti klassen. I Scala kallas en sådan extra konstruktör för **hjälpkonstruktör** (eng. *auxiliary constructor*).

En hjälpkonstruktör skapar man i Scala genom att definiera en metod som har det speciella namnet `this`, alltså en deklaration `def this(...) = ...`. Hjälponstruktörer måste börja med att anropa en annan konstruktör, antingen den primära konstruktorn eller en tidigare definierad hjälpkonstruktör.

- a) Läs mer om hjälpkonstruktörer här:
www.artima.com/pins1ed/functional-objects.html#6.7
- b) Hitta på en egen uppgift med hjälpkonstruktörer, baserat på någon av klasserna i tidigare övningar.

6.2 Laboration: turtlegraphics

Mål

- Kunna skapa egna klasser.
- Förstå skillnaden mellan klasser och objekt.
- Förstå skillnaden mellan muterbara och omuterbara objekt.
- Förstå hur ett objekt kan innehålla referenser till objekt av andra klasser, och varför detta kan vara användbart.
- Träna på att fatta beslut om vilka datatyper som bäst passar en viss tillämpning.

Förberedelser

- Gör övning `classes` i avsnitt 6.1.
- Studera dokumentationen för klassen `cslib.window.SimpleWindow` här: <http://cs.lth.se/pgk/api/>

6.2.1 Bakgrund

Under den här laborationen ska du skriva en samling av klasser som tillsammans kan användas för att rita i ett fönster. För att ge dig en grund att stå på får du tillgång till den färdigskrivna klassen `SimpleWindow`. `SimpleWindow` är en Java-klass som kan skapa ett enkelt ritfönster på skärmen, med metoder för att rita linjer, etc. `SimpleWindow` håller koll på en ”penna” som representerar *aktuell ritposition*. Det finns metoder för att flytta pennan och att rita en rak linje från pennans aktuella ritposition till en ny pennposition.

Med hjälp av `SimpleWindow` ska du skapa en `Turtle`-klass, som ska fungera likt sköldpaddan i Kojo, vilken du använder i laborationen i avsnitt 1.2. Delar av [dokumentationen](#) för `SimpleWindow` återspeglas i nedan specifikation.

```
class SimpleWindow

/** mouse click event type */
public final static int MOUSE_EVENT = 1;

/** key pressed event type */
public final static int KEY_EVENT = 2;

/** window closed event type */
public final static int CLOSE_EVENT = 3;

/** Creates a window and makes it visible. */
public SimpleWindow(int width, int height, String title);

/** Returns the width of the window. */
public int getWidth();

/** Returns the height of the window. */
public int getHeight();
```

```
/** Clears the window. */
public void clear();

/** Closes the window.*/
public void close();

/** Opens the window. */
public void open();

/** Moves the pen to a new position. */
public void moveTo(int x, int y) ;

/** Moves the pen to a new position while drawing a line. */
public void lineTo(int x, int y);

/** Writes a string at the current position.*/
public void writeText(String txt);

/** Draws a bitmap image at the current position.*/
public void drawImage(Image image);

/** Returns the pen's x coordinate. */
public int getX();

/** Returns the pen's y coordinate. */
public int getY();

/** Sets the line width. */
public void setLineWidth(int width);

/** Sets the line color. */
public void setLineColor(Color col);

/** Returns the current line width. */
public int getLineWidth();

/** Returns the current line color. */
public Color getLineColor();

/** Waits for a mouse click. */
public void waitForMouseClicked();

/** Returns the mouse x coordinate at the last mouse click. */
public int getMouseX();

/** Returns the mouse y coordinate at the last mouse click. */
public int getMouseY();

/** Adds a sprite to the window. */
public void addSprite(Sprite sprite);

/** Wait for a specified time. */
public static void delay(int ms);
```

6.2.2 Obligatoriska uppgifter

Uppgift 1. Skapa en klass Point för att beskriva en viss koordinat (x,y) i ett fönster. Klassen ska vara oföränderlig – man ska alltså inte kunna ändra på en koordinat efter att den har skapats. Notera att klassens attribut är av typen Double och inte Int, trots att de beskriver en diskret pixelposition. Anledningen till detta är att det kan uppstå avrundningsfel vid upprepade förflyttningar. Detta blir särskilt märkbart vid många små förflyttningar, som t.ex. när en Turtle används för att rita en cirkel.

Specification Point

```
package turtlegraphics

/** Represents a single Point in the x,y plane. */
case class Point(val x: Double, val y: Double) {
    /** Returns a new Point which has been moved some number of pixels */
    def translate(dx: Double, dy: Double) = ???
}
```

- a) Implementera klassen Point.

Uppgift 2. Skapa klassen Turtle:

Specification Turtle

```
package turtlegraphics

import cslib.window.SimpleWindow

/** A Kojo-like Turtle class that can be used to draw shapes in a SimpleWindow.
 *
 * @param window      The window the turtle should be placed in.
 * @param position    A Point representing the turtle's starting coordinates.
 * @param angle        The angle between the turtle direction and the X-axis
 *                     measured in degrees. Positive degrees indicate a counter
 *                     clockwise rotation.
 * @param isPenDown   A boolean representing the turtle's pen position. True if
 *                     the pen is down. */
class Turtle(window: SimpleWindow,
            private var position: Point,
            private var angle: Double,
            private var isPenDown: Boolean) {

    /** Gets the Turtle's current pixel position on the x axis */
    def x: Int = ???

    /** Gets the Turtle's current pixel position on the y axis
     * (measured from the top left) */
    def y: Int = ???

    /** Moves the turtle to a new position without drawing a line. */
    def jumpTo(newPosition: Point) = ???
```

```

/** Moves the turtle forward in its current direction, drawing a line if
 * the pen is down.
 * @param length The number of pixels to move forward. */
def forward(length: Double): Unit = ???

/** Turns the turtle to the left.
 *
 * @param turnAngle The number of degrees to turn. */
def turnLeft(turnAngle: Double): Unit = ???

/** Turns the turtle to the right.
 *
 * @param turnAngle The number of degrees to turn. */
def turnRight(turnAngle: Double): Unit = ???

/** Turns the turtle straight up. */
def turnNorth(): Unit = ???

/** Sets the turtle's pen down, causing it to draw lines when it moves */
def penDown(): Unit = ???

/** Lifts the turtle's pen, preventing it from drawing lines. */
def penUp(): Unit = ???
}

```

- a) Vilka attribut finns i klassen, och vilken synlighetsnivå har de? Vilken/vilka konstruktörer finns?
- b) Är klassen muterbar eller omuterbar? Motivera! Hade man kunnat göra tvärtom?
- c) Implementera klassen Turtle enligt specifikationen ovan. När klassen är färdigimplementerad ska den kunna användas för att rita figurer från labbens main-metod i filen Main.scala.

Specification Main

```

package turtlegraphics

import cslib.window.SimpleWindow

object Main {
  def main(args: Array[String]): Unit = {
    // Create the window and turtle objects
    val window = new SimpleWindow(500, 500, "Turtlegraphics")
    val t = new Turtle(window, Point(0, 0), 0, false)

    // Move to an initial position
    t.jumpTo(Point(100, 200));
    t.turnNorth()

    // Draw a rectangle
    t.penDown()
    for (i <- 1 to 4) {
      t.turnRight(90)
      t.forward(100)
    }
  }
}

```

```

    }
    t.penUp()

    // Move 50 pixels to the right of the first rectangle
    t.turnRight(90)
    t.forward(200)
    t.turnNorth()

    // Draw a rectangle
    t.penDown()
    for (i <- 1 to 4) {
        t.turnRight(90)
        t.forward(100)
    }
}
}
}

```

- d) Kör main-metoden ovan för att bekräfta att din implementation fungerar. Bilden ska visa två lika stora rektanglar i samma höjd.
- e) Just nu behöver användaren av en Turtle specificera alla detaljer om en Turtles ursprungliga tillstånd som parametervärden för att skapa den. För att underlätta för användaren ska du nu skapa en alternativ konstruktor som kräver färre parametrar. Vilka konstruktorparametrar skulle kunna bytas ut mot rimliga default-värden?
- f) Använd din Turtle för att rita en cirkel. För att göra detta kan du t.ex. låta din Turtle gå ett kort steg och svänga någon grad tills den har gjort ett fullt varv.
- g) Skapa två stycken Turtles i samma fönsterobjekt som rör sig alternerande. Fungerar allt som tänkt?
- **Tips:** SimpleWindow har sitt origo i övre vänstra hörnet (och inte det nedre vänstra hörnet som är vanligt inom matematik).

Uppgift 3. Skapa klassen Rectangle:

<i>Specification Rectangle</i>

```

package turtlegraphics

/** Immutable class representing a rectangle.
 * @param position a Point representing the upper left corner of the rectangle
 *                  (before rotation)
 * @param width    the width of the rectangle
 * @param height   the height of the rectangle
 * @param angle    the angle of the rectangle (rotated around the upper
 *                  left corner) Positive degrees indicate a counter clockwise
 *                  rotation measured from the X-axis
 */
case class Rectangle(position: Point,
                     width: Double,
                     height: Double,
                     angle: Double = 0)

```

```

        angle: Double) {
    /** Draws the rectangle using a turtle */
    def draw(turtle: Turtle): Unit = ???

    /** Returns a new Rectangle that is rotated to the left */
    def rotateLeft(degrees: Double): Rectangle = ???

    /** Returns a new Rectangle that is rotated to the right */
    def rotateRight(degrees: Double): Rectangle = ???

    /** Returns a new Rectangle that has been scaled by a size factor */
    def scale(factor: Double): Rectangle = ???

    /** Returns a new Rectangle that has been moved some number of pixels */
    def translate(dx: Double, dy: Double): Rectangle = ???
}

```

- Vilken synlighetsnivå bör konstruktörparametrarna ges? Motivera.
- I specifikationen står det att rektangeln roteras runt det övre vänstra hörnet, men finns det andra val av rotationsaxlar? Vilka fördelar/nackdelar finns för olika val? Välj den implementationen du anser lämpligast.
- Implementera klassen Rectangle enligt specifikationen ovan.
- Använd din Rectangle för att skapa en animation som utnyttjar skalning, rotation och förflyttning. Du kan skapa en animering genom att använda dig av SimpleWindow-objektens clear-metod för att rensa skärmen, samt SimpleWindow-klassens delay-metod. Notera att delay-metoden inte kan anropas på objektet. Se nedanstående exempel.

```

val w = new SimpleWindow(500,500, "Animation")
while(true){
    w.clear()
    // Draw something here
    SimpleWindow.delay(50)
}

```

6.2.3 Frivilliga extrauppgifter

Uppgift 4. Skapa en klass RectangleSequence. I denna klass skall draw-metoden rita ut ett antal rektanglar där varje rektangel har förflyttat sig, roterats och skalats jämfört med föregående rektangel i sekvensen. Se bilder nedan.

```
Specification RectangleSequence
package turtlegraphics

/** Represents a sequence of Rectangles that have been translated, rotated,
 * and scaled down.
 *
 * @param rectangle    the rectangle to use as a base for the composite shape
 * @param count        the number of rectangles to draw
 * @param startAngle   the number of degrees to rotate the image. Positive
 *                     degrees indicate a counter clockwise rotation measured
 *                     from the X-axis
 *
 * @param step          the number of pixels to move each rectangle
 *                      (in the direction of startAngle)
 * @param rotationStep the number of degrees to shift each rectangle with
 *                      each step of the animation
 * @param scaleStep     the scale factor to use in each step
 */
case class RectangleSequence(rectangle: Rectangle,
                             count: Int,
                             startAngle: Double,
                             step: Double,
                             rotationStep: Double,
                             scaleStep: Double) {

  /** Draws the image using a given Turtle */
  def draw(turtle: Turtle): Unit = ???

  /** Returns a new RectangleSequence that has been scaled with a size factor */
  def scale(factor: Double): RectangleSequence = ???

  /** Returns a new RectangleSequence that has been rotated to the left */
  def rotateLeft(degrees: Double): RectangleSequence = ???

  /** Returns a new RectangleSequence that has been rotated to the right */
  def rotateRight(degrees: Double): RectangleSequence = ???

  /** Returns a new RectangleSequence that has been moved a number of pixels */
  def translate(dx: Double, dy: Double): RectangleSequence = ???
}
```

- Implementera RectangleSequence.
- I SimpleWindow kan man ange en färg via metoden setLineColor som ska användas vid utritning. Nyttja detta för att göra en färggladare visualisering av rektangelsekvensen.

- c) RectangleSequence är resultatet av flera lager utav abstraktioner. Vilka abstraktionslager ser du? Skulle man kunna abstrahera ytterligare?



Figur 6.1: Resultatet av:

```
RectangleSequence(
    Rectangle(Point(200, 200), 50, 30, 0), 5, 0, 70, -30, 0.67
)
```

```
val w = new SimpleWindow(500, 500, "Shapes")
val t = new Turtle(w, new Point(200, 200), 0, false)
val rect = Rectangle(Point(225, 235), 50, 30, 0)
val roll = RectangleSequence(rect, 100, 0, 2, 0, 0.98)
for(i <- 0 to 360 by 20) {
    roll.rotateLeft(i).draw(t)
}
```

Figur 6.2: Kod som ritar bilden som visas i figur 6.3 på sidan 162.



Figur 6.3: Resultatet av koden i figur 6.2.

Uppgift 5. Studera dokumentationen för de SimpleWindow-metoder som erbjuder hantering av händelser (eng. *event*) och använd dessa för lösa deluppgifterna nedan.

- Gör så att en Turtle kan styras med hjälp av tangentbordstryckningar A–S–D–W för vänster–ner–höger–upp och att den ritar ett spår allteftersom den förflyttas.
- Gör så att en andra Turtle kan styras med hjälp av tangentbordstryckningar J–K–L–I för vänster–ner–höger–upp och att den också ritar ett spår allteftersom den förflyttas.
- Gör så att, när de två sköldpaddorna ovan befinner sig tillräckligt nära varandra, det ritas ut en rektangel med hörn där de två sköldpaddorna finns. (Denna uppgift är lite svårare och kan behöva delas upp i delar.)

Uppgift 6. Studera dokumentationen för de SimpleWindow-metoder som erbjuder hantering av flyttbara bilder (eng. *sprites*). Gör så att en fin Sprite ritas vid positionen för de styrbara sköldpaddorna i föregående uppgift.

Uppgift 7. En riktig utmaning, för den som har lust: Implementera spelet "Masken" som beskrivs här: <https://sv.wikipedia.org/wiki/Snake>.

Kapitel 7

Arv

Begrepp som ingår i denna veckas studier:

- arv
- polymorfism
- trait
- extends
- asInstanceOf
- with
- inmixning
- supertyp
- subtyp
- bastyp
- override
- klasshierarkin i Scala: Any AnyRef Object AnyVal Null Nothing
- referenstyper vs värde typer
- klasshierarkin i scala.collection
- Shape som bastyp till Rectangle och Circle
- accessregler vid arv
- protected
- final
- klass vs trait
- abstract class
- case-object
- typer med uppräknade värden

Medlemmar, arv och överskuggning

Olika sorters överskuggningsbara medlemmar i klasser och traits i **Scala**:

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

Olika sorters överskuggningsbara instansmedlemmar i **Java**:

- variabel
- metod

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

- När man överskuggar (eng. *override*) en medlemmen med en annan medlem med samma namn i en subtyp, får denna medlem en (ny) implementation.
- När man konstruerar ett objektorienterat språk gäller det att man definierar sunda överskuggningsregler vid arv. Detta är förvånansvärt knepigt.
- Singelobjekt kan ej ärvas (och medlemmar i singelobjekt kan därmed ej överskuggas).

Fördjupning: Regler för överskuggning i Scala

En medlem M1 i en supertyp får överskuggas av en medlem M2 i en subtyp, enligt dessa regler:

1. M1 och M2 ska ha samma namn och typerna ska matcha.
2. **def** får bytas ut mot: **def**, **val**, **var**, **lazy val**
3. **val** får bytas ut mot: **val**, och om M1 är abstrakt mot en **lazy val**.
4. **var** får bara bytas ut mot en **var**.
5. **lazy val** får bara bytas ut mot en **lazy val**.
6. Om en medlem i en supertyp är abstrakt *behöver* man inte använda nyckelordet **override** i subtypen. (Men det är bra att göra det ändå så att komplatorn hjälper dig att kolla att du verkligen överskuggar något.)
7. Om en medlem i en supertyp är konkret *måste* man använda nyckelordet **override** i subtypen, annars ges kompileringsfel.
8. M1 får inte vara **final**.
9. M1 får inte vara **private** eller **private[this]**, men kan vara **private[X]** om M2 också är **private[X]**, eller **private[Y]** om X innehåller Y.
10. Om M1 är **protected** måste även M2 vara det.

Trait eller abstrakt klass?

Använd en **trait** som supertyp om...

- ...du är osäker på vilket som är bäst. (Du kan alltid ändra till en abstrakt klass senare.)
- ...du vill kunna mixa in din trait tillsammans med andra traits.
- ...du bara har abstrakta medlemmar.

Använd en **abstract class** som supertyp om...

- ...du vill ge supertypen en parameter vid konstruktion.
- ...du vill ärva supertypen från klasser skrivna i Java.
- ...du vill minimera vad som behöver omkompileras vid ändringar.

7.1 Övning: traits

Mål

- Förstå följande begrepp: supertyp, subtyp, bastyp, abstrakt typ, polymorfism.
- Kunna deklarera och använda en arvhierarki i flera nivåer med nyckelordet **extends**.
- Kunna deklarera och använda inmixning med flera traits och nyckelordet **with**.
- Kunna deklarera och känna till nyttan med finala klasser och finala attribut och nyckelordet **final**.
- Känna till synlighetsregler vid arv och nyttan med privata och skyddade attribut.
- Kunna deklarera och använda skyddade attribute med nyckelordet **protected**.
- Känna till hur typtester och typkonvertering vid arv kan göras med metoderna `isInstanceOf` och `asInstanceOf` och känna till att detta görs bättre med **match**.
- Känna till begreppet anonym klass.
- Kunna deklarera och använda överskuggade metoder med nyckelordet **override**.
- Känna till reglerna som gäller vid överskuggning av olika sorters medlemmar.
- Kunna deklarera och använda hierarkier av klasser där konstruktörparametrar överförs till superklasser.
- Kunna deklarera och använda uppräknade värden med case-objekt och gemensam bastyp.

Förberedelser

- Studera begreppen i kapitel 7.

7.1.1 Grunduppgifter

Uppgift 1. *Gemensam bastyp.* Man vill ofta lägga in objekt av olika typ i samma samling.

```

1 class Gurka(val vikt: Int)
2 class Tomat(val vikt: Int)
3 val gurkor = Vector(new Gurka(100), new Gurka(200))
4 val grönsaker = Vector(new Gurka(300), new Tomat(42))

```

- a) Om en samling innehåller objekt av flera olika typer försöker kompilatorn härleda den mest specifika typen som objekten har gemensamt. Vad blir det för typ på värdet grönsaker ovan?
- b) Försök ta reda på summan av vikterna enligt nedan. Vad ger andra raden för felmeddelande? Varför?

```
1 gurkor.map(_.vikt).sum
2 grönsaker.map(_.vikt).sum
```

- c) Vi kan göra så att vi kan komma åt vikten på alla grönsaker genom att ge gurkor och tomater en gemensam bastyp som de olika konkreta grönsakstyperna utvidgar med nyckelordet **extends**. Man säger att subtyperna Gurka och Tomat **ärver** egenskaperna hos supertypen Grönsak.

Attributet `vikt` i traiten Grönsak nedan initialiseras inte förrän konstruktörerna anropas när vi gör `new` på någon av klasserna Gurka eller Tomat.

```
1 trait Grönsak { val vikt: Int }
2 class Gurka(val vikt: Int) extends Grönsak
3 class Tomat(val vikt: Int) extends Grönsak
4 val gurkor = Vector(new Gurka(100), new Gurka(200))
5 val grönsaker = Vector(new Gurka(300), new Tomat(42))
```

- d) Vad blir det nu för typ på variabeln `grönsaker` ovan?
- e) Fungerar det nu att räkna ut summan av vikterna i `grönsaker` med `grönsaker.map(_.vikt).sum`?
- f) En trait liknar en klass, men man kan inte instansiera den och den kan inte ha några parametrar. En typ som inte kan instansieras kallas **abstrakt** (eng. *abstract*). Vad blir det för felmeddelande om du försöker göra `new` på en trait enligt nedan?

```
1 trait Grönsak{ val vikt: Int }
2 new Grönsak
```

- g) Traiten Grönsak har en abstrakt medlem `vikt`. Den sägs vara abstrakt eftersom den saknar definition – medlemmen har bara ett namn och en typ men inget värde. Du kan instansiera den abstrakta traiten Grönsak om du fyller i det som ”fattas”, nämligen ett värde på `vikt`. Man kan fylla på det som fattas i genom att ”hänga på” ett block efter typens namn vid instansiering. Man får då vad som kallas en **anonym** klass, i detta fall en ganska konstig grönsak som inte är någon speciell sorts grönsak med som ändå har en vikt.

Vad får `anonymGrönsak` nedan för typ och strängrepresenation?

```
1 val anonymGrönsak = new Grönsak { val vikt = 42 }
```

Uppgift 2. *Polymorfism i samband med arv.* Polymorfism betyder ”många skepnader”. I samband med arv innebär det att flera subtyper, till exempel Ko och Gris, kan hanteras gemensamt som om de vore instanser av samma supertyp, så som Djur. Subklasser kan implementera en metod med samma namn på olika sätt. Vilken metod som exekveras bestäms vid körtid beroende på vilken subtyp som instansieras. På så sätt kan djur komma i många skepnader.

- a) Implementera funktionen `skapaDjur` nedan så att den returnerar antingen en ny Ko eller en ny Gris med lika sannolikhet.

```
1 scala> trait Djur { def väsnas: Unit }
```

```

2 scala> class Ko    extends Djur { def väsnas = println("Muuuuuuu") }
3 scala> class Gris extends Djur { def väsnas = println("Nöffenöff") }
4 scala> def skapaDjur: Djur = ???
5 scala> val bondgård = Vector.fill(42)(skapaDjur)
6 scala> bondgård.foreach(_.väsnas)

```

- b) Lägg till ett djur av typen Häst som väsnas på lämpligt sätt och modifiera `skapaDjur` så att det skapas kor, grisar och hästar med lika sannolikhet.

Uppgift 3. *Bastypen Shape och subtyperna Rectangle och Circle.* Du ska nu skapa ett litet bibliotek för geometriska former med oföränderliga objekt implementerade med hjälp av case-klasser. De geometriska formerna har en gemensam bastyp kallad Shape. Skriv nedan kod i en editor och klistra sedan in den i REPL med kommandot :paste.

```

case class Point(x: Double, y: Double) {
  def move(dx: Double, dy: Double): Point = Point(x + dx, y + dy)
}

trait Shape {
  def pos: Point
  def move(dx: Double, dy: Double): Shape
}

case class Rectangle(
  pos: Point,
  dx: Double,
  dy: Double
) extends Shape {
  override def move(dx: Double, dy: Double): Rectangle =
    Rectangle(pos.move(dx, dy), this.dx, this.dy)
}

case class Circle(pos: Point, radius: Double) extends Shape {
  override def move(dx: Double, dy: Double): Circle =
    Circle(pos.move(dx, dy), radius)
}

```

- a) Instansiera några cirklar och rektanglar och gör några relativt förflyttningar av dina instanser genom att anropa `move`.
- b) Lägg till metoden `moveTo` i `Point`, `Shape`, `Rectangle` och `Circle` som gör en absolut förflyttning till koordinaterna `x` och `y`. Klistra in i REPL och testa på några instanser av `Rectangle` och `Circle`.
- c) Lägg till metoden `distanceTo(that: Point): Double` i case-klassen `Point` som räknar ut avståndet till en annan punkt med hjälp av `math.hypot`. Klistra in i REPL och testa på några instanser av `Point`.

- d) Lägg till en konkret metod `distanceTo(that: Shape): Double` i traiten `Shape` som räknar ut avståndet till positionen för en annan `Shape`. Klistra in i REPL och testa på några instanser av `Rectangle` och `Circle`.

Uppgift 4. Inmixning. Man kan utvidga en klass med multipla traits med nyckelordet `with`. På så sätt kan man fördela medlemmar i olika traits och återanvända gemensamma koddelar genom så kallad **inmixning**, så som nedan exempel visar.

En alternativ fågeltaxonomi, speciellt populär i Skåne, delar in alla fåglar i två specifika kategorier: Kråga respektive Ånka. Krågor kan flyga men inte simma, medan Ångor kan simma och oftast även flyga. Fågel i generell, kollektiv bemärkelse kallas på gammal skånska för Fyle.¹ Skriv in nedan kod i en editor och spara den för kommande uppgifter. Klistra in koden i REPL med kommandot `:paste`.

```
trait Fyle {
    val läte: String
    def väsnas: Unit = print(läte * 2)
    val ärSimkunnig: Boolean
    val ärFlygkunnig: Boolean
}

trait KanSimma { val ärSimkunnig = true }
trait KanInteSimma { val ärSimkunnig = false }
trait KanFlyga { val ärFlygkunnig = true }
trait KanKanskeFlyga { val ärFlygkunnig = math.random < 0.8 }

class Kråga extends Fyle with KanFlyga with KanInteSimma {
    val läte = "krax"
}

class Ånka extends Fyle with KanSimma with KanKanskeFlyga {
    val läte = "kvack"
    override def väsnas = print(läte * 4)
}
```

- a) En flitig ornitolog hittar 42 fåglar i en perfekt skog där alla fågelsorter är lika sannolika, representerat av vektorn `fyle` nedan. Skriv i REPL ett uttryck som undersöker hur många av dessa som är flygkunniga Ångor, genom att använda metoderna `filter`, `isInstanceOf`, `ärFlygkunnig` och `size`.

```
1 scala> val fyle =
2           Vector.fill(42)(if (math.random > 0.5) new Kråga else new Ånka)
3 scala> fyle.foreach(_.väsnas)
4 scala> val antalFlygångor: Int = ???
```

¹www.klangfix.se/ordlista.htm

- b) Om alla de fåglar som ornitologen hittade skulle väsnas exakt en gång var, hur många krax och hur många kvack skulle då höras? Använd metoderna `filter` och `size`, samt predikatet `ärSimKunnig` för att beräkna antalet krax respektive kvack.

```
1 scala> val antalKrax: Int = ???  
2 scala> val antalKvack: Int = ???
```

Uppgift 5. *Finala klasser.* Om man vill förhindra att man kan göra `extends` på en klass kan man göra den final genom att placera nyckelordet `final` före nyckelordet `class`.

- a) Eftersom klassificeringen av fåglar i uppgiften ovan i antingen Ångor eller Krågor är fullständig och det inte finns några subtyper till varken Ångor eller Krågor är det lämpligt att göra dessa finala. Ändra detta i din kod.
 b) Testa att ändå försöka göra en subklass `Simkråga` `extends` `Kråga`. Vad ger kompilatorn för felmeddelande om man försöker utvidga en final klass?

Uppgift 6. *Accessregler vid arv och nyckelordet `protected`.* Om en medlem i en supertyp är privat så kan man inte komma åt den i en subklass. Ibland vill man att subklassen ska kunna komma åt en medlem även om den ska vara otillgänglig i annan kod.

```
1 trait Super {  
2   private val minHemlis = 42  
3   protected val vårHemlis = 42  
4 }  
5 class Sub extends Super {  
6   def avslöja = minHemlis  
7   def kryptisk = vårHemlis * math.Pi  
8 }
```

- a) Vad blir felmeddelandet när klassen `Sub` försöker komma åt `minHemlis`?
 b) Deklarera `Sub` på nytt, men nu utan den förbjudna metoden `avslöja`. Vad blir felmeddelandet om du försöker komma åt `vårHemlis` via en instans av klassen `Sub`? Prova till exempel med (`new Sub`).`vårHemlis`
 c) Fungerar det att anropa metoden `kryptisk` på instanser av klassen `Sub`?

Uppgift 7. *Använing av `protected`.* Den flitige ornitologen från uppgift 4 ska ringmärka alla 42 fåglar hen hittat i skogen. När hen ändå håller på bestämmer hen att i även försöka ta reda på hur mycket oväsen som skapas av respektive fågelsort. För detta ändamål apterar den flitige ornitologen en linuxdator på allt infångat fyle. Du ska hjälpa ornitologen att skriva programmet.

- a) Inför en `protected var` `räknaläte` i traiten `Fyle` och skriv kod på lämpliga ställen för att räkna hur många läten som respektive fågelinstans yttrar.
 b) Inför en metod `antalLäten` som returnerar antalet krax respektive kvack som en viss fågel yttrat sedan dess skapelse.
 c) Varför inte använda `private` i stället för `protected`?



-  d) Varför är det bra att göra räknar-variabeln åtkomlig från ”utsidan”?

Uppgift 8. Typtester med `isInstanceOf` och typkonvertering med `asInstanceOf`. Gör nedan deklarationer.

```

1 scala> trait A; trait B extends A; class C extends B; class D extends B
2 scala> val (c, d) = (new C, new D)
3 scala> val a: A = c
4 scala> val b: B = d

```

- a) Rita en bild över vilka typer som ärver vilka.
 b) Vilket resultat ger dessa typtester? Varför?

```

1 scala> c.isInstanceOf[C]
2 scala> c.isInstanceOf[D]
3 scala> d.isInstanceOf[B]
4 scala> c.isInstanceOf[A]
5 scala> b.isInstanceOf[A]
6 scala> b.isInstanceOf[D]
7 scala> a.isInstanceOf[B]
8 scala> c.isInstanceOf[AnyRef]
9 scala> c.isInstanceOf[Any]
10 scala> c.isInstanceOf[AnyVal]
11 scala> c.isInstanceOf[Object]
12 scala> 42.asInstanceOf[Object]
13 scala> 42.asInstanceOf[Any]

```

- c) Vilka av dessa typkonverteringar ger felmeddelande? Vilket och varför?

```

1 scala> c.asInstanceOf[B]
2 scala> c.asInstanceOf[A]
3 scala> d.asInstanceOf[C]
4 scala> a.asInstanceOf[B]
5 scala> a.asInstanceOf[C]
6 scala> a.asInstanceOf[D]
7 scala> a.asInstanceOf[E]
8 scala> b.asInstanceOf[A]

```

Uppgift 9. Regler för `override`, `private` och `final`.

- a) Undersök överskuggning av abstrakta, konkreta, privata och finala medlemmar genom att skriva in raderna nedan en i taget i REPL. Vilka rader ger felmeddelande? Varför? Vid felmeddelande: notera hur felmeddelandet lyder och ändra deklarationen av den felande medlemmen så att koden blir kompilerbar (eller om det är enda rimliga lösningen: ta bort den felaktiga medlemmen), innan du provar efterkommande rad.

```

1 trait Super1 { def a: Int; def b = 42; private def c = "hemlis" }
2 class Sub2 extends Super1 { def a = 43; def b = 43; def c = 43 }
3 class Sub3 extends Super1 { def a = 43; override def b = 43 }
4 class Sub4 extends Super1 { def a = 43; override def c = "43" }
5 trait Super5 { final def a: Int; final def b = 42 }
6 class Sub6 extends Super5 { override def a = 43; def b = 43 }
7 class Sub7 extends Super5 { def a = 43; override def b = 43 }

```

```

8 class Sub8 extends Super5 { def a = 43; override def c = "43" }
9 trait Super9 { val a: Int; val b = 42; lazy val c: String = "lazy" }
10 class Sub10 extends Super9 { override def a = 43; override val b = 43 }
11 class Sub11 extends Super9 { val a = 43; override lazy val b = 43 }
12 class Sub12 extends Super9 { val a = 43; override var b = 43 }
13 class Sub13 extends Super9 { val a = 43; override lazy val c = "still lazy" }
14 class SubSub extends Sub13 { override val a = 44}
15 trait Super14 { var a: Int; var b = 42; var c: String }
16 class Sub15 extends Super14 { def a = 43; override var b = 43; val c = "?" }
```

- b) Skapa instanser av klasserna Sub3, Sub13 och SubSub från ovan deluppgift och undersök alla medlemmarnas värden för respektive instans. Förklara varför de har dessa värden.
- c) Läs igenom reglerna i kapitel 7 om vad som gäller vid arv och överskuggning av medlemmar. Gör några egna undersökningar i REPL som försöker bryta mot någon regel som inte testades i deluppgift a.

Uppgift 10. *Supertyp med parameter.* En trait kan inte ha någon parameter. Vill man ha en parameter till supertypen måste man använda en klass istället, enligt nedan exempel.

Utbildningsdepartementet vill i sitt system hålla koll på vissa personer och skapar därför en klasshierarki enligt nedan. Skriv in kodens i en editor och klipp sedan in den i REPL.

```

class Person(val namn: String)

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

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

class Forskare(
    namn: String,
    universitet: String,
    titel: String) extends Akademiker(namn, universitet)
```

- a) Deklarera fyra olika **val**-variabler med lämpliga namn som refererar till olika instanser av alla olika klasser ovan och ge attributen valfria initialvärden genom olika parametrar till konstruktorerna.
- b) Skriv två satser: en som först stoppar in instanserna i en Vector och en som sedan loopar igenom vektorn och skriv ut alla instansers `toString` och `namn`.
- c) Utbildningsdepartementet vill att det inte ska gå att instansiera objekt av typerna Person och Akademiker. Det kan åstadkommas genom att placera

nyckelordet **abstract** före **class**. Uppdatera koden i enlighet med detta. Vilket blir felmeddelande om man försöker instansiera en **abstract class**?

d) Utbildningsdepartementet vill slippa implementera `toString` och slippa skriva `new` vid instansiering. Gör därför om typerna `Student` och `Forskare` till case-klasser. *Tips:* För att undkomma ett kompileringsfel (vilket?) behöver du använda **override val** på lämpligt ställe.

Skapa instanser av de nya case-klasserna `Student` och `Forskare` och skriv ut deras `toString`. Hur ser utskriften ut?

e) Eftersom `Person` och `Akademiker` nu är abstrakta, vill utbildningsdepartementet att du gör om dessa typer till traits med abstrakta attribut istället för klasser. Du kan då undvika **override val** i klassparametrarna till de konkreta case-klasserna.

Man inför också en case-klass `IckeAkademiker` som man tänker använda i olika statistiska medborgarundersökningar.

Dessutom förser man alla personer med ett personnummer representerat som en `Int`.

Hur ser utbildningsdepartementets kod ut nu, efter alla ändringar? Skriv ett testprogram som skapar några instanser och skriver ut deras attribut.

-  f) I vilka sammanhang är det nödvändigt att använda en **trait** respektive en **class**.

Uppgift 11. *Uppräknade värden.* Ett sätt att hålla reda på uppräknade värden, t.ex. färgen på olika kort i en kortlek, är att använda olika heltal som får representera de olika värdena, till exempel så här:²

```
object Färg {
    val Spader = 1
    val Hjärter = 2
    val Ruter = 3
    val Klöver = 4
}
```

Dessa kan sedan användas som parametrar till denna case-klass vid skapande av kortobjekt:

```
case class Kort(färg: Int, valör: Int)
```

Man kan hålla reda på färgen med en variabel av typen `Int` och tilldela den en viss färg med ovan konstanter. Och när man skapar ett kort behöver man inte komma ihåg vilket numret är.

```
1 scala> val f = Färg.Spader
2 scala> import Färg.-
3 scala> Kort(Ruter, 7)
```

En annan fördelen med detta är att man lätt kan loopa från 1 till 4 för att gå igenom alla färger.

²Om namnkonventioner för konstanter i Scala: läs under rubriken "Constants, Values, Variable and Methods" här docs.scala-lang.org/style/naming-conventions.html

```
1  scala> val kortlek = for (f <- 1 to 4; v <- 1 to 13) yield Kort(f, v)
```

Nackdelen är att kompilatorn vid kompileringstid inte kollar om variablerna av misstag råkar ges något värde utanför det giltiga intervallet, t.ex. 42. Detta får vi själv hålla koll på vid körtid, till exempel med funktionen `require` eller `if`-satser, etc.

Istället kan man använda case-objekt enligt nedan deluppgifter och få hjälp av kompilatorn att hitta eventuella fel vid kompileringstid. Ett case-objekt är som ett vanligt singelton-objekt men det får automatiskt en `toString` sammansom namnet och kan användas i matchningar etc. (mer om `match` i kapitel 8).

a) Deklarera följande uppräknade värden som singelton objekt med gemensam bastyp i en editor och klistra in i REPL med kommandot :paste. Med nyckelordet `sealed` så ”försegglas” klassen och inga andra direkta subtyper tillåts förutom de som finns i samma kod-fil eller block. I detta exempel med kortfärgar vet vi ju att det inte finns fler än dessa fyra färger.

```
sealed trait Färg
case object Spader extends Färg
case object Hjärter extends Färg
case object Ruter extends Färg
case object Klöver extends Färg
```

Dessa kan sedan användas som parametrar till denna case-klass vid skapande av kortobjekt:

```
case class Kort(färg: Färg, valör: Int)
```

Skapa därefter några exemplinstanser av klassen `Kort`. Vad är fördelen med ovan angreppssätt jämfört med att använda heltalskonstanter?

b) Om man vill kunna iterera över alla värden är det bra om de finns i en samling med alla värden. Vi kan lägga en sådan i ett kompanjonsobjekt till bastypen. Uppdatera koden enligt nedan och klistra in på nytt i REPL med kommandot :paste. Skriv ut alla färgvärden med en `for`-sats.

```
sealed trait Färg
object Färg {
    val values = Vector(Spader, Hjärter, Ruter, Klöver)
}
case object Spader extends Färg
case object Hjärter extends Färg
case object Ruter extends Färg
case object Klöver extends Färg
```

Skapa en kortlek med 52 olika kort och blanda den sedan med `Random.shuffle` enligt nedan. Använd en dubbel `for`-sats och `yield`.

```
1  scala> val kortlek: Vector[Kort] = ???
2  scala> val blandad = scala.util.Random.shuffle(kortlek)
```

- c) Skriv en funktion `def blandadKortlek: Vector[Kort] = ???` som ger en ny blandad kortlek varje gång den anropas med metoden i föregående uppgift.
- d) Om man även vill ha en heltalsrepresentation med en medlem `toInt` för alla värden, kan man ge bastypen en parameter och i stället för en trait (som inte kan ha några parametrar) använda en abstrakt klass.

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

Skapa en funktion `färgPoäng` som räknar ut summan av heltalsrepresentationen av alla färger hos en vektor med kort, och använd den sedan för att räkna ut `färgPoäng` för de första fem korten.

```
1 scala> def blandadKortlek: Vector[Kort] = ???
2 scala> def färgPoäng(xs: Vector[Kort]): Int = ???
3 scala> färgPoäng(blandadKortlek.take(5))
```

7.1.2 Extrauppgifter

Uppgift 12. Det visar sig att vår flitige ornitolog från uppgift 4 på sidan 169sov på en av föreläsningarna i zoologi när hen var nolla på Natfak, och därför helt missat fylekategorin Pjodd. Hjälp vår stackars ornitolog så att fylehierarkin nu även omfattar Pjoddar. En Pjodd kan flyga som en Kråga men den Ärliten medan en Kråga ÄrStor. En Pjodd kvittrar dubbelt så många gånger som en Ånka kvackar. En Pjodd KanKanskeSimma där simkunnighetssannolikheten är 0.2. Låt ornitologen ånyo finna 42 slumpmässiga fåglar i skogen och filtrera fram lämpliga arter. Undersök sedan hur dessa väsnas, i likhet med deluppgift 4b.

7.1.3 Fördjupningsuppgifter

Uppgift 13. Hitta på en egen fördjupningsuppgift inspirerat av denna artikel på Stackoverflow: <http://stackoverflow.com/questions/16173477/usages-of-null-nothing-unit-in-scala>

Uppgift 14. Studera den djupa arvshierarkin i paketet numbers nedan som modellerar olika sorters tal i matematiken. Du kan även ladda ner koden här: github.com/lunduniversity/introprog/blob/master/compendium/examples/numbers.scala. Notera metoden reduce som reducerar ett tal till sin enklaste form och dess implementation överskuggas på lämpliga ställen med relevant reduktion.

- a) Skriv kod som använder de olika konkreta klasserna i **package** numbers. Om du kompilerar koden i samma bibliotek som du kör igång REPL är det bara att använda paketet direkt:

```

1 $ scalac numbers.scala
2 $ scala
3 scala> numbers. // Tryck Tab
4 AbstractComplex  AbstractNatural   AbstractReal   Frac      Nat       Polar
5 AbstractInteger  AbstractRational  Complex        Integ     Number    Real
6
7 scala> numbers.Integ(12)
8 res0: numbers.Integ = Integ(12)
9
10 scala> import numbers.Syntax._
11 import numbers.Syntax._
12
13 scala> 42.j
14 res1: numbers.Complex = Complex(Real(0),Real(42))
15
16 scala> 42.42.j
17 res2: numbers.Complex = Complex(Real(0),Real(42.42))

```

- b) Andra på metoden + i **trait** Number så att den blir abstrakt och implementera den i alla konkreta klasser.
- c) Implementera fler räknesätt och bygg vidare på koden så som du finner intressant.
- d) Gör så att metoden reduce i klassen AbstractRational använder algoritmen Greatest Common Divisor (GCD)³ så som beskrivs här: www.artima.com/pins1ed/functional-objects.html#6.8 så att täljare och nämnare blir så små som möjligt.

```

1 package numbers
2
3 trait Number {
4   def reduce: Number = this
5   def isZero: Boolean
6   def isOne: Boolean
7   def +(that: Number): Number = ???
8 }

```

³https://sv.wikipedia.org/wiki/St%C3%B6rsta_gemensamma_delare

```

9
10 object Number {
11   def Zero = Nat(0)
12   def One = Nat(1)
13   def Im(im: BigDecimal) = Complex(Real(0), Real(im))
14   def Im(im: Real) = Complex(Real(0), im)
15   def j = Complex(Real(0), Real(1))
16   def Re(re: BigDecimal) = Complex(Real(re), Real(0))
17   def Re(re: Real) = Complex(re, Real(0))
18 }
19
20 trait AbstractComplex extends Number {
21   def re: AbstractReal
22   def im: AbstractReal
23   def abs = Real(math.hypot(re.decimal.toDouble, im.decimal.toDouble))
24   def fi = Real(math.atan2(im.decimal.toDouble, re.decimal.toDouble))
25   override def isZero: Boolean = re.decimal == 0 && im.decimal == 0
26   override def isOne: Boolean = abs.decimal == 1.0
27   override def reduce: AbstractComplex = if (im.decimal == 0) re.reduce else this
28 }
29
30 case class Complex(re: Real, im: Real) extends AbstractComplex
31
32 object Complex {
33   def apply(re: BigDecimal, im: BigDecimal) = new Complex(Real(re), Real(im))
34 }
35
36 case class Polar(
37   override val abs: Real,
38   override val fi: Real
39 ) extends AbstractComplex {
40   override def re = Real(abs.decimal.toDouble * math.cos(fi.decimal.toDouble))
41   override def im = Real(abs.decimal.toDouble * math.sin(fi.decimal.toDouble))
42 }
43
44 object Polar {
45   def apply(abs: BigDecimal, fi: BigDecimal) = new Polar(Real(abs), Real(fi))
46 }
47
48
49 trait AbstractReal extends AbstractComplex {
50   def decimal: BigDecimal
51   override def isZero = decimal == 0
52   override def isOne = decimal == 1
53   override def re = this
54   override def im = Number.Zero
55   override def reduce: AbstractReal =
56     if (decimal == 0) Number.Zero else if (decimal == 1) Number.One else this
57 }
58
59 case class Real(decimal: BigDecimal) extends AbstractReal
60
61 trait AbstractRational extends AbstractReal {
62   def numerator: AbstractInteger
63   def denominator: AbstractInteger
64   override def decimal = BigDecimal(numerator.integ)
65   override def isOne = numerator.integ == denominator.integ
66   override def reduce: AbstractRational =
67     if (denominator.isOne) numerator.reduce else this // should use GCD
68 }
69

```

```
70 case class Frac(numerator: Integ, denominator: Integ) extends AbstractRational {
71   require(denominator.integ != 0, "denominator must be non-zero")
72 }
73
74 object Frac {
75   def apply(n: BigInt, d: BigInt) = new Frac(Integ(n), Integ(d))
76 }
77
78 trait AbstractInteger extends AbstractRational {
79   def integ: BigInt
80   override def numerator = this
81   override def denominator = Number.One
82   override def isZero = integ == 0
83   override def isOne = integ == 1
84   override def decimal: BigDecimal = BigDecimal(integ)
85   override def reduce: AbstractInteger =
86     if (isZero) Number.Zero else if (isOne) Number.One else this
87 }
88
89 case class Integ(integ: BigInt) extends AbstractInteger
90
91 trait AbstractNatural extends AbstractInteger
92
93 case class Nat(integ: BigInt) extends AbstractNatural{
94   require(integ >= 0, "natural numbers must be non-negative")
95 }
96
97 object Syntax {
98   implicit class IntDecorator(i: Int){ def j = Number.Im(i) }
99   implicit class DoubleDecorator(d: Double){ def j = Number.Im(d) }
100 }
```

7.2 Grupplaboration: turtlerace - team

Mål

- Kunna skapa och använda arvhierarkier och förstå dynamisk bindning.
- Kunna skapa använda en trait som bastyp i en arvhierarki.

Förberedelser

- Gör övning traits i kapitel 7.1.
- Läs på om och förstå arv
- Diskutera i din samarbetsgrupp hur ni ska dela upp koden mellan er i flera olika delar, som ni kan arbeta med var för sig. En sådan del kan vara en klass, en trait, ett objekt, ett paket, eller en funktion.
- Varje del ska ha en *huvudansvarig* individ.
- Arbetsfördelningen ska vara någorlunda jämt fördelad mellan gruppmedlemmarna.
- När ni redovisar er lösning ska ni börja med att redogöra för handledaren hur ni delat upp koden och vem som är huvudansvarig för vad.
- Den som är huvudansvarig för en viss del redovisar den delen.
- Grupplaborationer görs i huvudsak som hemuppgift. Salstiden används primärt för redovisning.

7.2.1 Bakgrund

I labben kommer sköldpaddor, Turtle, att få tävla mot varandra i ett lopp. Racet kommer att köras i ett RaceWindow som ärver från SimpleWindow. Först kommer en RaceTurtle att skapas, som ärver från Turtle och sedan kommer sköldpaddor med olika egenskaper att skapas. Dessa sköldpaddor kommer sedan att tävla i en turnering med 32 sköldpaddor. Det kommer att vara åtta sköldpaddor i varje deltävling och turneringen kommer att bestå av fyra kvartsfinaler, två semifinaler och tillsist en final.

7.2.2 Obligatoriska uppgifter

Uppgift 1. ColorTurtle.

- a) Om du använder Eclipse: högerklicka på projektet w07_turtlerace_team och välj Build Path → Configure Build Path. Välj fliken Projects och tryck Add.... Markera projektet w06_turtlegraphics och tryck OK.

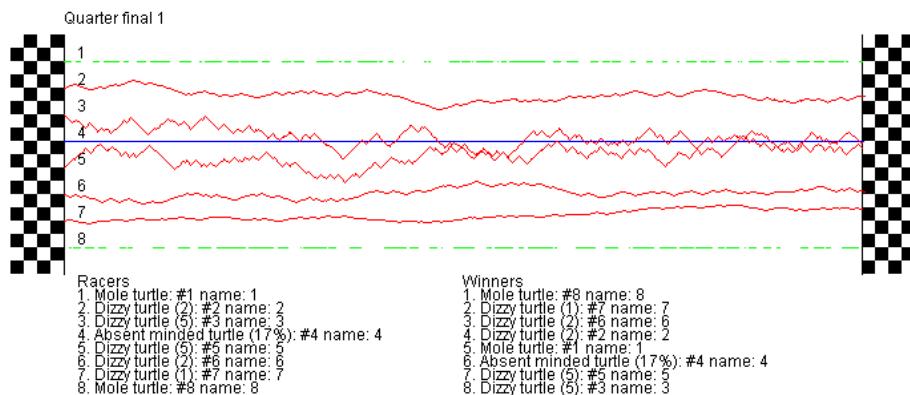
Om du använder IntelliJ: tryck på File → Project Structure.... Välj sedan modules och markera w07_turtlerace_team och tryck på fliken Dependencies. Tryck på det gröna plusstecknet till höger och välj JARs or directories.... Navigera till w06_turtlegraphics och tryck på OK.

- b) Skapa en ny klass ColorTurtle som ärver från Turtle och tar in en extra parameter color av typen java.awt.Color. Turtle måste importeras från paketet turtlegraphics.

- c) Använd **override** på metoden `forward`, där färgen i `SimpleWindow` sparas undan och sedan ändras till den angivna färgen `color`. Anropa sedan metoden `forward` i `Turtle` och byt tillbaka färgen i `SimpleWindow` till den färg som sparades undan.

Uppgift 2. RaceWindow

- a) Skapa `RaceWindow` som ärver från `SimpleWindow` med fix storlek och förbestämd titel. Lämplig storlek är 800x400. Det ska finnas två attribut `startX` och `endX` som är förbestämda och motsvarar start och mål.
- b) Skapa tre metoder `startX`, `endX` och `startY`. `startX` och `endX` ska returnera värdet på motsvarande attribut. `startY` ska ta in ett heltalet `nbr` som är ett startnummer och returnera y-värdet för det startnumret. `startX` och `endX` är x-värdet för start- resp. mållinjen och `startY` är y-värdet för varje sköldpaddas startposition på startlinjen.
- c) Skapa en metod `draw` som ritar upp ett race i fönstret och skriver ut startnummer. Definiera gärna en klass som ritar upp ett start-/målfält. Bilden är enbart ett exempel på vad man kan göra. Rita något kul! Titeln och alla tävlade turtles kommer att skrivas ut senare i uppgiften.



- d) Skapa en metod `printTitle` som tar in en sträng och skriver ut den som titel på racet. Denna sträng skiljer sig från `SimpleWindow`-konstruktorns titelsträng, och används för att inuti fönstret skriva ut titeln för det nuvarande racet. Titelsträngen i `SimpleWindow` anger istället titeln för hela fönstret.

```
class RaceTurtle(private val w: RaceWindow,
var nbr: Int, val name: String) {
/**
 * Takes one step of a random length 1 to 5
 */
def raceStep(): Unit = ???
```

```

/***
 * Restarts the turtle at the finish line.
 * To be used before each race
 */
def restart: Unit = ???

override def toString: String = ???
}

```

Uppgift 3. RaceTurtle.

- a) Implementera klassen RaceTurtle som ska ärva från Turtle. Turtle och Point behöver importeras från `turtlegraphics`. Startpositionen för en RaceTurtle hämtas från RaceWindow.
Exempel på import-sats: `import turtlegraphics.Turtle`.
- b) Skapa en metod `printRacers` i RaceWindow som tar in en lista med RaceTurtle, ett *x*-värde för placering av utskriften i x-led, samt en titel på listan. Därefter skrivas listan ut vid angiven *x*-position och vid lämplig *y*-position. Se exemplet med listan *Racers* och *Winners*.
- c) Det går nu att testa RaceTurtle (för det krävs `TurtleRace`).
- d) Ändra så att RaceTurtle istället ärver från ColoTurtle. RaceTurtle behöver då ta in en parameter av typen `Color`.

Uppgift 4. TurtleRace

- a) Implementera metoden `race` som ska börja med att skriva ut tävlande och titel i RaceWindow. Skapa en tom `ArrayBuffer` med namnet `winners` (för detta behöver man importera `scala.collection.mutable.ArrayBuffer`). Låt varje RaceTurtle ta ett steg tills en passerar mållinjen. Lägg över alla som passerat mållinjen i `winners` och kör detta tills alla RaceTurtle har hamnat i `winners`. Använd `SimpleWindow.delay(5)` mellan varje steg för att se animeringen. Skriv ut vinnarna i RaceWindow och vänta på musklick. Returnera listan med vinnare.
- b) Testa att köra ett race mellan åtta sköldpaddor.

Uppgift 5. Dizziness, AbsentMindness och Mole

- a) Implementera tre `trait` som ärver från `RaceTurtle` och som `override` `raceStep` och `toString`. `toString` ska utöver `toString` från `RaceTurtle` även bestå av vilken typ av `RaceTurtle` det är och graden av yrsel/tankspriddhet den har.
- **Dizziness.** Slumpa ett heltalet `dizziness` mellan 1 och 5. För varje steg ska en riktningsförändring slumpas fram som blir större desto större `dizziness` är. Slumpa även om den avviker åt höger eller vänster och använd `turnRight` och `turnLeft`.
 - **AbsentMindness.** Slumpa ett heltalet `absent` mellan 0 och 99 som anger

i procent hur tankspridd RaceTurtle är. För varje steg ska det vara `absent%` chans att ett steg inte tas.

- Mole. Med 50% sannolikhet ska denna typ RaceTurtle gräva ner sig i marken. För varje steg ska det vara 50% chans att pennan är uppe och 50% chans att pennan är nere. Använd metoderna `penUp` och `penDown`.

Uppgift 6. TurtleTournament

- Börja med att skapa en hjälpmetod `randTurtle` som tar in ett `RaceWindow`, ett nummer och ett namn som parameter. Slumpa med lika stor sannolikhet mellan att skapa en `ColorTurtle` med en av de tre olika egenskaperna och låt de olika egenskaperna ha olika färger.
- Skapa ett `RaceWindow`. Slumpa fram 32 sköldpaddor och låt åtta av dem tävla åt gången, i totalt fyra st `TurtleRace`. Ta vara på vinnarna och låt de fyra bästa från varje lopp köra två lopp till. De fyra bästa från båda dessa loppen går vidare till finalen. **Tänk på att rensa `RaceWindow` efter varje lopp och rita ut det på nytt innan varje lopp.**

Kapitel 8

Repetition, specialundervisning

Begrepp som ingår i denna veckas studier:

- REBOOT CAMP: repetera
- identifiera kunskapsluckor
- kom-i-kapp med övningar och labbar
- specialundervisning med hårdträning för behövande

Kapitel 9

Mönster, undantag

Begrepp som ingår i denna veckas studier:

- mönstermatchning
- match
- Option
- throw
- try
- catch
- Try
- unapply
- sealed
- flatten
- flatMap
- partiella funktioner
- collect
- speciella matchningar: wildcard pattern; variable binding; sequence wildcard; back-ticks
- equals
- hashCode
- exempel: equals för klassen Complex
- switch-sats i Java

Javas switch-sats

```
public class Switch {  
    public static void main(String[] args) {  
        String favorite = "selleri";  
        if (args.length > 0) {  
            favorite = args[0];  
        }  
        System.out.println("Din favoritgrönsak: " + favorite);  
        char firstChar = Character.toLowerCase(favorite.charAt(0));  
        System.out.print("Jag tycker ");  
        switch (firstChar) {  
            case 'g':  
                System.out.println("gurka är gott!");  
                break;  
            case 't':  
                System.out.println("tomat är gott!");  
                break;  
            case 'b':  
                System.out.println("brocolli är gott!");  
                break;  
            default:  
                System.out.println(favorite + " är mindre gott...");  
                break;  
        }  
    }  
}
```

9.1 Övning: matching

Mål

- Kunna skapa och använda **match**-uttryck med konstanta värden, garder och mönstermatchning med case-klasser.
- Kunna skapa och använda case-objekt för matchningar på uppräknade värden.
- Känna till betydelsen av små och stora begynnelsebokstäver i case-grenar i en matchning, samt förstå hur namn binds till värden in en case-gren.
- Kunna hantera saknade värden med hjälp av typen Option och mönstermatchning på Some och None.
- Känna till hur metoden unapply används vid mönstermatchning.
- Känna till nyckelordet **sealed** och förstå nyttan med förseglade typer.
- Känna till **switch**-satser i Java.
- Känna till **null**.
- Kunna fånga undantag med **try-catch** och `scala.util.Try`.
- Känna till skillnaderna mellan **try-catch** i Scala och java.
- Kunna implementera equals med hjälp av en **match**-sats, som fungerar för finala klasser utan arv.
- Känna till relationen mellan hashCode och equals.
- Kunna använda flatMap tillsammans med Option och Try. ???
- Kunna skapa partiella funktioner med case-uttryck. ???

Förberedelser

- Studera begreppen i kapitel 9.

9.1.1 Grunduppgifter

Uppgift 1. Hur fungerar en **switch**-sats i Java (och flera andra språk)? Det händer ofta att man vill testa om ett värde är ett av många olika alternativ. Då kan man använda en sekvens av många **if-else**, ett för varje alternativ. Men det finns ett annat sätt i Java och många andra språk: man kan använda **switch** som kollar flera alternativ i en och samma sats, se t.ex. en.wikipedia.org/wiki/Switch_statement.

- a) Skriv in nedan kod i en kodeditor. Spara med namnet Switch.java och kompilera filen med kommandot javac Switch.java. Kör den med java Switch och ange din favoritgrönsak som argument till programmet. Vad händer? Förklara hur **switch**-satsen fungerar.

```

1 public class Switch {
2     public static void main(String[] args) {
3         String favorite = "selleri";
4         if (args.length > 0) {
5             favorite = args[0];
6         }
7         System.out.println("Din favoritgrönsak: " + favorite);

```

```

8   char firstChar = Character.toLowerCase(favorite.charAt(0));
9   System.out.print("Jag tycker att ");
10  switch (firstChar) {
11    case 'g':
12      System.out.println("gurka är gott!");
13      break;
14    case 't':
15      System.out.println("tomat är gott!");
16      break;
17    case 'b':
18      System.out.println("broccoli är gott!");
19      break;
20    default:
21      System.out.println(favorite + " är mindre gott...");
22      break;
23  }
24 }
25 }
```

- b) Vad händer om du tar bort **break**-satsen på rad 16?

Uppgift 2. Matcha på konstanta värden. I Scala finns ingen **switch**-sats. I stället har Scala ett **match**-uttryck som är mer kraftfullt. Dock saknar Scala nyckelordet **break** och Scalas **match**-uttryck kan inte ”falla igenom” som skedde i uppgift 1b.

- a) Skriv nedan program med en kodeditor och spara i filen `Match.scala`. Kompilera med `scalac Match.scala`. Kör med `scala Match` och ge som argument din favoritgrönsak. Vad händer? Förklara hur ett **match**-uttryck fungerar.

```

1 object Match {
2   def main(args: Array[String]): Unit = {
3     val favorite = if (args.length > 0) args(0) else "selleri"
4     println("Din favoritgrönsak: " + favorite)
5     val firstChar = favorite.toLowerCase.charAt(0)
6     val meThink = firstChar match {
7       case 'g' => "gurka är gott!"
8       case 't' => "tomat är gott!"
9       case 'b' => "broccoli är gott!"
10      case _ => s"$favorite är mindre gott..."
11    }
12    println(s"Jag tycker att $meThink")
13  }
14 }
```

- b) Vad blir det för felmeddelande om du tar bort case-grenen för defaultvärdet och indata väljs så att inga case-grenar matchar? Är det ett exekveringsfel eller ett kompileringsfel?
- c) Beskriv några skillnader i syntax och semantik mellan Javas flervalssats  **switch** och Scalas flervalsuttryck **match**.

Uppgift 3. Gard i case-grenar. Med hjälp en gard (eng. *guard*) i en case-gren kan man begränsa med ett villkor om grenen ska väljas.

Utgå från koden i uppgift 2a och byt ut case-grenen för 'g'-matchning till nedan variant med en gard med nyckelordet **if** (notera att det inte behövs parenteser runt villkoret):

```
case 'g' if math.random > 0.5 => "gurka är gott ibland..."
```

Kompilera om och kör programmet upprepade gånger med olika indata tills alla grenar i **match**-uttrycket har exekverats. Föklara vad som händer.

Uppgift 4. *Mönstermatcha på attributen i case-klasser.* Scallas **match**-uttryck är extra kraftfulla om de används tillsammans med **case**-klasser: då kan attribut extraheras automatiskt och bindas till lokala variabler direkt i case-grenen som nedan exempel visar (notera att **v** och **rutten** inte behöver deklareraras **explicit**). Detta kallas för **mönstermatchning**.

- a) Vad skrivs ut nedan? Varför? Prova att byta namn på **v** och **rutten**.

```
1 scala> case class Gurka(vikt: Int, ärRutten: Boolean)
2 scala> val g = Gurka(100, true)
3 scala> g match { case Gurka(v,rutten) => println("G" + v + rutten) }
```

- b) Skriv sedan nedan i REPL och tryck TAB två gånger efter punkten. Vad har **unapply**-metoden för resultattyp?

```
1 scala> Gurka.unapply // Tryck TAB två gånger
```

Bakgrund för kännedom: Case-klasser får av komplatorn automatiskt ett kompanjonsobjekt (eng. *companion object*), i detta fallet **object** **Gurka**. Det objektet får av komplatorn automatiskt en **unapply**-metod. Det är **unapply** som anropas ”under huven” när case-klassernas attribut extraheras vid mönstermatchning, men detta sker alltså automatiskt och man behöver inte explicit nyttja **unapply** om man inte själv vill implementera s.k. extraherare (eng. *extractors*); om du är nyfiken på detta, se fördjupningsuppgift 22.

- c) Anropa **unapply**-metoden enligt nedan. Vad blir resultatet?

```
1 scala> Gurka.unapply(g)
```

Vi ska i senare uppgifter undersöka hur typerna **Option** och **Some** fungerar och hur man kan ha nytta av dessa i andra sammanhang.

- d) Spara programmet nedan i filen **vegomatch.scala** och kompilera med **scalac vegomatch.scala** och kör med **scala vegomatch.Main 1000** i terminalen. Föklara hur predikatet **ärÄtvärd** fungerar.

```
1 package vegomatch
2
3 trait Grönsak {
4   def vikt: Int
5   def ärRutten: Boolean
6 }
7
8 case class Gurka(vikt: Int, ärRutten: Boolean) extends Grönsak
9 case class Tomat(vikt: Int, ärRutten: Boolean) extends Grönsak
```

```

10
11 object Main {
12   def slumpvikt: Int      = (math.random * 420 + 42).toInt
13   def slumprutten: Boolean = math.random > 0.8
14   def slumpgurka: Gurka    = Gurka(slumpvikt, slumprutten)
15   def slumptomat: Tomat    = Tomat(slumpvikt, slumprutten)
16   def slumpgrönsak: Grönsak =
17     if (math.random > 0.2) slumpgurka else slumptomat
18
19   def ärÄtvärd(g: Grönsak): Boolean = g match {
20     case Gurka(v, rutten) if v > 100 && !rutten => true
21     case Tomat(v, rutten) if v > 50 && !rutten => true
22     case _ => false
23   }
24
25   def main(args: Array[String]): Unit = {
26     val skörd = Vector.fill(args(0).toInt)(slumpgrönsak)
27     val ätvärda = skörd.filter(ärÄtvärd)
28     println("Antal skördade grönsaker: " + skörd.size)
29     println("Antal ätvärda grönsaker: " + ätvärda.size)
30   }
31 }
```

Uppgift 5. Man kan åstadkomma urskiljningen av de ätbara grönsakerna i uppgift 4 med polymorfism i stället för **match**.

a) Gör en ny variant av ditt program enligt nedan riktlinjer och spara den modifierade koden i filen `vegopoly.scala` och kompilera och kör.

- Ta bort predikatet `ärÄtvärd` i objektet `Main` och inför i stället en abstrakt metod `def ärÄtbar: Boolean` i traiten `Grönsak`.
- Inför konkreta `val`-medlemmar i respektive grönsak som definierar ätbarheten.
- Ändra i huvudprogrammet i enlighet med ovan ändringar så att `ärÄtvärd` anropas som en metod på de skördade grönsaksobjekten när de ätvärda ska filtreras ut.

b) Lägg till en ny grönsak `case class Broccoli` och definiera dess ätbarhet. Ändra i slump-funktionerna så att broccoli blir ovanligare än gurka.

c) Jämför lösningen med `match` i uppgift 4 och lösningen ovan med polymorfism. Vilka är för- och nackdelarna med respektive lösning. Diskutera två olika situationer på ett hypotetiskt företag som utvecklar mjukvara för jordbrukssektorn: 1) att uppsättningen grönsaker inte ändras särskilt ofta medan definitionerna av ätbarhet ändras väldigt ofta och 2) att uppsättningen grönsaker ändras väldigt ofta men att ätbarhetsdefinitionerna inte ändras särskilt ofta.



Uppgift 6. Matcha på case-objekt och nyttan med `sealed`. Skapa nedan kod i en editor, och klistica in i REPL med kommandot :pa. Notera nyckelordet `sealed` som används för att förseglia en typ. En **förseglad typ** måste ha alla sina subtyper i en och samma kodfil.

```
sealed trait Färg
object Färg {
    val values = Vector(Spader, Hjärter, Ruter, Klöver)
}
case object Spader extends Färg
case object Hjärter extends Färg
case object Ruter extends Färg
case object Klöver extends Färg
```

- a) Skapa en funktion **def** parafärg(f: Färg): Färg i en editor, som med hjälp av ett match-uttryck returnerar parallelfärgen till en färg. Parallelfärgen till Hjärter är Ruter och vice versa, medan parallelfärgen till Klöver är Spader och vice versa. Klistra in funktionen i REPL.

```
1 scala> parafärg(Spader)
2 scala> val xs = Vector.fill(5)(Färg.values((math.random * 4).toInt))
3 scala> xs.map(parafärg)
```

- b) Vi ska nu undersöka vad som händer om man glömmer en av case-grenarna i matchningen i parafärg? "Glöm" alltså avsiktligt en av case-grenarna och klistra in den nya parafärg med den ofullständiga matchningen. Hur lyder varningen? Kommer varningen vid körtid eller vid kompilering?
- c) Anropa parafärg med den "glömda" färgen. Hur lyder felmeddelandet? Är det ett kompileringsfel eller ett körtidsfel?
-  d) Förlara vad nyckelordet **sealed** innebär och vilken nytta man kan ha av att **försägla** en supertyp.

Uppgift 7. *Betydelsen av små och stora begynnelsebokstäver vid matchning.* För att åstadkomma att namn kan bindas till variabler vid matchning utan att de behöver deklarerats i förväg (som vi såg i uppgift 4a) så har identifierare med liten begynnelsebokstav fått speciell betydelse: den tolkas av kompilatorn som att du vill att en variabel binds till ett värde vid matchningen. En identifierare med stor begynnelsebokstav tolkas dock emot som ett konstant värde (t.ex. ett case-objekt eller ett case-klass-mönster).

- a) *En case-gren som fångar allt.* En case-gren med en identifierare med liten begynnelsebokstav som saknar gard kommer att matcha allt. Prova nedan i REPL, men försök lista ut i förväg vad som kommer att hänta. Vad händer?

```
1 scala> val x = "urka"
2 scala> x match {
3     case str if str.startsWith("g") => println("kanske gurka")
4     case vadsomhelst => println("ej gurka: " + vadsomhelst)
5 }
6 scala> val g = "gurka"
7 scala> g match {
8     case str if str.startsWith("g") => println("kanske gurka")
9     case vadsomhelst => println("ej gurka: " + vadsomhelst)
10 }
```

- b) *Fallgrop med små begynnelsbokstäver.* Innan du provar nedan i REPL, försök gissa vad som kommer att hänta. Vad händer? Hur lyder varningarna och vad innebär de?

```

1 scala> val any: Any = "gurka"
2 scala> case object Gurka
3 scala> case object tomat
4 scala> any match {
5     case Gurka => println("gurka")
6     case tomat => println("tomat")
7     case _ => println("allt annat")
8 }
```

- c) *Använd backticks för att tvinga fram match på konstant värde.* Det finns en utväg om man inte vill att kompilatorn ska skapa en ny lokal variabel: använd specialtecknet *backtick*, som skrivs ` och kräver speciella tangentbordstryck.¹ Gör om föregående uppgift men omgärda nu identifieraren *tomat* i *tomat*-case-grenen med backticks, så här: **case `tomat` => ...**

Uppgift 8. *Använda Option och matcha på värden som kanske saknas.* Man behöver ofta skriva kod för att hantera värden som eventuellt saknas, t.ex. saknade telefonnummer i en persondatabas. Denna situation är så pass vanlig att många språk har speciellt stöd för saknande värden.

I Java² används värdet **null** för att indikera att en referens saknar värde. Man får då komma ihåg att testa om värdet saknas varje gång sådana värden ska behandlas, , t.ex. med **if (ref != null) { ... } else { ... }**. Ett annat vanligt trick är att låta -1 indikera saknade positiva heltal, till exempel saknade index, som får behandlas med **if (i != -1) { ... } else { ... }**.

I Scala finns en speciell typ *Option* som möjliggör smidig och typsäker hantering av saknade värden. Om ett kanske saknat värde packas in i en *Option* (eng. *wrapped in an Option*), finns det i en speciell slags samling som bara kan innehålla *inget* eller *något* värde, och alltså har antingen storleken 0 eller 1.

- a) Förklara vad som händer nedan.

```

1 scala> var kanske: Option[Int] = None
2 scala> kanske.size
3 scala> kanske = Some(42)
4 scala> kanske.size
5 scala> kanske.isEmpty
6 scala> kanske.isDefined
7 scala> def ökaOmFinns(opt: Option[Int]): Option[Int] = opt match {
8     case Some(i) => Some(i + 1)
9     case None     => None
10    }
11 scala> val annanKanske = ökaOmFinns(kanske)
12 scala> def öka(i: Int) = i + 1
13 scala> val merKanske = kanske.map(öka)
```

¹Fråga någon om du inte hittar hur man gör backtick ` på ditt tangentbord.

²Scala har också **null** men det behövs bara vid samverkan med Java-kod.

- b) Mönstermatchingen ovan är minst lika knölig som en **if**-sats, men tack vare att en Option är en slags (liten) samling finns det smidigare sätt. Förklara vad som händer nedan.

```

1 val meningen = Some(42)
2 val ejMeningen = Option.empty[Int]
3 meningen.map(_ + 1)
4 ejMeningen.map(_ + 1)
5 ejMeningen.map(_ + 1).orElse(Some("saknas")).foreach(println)
6 meningen.map(_ + 1).orElse(Some("saknas")).foreach(println)

```

- c) *Samlingsmetoder som ger en Option.* Förklara för varje rad nedan vad som händer. En av raderna ger ett felmeddelande; vilken rad och vilket felmeddelande?

```

1 val xs = (42 to 84 by 5).toVector
2 val e = Vector.empty[Int]
3 xs.headOption
4 xs.headOption.get
5 xs.headOption.getOrElse(0)
6 xs.headOption.orElse(Some(0))
7 e.headOption
8 e.headOption.get
9 e.headOption.getOrElse(0)
10 e.headOption.orElse(Some(0))
11 Vector(xs, e, e, e)
12 Vector(xs, e, e, e).map(_.lastOption)
13 Vector(xs, e, e, e).map(_.lastOption).flatten
14 xs.lift(0)
15 xs.lift(1000)
16 e.lift(1000).getOrElse(0)
17 xs.find(_ > 50)
18 xs.find(_ < 42)
19 e.find(_ > 42).foreach(_ => println("HITTAT!"))

```

-  d) Vilka är fördelarna med Option jämfört med **null** eller -1 om man i sin kod glömmer hantera saknade värden?

Uppgift 9. Kasta undantag. Om man vill signalera att ett fel eller en onormal situation uppstått så kan man **kasta** (eng. *throw*) ett **undantag** (eng. *exception*). Då avbryts programmet direkt med ett felmeddelande, om man inte väljer att **fångा** (eng. *catch*) undantaget.

- a) Vad händer nedan?

```

1 scala> throw new Exception("PANG!")
2 scala> java.lang. // Tryck TAB efter punkten
3 scala> throw new IllegalArgumentException("fel fel fel")
4 scala> val carola = try {
5         throw new Exception("stormvind!")
6         42
7     } catch { case e: Throwable => println("Fångad av en " + e); -1 }

```

- b) Nämn ett par undantag som finns i paketet `java.lang` som du kan gissa vad de innebär och i vilka situationer de kastas. 
- c) Vilken typ har variabeln `carola` ovan? Vad hade typen blivit om `catch`-grenen hade returnerat en sträng i stället? 

Uppgift 10. Fånga undantag i Java med en **try-catch**-sats. Det finns som vi såg i förra uppgiften inbyggt stöd i JVM för att hantera när program avbryts på oväntade sätt, t.ex. på grund av division med noll eller ej förväntade indata från användaren. Skriv in nedan Java-program i en editor och spara i en fil med namnet `TryCatch.java` och kompilera med `javac TryCatch.java` i terminalen.

```

1 // TryCatch.java
2
3 public class TryCatch {
4     public static void main(String[] args) {
5         int input;
6         int output;
7         if (args[0].equals("safe")) {
8             try {
9                 input = Integer.parseInt(args[1]);
10                System.out.println("Skyddad division!");
11                output = 42 / input;
12            } catch (Exception e) {
13                System.out.println("Undantag fångat: " + e);
14                System.out.println("Dividerar ändå med säker default!");
15                input = 1;
16                output = 42 / input;
17            }
18        } else {
19            input = Integer.parseInt(args[0]);
20            System.out.println("Oskyddad division!");
21            output = 42 / input;
22        }
23        System.out.println("42 / " + input + " == " + output);
24    }
25 }
```

- a) Förklara vad som händer när du kör programmet med olika indata:

```

1 $ java TryCatch 42
2 $ java TryCatch 0
3 $ java TryCatch safe 42
4 $ java TryCatch safe 0
5 $ java TryCatch
```

- b) Vad händer om du ”glömmer bort” raden 15 och därmed missar att initialisera `input`? Hur lyder felmeddelandet? Är det ett körtidsfel eller kompileringsfel?
- c) Beskriv några skillnader och likheter i syntax och semantik mellan **try-catch** i Java respektive Scala. 

Uppgift 11. Fånga undantag i Scala med `scala.util.Try`. I paketet `scala.util` finns typen `Try` med stort T som är som en slags samling som kan innehålla antingen ett "lyckat" eller "misslyckat" värde. Om beräkningen av värdet lyckades och inga undantag kastas blir värdet inkapslat i en `Success`, annars blir undantaget inkapslat i en `Failure`. Man kan extrahera värdet, respektive undantaget, med mönstermatchning, men det är oftast smidigare att använda samlingsmetoderna `map` och `foreach`, i likhet med hur `Option` används. Det finns även en smidig metod `recover` på objekt av typen `Try` där man kan skicka med kod som körs om det uppstår en undantags situation.

- a) Förklara vad som händer nedan.

```

1 scala> def pang = throw new Exception("PANG!")
2 scala> import scala.util.{Try, Success, Failure}
3 scala> Try{pang}
4 scala> Try{pang}.recover{case e: Throwable => s"desarmerad bomb: $e"}
5 scala> Try{"tyst"}.recover{case e: Throwable => s"desarmerad bomb: $e"}
6 scala> def kanskePang = if (math.random > 0.5) "tyst" else pang
7 scala> def kanskeOk = Try{ kanskePang}
8 scala> val xs = Vector.fill(100)(kanskeOk)
9 scala> xs(13) match {
10     case Success(x) => ":)"
11     case Failure(e) => ":( " + e
12 }
13 scala> x(13).isSuccess
14 scala> x(13).isFailure
15 scala> xs.count(_.isFailure)
16 scala> xs.find(_.isFailure)
17 scala> val badOpt = xs.find(_.isFailure)
18 scala> val goodOpt = xs.find(_.isSuccess)
19 scala> badOpt
20 scala> badOpt.get
21 scala> badOpt.get.get
22 scala> badOpt.map(_.getOrElse("bomben desarmerad!")).get
23 scala> goodOpt.map(_.getOrElse("bomben desarmerad!")).get
24 scala> xs.map(_.getOrElse("bomben desarmerad!")).foreach(println)
25 scala> xs.map(_.toOption)
26 scala> xs.map(_.toOption).flatten
27 scala> xs.map(_.toOption).flatten.size

```

- b) Vad har funktionen `pang` för returtyp?
 c) Varför får funktionen `kanskePang` den härledda returtypen `String`?

Uppgift 12. Metoden `equals`. Om man överskuggar den befintliga metoden `equals` så kommer metoden `==` att fungera annorlunda. Man kan då själv åstadkomma innehållslikhet i stället för referenslikhet. Vi börjar att studera den befintliga `equals` med referenslikhet.

- a) Vad händer nedan? Om du trycker TAB två gånger efter ett metodnamn får du se metodens signatur. Vilken signatur har metoden `equals`?

```

1 scala> class Gurka(val vikt: Int, ärÄtbar: Boolean)
2 scala> val g1 = new Gurka(42, true)
3 scala> val g2 = g1

```

```

4 scala> val g3 = new Gurka(42, true)
5 scala> g1 == g2
6 scala> g1 == g3
7 scala> g1.equals // tryck TAB två gånger

```

- b) Rita minnessituationen efter rad 4.
- c) Överskugga metoderna `equals` och `hashCode`.

Bakgrund för kännedom: Det visar sig förvånande komplicerat att implementera innehållslikhet med metoden `equals` så att den ger bra resultat under alla speciella omständigheter. Till exempel måste man även överskugga en metod vid namn `hashCode` om man överskuggar `equals`, eftersom dessa båda används gemensamt av effektivitetsskäl för att skapa den interna lagringen av objekten i vissa samlingar. Om man missar det kan objekt bli ”osynliga” i `hashCode`-baserade samlingar – men mer om detta i senare kurser. Om objekten ingår i en öppen arvhierarki blir det också mer komplicerat; det är enklare om man har att göra med finala klasser. Dessutom krävs speciella hänsyn om klassen har en typparameter.

Definiera klassen nedan i REPL med överskuggade `equals` och `hashCode`; den ärver inte något och är final.

```

// fungerar fint om klassen är final och inte ärver något
final class Gurka(val vikt: Int, ärÄtbar: Boolean) {
    override def equals(other: Any): Boolean = other match {
        case that: Gurka => this.vikt == that.vikt
        case _ => false
    }
    override def hashCode: Int = (vikt, ärÄtbar).## //förklaras sen
}

```

- d) Vad händer nu nedan, där `Gurka` nu har en överskuggad `equals` med innehållslikhet?

```

1 scala> val g1 = new Gurka(42, true)
2 scala> val g2 = g1
3 scala> val g3 = new Gurka(42, true)
4 scala> g1 == g2
5 scala> g1 == g3

```

- e) Hur märker man ovan att den överskuggade `equals` medför att `==` nu ger innehållslikhet? Jämför med deluppgift a.

I uppgift 19 får du prova på att följa det fullständiga receptet i 8 steg för att överskugga en `equals` enligt konstens alla regler. I efterföljande kurs kommer mer träning i att hantera innehållslikhet och hash-koder. I Scala får man ett objekts hash-kod med metoden `##`.³

³Om du är nyfiken på hash-koder, läs mer här: [en.wikipedia.org/wiki/Java_hashCode\(\)](https://en.wikipedia.org/wiki/Java_hashCode()).

9.1.2 Extrauppgifter

Uppgift 13. *Polynom.* Med hjälp av koden nedan, kan man göra följande:

```

1  scala> :pa polynomial.scala
2
3  scala> import polynomial._
4
5  scala> Const(1) * x
6  res0: polynomial.Term = x
7
8  scala> (x*5)^2
9  res1: polynomial.Prod = 25x^2
10
11 scala> Poly(x*(-5), y^4, (z^2)*3)
12 res2: polynomial.Poly = -5x + y^4 + 3z^2

```

-  a) Förlära vad som händer ovan genom att studera koden för **object** `polynomial` nedan i filen `polynomial.scala`.⁴

```

1  object polynomial {
2
3    sealed trait Term {
4      def *(that: Term): Term
5    }
6
7    case class Const(value: BigDecimal) extends Term {
8
9      def toSilentString: String = this match {
10        case Const.One      => ""
11        case Const.MinusOne => "-"
12        case _               => value.toString
13      }
14
15      override def toString = value.toString
16
17      override def *(that: Term): Term = that match {
18        case Const(d)   => Const(d * value)
19        case v: Var    => Prod(this, Set(v))
20        case Prod(c, vs) => Prod(Const(c.value * value), vs)
21      }
22
23      def *(d: BigDecimal): Const = Const(d * value)
24
25      def ^(e: Int): Const = Const(value.pow(e))
26
27    }
28
29    object Const {
30      final val Zero     = Const(BigDecimal(0))
31      final val One      = Const(BigDecimal(1))
32      final val MinusOne = Const(BigDecimal(-1))
33    }
34  }

```

⁴Koden finns även här:

github.com/lunduniversity/introprog/tree/master/compendium/examples/polynomial

```

33 }
34
35 case class Var(name: Char, exp: Int = 1) extends Term {
36
37   private def silentExpString: String =
38     if (exp == 1) "" else "^"+exp.toString
39
40   override def toString = s"$name$silentExpString"
41
42   def ^(e: Int): Var = Var(name, e * exp)
43
44   def *(c: BigDecimal) = Prod(Const(c), Set(this))
45
46   override def *(that: Term): Term = that match {
47     case c: Const => Prod(c, Set(this))
48
49     case v: Var =>
50       if (v.name == name) Var(name, v.exp + exp)
51       else Prod(Const.One, Set(this, v))
52
53     case p: Prod => p * this
54   }
55
56 }
57
58 object Var{
59
60   def apply(d: BigDecimal, name: Char): Prod =
61     Prod(Const(d), Set(Var(name)))
62
63   def apply(d: BigDecimal, name: Char, exp: Int): Prod =
64     Prod(Const(d), Set(Var(name, exp)))
65
66   def addExp(v1: Var, v2: Var): Var = Var(v1.name, v1.exp + v2.exp)
67
68   def multiply(v1: Var, vs: Set[Var]): Set[Var] = {
69     if (!vs.contains(v1)) vs + v1
70     else vs.map(v2 => if (v1.name == v2.name) addExp(v1, v2) else v2)
71   }
72
73   def multiply(vs1: Set[Var], vs2: Set[Var]): Set[Var] = {
74     var result = vs2
75     vs1.foreach{ v1 => result = multiply(v1, result) }
76     result
77   }
78
79 }
80
81 case class Prod(const: Const, vars: Set[Var]) extends Term {
82
83   override def toString = s"${const.toSilentString}${vars.mkString}"
84
85   override def *(that: Term): Term = that match {
86     case Const(d) => Prod(Const(d * const.value), vars)
87
88     case v: Var => Prod(const, Var.multiply(v, vars))

```

```

89
90     case Prod(Const(d), vs)  =>
91         Prod(Const(const.value * d), Var.multiply(vs, vars))
92     }
93
94     def ^(e: Int) = Prod(const ^ e, vars.map(_ ^ e))
95   }
96
97   case class Poly(xs: Set[Term]) {
98     override def toString = xs.mkString(" + ")
99   }
100
101 object Poly {
102   def apply(ts: Term*) : Poly = Poly(ts.toSet)
103 }
104
105 val (x, y, z, s, t) = (Var('x'), Var('y'), Var('z'), Var('s'), Var('t'))
106 }
107 }
```

- b) Bygg vidare på **object** polynomial och implementera addition mellan olika termer.



Uppgift 14. Studera dokumentationen för Option här och se om du känner igen några av metoderna som också finns på samlingen Vector:
www.scala-lang.org/api/current/index.html#scala.Option
Förklara hur metoden contains på en Option fungerar med hjälp av dokumentationens exempel.

Uppgift 15. Gör motsvarande program i Scala som visas i uppgift 10, men utnyttja att Scalas **try-catch** är ett uttryck. Kompilera och kör och testa så att de ur användarens synvinkel fungerar precis på samma sätt. Notera de viktigaste skillnaderna mellan de båda programmen.

9.1.3 Fördjupningsuppgifter

Uppgift 16. Bygg vidare på **object** `polynomial` i uppgift 13 på sidan 197 och implementera metoden **def** `reduce`: `Poly` i case-klassen `Poly` som förenklar polynom om flera Prod-termer kan adderas.

Uppgift 17. Läs om hash-koder här: [`en.wikipedia.org/wiki/Java_hashCode\(\)`](https://en.wikipedia.org/wiki/Java_hashCode())

Vad ger metoden `##` i `scala.Any` för resultat? Läs dokumentationen här:

[`www.scala-lang.org/api/current/#scala.Any`](https://www.scala-lang.org/api/current/#scala.Any)



Uppgift 18. *Typsäker innehållstest med metoden `==`.* Metoderna `equals` och `==` tillåter jämförelse med vad som helst. Ibland vill man ha en typsäker innehållsjämförelse som bara tillåter jämförelse av objekt av en mer specifik typ och ger kompileringsfel annars. Man brukar då definiera en metod `==` som har en parameter `that` som har en så specifik typ som önskas. Inför nedan abstrakta metod `==` i traiten `polynomial.Term` i uppgift 13 på sidan 197 och överskugga den sedan i alla subklasser till `Term`. Testa så att du får kompileringsfel om du försöker jämföra en `Term` med något helt annat, t.ex. en `String` eller `Vector`.

```
def ==(that: Term): Boolean
```

Uppgift 19. *Överskugga `equals` med innehållslikhet även för icke-finala klasser.* Nedan visas delar av klassen `Complex` som representerar ett komplext tal med realdel och imaginärdel. I stället för att, som man ofta gör i Scala, använda en case-klass och en `equals`-metod som automatiskt ger innehållslikhet, ska du träna på att implementera en egen `equals`.

```
class Complex(re: Double, im: Double) {
    def abs: Double = math.hypot(re, im)
    override def toString = s"Complex($re, $im)"
    def canEqual(other: Any): Boolean = ???
    override def hashCode: Int = ???
    override def equals(other: Any): Boolean = ???
}
case object Complex {
    def apply(re: Double, im: Double): Complex = new Complex(re, im)
}
```

Följ detta **receipt**⁵ i 8 steg för att överskugga `equals` med innehållslikhet som fungerar även för klasser som inte är **final**:

1. Inför denna metod: **def** `canEqual(other: Any): Boolean`
Observera att typen på parametern ska vara `Any`. Om detta görs i en subklass till en klass som redan implementerat `canEqual`, behövs även **override**.

⁵Detta recept bygger på [`http://www.artima.com/pinsled/object-equality.html`](http://www.artima.com/pinsled/object-equality.html)

2. Metoden `canEqual` ska ge `true` om `other` är av samma typ som `this`, alltså till exempel:
`def canEqual(other: Any): Boolean = other.isInstanceOf[Complex]`
3. Inför metoden `equals` och var noga med att parametern har typen `Any`:
`override def equals(other: Any): Boolean`
4. Implementera metoden `equals` med ett `match`-uttryck som börjar så här:
`other match { ... }`
5. Match-uttrycket ska ha två grenar. Den första grenen ska ha ett typat mönster för den klass som ska jämföras:
`case that: Complex =>`
6. Om du implementerar `equals` i den klass som inför `canEqual`, börja uttrycket med:
`(that canEqual this) &&`
 och skapa därefter en fortsättning som baseras på innehållet i klassen, till exempel: `this.re == that.re && this.im == that.im`
 Om du överskuggar en *annan* `equals` än den standard-equals som finns i `AnyRef`, vill du förmögligen börja det logiska uttrycket med att anropa superklassens `equals`-metod: `super.equals(that)` && men du får fundera noga på vad likhet av underklasser egentligen ska innehåra i ditt speciella fall.
7. Den andra grenen i matchningen ska vara: `case _ => false`
8. Överskugga `hashCode`, till exempel genom att göra en tupel av innehållet i klassen och anropa metoden `##` på tupeln så får du i en bra hashcode:
`override def hashCode: Int = (re, im).##`

Uppgift 20. Bygg vidare på exemplet nedan och överskugga `equals` vid arv, genom att följa receptet i uppgift 19.

```
trait Number {
  override def equals(other: Any): Boolean = ???
}
class Complex(re: Double, im: Double) extends Number {
  override def equals(other: Any): Boolean = ???
}
class Rational(numerator: Int, denominator: Int) extends Number {
  override def equals(other: Any): Boolean = ???
}
```

Uppgift 21. Läs om olika speciella matchningar här:
www.artima.com/pins1ed/case-classes-and-pattern-matching.html

- a) Prova variabelbinding med @ i en matchning i REPL.

- b) Prova sekvensmönster med `_` och `_*` i en matching i REPL.

Uppgift 22. Läs mer om extraktorer här:

www.artima.com/pins1ed extractors.html

Skapa ditt eget extraktor-objekt för http-addresser som i t.ex.:

`http://my.host.domain/path/to/this`

extraherar `my.host.domain` och `path/to/this` med metoden `unapply` och testa i en matchning.

Uppgift 23. En rejäl utmaning: Implementera polynomdivision på lämpligt sätt genom att bygga vidare på **object polynomial** i uppgift 13 på sidan 197.

Läs mer om polynomdivision här: sv.wikipedia.org/wiki/Polynomdivision

9.2 Grupplaboration: chords - team

Mål

- Kunna använda mönstermatchning
- Känna till exceptions
- Förstå hur Try fungerar

Förberedelser

- Gör övning Repetera i kapitel 8.
- Läs om exceptions och felhantering.
- Bonus: ha tillgång till en dator där ni kan spela upp ljud.
- Diskutera i din samarbetsgrupp hur ni ska dela upp koden mellan er i flera olika delar, som ni kan arbeta med var för sig. En sådan del kan vara en klass, en trait, ett objekt, ett paket, eller en funktion.
- Varje del ska ha en *huvudansvarig* individ.
- Arbetsfördelningen ska vara någorlunda jämt fördelad mellan gruppmedlemmarna.
- När ni redovisar er lösning ska ni börja med att redogöra för handledaren hur ni delat upp koden och vem som är huvudansvarig för vad.
- Den som är huvudansvarig för en viss del redovisar den delen.
- Grupplaborationer görs i huvudsak som hemuppgift. Salstiden används primärt för redovisning.

9.2.1 Bakgrund

Inom musik utgår man från en skala med 12 olika toner (C, C#, D, D#, E, F, F#, G, G#, A, A#, B). Nästa ton efter B är C och tillhör nästa oktav. Ett ackord är uppbyggt av ett antal olika toner som spelas tillsammans. Laborationen kommer att utgå från två olika instrument: gitarr och ukulele. Skillnaden mellan dessa två instrument är antalet strängar och vilken tonart de är stämda i. Båda instrumenten har en greppbräda med ett antal olika band. Ett ackord spelas genom att man med ett finger trycker ner på strängen på band *i*. Om strängen spelas kommer tonen att vara ett halvt tonsteg högre än om man håller ner strängen på plats *i* – 1.

Laborationen kommer att bestå av ett textbaserat användargränssnitt där man kommer att ha möjlighet att bland annat lägga till nya ackord, rita upp ackord och spela ackord. En ton anges på följande format "E2", vilket innebär andra oktaven tonen E. Rekommenderad stämning för gitarr resp. ukulele är: E2, A2, D3, G3, B3, E4 resp. G4, C4, E4, A4. Denna stämning kommer att fungera för de fördefinierade ackorden i filen chords.txt.

Om ni är fler än fyra i gruppen behöver ni även göra extrauppgiften, men om ni ändå är tre behöver ni inte implementera play.

De givna kodfilerna är:⁶

- Chord.scala innehåller ett **trait** Chord som beskriver ett ackord med tillhörande funktionalitet som rör ackordhantering.
- ChordDraw.scala innehåller ett **object** ChordDraw med metoder för att rita ut ett ackord i SimpleWindow.
- ChordPlayer.scala innehåller ett **object** ChordPlayer med en metod **def apply(chord: Chord, time: Int)** som spelar upp ett ackord den angivna tiden.
- database.scala innehåller metoder för att lägga till, ta bort och hantera ackord i databasen.
- io.scala används för läsning/skrivning till/från fil.
- Main.scala Startar användargränssnittet och läser in kommandon från användaren.
- Notes.scala innehåller ett object med metoder som används för att konvertera en not till ett nummer och vice versa.
- SimpleNotePlayer.scala innehåller ett hjälpprojekt för uppspelning av noter med hjälp av inbyggd MIDI-spelare.
- textui.scala innehåller en modul som har hand om det textuella användargränssnittet och alla kommandon.

9.2.2 Obligatoriska uppgifter

Uppgift 1. Notes. Objektet ska kunna omvandla en tons namn (t.ex. "E2") till en heltalsrepresentation och tvärtom.

- a) Implementera först metoden `fromNbrToNote` med hjälp av %-operatorn
- b) Implementera metoden `unapply` som ska ta in en sträng som innehåller tonens namn och oktavnummer (t.ex. "C2"). Tänk på att en not kan vara på formen "C2" och "C#2". "E2" kommer översättas till 16 och "C1" till 0 och tänk på att hantera specialfallet då till exempel "C#1" ska ge värdet 1. Även felaktiga ackord ska hanteras, t.ex. om användaren anger två bokstäver eller inte anger oktavnummer. Använd attributet `toNumber` för att översätta en ton ("C#", "E") till ett nummer. Titta gärna på vad metoden `zipWithIndex.toMap` gör och använd den i `toNumber`.
- c) Använd `TestNotes` som ligger i paketet `test` för att se till så att `Notes` är rätt implementerat. Starta `TestNotes` och rätta till eventuella fel.

⁶Du kan bläddra bland klasserna i paketet Chords här:
https://github.com/lunduniversity/introprog/tree/master/workspace/w08_chords_team/src/main/scala/chord

Uppgift 2. Chord. Representation av de två olika ackorden. Lägg märke till hur stämning (eng. *tuning*) och ett grepp (eng. *grip*) representeras i Chord. –1 i ett grepp betyder att strängen inte ska spelas.

- Implementera `toString` i **trait** `Chord` så att den matchar utskriften i filen `chords.txt`. I klassen `Guitar` ska `toString` vara på formen `git:D:-1 -1 0 2 3 2`
- Implementera metoden `isIncludedBy`. Använd `.split(" ; ")` för att dela upp strängen `filter` i de olika filtersträngarna. För varje filtersträng ska det kollas om denna förekommer i strängen som returneras från `toString`. Exempel på filtersträng: `git:G;B;uku`". **Tänk på att metoden `forall` kollar om ett villkor gäller för alla möjligheter. I det här fallet räcker det att det gäller för ett av fallen**
- Implementera `isGit` och `isUku` som jämför början av strängen `s` och returnerar `true` om strängen matcher ett gitarrackord resp. ukuleleackord.
- Implementera klassen `Guitar`. `toString` ska vara på samma format som i filen `chords.txt`. `tuning` ska vara en strängrepresentation av gitarrrens stämning, alltså `E2 A2 D3 G3 B3 E4`.
- Skapa en ny klass `Ukulele` som har liknande beteende som `Guitar`.
- Implementera `instToChord` med hjälp av `matching` och låt basfallet (om ackordet är varken gitarrackord eller ukuleleackord) returnera `None`.

Uppgift 3. database. Kommer att hålla reda på alla ackord i programmet. Testa varje metod innan nästa blir implementerad. Det blir då lättare att upptäcka fel i koden. Metoderna kan testas genom att skriva ett testprogram eller efterhand som uppgift 4 implementeras.

- Börja med att implementera `add` och `allChords`.
- Implementera `find` och `updateFilter`. `find` ska returnera alla ackord som innehåller söksträngen, alltså inte bara första förekomsten. Metoden `find` i `Vector` returnerar enbart första förekomsten. `updateFilter` ska ändra filtersträngen i `database` till den angivna strängen.
- Implementera `filteredChords` och `sort` med hjälp av metoden `isIncludedBy` i `Chord`. Tom filtersträng innebär att inget filter appliceras. Vid `sort` kan man ta hjälp av metoden `sortBy` i `Vector`. När man listar alla ackord ska man skriva ut enbart de ackord som matchar det applicerade filtret. Listan ska vara numrerad så att man utgår från dessa nummer när man väljer ackord i listan.
- Implementera `delete`. **Tänk på att `delete` ska radera ackordet på plats *i* i `filteredChords` och inte i `allChords`**. För detta kan man använda filter för att filtrera ut det ackord på plats *i* i `filteredChords` så här: `db.filter(_ != filteredChords(i))`.

Uppgift 4. textui

- Provör programmet och prova några olika kommandon (t.ex. `add`, `del`, `help`). Använd metoderna i `database` för att implementera kommandona.

b) Titta på objektet Help. Använd liknande matching för att ta hand om alla fall för de olika kommandona.

- Add. Argumenten måste först bli en sträng med hjälp av `mkString(" ")` för att sedan delas upp vid ';'. Användaren kan skriva in felaktiga ackord, vilket måste hanteras. Metoden `fromString` i Chord retunerar ett `Option[Chord]`. Titta närmare på metoden `flatMap`.
- Lst. Använd matching för att ta hand om fallet när användaren anger argument till kommandot, vilket inte ska vara med. Använd metoden i `database`. Listan ska vara numrerad från 1.
- Del. Använd matching för att ta hand om felfallen inget argument och fler än ett argument. Ett tredje felfall är om användaren anger något annat än en siffra som argument. Använd Try och matching för att ta hand om felet.
- Filter. Använd metoderna i `database`. De filtrerade ackorden behöver inte skrivas ut efter filtrering, utan användaren behöver använda kommandot `list` för att lista de filtrerade ackorden.
- Find. Använd matching för att ta hand om felfallen inget argument eller fler än ett argument.
- Load. Använd matching för att ta hand om felfallen inget argument eller fler än ett argument. Använd metoden `load` i objektet `io` för att läsa in från den angivna filen. Metoden kommer att likna metoden `Add`.
- Save. Använd matching för att ta hand om felfallen inget argument eller fler än ett argument. Använd metoden `save` i objektet `io` för att skriva till den angivna filen. Om filen inte finns kommer den att skapas.
- Sort. Använd matching för att ta hand om fallet när användaren anger argument till kommandot, vilket inte behövs. Använd metoden i `database`.
- Quit. Använd matching för att ta hand om fallet när användaren anger argument till kommandot, vilket inte behövs. Skapa en hjälpmetod `quitPrompt` som returnerar en Boolean med värdet `true` om användaren anger 'y', `false` om användaren anger 'n' och som körs igen vi felaktigt svar (allt annat än 'y' och 'n'). Även 'Y' och 'N' ska vara ett acceptabelt svar.

Uppgift 5. ChordPlayer. Ska spela upp enstaka ackord chord med en viss längd time i millisekunder.

- a) Titta vilka parametrar metoden `play` i `SimpleNotePlayer` behöver. Heltalet note ska vara heltalsrepresentationen av en ton.
- b) För att omvandla ett ackord till toner behöver först stämningen, tuning, omvandlas till sin heltalsrepresentation och sen ska greppet, grip, adderas

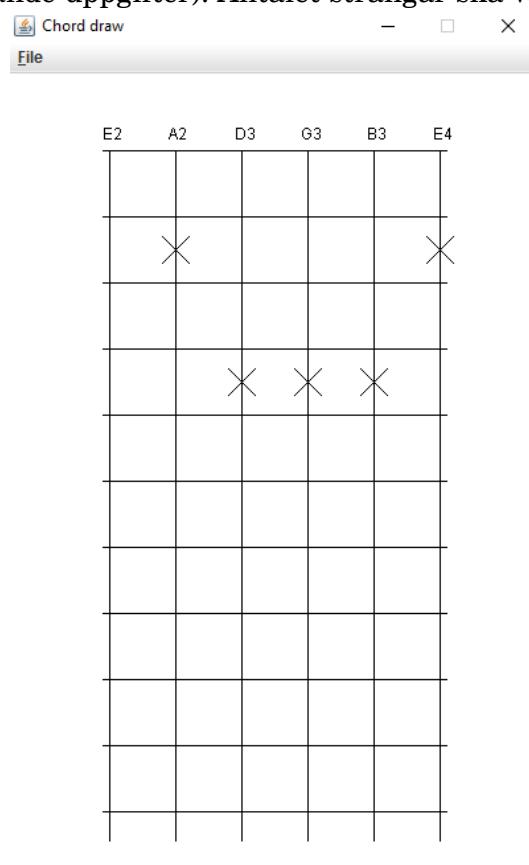
till stämningen för varje sträng. Varje strängs ton ska sedan spelas upp under den angivna tiden. Koden för att spela ackordet ska placeras ovanför den existerande koden. **Tänk på att strängar med greppet -1 inte ska spelas**

- c) Lägg till menyvalet play i textiu. Använd matching för att bryta ut de olika argumenten och för att ta hand om felfallet inga argument. Första argumentet är tempo i millisekunder och resten är siffror som motsvarar platsen för ackordet i den filtrerade listan. Varje sträng i argumentlistan ska omvandlas till Int och det är viktigt att ta hand om fallet då användaren anger något annat än en siffra. Använd Try och matching för att ta hand om felet. Använd ChordPlayer för att spela upp ett ackord. **Kom ihåg att lägga till kommandot i listan med kommandon i doCommand**

9.2.3 Extrauppgifter

Uppgift 6. ChordDraw

- a) Rita upp en greppbräda liknande bilden nedan (kryssen läggs till i kommande uppgifter). Antalet strängar ska variera beroende på instrument.



- b) Skapa en hjälpmetod cross som tar in två heltal x och y . Metoden ska rita upp ett kryss som är 20x20 pixlar och har sitt centrum i den angivna koordinaten.
- c) Rita ut ett kryss där en sträng trycks ner. **Tänk på att -1 och 0 anger att en sträng inte trycks ner.**

- d) Implementera metoden `play` som börjar med att vänta på ett event från `SimpleWindow`, sedan kollar om eventet är av typen `SimpleWindow.MOUSE_EVENT`. Sedan ska man kolla om användaren tryckte på någon sträng (ett intervall på -10 till +10 i förhållande till strängens x-koordinat kan anses vara på strängen). Om användaren tryckt på en sträng ska denna spelas med hjälp av `SimpleNotePlayer`. Metoden `play` ska köras tills användaren kryssar ner fönstret, vilket motsvarar `SimpleWindow.CLOSE_EVENT`.
- e) Lägg till menyvalet `draw` i `textui`. Använd `matching` för att ta hand om felfallen inget argument eller fler än ett argument. Argumentet motsvarar ackordets plats i den filtrerade litan. Använd `Try` och `matching` för att ta hand om felet att användaren anger något annat än en siffra. Använd `ChordDraw` för att rita upp ackord. **Kom ihåg att lägga till kommandot i listan med kommandon i `doCommand`**

Kapitel 10

Matriser, typparametrar

Begrepp som ingår i denna veckas studier:

- matris
- nästlad samling
- nästlad for-sats
- typparameter
- generisk funktion
- generisk klass
- fri vs bunden typparameter
- matriser i Java vs Scala
- allokkering av nästlade arrayer i Scala och Java

10.1 Övning: matrices

Mål

- Kunna skapa och använda matriser med nästlade strukturer av Vector.
- Kunna iterera över elementen i en matris med nästlade **for**-satser och **for-yield**-uttryck, samt nästlad applicering av map respektive foreach.
- Kunna skapa och använda funktioner som tar matriser som parametrar.
- Känna till generiska funktioner.
- Känna till generiska klasser.
- Kunna skapa och använda matriser med hjälp inbyggda arrayer i Java.
- Kunna använda nästlade **for**-satser i Java för att iterera över elementen i en matris.

Förberedelser

- Studera begreppen i kapitel 10.

10.1.1 Grunduppgifter

Uppgift 1. Skapa matriser med hjälp av nästlade samlingar. Man kan i ett datorprogram, med hjälp av samlingar som innehåller samlingar, skapa nästlade strukturer som kan indexeras i två dimensioner och på så sätt representera en matematisk **matris**.¹

Bakgrund för kännedom: En **matris** inom matematiken innehåller ett antal rader och kolumner (även kallade kolonner). I en matematisk matris har alla rader lika många element och även alla kolumner har lika många element. En matris av dimension $m \times n$ har $m \cdot n$ stycken element, där m är antalet rader och n är antalet kolumner. En matris $A_{m,n}$ av dimension $m \times n$ ritas ofta så här:

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

Exempel: En heltalsmatris $M_{2,5}$ av dimension 2×5 där element $m_{2,5} = 7$:

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

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

```

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

```

¹sv.wikipedia.org/wiki/Matris

- b) Vad ger uttrycken på raderna 2, 3 och 4 ovan för värden och typ?
- c) Man kan i ett datorprogram mycket väl skapa tvådimensionella, nästlade strukturer där raderna *inte* innehåller samma antal element. Det blir då ingen äkta matris i strikt matematisk mening, men man kallar ofta ändå en sådan struktur för en "matris". Vilken typ har variablerna `m2`, `m3`, `m4` och `m5` nedan?

```

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

- d) Rita minnessituationen efter tilldelingen av `m2` på rad 1 ovan.
- e) Vilken av variablerna `m2`, `m3`, `m4` och `m5` ovan representerar en äkta matris i matematisk mening? Vilken är dess dimensioner?

Uppgift 2. *Skapa och itererar över matriser.* Vi ska skapa matriser där varje rad representerar 5 kast med en tärning spelet Yatzy.²

- a) Definiera i REPL en funktion `def throwDie: Int = ???` som returnerar ett slumptal mellan 1 och 6.
- b) Skapa nedan heltalsmatris i REPL. Vilken dimension får matrisen?

```

1  val ds1 = for (i <- 1 to 1000) yield {
2      for (j <- 1 to 5) yield throwDie
3  }
```

- c) Man kan också använda nedan varianter för att skapa en heltalsmatris. Vilken av variантerna `ds1` ... `ds5` tycker du är lättast att läsa och förstå? Prova respektive variant i REPL och ange vilken typ på `ds1` ... `ds5` som härleddes av kompilatorn.

```

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

- d) Definiera en funktion
`def roll(n: Int): Vector[Int] = ???`
som ger en heltalsvektor med n stycken slumpvisa tärningskast. Kasten ska vara sorterade i växande ordning; använd för detta ändamål samlingsmetoden `sorted`.
- e) Definiera i REPL en funktion `isYatzy(xs: Vector[Int]): Boolean = ???` som testar om alla elementen i en heltalsvektor är samma. Använd samlingsmetoden `forall`.
- f) Implementera `isYatzy` igen med ett imperativt angreppssätt som använder en `while`-sats (alltså utan att använda funktionella `forall`). Ta hjälp av en variabel `i` som håller reda på index och en variabel `foundDiff` som håller reda

²sv.wikipedia.org/wiki/Yatzy

på om ett avvikande värde upptäcks. Funktionen blir ca 10 rader, så det kan vara lämpligt att öppna en editor att skriva i medan du klurar ut lösningen. Börja med att skriva pseudokod, gärna med penna på papper. Prova genom att klistra in i REPL.

g) Skapa en funktion

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

h) Skapa en funktion som returnerar en utskriftsvänlig sträng

def diceMatrixToString(xss: Vector[Vector[Int]]): String = ???
 med hjälp av `map` och `mkString`, som fungerar enligt nedan.

```
1  scala> println(diceMatrixToString(diceMatrix(10, 5)))
2  4 5 5 3 3
3  1 4 1 3 1
4  1 3 1 5 5
5  6 4 4 5 5
6  2 1 5 6 5
7  1 2 2 3 6
8  1 3 2 4 5
9  2 2 3 2 2
10 2 6 3 4 6
11 4 5 5 2 3
```

i) Ett imperativt sätt³ att göra detta på visas nedan. Förklara hur nedan kod fungerar. Vad händer om `xss` är tom? Vad händer om `xss` bara innehåller tomma vektorer? Nämn en fördel och en nackdel med att använda `val sb: StringBuilder` och `append`, jämfört med en vanlig `var s: String` och `+ tillägg` i slutet.

```
def diceMatrixToString(xss: Vector[Vector[Int]]): String = {
  val sb = new StringBuilder()
  for(m <- 0 until xss.size) {
    for(n <- 0 until xss(m).size) {
      sb.append(xss(m)(n))
      if (n < xss(m).size - 1) sb.append(" ")
      else if (m < xss.size - 1) sb.append("\n")
    }
  }
  sb.toString
}
```

j) Implementera funktionen

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

³Imperativa anreppssätt är nödvändiga att kunna när du stöter på samlingar och/eller språk som saknar funktionsprogrammeringsmöjligheter med `map`, `mkString` etc.

```

1  scala> println(diceMatrixToString(filterYatzy(diceMatrix(10000, 5))))
2  2 2 2 2 2
3  3 3 3 3 3
4  1 1 1 1 1
5  3 3 3 3 3
6  4 4 4 4 4
7  6 6 6 6 6
8  2 2 2 2 2
9  3 3 3 3 3
10 2 2 2 2 2
11 6 6 6 6 6
12 4 4 4 4 4
13 2 2 2 2 2
14 4 4 4 4 4

```

- k) Gör som träning en imperativ implementation av `filterYatzy` med en **for**-sats (alltså utan att använda `filter`, och utan att använda `yield`).
-  l) Tycker du din imperativa lösning är lättare eller svårare att läsa och förstå jämfört nedan funktionella lösning med ett **for**-uttryck och `yield`?

```

def filterYatzy(xss: Vector[Vector[Int]]): Vector[Vector[Int]] = {
  for (i <- 0 until xss.size if isYatzy(xss(i))) yield xss(i)
}.toVector

```

- m) Implementera funktionen

`def yatzyPips(xss: Vector[Vector[Int]]): Vector[Int]`
 som ger en vektor med tärningsvärdena för de kast i matrisen `xss` som gav yatzy enligt nedan. Använd din funktion `isYatzy` och samlingsmetoden `filter`.

```

1  scala> yatzyPips(diceMatrix(10000, 5))
2  res42: Vector[Int] = Vector(3, 5, 6, 6, 3, 3, 2, 6, 1, 3)

```

Uppgift 3. *Strängtabell med rubrikrad.* Denna övning utgör en början på det du ska göra under veckans laboration.

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

Tips:

- Värdet `index0fHeading` kan skapas med hjälp av metoden `zipWithIndex` som fungerar på alla sekvenssamlingar, samt metoden `toMap` som fungerar på sekvenser av 2-tupler. Undersök först hur metoderna fungerar i REPL och sök upp deras dokumentation.
- Skapa en indatafil som du kan använda för att testa att `Table` fungerar.

Specification Table

```

case class Table(
  data: Vector[Vector[String]],

```

```

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

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

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

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

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

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

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

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

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

```

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

Uppgift 4. Generiska funktioner. En generisk funktion har (minst) en typparameter inom klammerparenteser efter namnet, till exempel `[T]`. Denna typ förekommer sedan som typ på (någon av) parametrarna i parameterlistan. Kompilatorn härleder en konkret typ vid kompileringstid och ersätter typparametern med denna konkreta typ. På så sätt kan en funktion fungera för många olika typer.

- a) Förklara för varje rad nedan vad som händer.

```

1 scala> def tnirp[T](x: T): Unit = println(x.toString.reverse)
2 scala> tnirp(42)
3 scala> tnirp("hej")
4 scala> case class Gurka(vikt: Int)
5 scala> tnirp(Gurka(42))
6 scala> tnirp[String](42)
7 scala> tnirp[Double](42)

```

- b) Man kan kombinera generiska funktioner med funktioner som tar funktioner som parametrar. Det är så `map` och `foreach` är implementerade. Förklara

för varje rad nedan vad som händer.

```

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

```

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

Uppgift 5. *Generiska klasser.* Även klasser kan vara generiska. En generisk klass har (minst) en typparameter inom klammerparenteser efter klassens namn.

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

```

1  scala> class Cell[T](var value: T){
2      override def toString = "Cell(" + value + ")"
3  }
4  scala> new Cell(42)
5  scala> new Cell("hej")
6  scala> new Cell(new Cell(math.Pi))
7  scala> new Cell[String](42)
8  scala> new Cell[Double](42)

```

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

```

1  scala> val a = new Cell("hej")
2  scala> val b = new Cell(42)
3  scala> a concat b

```

- ☞ c) Vilken sorts celler kan du konkatenera om du tar bort typparameternamnet `U` i `concat` samtidigt som du använder `Cell[T]` som typ på värdeparametern `that`? Vad ger det för konsekvenser för celler av annan typ än `Cell[String]`?
- ☞ d) Denna uppgift illustrerar grunderna för att hur generiska samlingar är konstruerade, men vi går inte djupare här (det kommer mer i fördjupningskursen). Fundera om du vill på hur en generisk Matris-klass skulle kunna se ut och om du är intresserad av att fördjupa dig så gör fördjupningsuppgift 8.

Uppgift 6. *Matriser med array i Java.* Om man redan vid allokeringsvet hur många element en matris ska ha, använder man i Java gärna en array av arrayer. En heltalsmatris (en array av array av heltal) skrivs i Java med dubbla hakparentespar `int[][]` direkt efter typen. Vid allokeringsanvändning använder man nyckelordet `new` och antalet element i respektive dimension anges inom hakparenteserna; t.ex. så ger `new int[42][21]` en matris med 42 rader och 21 kolumner,

vilket motsvarar att man i Scala skriver⁴ `Array.ofDim[Int](42, 21)`. Alla element får defaultvärdet för typen, som är 0 för typen Int i Scala, motsvarande `int` i Java.

- a) Skriv nedan program i en editor och spara koden i filen `ArrayMatrix.java` och kompilera med `javac ArrayMatrix.java` och kör i terminalen med `java ArrayMatrix` och undersök utskriften. Förklara vad som händer. Notera några skillnader i hur matriser används i Scala och Java.

```
// ArrayMatrix.java

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 main(String[] args) {
        System.out.println("ArrayMatrix test");
        int[][] XSS = new int[10][5];
        showMatrix(XSS);
    }
}
```

- b) Implementera nedan metod `fillRnd` inuti klassen `ArrayMatrix`. Skriv kod som fyller matrisen `m` med slumptal mellan 1 och n.

```
public static void fillRnd(int[][] m, int n){
    /* ??? */
}
```

Tips: med detta uttryck skapas ett slumptal mellan 1 och 42 i Java:

`(int) (Math.random() * 42 + 1)`

där typkonverteringen `(int)` ger samma effekt som ett anrop av metoden `toInt` i Scala; alltså att dubbelprecisionssflyttal omvandlas till heltal genom avkortning av alla eventuella decimaler.

Ändra huvudprogrammet till:

⁴Ett annat längre, men kanske tydligare, sätt att skriva detta i Scala där initialvärdet framgår explicit: `Array.fill(42)(Array.fill(21)(0))`

```
public static void main(String[] args) {  
    System.out.println("ArrayMatrix test");  
    int[][] xss = new int[10][5];  
    showMatrix(xss);  
    fillRnd(xss, 6);  
    showMatrix(xss);  
}
```

Programmet ska ge en utskrift som liknar följande:

```
1 $ javac ArrayMatrix.java  
2 $ java ArrayMatrix  
3 ArrayMatrix test  
4  
5 --- showMatrix ---  
6 [0][0] = 0; [0][1] = 0; [0][2] = 0; [0][3] = 0; [0][4] = 0;  
7 [1][0] = 0; [1][1] = 0; [1][2] = 0; [1][3] = 0; [1][4] = 0;  
8 [2][0] = 0; [2][1] = 0; [2][2] = 0; [2][3] = 0; [2][4] = 0;  
9 [3][0] = 0; [3][1] = 0; [3][2] = 0; [3][3] = 0; [3][4] = 0;  
10 [4][0] = 0; [4][1] = 0; [4][2] = 0; [4][3] = 0; [4][4] = 0;  
11 [5][0] = 0; [5][1] = 0; [5][2] = 0; [5][3] = 0; [5][4] = 0;  
12 [6][0] = 0; [6][1] = 0; [6][2] = 0; [6][3] = 0; [6][4] = 0;  
13 [7][0] = 0; [7][1] = 0; [7][2] = 0; [7][3] = 0; [7][4] = 0;  
14 [8][0] = 0; [8][1] = 0; [8][2] = 0; [8][3] = 0; [8][4] = 0;  
15 [9][0] = 0; [9][1] = 0; [9][2] = 0; [9][3] = 0; [9][4] = 0;  
16  
17 --- showMatrix ---  
18 [0][0] = 6; [0][1] = 2; [0][2] = 6; [0][3] = 3; [0][4] = 5;  
19 [1][0] = 2; [1][1] = 4; [1][2] = 6; [1][3] = 1; [1][4] = 1;  
20 [2][0] = 5; [2][1] = 4; [2][2] = 4; [2][3] = 1; [2][4] = 5;  
21 [3][0] = 4; [3][1] = 6; [3][2] = 6; [3][3] = 1; [3][4] = 3;  
22 [4][0] = 4; [4][1] = 6; [4][2] = 2; [4][3] = 3; [4][4] = 2;  
23 [5][0] = 2; [5][1] = 4; [5][2] = 5; [5][3] = 5; [5][4] = 3;  
24 [6][0] = 6; [6][1] = 5; [6][2] = 2; [6][3] = 4; [6][4] = 3;  
25 [7][0] = 1; [7][1] = 6; [7][2] = 1; [7][3] = 6; [7][4] = 2;  
26 [8][0] = 1; [8][1] = 1; [8][2] = 5; [8][3] = 3; [8][4] = 2;  
27 [9][0] = 1; [9][1] = 1; [9][2] = 1; [9][3] = 5; [9][4] = 4;
```

10.1.2 Extrauppgifter

Uppgift 7. Skapa ett yatzy-spel för användning i terminalen.

- a) Skapa med en editor en klass enligt nedan specifikation. Läs om hur de olika predikaten för att kolla olika giltiga kombinationer i Yatzy ska fungera här: en.wikipedia.org/wiki/Yahtzee. Bygg ett huvudprogram som testar dina funktioner. Kompilera och testa i terminalen allteftersom du lägger till nya funktioner.

Specification YatzyRows

```
/** En skiss på en klass som kan användas till ett förenklat yatzy-spel */
case class YatzyRows(val rows: Vector[Vector[Int]]) {
    /** A new YatzyRows with a new row of 5 dice rolls appended to rows */
    def roll: YatzyRows = ???

    /** A new YatzyRows with some indices of the last row re-rolled */
    def reroll(indices: Vector[Int]): YatzyRows = ???
}

object YatzyRows {
    def isYatzy(xs: Vector[Int]): Boolean = ???
    def isThreeOfAKind(xs: Vector[Int]): Boolean = ???
    def isFourOfAKind(xs: Vector[Int]): Boolean = ???
    def isFullHouse(xs: Vector[Int]): Boolean = ???
    def isSmallStraight(xs: Vector[Int]): Boolean = ???
    def isLargeStraight(xs: Vector[Int]): Boolean = ???
}
```

- b) Använd YatzyRows för att med hjälp av många tärningskast beräkna sannolikheter för några olika giltiga kombinationer. Använd, om du vill, möjligheten som reglerna ger att slå om tärningar i två ytterliggare kast, där de tärningar som slås om väljs slumpmässigt.
- c) Bygg ett förenklat yatzy-spel i terminalen där användaren kan bestämma vilka tärningar som ska slås om. Använd Scanner för att läsa indata från användaren. Börja med något riktigt enkelt och bygg sedan vidare på ditt spel genom att införa fler och fler funktioner.

10.1.3 Fördjupningsuppgifter

Uppgift 8. Skapa en generisk, oföränderlig matrisklass. Med hjälp av en typparameter kan vi skapa en matrisklass som kan innehålla vilka element som helst. Implementera nedan specifikation. Testa din matrisklass i REPL för olika typer av element.

```
Specification Matrix[T]

case class Matrix[T](data: Vector[Vector[T]]){

    def map[U](f: T => U): Matrix[U] = Matrix(data.map(_.map(f)))

    def foreachRowCol[U](f: (Int, Int, T) => Unit): Unit =
        for (r <- 0 until data.size) {
            for (c <- 0 until data(r).size) {
                f(r, c, data(r)(c))
            }
        }

    /** The element at row r and column c */
    def apply(r: Int, c: Int): T = ???

    /** Gives Some[T](element) at row r and column c
     *  if r and c are within index bounds, else None */
    def get(r: Int, c: Int): Option[T] = ???

    /** The row vector of row r */
    def row(r: Int): Vector[T] = ???

    /** The column vector of column c */
    def col(c: Int): Vector[T] = ???

    /** A new Matrix with element at row r and col c updated */
    def updated(r: Int, c: Int, value: T): Matrix[T] = ???
}

object Matrix {
    def fill[T](rowSize: Int, colSize: Int)(init: T): Matrix[T] =
        new Matrix(Vector.fill(rowSize)(Vector.fill(colSize)(init)))
}
```

Uppgift 9. Använd matrisklassen från uppgift 8 för att göra en SpriteEditor med JColorChoser enligt nedan skiss.

```
object ColorChooser {
    import java.awt.Color
    import javax.swing.JColorChooser

    var title = "Pick Color"
    private val chooser = new JColorChooser(Color.BLACK)
    private val dialog = JColorChooser.
        createDialog(null, title, true, jcs, null, null)
```

```

def getColor(initColor: Color = Color.BLACK): Color = {
    chooser.setColor(initColor)
    dialog.setVisible(true)
    chooser.getColor
}
}

class Sprite(// en bild med många lager av pixlar i olika färger
    val id: String,
    val size: (Int, Int),
    val pixels: Matrix[Option[Int]],//Some(color),None=genomskinlig
    var scale: Int, //uppskalning av storlek i pixlar
    var colors: Vector[java.awt.Color], //tillgängliga färger
    var pos: (Int, Int, Int) // (row, col, layer)
){
    def row = pos._1
    def col = pos._2
    def layer = pos._3
}

class SpriteEditor(
    rows: Int = 64, cols: Int = 64,
    scale: Int = 16, nColors: Int = 16) {
    private val w = new SimpleWindow(?)
    def edit: Unit = ???
}

```

Uppgift 10. Klasser för tät och glesa matematiska matriser med flyttal. Läs om matrisräkning här: sv.wikipedia.org/wiki/Matris

- Skapa en oföränderlig, final klass `DenseMatrix` för matematiska matriser med dubbelprecisionssflyttal. `DenseMatrix` ska internt lagra elementen i en privat *endimensionell* array av flyttal av typen `Array[Double]`. Klassen ska inte vara en case-klass. Det ska gå att skapa matriser med uttrycket `DenseMatrix.ofDim(3,7)(1.0,42,3.2,1.0,2.2,3)` tack vare ett kompanjonsobjekt med lämplig fabriksmetod som anropar den privata konstruktorn. Om antalet element är för litet i förhållande till den angivna dimensionen så fyll på med nollor.
- Överskugga metoderna `equals` och `hashCode` och ge `DenseMatrix` innehållslikhet i stället för referenslikhet.
- Implementera egna innehållslikhetsmetoder med namnet `==` på `DenseMatrix` som är typsäker, d.v.s. bara tillåter innehållsjämförelse mellan tät matriser.
- Läs om glesa matriser här: https://sv.wikipedia.org/wiki/Gles_matris och implementera `SparseMatrix` med ett privat attribut av typen

- `mutable.Map[(Int, Int), Double]` som bara lagrar index som inte är noll.
- e) Skapa ett **trait** `Matrix` som både `DenseMatrix` och `SparseMatrix` ärver, med lämpliga abstrakta och konkreta medlemmar. Implementera addition, subtraktion och multiplikation av täta och glesa matriser.

10.2 Laboration: maze

Mål

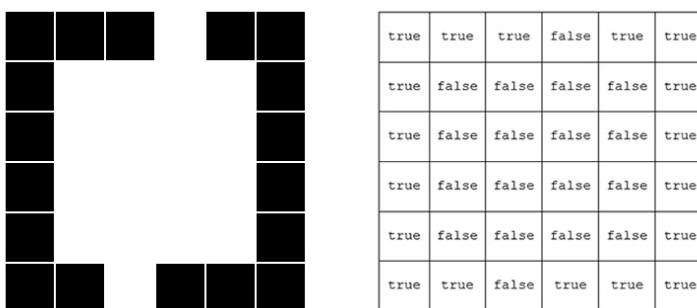
- Kunna skapa och använda matriser.
- Kunna iterera över matriser med nästlade for-loopar.
- Kunna använda sig av och förstå arv.
- Förstå och träna på olika villkor med if-satser.
- Känna till algoritmer för att lösa problem så som att ta sig igenom en labyrinth eller slumpmässigt skapa en labyrinth.

Förberedelser

- Gör veckans övningsuppgifter.
- Läs om matriser i Scala-boken på sida (??).
- Läs om arv i Scala-boken på sida (??).
- Läs om olika algoritmer för att ta sig igenom en labyrinth: https://en.wikipedia.org/wiki/Maze_solving_algorithm
- (Frivillig) Läs om olika algoritmer för att skapa en slumpmässig labyrinth: https://en.wikipedia.org/wiki/Maze_generation_algorithm.

10.2.1 Bakgrund

I denna laboration ska du rita labyrinter och sedan implementera en algoritm för att ta dig igenom dessa. En labyrinth är ett rum som har en ingång och en utgång. Ingången är i denna laboration alltid längst ner i labyrinthen, och utgången alltid högst upp. Alla väggar är också parallella med antingen x-axeln eller y-axeln. Men kan beskriva en sådan labyrinth i kod med hjälp av en matris av booleiska element. Varje element i matrisen motsvarar då en ruta i labyrinthen. Värdet `true` i matrisen representerar att det finns en vägg på motsvarande ruta i labyrinthen och `false` representerar att där istället är en väg att gå på.



Figur 10.1: Exempel på en matris-representation av en labyrinth.

Det finns många olika sätt att ta sig igenom en labyrinth. Ett tänkbart sätt att hitta utgången, som kanske inte följer närmsta vägen, är att hålla vänster

hand i vänster vägg och följa väggen med handen tills man når öppningen. Detta fungerar för alla labyrinter där väggarna från ingången till utgången är sammankopplade.

När man går igenom labyrinten finns det fyra olika riktningar att välja mellan som alla kan beskrivas i antal grader räknat från x-axeln, där höger motsvaras av 0 grader, uppåt motsvaras av 90 grader, vänster av 180 grader och nedåt av 270 grader.

I denna laboration ska du använda en färdig klass `Maze` som representerar en labyrint med hjälp av en booleisk matris implementerad som en `Vector[Vector[Boolean]]`. Klassen har följande specifikation:

<i>Specification Maze</i>
<pre>/** * A class representing a maze. */ case class Maze(data: <code>Vector[Vector[Boolean]]</code>) { /** * Returns a corresponding char from a boolean. * @param b The boolean which to convert to a char */ def boolToChar(b: Boolean): Char /** * Returns a String representation of the maze. */ override def toString: Unit /** * Checks if the coordinates x, y is inside the maze and if * so returns true, otherwise false. * @param x The x coordinate * @param y The y coordinate */ private def insideMaze(x: Int, y: Int): Boolean /** * Returns the x coordinate of the entry of the maze. */ def getXEntry(): Int /** * Returns the y coordinate of the entry of the maze. */ def getYEntry(): Int /** * Checks if there is a wall left of the coordinates x, y at * given direction and if so returns true, otherwise false. * @param direction The direction of the turtle * @param x The x coordinate * @param y The y coordinate */ def wallAtLeft(direction: Int, x: Int, y: Int): Boolean</pre>

```

    /**
     * Checks if there is a wall in front of the coordinates x, y at
     * given direction and if so returns true, otherwise false.
     * @param direction The direction of the turtle
     * @param x          The x coordinate
     * @param y          The y coordinate
    */
    def wallInFront(direction: Int, x: Int, y: Int): Boolean

    /**
     * Checks if the coordinates x, y is at the exit of the maze.
     * @param x          The x coordinate
     * @param y          The y coordinate
    */
    def atExit(x: Int, y: Int): Boolean

    /**
     * Goes through the the maze and for every spot that is a wall
     * draws a brick of size blockSize in SimpleWindow.
     * @param w      The window in which to draw the maze
    */
    def draw(w: SimpleWindow): Unit = ???
}

/**
 * An object representing a maze.
 */
object Maze {

    /**
     * Returns a Maze from a vector of Strings.
     * @param xs The vector of Strings that represent the maze
    */
    def fromStrings(xs: Vector[String]): Maze

    /**
     * Returns a Maze from a specified file.
     * @param fileName The name of the file that represent the maze
    */
    def fromFile(fileName: String): Maze

    /**
     * Returns a Maze from a sequence of Strings.
     * @param rows The sequence of Strings that represent the maze
    */
    def apply(rows: String*): Maze

    /**
     * Creates and returns a random maze.
     * @param rows   The number of rows for the maze
     * @param cols   The number of columns for the maze
    */
    def random(rows: Int, cols: Int): Maze = ???
}

```

Den frivilliga uppgiften

I den frivilliga uppgiften ska en slumpmässig labyrint genereras. Det finns flera olika algoritmer för att göra detta. Vi kommer här använda en slumpmässig variant av den s.k. *Prims algoritm*. Du kan läsa mer om Prims algoritm här: en.wikipedia.org/wiki/Maze_generation_algorithm (De representerar en labyrint på ett annat sätt än vi gör här, vilket innebär att algoritmen blir lite annorlunda).

10.2.2 Obligatoriska uppgifter

Uppgift 1. I den här uppgiften ska du implementera en metod som ritar upp en labyrint i SimpleWindow. Studera kodfilerna som ges inför denna laboration här: github.com/lunduniversity/introprog/tree/master/workspace/w09_maze/src/main/scala/maze

Läs igenom case-klassen Maze och dess kompanjonsobjekt och se till att du förstår det mesta, annars fråga. Objektet Maze tar in rader med strängar, antingen direkt som argument eller genom att läsa in från en fil. Utifrån detta skapar den sedan en booleisk matris som representation av labyrinten, där tecknet '#' i en sträng representerar en bit av en vägg medan blanksteg representerar en bit av en gång.

- Du ska nu implementera metoden draw i klassen Maze ska rita upp labyrinten i SimpleWindow. För att göra detta behöver du gå igenom matrisen data och undersöka elementen på varje plats. Om elementet på en viss plats är true ska det ritas upp en bit av en vägg på motsvarande plats i SimpleWindow. Detta kan du göra med hjälp av den färdigskrivna metoden brickInTheWall. Om elementet på en viss plats är false ska ingenting ritas upp.

Uppgift 2. I den här uppgiften ska du skapa en klass med en main-metod som anropar din metod för att rita upp en labyrint.

- Skapa en ny klass AMazeIngRace. I denna klass ska du skriva en main-metod där du skapar ett objekt av Maze genom att läsa in från en fil (börja exempelvis med filen maze1.txt som ligger i w09_maze/src/main/resources/). Du måste även skapa ett objekt av SimpleWindow som ska skickas med när du anropar metoden draw. Anropa metoden draw på Maze-objektet och kontrollera att labyrinten ritas upp som den ska. Gör samma sak för resterande av filerna maze2.txt – maze4.txt; alla ska kunna ritas upp korrekt.
- Testa att rita en egen labyrint genom att skapa en textfil maze5.txt och lägg i samma mapp som övriga maze-filer. Kontrollera så att även denna labyrint ritas upp som den ska, och fixa annars till metoden draw så att den fungerar som tänkt. Din labyrint inte alls behöver vara avancerad; det är inte meningen att den här uppgiften ska ta lång tid.

Uppgift 3. I den här uppgiften ska du implementera en algoritm för att få en sköldpadda att ta sig genom en labyrint genom att alltid hålla i väggen med vänster hand (eller kanske fot i det här fallet). För detta ska du skapa en ny

klass MazeTurtle som ska ha följande specifikation:

Specification MazeTurtle

```
/**
 * A Turtle that can walk through a maze with a specified color.
 * @param window    The window the turtle should be placed in.
 * @param position  A Point representing the turtle's starting coordinates.
 * @param angle     The angle between the turtle direction and the X-axis
 *                  measured in degrees.
 * @param isPenDown A boolean representing the turtle's pen position.
 *                  True if the pen is down.
 * @param color     The color with which the turtle will walk.
 * @param maze      The maze in which the turtle will walk.
 */
class MazeTurtle(
    window: SimpleWindow,
    private var position: Point,
    private var angle: Double,
    private var isPenDown: Boolean,
    color: Color,
    maze: Maze
) extends ColorTurtle(window, position, angle, isPenDown, color) {
    /**
     * Lets the turtle walk through the maze from entry to exit
     * by following the wall to left side of the turtle.
     */
    def walk(): Unit = ???
```

- Skapa en ny klass MazeTurtle som ärver från klassen ColorTurtle. MazeTurtle ska ta in ett extra argument, nämligen ett av typen Maze som representerar den labryrint som sköldpaddan ska gå i. Observera att du här behöver importera klasserna Turtle och ColorTurtle från en tidigare laboration. Detta gör du genom att högerklicka på projektet, välj Build Path -> Configure Build Path, välj fliken Projects, klicka på Add... och markera rätt laboration och klicka slutligen på OK.
- Lägg till och implementera metoden walk i MazeTurtle. I metoden ska sköldpaddan, med hjälp av tekniken att alltid hålla vänster extremitet i väggen, ta sig genom labryrinten Maze, från början till slut. Varje steg motsvarar att flytta sig från en ruta till en annan i Boolean-matrisen i Maze. Sköldpaddan kommer alltså ta sig fram i labryrinten genom att undersöka för varje steg om den borde svänga vänster, gå rakt fram eller svänga höger, beroende på hur den står i förhållande till vänster vägg. För att avgöra hur sköldpaddan ska gå kan den använda sig av metoderna wallInFront och wallAtLeft som finns i Maze.
- Lägg till kod i AMazeIngRace som skapar en sköldpadda av typ MazeTurtle och sedan låter denna gå igenom en labrynt med metoden walk. Testa att din MazeTurtle fungerar som den ska. Sköldpaddan ska klara att ta sig igenom alla labrynter i filerna maze1.txt-maze4.txt samt labrynten som du skapat själv.

10.2.3 Frivillig extrauppgift

Prims algoritm används för att koppla ihop alla punkter i en graf med så få antal kopplingar som möjligt. Tänk dig att varje punkt i en graf representerar en ruta i en labyrint och att en koppling motsvarar ställen där du kan gå i labyringen, det vill säga två öppna rutor (rutor där det inte finns någon vägg och som du därmed kan gå på) intill varandra.

I algoritmen väljs en slumpmässig ruta intill en redan öppen ruta och sedan undersöks om det finns någon annan öppen ruta intill. Om så är fallet låter man rutan förbli en vägg, annars öppnas den rutan (det vill säga rutan görs om från en vägg till en gång). Detta upprepas tills dess att alla rutor har undersökts.

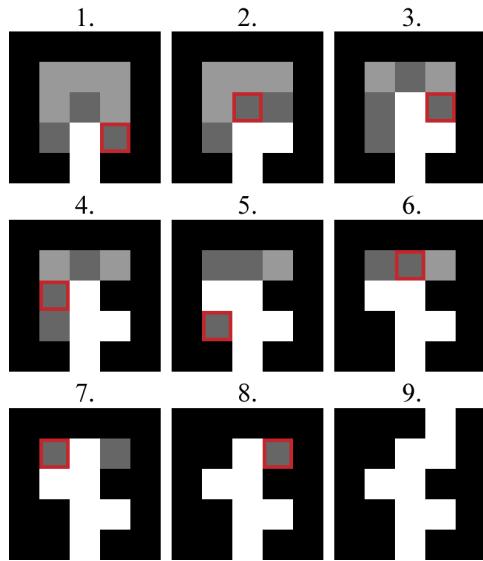
Eftersom det i vårt fall endast ska finnas en enda ingång och en enda utgång i labyringen, måste detta hanteras speciellt.

Nedan följer en grov beskrivning av algoritmen. Pseudokod med mer detaljer återfinns i figur 10.2.

1. Börja med att slumpmässigt välja en av rutorna i nedersta raden och öppna den rutan samt rutan ovanför. Detta motsvarar ingången till labyringen.
2. Applicera väggöppningsalgoritmen som visualiseras i figur 10.3 på sida 228. Observera att matrisens ytterkantväggar inte ska öppnas (förutom ingången och utgången).
3. Slutligen letar vi efter en slumpmässig ruta på näst översta raden som är öppen och öppnar rutan ovanför denna, vilket kommer bli utgången för labyringen.

```
def random(rows, cols): Maze = {
    labyrinth = matris(rows, cols) med inga öppna rutor
    buffer = lista(koordinater till rutor)
    Välj en slumpmässig ruta i nedersta raden och öppna den.
    Öppna rutan ovanför.
    Lägg till rutorna runt om till buffer.
    while (buffer inte är tom)
        element = slumpmässigt element i buffer
        if (element har tre väggar intill sig)
            Öppna rutan för de koordinater som element innehåller.
            Lägg till rutorna runt om element till buffer.
        Ta bort element från buffer.
        Hitta en ruta som är öppen i näst översta raden.
        Öppna rutan ovanför.
    Returnera: labyrinth
}
```

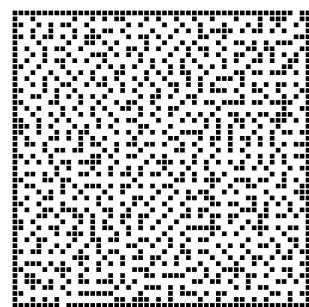
Figur 10.2: Pseudokod för generering av slumpmässig labyrint.



Figur 10.3: Visualisering av algoritmen när den genererar en labyrint av storleken 5x5.

Uppgift 4. I den här uppgiften ska du göra färdigt den påbörjade algoritmen som skapar en slumpmässig labyrint.

- Inspektera ovanstående pseudokod och försök förstå den. Fråga om något är oklart. Läs även de färdigskrivna metoderna `addWallToList` och `threeWallsAround` och studera den halvfärdiga metoden `random` i `Maze` i ditt workspace och se om du kan förstå vad de gör.
- Implementera metoden `random` i `Maze` som skapar och returnerar en slumpmässigt utformad labyrint med hjälp av pseudokoden ovan (eller på egen hand för den modiga/nyfikna). Ta hjälp av metoderna `addWallToList` och `threeWallsAround`.
- Skapa en ny slumpmässig labyrint i `AMazeIngRace` genom att anropa metoden `random`. Ett bra värde att använda när du anropar metoden är något tal mellan 50 och 100. Det vill säga anropa metoden genom att exempelvis skriva `random(50, 50)`. När du har slumpat fram en labyrint, testa att låta din sköldpadda gå igenom labyranten och se om den lyckas!



Figur 10.4: Ett exempel på hur en slumpmässigt utformad labyrint kan se ut.

Kapitel 11

Sökning, sorterings

Begrepp som ingår i denna veckas studier:

- strängjämförelse
- compareTo
- implicit ordning
- linjärsökning
- binärsökning
- algoritm: LINEAR-SEARCH
- algorit: BINARY-SEARCH
- algoritmisk komplexitet
- sorter till ny vektor
- sorter på plats
- insättningssortering
- urvalssortering
- algoritm: INSERTION-SORT
- algoritm: SELECTION-SORT
- Ordering[T]
- Ordered[T]
- Comparator[T]
- Comparable[T]

11.1 Övning: sorting

Mål

- Förstå hur sorteringsordningen är definierad för strängar.
- Förstå skillnaderna mellan strängjämförelser i Scala och Java, samt kunna jämföra strängar med jämförelsoperatorer i Scala och med `compareTo` i Java.
- Kunna sortera sekvenssamlingar innehållande objekt av grundtyper med hjälp av inbyggda och egendefinierade sorteringsordningar med metoderna `sorted`, `sortBy` och `sortWidth`.
- Kunna använda inbyggda linjärsöknings- och binärsökningssmetoder.
- Kunna implementera en egen sökalgoritm med linjärsökning och binärsökning.
- Förstå när binärsökning är lämplig och möjlig.
- Kunna implementera en enkel sorteringsalgoritm, t.ex. insättningssortering eller urvalssortering, både till ny samling och på plats.
- Känna till hur implicita sorteringsordningar används för grundtyperna och egendefinierade typer.
- Känna till existensen av, funktionen `hos`, och relationen mellan klasserna `Ordering` och `Comparator`, samt `Ordered` och `Comparable`.

Förberedelser

- Studera begreppen i kapitel 11.

11.1.1 Grunduppgifter

Uppgift 1. *Jämföra strängar i Scala.* I Scala kan strängar jämföras med operatorerna `==`, `!=`, `<`, `<=`, `>`, `>=`, där likhet/olikhet avgörs av om alla tecken i strängen är lika eller inte, medan större/mindre avgörs av sorteringsordningen i enlighet med varje teckens Unicode¹-värde.

a) Vad ger följande jämförelser för värde?

```

1 scala> 'a' < 'b'
2 scala> "aaa" < "aaaa"
3 scala> "aaa" < "bbb"
4 scala> "AAA" < "aaa"
5 scala> "ÄÄÄ" < "ÖÖÖ"
6 scala> "ÅÅÅ" < "ÄÄÄ"

```

Tyvärr så följer ordningen av ÅÄÖ inte svenska regler, men det ignorerar vi i fortsättningen för enkelhets skull; om du är intresserad av hur man kan fixa detta, gör uppgift 19.

b) Vilken av strängarna `s1` och `s2` kommer först (d.v.s. är ”mindre”) om `s1` utgör början av `s2` och `s2` innehåller fler tecken än `s1`? 

¹sv.wikipedia.org/wiki/Unicode

Uppgift 2. Jämföra strängar i Java. I Java kan man **inte** jämföra strängar med operatorerna <, <=, >, och >=. Dessutom ger operatorerna == och != inte innehålls(o)likhet utan referens(o)likhet. Istället får man använda metoderna equals och compareTo, vilka också fungerar i Scala eftersom strängar i Scala och Java är av samma typ, nämligen java.lang.String.

- a) Vad ger följande uttryck för värde?

```
1 scala> "hej".getClass.getTypeName
2 scala> "hej".equals("hej")
3 scala> "hej".compareTo("hej")
```

- b) Studera dokumentationen för metoden compareTo i java.lang.String² och skriv minst 3 olika uttryck i Scala REPL som testar hur metoden fungerar i olika fall.
- c) Studera dokumentationen compareToIgnoreCase³ och skriv minst 3 olika stränguttryck i Scala REPL som testar hur metoden fungerar i olika fall.
- d) Vad skriver följande Java-program ut?

```
public class StringEqTest {
    public static void main(String[] args){
        boolean eqTest1 =
            (new String("hej")) == (new String("hej"));
        boolean eqTest2 =
            (new String("hej")).equals(new String("hej"));
        int eqTest3 =
            (new String("hej")).compareTo(new String("hej"));
        System.out.println(eqTest1);
        System.out.println(eqTest2);
        System.out.println(eqTest3);
    }
}
```

Uppgift 3. Sortering med inbyggda sorteringsmetoder. För grundtyperna (Int, Double, String, etc.) finns en fördefinierad ordning som gör så att färdiga sorteringsmetoder fungerar på alla samlingar i scala.collection. Även jämförelseoperatorerna i uppgift 1 fungerar enligt den fördefinierade ordningsdefinitionen för alla grundtyper. Denna ordningsdefinition är *implicit tillgänglig* vilket betyder att kompilatorn hittar ordningsdefinitionen utan att vi explicit måste ange den. Detta fungerar i Scala även med primitiva Array.

- a) Testa metoden sorted på några olika samlingar. Förklara vad som händer. Hur lyder felmeddelande på sista raden? Varför blir det fel?

```
1 scala> Vector(1.1, 4.2, 2.4, 42.0, 9.9).sorted
2 scala> val xs = (100000 to 1 by -1).toArray
```

²docs.oracle.com/javase/8/docs/api/java/lang/String.html#compareTo-java.lang.String-

³docs.oracle.com/javase/8/docs/api/java/lang/String.html#compareToIgnoreCase--java.lang.String-

```

3 scala> xs.sorted
4 scala> xs.map(_.toString).sorted
5 scala> xs.map(_.toByte).sorted.distinct
6 scala> case class Person(firstName: String, familyName: String)
7 scala> val ps = Vector(Person("Robin", "Finkodare"), Person("Kim", "Fulhack"))
8 scala> ps.sorted

```

b) Om man har en samling med egendefinierade klasser eller man vill ha en annan sorteringsordning får man definiera ordningen själv. Ett helt generellt sätt att göra detta på illustreras i uppgift 16, men de båda hjälpmetoderna `sortWith` och `sortBy` räcker i de flesta fall. Hur de används illustreras nedan. Metoden `sortBy` kan användas om man tar fram ett värde av grundtyp och är nöjd med den inbyggda sorteringsordningen.

Metoden `sortWith` används om man vill skicka med ett eget jämförelsepredikat som ordnar två element; funktionen ska returnera **true** om det första elementet ska vara först, annars **false**.

```

1 scala> case class Person(firstName: String, familyName: String)
2 scala> val ps = Vector(Person("Robin", "Finkodare"), Person("Kim", "Fulhack"))
3 scala> ps.sortBy(_.firstName)
4 scala> ps.sortBy(_.familyName)
5 scala> ps.sortBy // tryck TAB två gånger för att se signaturen
6 scala> ps.sortWith((p1, p2) => p1.firstName > p2.firstName)
7 scala> ps.sortWith // tryck TAB två gånger för att se signaturen
8 scala> Vector(9,5,2,6,9).sortWith((x1, x2) => x1 % 2 > x2 % 2)

```

Vad har metoderna `sortWith` och `sortBy` för signaturer?

c) Lägg till attributet `age`: Int i case-klassen `Person` ovan och lägg till fler personer med olika namn och ålder i en vektor och sortera den med `sortBy` och `sortWith` för olika attribut. Välj själv några olika sätt att sortera på.

Uppgift 4. Tidmätning. I kommande uppgifter kommer du att ha nytta av funktionen `timed` enligt nedan.

```

def timed[T](code: => T): (T, Long) = {
  val now = System.nanoTime
  val result = code
  val elapsed = System.nanoTime - now
  println(s"\ntime: ${elapsed / 1e6} ms")
  (result, elapsed)
}

```

a) Klistra in `timed` i REPL och testa så att den fungerar, genom att mäta hur lång tid nedan uttryck tar att exekvera.

```

1 scala> val (v, t1) = timed{ (1 to 1000000).toVector.reverse }
2 scala> val (s, t2) = timed{ v.toSet }
3 scala> timed{ v.find(_ == 1) }
4 scala> timed{ s.find(_ == 1) }
5 scala> timed{ s.contains(1) }

```

-  b) Försök förklara skillnaderna i exekveringstid mellan de olika sätten att söka reda på talet 1 i samlingen. Ungefär hur många gånger behöver man använda `contains` på heltalsmängden `s` för att det ska löna sig att skapa `s` i stället för att linjärsöka i `v` med `find` i ovan exempel?

Uppgift 5. Sökning med inbyggda sökmetoder.

- a) *Linjärsökning framifrån med `indexOfSlice`.* Studera dokumentationen för Scalas samlingsmetod `indexOfSlice`⁴ och skriv 8 olika uttryck i REPL som, både med en sträng och med en vektor med heltalet, provar 4 olika fall: (1) finns i börja, (2) finns någonstans i mitten, (3) finns i slutet, samt (4) finns ej.
- b) *Linjärsökning bakifrån med `lastIndexOfSlice`.* Studera dokumentationen för Scalas samlingsmetod `lastIndexOfSlice`⁵ och skriv 8 olika uttryck i REPL som, både med en sträng och med en vektor med heltalet, provar 4 olika fall: (1) finns i börja, (2) finns någonstans i mitten, (3) finns i slutet, samt (4) finns ej.
- c) *Sökning med inbyggd binärsökning.* Om en samling är sorterad kan man utnyttja detta för att göra snabbare sökning. Vid **binärsökning** (eng. *binary search*)⁶ börjar man på mitten och kollar vilken halva att söka vidare i; sedan delar man upp denna halva på mitten och kollar vilken fjärdedel att söka vidare i, etc.

I objektet `scala.collection.Searching`⁷ finns en metod `search` som, om den importeras, erbjuder binärsökning för alla sorterade sekvenssamlingar. Om samlingen är sorterad ger den ett objekt av case-klassen `Found` som innehåller indexet för platsen där elementet först hittats; alternativt om det som eftersöks ej finns, ges ett objekt av case-klassen `InsertionPoint` som innehåller indexet där elementet borde ha varit placerad om det funnits i samlingen. Observera att om samlingen inte är sorterad är resultatet ”odefinierat”, d.v.s. något returneras men det är *inte* att lita på; man måste alltså först sortera samlingen eller vara helt säker på att den är sorterad.

Undersök hur `search` fungerar genom att förklara vad som händer nedan. Vilken är snabbast av `lin` och `bin` nedan? Använd `timed` från uppgift 4.

```

1  scala> val udda = (1 to 1000000 by 2).toVector
2  scala> import scala.collection.Searching._
3  scala> udda.search(udda.last)
4  scala> udda.search(udda.last + 1)
5  scala> udda.reverse.search(udda(0))
6  scala> def lin(x: Int, xs: Seq[Int]) = xs.indexOf(x)
7  scala> def bin(x: Int, xs: Seq[Int]) = xs.search(x) match {
8      case Found(i) => i
9      case InsertionPoint(i) => -i
10     }
11  scala> timed{ lin(udda.last, udda) }
12  scala> timed{ bin(udda.last, udda) }
```

⁴docs.scala-lang.org/overviews/collections/seqs.html

⁵docs.scala-lang.org/overviews/collections/seqs.html

⁶en.wikipedia.org/wiki/Binary_search_algorithm

⁷[http://www.scala-lang.org/api/current/#scala.collection.Searching\\$](http://www.scala-lang.org/api/current/#scala.collection.Searching$)

- d) Om en samling innehåller n element, hur många jämförelser behövs då vid binärsökning i värsta fall? *Tips:* Läs om algoritmen på wikipedia⁶.

Uppgift 6. Sök bland LTH:s kurser med linjärsökning.

- a) Surfa till denna URL:

http://kurser.lth.se/lot/?lasar=16_17&soek_text=&sort=kod&val=kurs&soek=t

och inspektera html-koden i din webbläsare genom att trycka *Ctrl+U* (fungerar i Firefox och Chrome). Rulla ner till rad 171 och framåt. Var finns antalet poäng för resp kurs i html-koden?

- b) Klistra in objektet courses med kommandot :paste i REPL.⁸ Vad gör koden? Hur många kurser innehåller lth2016?

```
object courses {
  def download(year: String = "16_17"): Vector[Course] = {
    val urlStart = s"http://kurser.lth.se/lot/?lasar=$year"
    val urlSearch = "&soek_text=&sort=kod&val=kurs&soek=t"
    val url = urlStart + urlSearch
    println("*** Downloading from: " + url)
    println("*** This may take a while...")
    val lines = scala.io.Source.fromURL(url).getLines.toVector
    lines.filter(_.contains("kurskod")).map(Course.fromHtml)
  }

  lazy val lth2016: Vector[Course] = download()

  case class Course(
    code: String,
    nameSv: String,
    nameEn: String,
    credits: Double,
    level: String
  )

  object Course {
    import scala.util.Try
    def fromHtml(s: String): Course = {
      def extract(s: String, init: String, stop: Char): String =
        s.replaceAllLiterally(init, "").takeWhile(_ != stop)
      val codeInit = """<a href="/lot/?val=kurs&kurskod="""
      val dataInit = """<td class="mitt">"""
      val xs = s.split("td>")
      val code = Try { extract(xs(1), codeInit, '') }.getOrElse("???")
      val credits = Try {
        val s = extract(xs(2), dataInit, '<')
        s.replaceAllLiterally(",",".").toDouble //fix decimals
      }.getOrElse(0.0)
      val level = Try { extract(xs(3), dataInit, '<') }.getOrElse("???")
      val nameSv = Try { xs(5).takeWhile(_ != '<') }.getOrElse("???")
      val nameEn = Try { xs(7).takeWhile(_ != '<') }.getOrElse("???")
      Course(code, nameSv, nameEn, credits, level)
    }
  }
}
```

⁸Du kan ladda ner koden från:

github.com/lunduniversity/introprog/tree/master/compendium/examples/lth-courses/courses.scala

```
    }
}
}
```

- c) *Linjärsökning med find.* Teknologen Oddput Clementina vill gå första bästa datavetenskapskurs som är på G2-nivå. Hjälp Oddput med att söka upp första bästa kurs genom linjärsökning med samlingsmetoden `find`. Kurskoder vid datavetenskap börjar på EDA eller ETS⁹. *Tips:* Du har nytta av att definiera predikatet `def isCS(s: String): Boolean` som i sin tur lämpligen nyttjar strängmetoden `startsWith`.

- d) *Implementera linjärsökning.* Som träning ska du nu implementera en egen linjärsökningsfunktion med signaturen:

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

Funktionen ska ta en sekvenssamling `xs` och ett predikat `p` som är en funktion som tar ett element och returnerar ett booleskt värde. Funktionen `p` ska ge `true` om parametern är ett eftersökt element. Funktionen `linearSearch` ska returnera index för första hittade elementet i `xs` där `p` gäller. Om det inte finns något element som uppfyller predikatet ska `-1` returneras. Skriv först pseudokod för funktionen med penna och papper. Använd `while`.

Typen `Seq` är supertyp till alla sekvenssamlingar, så om vi använder den som parametertyp för parametern `xs` så fungerar funktionen för `Vector`, `Array`, `List`, etc. Genom typparametern `T` blir funktionen generisk och fungerar för godtycklig typ.

- e) Skriv i en editor en funktion `def rndCode: String` som genererar slumpmässiga kurskoder som består av 6 tecken enligt dessa regler: de första tre tecknen är bokstäver mellan A och Z, de sista två är siffror mellan 0 och 9, medan det fjärde tecknet kan vara antingen en siffra mellan 0 och 9 eller ett av dessa tecken: ACFGLMNP. *Tips:* Använd REPL för att stegvis bygga upp hjälpfunktioner som du, när de fungerar som de ska, klistrar in i ett editorfönster som lokala funktioner där du utvecklar den slutliga koden för en lättläst, concis och fungerande `rndCode`.

- f) Använd `rndCode` från föregående deluppgift för att fylla en vektor kallad `xs` med en halv miljon slumpmässiga kurskoder. För varje slumpkod i `xs` sök med din funktion `linearSearch` efter index i vektorn `courses.lth2016` från deluppgift b). Mät totala tiden för de 500000 linjärsökningarna med hjälp av funktionen `timed` från uppgift 4. Hur många av de slumpmässiga kurskoderna hittades bland de verkliga kurskoderna på LTH?

- g) Hur kan du implementera `linearSearch` med den inbyggda samlingsmetoden `indexWhere`?

⁹Detta är en förenklad bild av LTH:s kurskodnamnsystem. Några kurser från EIT-institutionen kommer att slinka med, men det bortser vi ifrån i denna uppgift.

Uppgift 7. *Sök bland LTH:s kurser med binärsökning.*

Sökningsalgoritmen BINSEARCH kan formuleras med nedan pseudokod:

Indata	: En växande sorterad sekvens xs med n heltal och ett eftersökt heltal key
Resultat	: Ett heltal $i \geq 0$ som anger platsen där x finns, eller ett negativt tal $-i$ där $-i$ motsvarar platsen där x ska sättas in i sorterad ordning om x ej finns i samlingen.

```

1 sätt intervallet (low, high) till (0,  $n - 1$ )
2 found  $\leftarrow$  false
3 mid  $\leftarrow -1$ 
4 while low  $\leq$  high and not found do
5   mid  $\leftarrow$  platsen mitt emellan low och high
6   if xs(mid) == key then
7     | found  $\leftarrow$  true
8   else
9     | if xs(mid) < key then
10    |   | low  $\leftarrow$  mid + 1
11    | else
12    |   | high  $\leftarrow$  mid - 1
13    | end
14  end
15 end
16 if found then
17   | return mid
18 else
19   | return -(low + 1)
20 end

```

- a) Prova algoritmen ovan med penna och papper på en sorterade sekvens med mindre än 10 heltal. Prova om algoritmen fungerar med ett jämt antal tal, ett udda antal tal, en sekvens med ett heltal och en tom sekvens. Prova både om talet du letar efter finns och om det inte finns.
- b) Implementera binärsökning i en funktion med signaturen
`def binarySearch(xs: Seq[String], key: String): Int = ???`
och testa i REPL för olika fall. Vad händer om sekvensen inte är sorterad?
- c) Använd `binarySearch` för att leta efter LTH-kurser enligt nedan. Använd `rndCode`, `timed` och `courses` från tidigare uppgifter.

```

def binarySearch(xs: Seq[String], key: String): Int = ???

val lthCodesSorted = courses.lth2016.map(_.code).sorted
val xs = Vector.fill(500000)(rndCode)
val (_, elapsedBin) =
  timed{xs.map(x => binarySearch(lthCodesSorted, x))}
val (_, elapsedLin) =
  timed{xs.map(x => linearSearch(lthCodesSorted)(_ == x))}
```

```
println(elapsedLin / elapsedBin)
```

- d) Hur mycket snabbare blev binärsökningen jämfört med linjärsökningen?¹⁰

Uppgift 8. *Linjärsökning i Java.* Denna uppgift bygger vidare på uppgift 6 i kapitel 9. Du ska göra en variant på linjärsökning som innebär att leta upp första yatzy-raden i en matris där varje rad innehåller utfallet av 5 tärningskast.

- a) Du ska lägga till metoderna `isYatzy` och `findFirstYatzyRow` i klassen `ArrayMatrix` i uppgift 6 i kapitel 9 enligt nedan skiss. Vi börjar med metoden `isYatzy` i denna deluppgift (nästa deluppgift handlar om `findFirstYatzyRow`). OBS! Det finns en bug i `isYatzy` – rätta bugen och testa så att den fungerar.

```
public static boolean isYatzy(int[] dice){ /* has one bug! */
    int col = 1;
    boolean allSimilar = true;
    while (col < dice.length && allSimilar) {
        allSimilar = dice[0] == dice[col];
    }
    return allSimilar;
}

/** Finds first yatzy row in m; returns -1 if not found */
public static int findFirstYatzyRow(int[][][] m){
    int row = 0;
    int result = -1;
    while (????) {
        /* linear search */
    }
    return result;
}
```

- b) Implementera `findFirstYatzyRow`. Skapa först pseudo-kod för länjärsökningsalgoritmen innan du skriver implementationen i Java. Testa ditt program genom att lägga till följande rader i huvudprogrammet. Metoden `fillRnd` ingår i uppgift 6 i kapitel 9.

```
int[][][] yss = new int[2500][5];
fillRnd(yss, 6);
int i = findFirstYatzyRow(yss);
System.out.println("First Yatzy Index: " + i);
```

Uppgift 9. Implementera sorteringsalgoritmen **insättningssortering** (eng. *insertion sort*) i en funktion med följande signatur:

¹⁰Vid en körning på en i7-4970K med 4.0GHz tog `elapsedLin` cirka 3000 ms och `elapsedBin` cirka 60 ms. Binärsökning var alltså i detta fall ungefär 50 gånger snabbare än linjärsökning.

```
def insertionSort(xs: Seq[Int]): Seq[Int] = ???
```

Lösningssidé: Skapa en ny, tom sekvens som ska bli vårt sorterade resultat. För varje element i den osorterade sekvensen: Sätt in det på rätt plats i den nya sorterade sekvensen.

a) *Pseudokod:* Kör nedan pseudokod med papper och penna t.ex. på sekvensen 5 1 4 3 2 1. Rita minnessituationen efter varje runda i loopen. Här använder vi internt i funktionen föränderliga ArrayBuffer som är snabb på insättning och avslutar med toVector så att vi lämnar ifrån oss en oförändlig sekvens.

```
1 result ← en ny, tom ArrayBuffer
2 foreach element e in xs do
3   | pos ← leta upp rätt position i result
4   | stoppa in e på plats pos i result
5 end
6 result.toVector
```

b) Implementera insertionSort. Använd en **while**-loop för att implementera rad 3 i pseudokoden. Sök upp dokumentationen för metoden *insert* på ArrayBuffer. Testa *insert* på ArrayBuffer i REPL och verifiera att den kan användas för att stoppa in på slutet på den ”oanvända” positionen som är precis efter sista positionen. Vad händer om man gör *insert* på positionen *size + 2*?

Klistra in din implementation av insertionSort i REPL och testa så att allt fungerar:

```
1 scala> insertionSort(Vector())
2 res0: Seq[Int] = Vector()
3
4 scala> insertionSort(Vector(42))
5 res1: Seq[Int] = Vector(42)
6
7 scala> insertionSort(Vector(1,2,3))
8 res2: Seq[Int] = Vector(1, 2, 3)
9
10 scala> insertionSort(Vector(5,1,4,3,2,1))
11 res3: Seq[Int] = Vector(1, 1, 2, 3, 4, 5)
```

Uppgift 10. Implementera sortering på plats (eng. *in-place*) i en *Array[String]* med urvalssortering (eng. *selection sort*)

Lösningssidé: För alla index *i*: sök *minIndex* för ”minsta” strängen från plats *i* till sista plats och byt plats mellan strängarna på plats *i* och plats *minIndex*. Se även animering här: sv.wikipedia.org/wiki/Urvallssortering

Implementera enligt nedan skiss. *Tips:* Du har nytta av en modifierad variant av lösningen till uppgift 17 i kapitel 2.

```
def selectionSortInPlace(xs: Array[String]): Unit = {
  def indexOfMin(startFrom: Int): Int = ???
  def swapIndex(i1: Int, i2: Int): Unit = ???
```

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

11.1.2 Extrauppgifter

Uppgift 11. Undersök om en sekvens är sorterad. Ett enkelt och lättläst sätt att undersöka om en sekvens är sorterad visas nedan.

```
1 scala> def isSorted(xs: Vector[Int]): Boolean = xs == xs.sorted
```

- ☞ a) Om xs har 10^6 element, hur många jämförelser kommer i värsta fall att ske med `isSorted` enligt ovan. Metoden `sorted` använder algoritmen Timsort¹¹. Sök upp antalet jämförelser i värstafallet på wikipedia.⁼⁼⁼⁼⁼ Denna lösning är dock relativt långsam för stora samlingar. Man behöver ju inte först sortera för att avgöra om det är sorterat (om man inte ändå hade tänkt sortera av andra skäl), det räcker att kolla att elementen är i växande ordning.
- ☞ b) Om xs har n element, ungefär hur många jämförelser kommer i värsta fall att ske med `isSorted` ovan om man alltså först ska sortera och sedan jämföra den osorterade och den sorterade samlingen element för element? Metoden `sorted` använder algoritmen Timsort¹². Sök upp värstafallsprestandan för Timsort på wikipedia.¹³
- ☞ c) Implementera en effektivare variant av `isSorted` som använder en `while`-sats och kollar att elementen är i växande ordning.
- ☞ d) Vad blir antalet jämförelser i värstafallet med metoden i deluppgift c om du har n element?
- ☞ e) Man kan kolla om en sekvens är sorterad med det listiga tricket att först zippa sekvensen med sin egen svans och sedan kolla om alla element-par uppfyller sorteringskriteriet, alltså `xs.zip(xs.tail).forall(???)` där ??? byts ut mot lämpligt predikat. Vilken typ har 2-tupeln `xs.zip(xs.tail)` om xs är av typen `Vector[Int]`? Implementera `isSorted` med detta listiga trick. (Senare, i fördjupningsuppgift 15, ska vi göra `isSorted` generellt användbar för olika typer och olika ordningsdefinitioner.) ⁼⁼⁼⁼⁼ f) Man kan kolla om en sekvens är sorterad med det listiga tricket att först zippa sekvensen med sin egen svans och sedan kolla om alla element-par uppfyller sorteringskriteriet, alltså `xs.zip(xs.tail).forall(???)` där ??? byts ut mot lämpligt predikat. Vilken typ har 2-tupeln `xs.zip(xs.tail)` om xs är av typen `Vector[Int]`? Implementera `isSorted` med detta listiga trick. (I fördjupningsuppgift 15 görs denna variant av `isSorted` generellt användbar för olika typer och olika ordningsdefinitioner.)

¹¹stackoverflow.com/questions/14146990/what-algorithm-is-used-by-the-scala-library-method-vector-sorted

¹²stackoverflow.com/questions/14146990/what-algorithm-is-used-by-the-scala-library-method-vector-sorted

¹³en.wikipedia.org/wiki/Timsort

Uppgift 12. Implementera och testa sorteringsalgoritmen i pseudokod med *instickssortering*¹⁴.

- a) Implementera och testa funktionen nedan i Scala med följande signatur:

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

Placera metoden i ett objekt med lämpligt namn, samt skapa ett huvudprogram med testkod. Kompilera och kör från terminalen. Börja med att skriva sorteringsalgoritmen i pseudokod.

- b) Implementera och testa metoden nedan i Java med följande signatur:

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

Placera metoden i en klass med lämpligt namn, samt skapa ett huvudprogram med testkod. Börja med att skriva sorteringsalgoritmen i pseudokod.

Uppgift 13. Implementera och testa sorteringsalgoritmen till ny sekvens med urvalssortering¹⁵ i Scala, enligt nedan skiss. *Tips:* Du har nytta av lösningen till uppgift 17 i kapitel 2.

```
def selectionSort(xs: Seq[String]): Seq[String] = {
  def indexOfMin(xs: Seq[String]): Int = ???
  val unsorted = xs.toBuffer
  val result = scala.collection.mutable.ArrayBuffer.empty[String]
  /*
  så länge unsorted inte är tom {
    minPos = indexOfMin(unsorted)
    elem   = unsorted.remove(minPos)
    result.append(elem)
  }
  */
  result.toVector
}
```

11.1.3 Fördjupningsuppgifter

Uppgift 14. *Typklasser och implicita parametrar.* I Scala finns möjligheter till avancerad funktionsprogrammering med s.k. **typklasser**, som definierar generella beteenden som fungerar för befintliga typer utan att implementationen av dessa befintliga typer behöver ändras. Vi nosar i denna uppgift på hur implicita argument kan användas för att skapa typklasser, illustrerat med hjälp av implicita ordningarna, som är en typisk och användbar tillämpning av konceptet typklasser.

¹⁴en.wikipedia.org/wiki/Insertion_sort

¹⁵en.wikipedia.org/wiki/Selection_sort

- a) *Implicit parameter och implicit värde.* Med nyckelordet **implicit** framför en parameter öppnar man för möjligheten att låta kompilatorn ge argumentet "automatiskt" om den kan hitta ett värde med passande typ som också är deklarerat med **implicit**, så som visas nedan.

```

1  scala> def add(x: Int)(implicit y: Int) = x + y
2  scala> add(1)(2)
3  scala> add(1)
4  scala> implicit val ngtNamn = 42
5  scala> add(1)

```

Vad blir felmeddelandet på rad 3 ovan? Varför fungerar det på rad 5 utan fel?

- b) *Typklasser.* Genom att kombinera koncepten implicita värden, generiska klasser och implicita parametrar får man möjligheten att göra typklasser, så som `CanCompare` nedan, som vi kan få att fungera för befintliga typer utan att de behöver ändras.

Vad händer nedan? Vilka rader ger felmeddelande? Varför?

```

1  scala> trait CanCompare[T] { def compare(a: T, b: T): Int }
2  scala> def sort2[T](a: T, b: T)(implicit cc: CanCompare[T]): (T, T) =
3      if (cc.compare(a, b) > 0) (b, a) else (a, b)
4  scala> sort2(42, 41)
5  scala> implicit object intComparator extends CanCompare[Int]{
6      override def compare(a: Int, b: Int): Int = a - b
7  }
8  scala> sort2(42, 41)
9  scala> sort2(42.0, 41.0)

```

- c) Definiera ett implicit objekt som gör så att `sort2` fungerar för värden av typen `Double`.
- d) Definiera ett implicit objekt som gör så att `sort2` fungerar för värden av typen `String`.

Uppgift 15. *Användning av implicit ordning.* Vi ska nu göra `isSorted` från uppgift 11 mer generellt användbar genom att möjliggöra att implicita ordningsfunktioner finns tillgängliga för olika typer.

- a) Med signaturen `isSorted(xs: Vector[Int]): Boolean` så fungerar sorteringsmetoden bara för samlingar av typen `Vector[Int]`. Om vi i stället använder `isSorted(xs: Seq[Int]): Boolean` fungerar den för alla samlingar med heltal, även `Array` och `List`. Testa nedan funktion i REPL med heltalssekvenser av olika typ.

```
def isSorted(xs: Seq[Int]): Boolean = xs == xs.sorted
```

- b) Men vi vill gärna att `isSorted` ska fungera för godtyckliga typer `T` som har en ordningsdefinition. Detta kan göras med nedan funktion eftersom metoden `sorted` är definierad för alla samlingar där typen `T` har en implicit ordning. Speciellt gäller detta för alla de grundtyperna `Int`, `Double`, `String`, etc.

```
def isSorted[T](xs: Seq[T]): Boolean = xs == xs.sorted
```

Testa metoden ovan i REPL enligt nedan.

```
1 scala> isSorted(Vector(1,2,3))
2 scala> isSorted(Array(1,2,3,1))
3 scala> isSorted(Vector("A","B","C"))
4 scala> isSorted(List("A","B","C","A"))
5 scala> case class Person(firstName: String, familyName: String)
6 scala> val persons = Vector(Person("Kim", "Finkodare"), Person("Robin", "Fulhack"))
7 scala> isSorted(persons)
```

Vad ger sista raden för felmeddelande? Varför?

- c) Vi vill gärna kunna jämföra element av godtycklig typ T , så att vi till exempel ska kunna implementera en generisk `isSorted` med `while` eller vårt zip-trick från uppgift 11f. Men det blir problem enligt nedan. Hur lyder felmeddelandet? Vad saknas?

```
1 scala> def isSorted[T](xs: Seq[T]): Boolean =
2     xs.zip(xs.tail).forall(x => x._1 <= x._2)
```

- d) Det blir även problem med denna implementation. Hur lyder felmeddelandet? Vad saknas?

```
scala> def isSorted[T](xs: Seq[T]): Boolean = xs == xs.sorted
```

- e) *Implicita ordningar.* Man kan berätta för kompilatorn att den ska leta efter implicita ordningar av typen T . Detta kan göras genom att utöka signaturen för `isSorted` med en andra parameterlista, som tar en `implicit` parameter enligt följande:

```
def isSorted[T](xs: Seq[T])(implicit ord: Ordering[T]): Boolean =
  xs.zip(xs.tail).forall(x => ord.lteq(x._1, x._2))
```

Det finns fördefinierade implicita objekt `Ordering[T]` för alla grundtyper, alltså t.ex. `Ordering[Int]`, `Ordering[String]`, etc. Objekt av typen `Ordering` har jämförelsemetoder som t.ex. `lteq` (förk. *less than or equal*) och `gt` (förk. *greater than*). Testa så att kompilatorn hittar ordningen för samlingar med värden av några grundtyper. Kontrollera även enligt nedan att det fortfarande blir problem för egendefinierade klasser, t.ex. `Person` enligt tidigare (detta ska vi råda bot på i uppgift 16).

```
1 scala> isSorted(Vector(1,2,3))
2 scala> isSorted(Array(1,2,3,1))
3 scala> isSorted(Vector("A","B","C"))
4 scala> isSorted(List("A","B","C","A"))
5 scala> class Person(firstName: String, familyName: String)
6 scala> val persons = Vector(Person("Kim", "Finkodare"), Person("Robin", "Fulhack"))
7 scala> isSorted(persons)
```

- f) *Importera implicita ordningsoperatorer från en Ordering.* Om man gör import på ett `Ordering`-objekt får man tillgång till implicita konverteringar som gör att jämförelseoperatorerna fungerar. Testa nedan variant av `isSorted` på olika sekvenstyper och verifiera att \leq , $>$, etc., nu fungerar enligt nedan.

```
def isSorted[T](xs: Seq[T])(implicit ord: Ordering[T]): Boolean = {
    import ord._
    xs.zip(xs.tail).forall(x => x._1 <= x._2)
}
```

Uppgift 16. Skapa egen implicit ordning med Ordering.

- a) Ett sätt att skapa en egen, specialanpassad ordning är att mappa dina objekt till typer som redan har en implicit ordning. Med hjälp av metoden `by` i objektet `scala.math.Ordering` kan man skapa ordningar genom bifoga en funktion `T => S` där `T` är typen för de objekt du vill ordna och `S` är någon annan typ, t.ex. `String` eller `Int`, där det redan finns en implicit ordning.

```
1  scala> case class Team(name: String, rank: Int)
2  scala> val xs =
3      Vector(Team("fnatic", 1499), Team("nip", 1473), Team("lumi", 1601))
4  scala> xs.sorted // Hur lyder felmeddelandet? Varför blir det fel?
5  scala> val teamNameOrdering = Ordering.by((t: Team) => t.name)
6  scala> xs.sorted(teamNameOrdering) //explicit ordning
7  scala> implicit val teamRankOrdering = Ordering.by((t: Team) => t.rank)
8  scala> xs.sorted // Varför funkar det nu?
```

- b) Vill man sortera i omvänt ordning kan man använda `Ordering.fromLessThan` som tar en funktion `(T, T) => Boolean` vilken ska ge `true` om första parametern ska komma före, annars `false`. Om vi vill sortera efter rank i omvänt ordning kan vi göra så här:

```
1  scala> val highestRankFirst =
2      Ordering.fromLessThan[Team]((t1, t2) => t1.rank > t2.rank)
3  scala> xs.sorted(highestRankFirst)
```

- c) Om du har en case-klass med flera fält och vill ha en fördefinierad implicit sorteringsordning samt även erbjuda en alternativ sorteringsordning kan du placera olika ordningsdefinitioner i ett kompanjonobjekt; detta är nämligen ett av de ställen där komplatorn söker efter eventuella implicita värden innan den ger upp att leta.

```
case class Team(name: String, rank: Int)
object Team {
    implicit val highestRankFirst = Ordering.fromLessThan[Team]{
        (t1, t2) => t1.rank > t2.rank
    }
    val nameOrdering = Ordering.by((t: Team) => t.name)
}
```

```
1  scala> :pa
2  // Exiting paste mode, now interpreting.
3  case class Team(name: String, rank: Int)
4  object Team {
5      implicit val highestRankFirst =
```

```

6   Ordering.fromLessThan[Team]((t1, t2) => t1.rank > t2.rank}
7   val nameOrdering = Ordering.by((t: Team) => t.name)
8 }
9 scala> val xs =
10    Vector(Team("fnatic", 1499), Team("nip", 1473), Team("lumi", 1601))
11 scala> xs.sorted
12 scala> xs.sorted(Team.nameOrdering)

```

- d) Det går också med kompanjonsobjektet ovan att få jämförelseoperatorer att fungera med din case-klass, genom att importera medlemmarna i lämpligt ordningsobjekt. Verifiera att så är fallet enligt nedan:

```

1 scala> Team("fnatic",1499) < Team("gurka", 2) // Vilket fel? Varför?
2 scala> import Team.highestRankFirst._
3 scala> Team("fnatic",1499) < Team("gurka", 2) // Inget fel? Varför?

```

Uppgift 17. *Specialanpassad ordning genom att ärva från Ordered.* Om det finns en väldefinierad, specifik ordning som man vill ska gälla för sina case-klass-instanser kan man göra den ordnad genom att låta case-klassen mixa in traiten Ordered och implementera den abstrakta metoden compare.

Bakgrund för kännedom: En trait som används på detta sätt kallas **gränssnitt** (eng. *interface*), och om man *implementerar* ett gränssnitt så uppfyller man ett ”kontrakt”, som i detta fall innebär att man implementerar det som krävs av ordnade objekt, nämligen att de har en konkret compare-metod. Du lär dig mer om gränssnitt i kommande kurser.

- a) Implementera case-klassen Team så att den är en subtyp till Ordered enligt nedan skiss. Högre rankade lag ska komma före lägre rankade lag. Metoden compare ska ge ett heltal som är negativt om **this** kommer före **that**, noll om de ordnas lika, annars positivt.

```

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

```

Tips: Du kan anropa metoden compare på alla grundtyper, t.ex. Int, eftersom de är implicit ordnade. Genom att negera uttrycket blir ordningen den omvänta.

```

1 scala> -(2.compare(1))

```

- b) Testa att din case-klass nu uppfyller det som krävs för att vara ordnad.

```

1 scala> Team("fnatic",1499) < Team("gurka", 2)

```

Uppgift 18. *Jämförelsestöd i Java.* Java har motsvarigheter till Ordering och Ordered, som heter java.util.Comparator och java.lang.Comparable. I själva verket så är Scalas Ordering en subtyp till Javas Comparator, medan Scalas Ordered är en subtyp till Javas Comparable.

- Javas Comparator och Scalas Ordering används för att skapa fristående ordningar som kan jämföra *två olika* objekt. I Scala kan dessa göras implicit tillgängliga. I Javas samlingsbibliotek skickas instanser av Comparator med som expilicita argument.
- Javas Comparable och Scalas Ordered används som supertyp för klasser som vill kunna jämföra "sig själv" med andra objekt och har *en* naturlig ordningsdefinition.

-  a) Sök upp dokumentationen för java.util.Comparator. Vilken abstrakt metod måste implementeras och vad gör den?
- b) I paketet java.util.Arrays finns en metod sort som tar en Array[T] och en Comparable[T]. Testa att använda dessa i REPL enligt nedan skiss. Starta om REPL så att ev. tidigare implicita ordningar för Team inte finns kvar.

```

1  scala> import java.util.Comparator
2  scala> val teamComparator = new Comparator[Team]{
3      def compare(o1: Team, o2: Team) = ???
4  }
5  scala> val xs =
6      Array(Team("fnatic", 1499), Team("nip", 1473), Team("lumi", 1601))
7  scala> java.util.Arrays.sort(xs.toArray, teamComparator)
8  scala> xs

```

- c) I Scala finns en behändig metod Ordering.comparatorToOrdering som skapar en implicit tillgänglig ordning om man har en java.util.Comparator. Testa detta enligt nedan i REPL, med deklarationerna från föregående deluppgift.

```

1  scala> implicit val teamOrd = Ordering.comparatorToOrdering(teamComparator)
2  scala> xs.sorted

```

-  d) Sök upp dokumentationen för java.lang.Comparable. Vilken abstrakt metod måste implementeras och vad gör den?
- e) Gör så att klassen Point är Comparable och att punkter närmare origo sorteras före punkter som är längre ifrån origo enligt nedan skiss. I Scala är typer som är Comparable implicit även Ordered, varför sorteringen nedan funkar. Verfiera detta i REPL när du klurat ut hur implementera compareTo.

```

case class Point(x: Int, y: Int) extends Comparable[Point] {
    def distanceFromOrigin: Double = ???
    def compareTo(that: Point): Int = ???
}

```

```

1  scala> val xs = Seq(Point(10,10), Point(2,1), Point(5,3), Point(0,0))
2  scala> xs.sorted

```

Uppgift 19. Fixa svensk sortningsordning av ÅÅÖ. Svenska bokstäver kommer i, för svenskar, konstig ordning om man inte vidtar speciella åtgärder. Med

hjälp av klassen `java.text.Collator` kan man få en `Comparator` för strängar som följer lokala regler för en massa språk på planeten jorden.

- a) Verifiera att sorteringsordningen blir rätt i REPL enligt nedan.

```
1 scala> val fel = Vector("ö", "å", "ä", "z").sorted
2 scala> val svColl = java.text.Collator.getInstance(new java.util.Locale("sv"))
3 scala> val svOrd = Ordering.comparatorToOrdering(svColl)
4 scala> val rätt = Vector("ö", "å", "ä", "z").sorted(svOrd)
```

- b) Använd metoden ovan för att skriva ett program som skriver ut raderna i en textfil i korrekt svensk sorteringsordning. Programmet ska kunna köras med kommandot:

`scala sorted -sv textfil.txt`

- c) Läs mer här:

stackoverflow.com/questions/24860138/sort-list-of-string-with-localization-in-scala

Uppgift 20. I klassen `java.util.Arrays`¹⁶ finns en statisk metod `binarySearch` som kan användas enligt nedan.

```
1 scala> val xs = Array(5,1,3,42,-1)
2 scala> java.util.Arrays.sort(xs)
3 scala> xs
4 scala> java.util.Arrays.binarySearch(xs, 42)
5 scala> java.util.Arrays.binarySearch(xs, 43)
```

Skriv ett valfritt Java-program som testar `java.util.Arrays.binarySearch`. Använd en array av typen `int[]` med några heltal som först sorteras med `java.util.Arrays.sort`. Skriv ut det som returneras från `java.util.Arrays.binarySearch` i olika fall genom att asöka efter tal som finns först, mitt i, sist och tal som saknas. *Tips:* Man kan deklarera en array, allokerha den och fylla den med värden så här i Java:

`int[] xs = new int[]{5, 1, 3, 42, -1};`

Uppgift 21. Fördjupa dig inom webbteknologi.

- a) Lär dig om HTML här: <http://www.w3schools.com/html/>
- b) Lär dig om Javascript här: <http://www.w3schools.com/js/>
- c) Lär dig om CSS här: <http://www.w3schools.com/css/>
- d) Lär dig om Scala.JS här: <http://www.scala-js.org/>

¹⁶docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

11.2 Laboration: survey

Mål

- Kunna använda inbyggda sorteringsfunktioner.
- Kunna använda inbyggda sökfunktioner.
- Känna till hur strängar ordnas.
- Kunna läsa text i tabellform från fil och webbadress.
- Kunna använda registrering (frekvensräkning) för enkla statistikberäkningar.
- Kunna skriva till fil.
- Kunna omvandla startargument till kommandon.
- Kunna skriva rekursiva funktioner.

Förberedelser

- Studera begreppen i kapitel 10.
- Gör övning sorting i avsnitt 11.1.
- Läs igenom och begrunda hela laborationsinstruktionen.
- Fyll i denna enkät: <https://goo.gl/forms/hC6JK2UQXVpbGEc2>
I enkäten ska du för olika flervalsalternativlistor besvara frågan:
Vilket är ditt favoritalternativ?

11.2.1 Bakgrund

I den här veckans laboration ska du utveckla ett program som analyserar svar på enkäter med flervalsfrågor. Indata utgörs av text i form av **kolumnseparerade värden**, där varje persons svar finns på en egen rad och varje svarsrad innehåller svarsalternativ separerade med en **kolumnseparator** som till exempel kan vara “\t” eller “,”. Första raden i textfilen anger kolumnernas namn. Exempelindata finns i filen `favorit.csv` i mappen resources:

```
Program,Indent,UI,Lang,OS,Browser,DE
D,Spaces,Terminal,C,BSD,Firefox,Emacs
C,Spaces,Terminal,Javascript,Windows 7,Chrome,Notepad++
D,Spaces,GUI,Java,macOS,Safari,Gedit
I,Tabs,Terminal,PHP,Windows 10,Edge,Notepad++
C,Spaces,GUI,Java,Windows 8,Firefox,Eclipse
D,Spaces,Terminal,Java,Windows 8,Edge,Eclipse
F,Spaces,Terminal,C,Linux,Chrome,Emacs
D,Spaces,GUI,C,Linux,Firefox,Vim
Nano,Tabs,Terminal,Javascript,macOS,Safari,Vim
C,Tabs,Terminal,C#,Windows 10,Edge,Visual Studio
D,Tabs,GUI,Javascript,macOS,Chrome,Emacs
D,Spaces,GUI,Python,Windows 7,Chrome,Notepad++
E,Spaces,Terminal,Java,Linux,Chromium,Eclipse
I,Tabs,Terminal,Python,Windows 10,Chrome,Notepad++
K,Tabs,GUI,C#,Windows 7,Firefox,Visual Studio
F,Spaces,Terminal,C,Linux,Firefox,Vim
D,Tabs,GUI,C,Linux,Chrome,Gedit
```

Ditt program ska bestå av följande delar:

- En case-klass för strängmatriser som heter Table med funktioner för inläsning av tabellformatterad text, sortering, filtrering och registrering med avseende på en viss kolumn.
- Ett objekt för argumentparsning och kommandoexekvering som heter Command.
- En given Main-fil som ska kunna köra det slutgiltiga programmet.
- Funktion för att presentera statistik från enkätdaten med hjälp av registrering.

11.2.2 Given kod

Given kod finns här:

https://github.com/lunduniversity/introprog/tree/master/workspace/w10_survey/src/main

Så här ser huvudprogrammet ut:

```
package stats

import scala.util.Try

object Main {

  val usage =
    s"Usage: <uri> <separator> [Commands]\n\n${Command.list}"

  def main(args: Array[String]): Unit = args.toVector match {

    case uri :: sep :: rest if rest != Vector() => {
      Try {
        println(s"""Loading "$uri" "$sep" to table ...""")
        val t = Table.fromFile(uri, sep)
        println(s"Done. Size: ${t.dim._1}x${t.dim._2}.\n")
        val commands = Command.parseAll(rest)
        Command.runAllWith(commands, t)
      }.recover {
        case e: Exception => println(s"Error: $e\n\n$usage")
      }
    }
    case _ => println(s"Not enough arguments.\n\n$usage")
  }
}
```

11.2.3 Obligatoriska uppgifter

Uppgift 1. Implementera Table enligt specifikation:

<i>Specification Table</i>
<pre> package stats /** * A representation of text data in matrix form. * * @param matrix The data. Rows in the outer Vector, columns in the inner. * @param headings Column headings. * @param sep The String character that separates the columns. */ case class Table(matrix: Vector[Vector[String]], headings: Vector[String], sep: String) { /** Returns the width (columns) and height (rows) of the matrix data. */ val dim: (Int, Int) = ??? /** Returns the values from a specified column. */ def col(c: Int): Vector[String] = ??? /** Returns the matrix in text format */ override lazy val toString: String = ??? /** Returns a copy of itself with the rows sorted on the specified column. */ def sort(c: Int): Table = ??? /** * Returns a copy of itself with the rows filtered so that the given column * only contains the wanted values. */ def filter(c: Int, wanted: Vector[String]): Table = ??? /** * Returns the distinct values for the given column coupled with the number * of occurrences for that value. The tuples are sorted descendingly on the * number of occurrences. The first element is the column header together * with the total number of occurrences for all values. */ def register(c: Int): Vector[(String, Int)] = ??? } object Table { /** * Reads column separated text data from either a file or a URL into a Table. * * @param uri The location of the data. * @param sep The character that separates the columns. */ def fromFile(uri: String, sep: String): Table = ??? /** Writes a text formatted Table to disk. */ } </pre>

```

def toFile(path: String, table: Table): Unit = ???

/** To test Table */
def main(args: Array[String]): Unit = {
  val table = Table.fromFile("src/main/resources/favorit.csv", ",")
  val tablefs = table.filter(4, Vector("Linux")).sort(6)
  println(tablefs.register(6).mkString("\n"))
  toFile("src/main/resources/out.csv", tablefs)
}
}

```

- a) Börja med att implementera klassen Table men vänta med register till nästa uppgift. I sort och filter ska inbyggda funktioner för sorterings och filtrering användas. Eftersom Table är omuterbar måste nya instanser eller kopior skapas av Table vid varje filtrering eller sorterings.
- b) Implementera register som senare kommer användas för att skapa diagram som presenterar den registrerade datan. För att registrera antalet förekomster av varje unikt värde kan funktionerna groupBy och mapValues användas.

Omvandlingen går från Vector[String] till Map[String, Vector[String]] med groupBy och efter det till Map[String, Int] med mapValues. Exempel med groupBy:

```

scala> Vector("ICA", "COOP", "Konsum", "ICA", "ICA", "COOP").groupBy(v => v)

res0: Map[String,Vector[String]] = Map(COOP -> Vector(COOP, COOP),
                                         Konsum -> Vector(Konsum),
                                         ICA -> Vector(ICA, ICA, ICA))

```

- c) Implementera funktionen fromFile i objektet Table. Parametern uri är antingen en webbadress eller en lokal sökväg. Det räcker med att anta att en webbadress börjar på "http".

En fil som innehåller tre rader med 3 kolumner på varje rad läses in till vektorn så här:

```

Vector(Vector(rad1kolumn1,rad1kolumn2,rad1kolumn3),
      Vector(rad2kolumn1,rad2kolumn2,rad2kolumn3),
      Vector(rad3kolumn1,rad3kolumn2,rad3kolumn3))

```

- d) Implementera toFile med valfri metod för att skriva till fil.
- e) Testa Table genom att köra main-funktionen. Om allt står rätt till finns det nu en fil out.csv i mappen resources och i konsollen lyder utskriften:

```

(DE,5)
(Vim,2)
(Eclipse,1)
(Emacs,1)
(Gedit,1)

```

Uppgift 2. Komplettera Command enligt specifikation:

Specification	Command
	<pre>package stats /** Defines, creates and executes commands. */ object Command { val list = """Commands: -save <filepath> Saves the table to <filepath>. -sep <separator> Changes the separator of the table. -filter <column> <values: a b c> Filters the table so <column> only contains <values>. -sort <column> Sorts the table on <column>. -printchart <columns: 3 4 5> Prints barcharts of the <columns> to the console. -print Prints the table to the console.""".stripMargin type Command = Table => Table /** * Takes arguments and matches them to a single command or throws an * exception if no match is found. */ def parseOne(args: Vector[String]): Command = ??? /** Turns a collection of arguments into a collection of commands. */ def parseAll(args: Vector[String]): Vector[Command] = ??? /** * Executes the specified commands in a chain. The input to the next command * is the output from the previous. */ def runAllWith(commands: Vector[Command], table: Table): Table = ??? }</pre>

Observera att i objektet Command definieras typen Command. En instans av typen Command är det som *kommando* hädanefter kommer åsyfta. Det är helt enkelt en funktion som tar emot en instans av Table, gör något med det och returnerar tillbaka en instans av Table. På så vis kan flera kommandon exekveras som en kedja där nästa kommandos indata är det föregående kommandots utdata. Exempel på ett kommando som lägger till ett utropstecken på kolumnrubrikerna:

```
t: Table => t.copy(headings = t.headings.map(_ + "!"))
```

Metoderna Command.parseOne och Command.parseAll omvandlar argument till kommandon.

- Implementera parseOne. Lägg märke till att likt parseAll är parametertypen Vector[String], men skillnaden är att parseOne ska matcha och

returnera exakt ett kommando. I annat fall kastas ett generellt undantag: `throw new Exception("meddelande")`. Till exempel “-sort 4” funkar men “-sprt 4” och “-sort 4 asdf” funkar inte. De giltiga argumenten som ska kunna tas emot finns i `Command.list`. Undantaget som kastas då något annat än ett strängheltal ska göras om till en `Int` tas hand om i `Main` och behöver inte hanteras.

Om “-printchart 6” körs på datan från `favorit.csv` ska följande diagram produceras:

```
Column: DE (17)
Notepad++: 4 occurrences
****
Eclipse: 3 occurrences
 ***
Emacs: 3 occurrences
 ***
Vim: 3 occurrences
 ***
Gedit: 2 occurrences
 **
Visual Studio: 2 occurrences
 **
```

Om flera kolumner efterfrågas, till exempel “-printchart 3 5 6”, skrivs diagrammen ut direkt efter varandra.

b) Implementera `parseAll` som en rekursiv funktion. Använd minustecknet som markerar starten för ett nytt kommando för att dela upp argumenten `args` i delar som sedan var för sig skickas med i anrop till `parseOne`. Kanske funktionen `span` kan vara till nytta:

```
scala> Vector("myra", "panda", "", "ekorre").span(_.nonEmpty)

res1: (Vector[String], Vector[String]) = (Vector(myra, panda),
                                            Vector("", ekorre))
```

c) Implementera `runAllWith`. Eftersom kommandona ska exekveras i en kedja passar det väldigt bra att även göra den här funktionen rekursiv.

d) Testa ditt program med `Main-filen` och `favorit.csv`.

```
>scala stats.Main favorit.csv , -filter 2 Terminal -sort 3 -sep "|" -print
-printchart 5
Loading "favorit.csv" "," to table ...

Done. Size: 17x7.

Program|Indent|UI|Lang|OS|Browser|DE
D|Spaces|Terminal|C|BSD|Firefox|Emacs
F|Spaces|Terminal|C|Linux|Chrome|Emacs
F|Spaces|Terminal|C|Linux|Firefox|Vim
C|Tabs|Terminal|C#|Windows 10|Edge|Visual Studio
D|Spaces|Terminal|Java|Windows 8|Edge|Eclipse
E|Spaces|Terminal|Java|Linux|Chromium|Eclipse
C|Spaces|Terminal|Javascript|Windows 7|Chrome|Notepad++
```

```
Nano|Tabs|Terminal|Javascript|macOS|Safari|Vim  
I|Tabs|Terminal|PHP|Windows 10|Edge|Notepad++  
I|Tabs|Terminal|Python|Windows 10|Chrome|Notepad++  
  
Column: Browser (10)  
Chrome: 3 occurrences  
***  
Edge: 3 occurrences  
***  
Firefox: 2 occurrences  
**  
Chromium: 1 occurrence  
*  
Safari: 1 occurrence  
*
```

Uppgift 3. Avslutningsvis, testa att programmet fungerar tillsammans med länken till enkätsvaren: <https://goo.gl/qPcuA0>.

11.2.4 Frivilliga extrauppgifter

Uppgift 4. Gör en variant av register med `foldLeft` istället för `groupBy` och `mapValues`.

- Implementera funktionen och testa att den fungerar. Överväg fördelarna hos respektive implementation.

Uppgift 5. Inför lämpliga argument till programmet som medför att fina tårtdiagram och/eller stapeldiagram i SimpleWindow med automatiska färger ritas ut som illustrationer till frekvenserna i de olika kolumnerna.

Kapitel 12

Scala och Java

Begrepp som ingår i denna veckas studier:

- syntaxskillnader mellan Scala och Java
- klasser i Scala vs Java
- referensvariabler vs enkla värden i Java
- referenstilldelning vs värdetilldelning i Java
- alternativ konstruktör i Scala och Java
- for-sats i Java
- java for-each i Java
- java.util.ArrayList
- autoboxing i Java
- primitiva typer i Java
- wrapperklasser i Java
- samlingar i Java vs Scala
- scala.collection.JavaConverters
- namnkonventioner för konstanter

12.1 Övning: scalajava

Mål

- Kunna förklara och beskriva viktiga skillnader mellan Scala och Java.
- Kunna översätta enkla algoritmer, klasser och singeltonobjekt från Scala till Java och vice versa.
- Känna till vad en case-klass innehåller i termer av en Javaklass.
- Kunna använda Javatyperna List, ArrayList, Set, HashSet och översätta till deras Scalamotsvarigheter med JavaConverters.
- Kunna förklara hur autoboxning fungerar i Java, samt beskriva fördelar och fallgropar.

Förberedelser

- Studera begreppen i kapitel 12.

12.1.1 Grunduppgifter

Uppgift 1. Översätta metoder från Java till Scala. I denna uppgift ska du översätta en Java-klass som används som en modul¹ och bara innehåller statiska metoder och inget varaktigt tillstånd. (I nästa uppgift ska du sedan översätta klasser med attribut och varaktiga tillstånd.)

Vi börjar med att göra översättningen från Java till Scala rad för rad och du ska behålla så mycket som möjligt av syntax och semantik så att Scala-koden blir så Java-lik som möjligt. I efterföljande deluppgift ska du sedan omforma översättningen så att Scala-koden blir mer idiomatisk².

a) Studera klassen Hangman nedan. Du ska översätta den från Java till Scala enligt de riklinjer och tips som följer efter koden. Läs igenom alla riklinjer och tips innan du börjar.

```

1 import java.net.URL;
2 import java.util.ArrayList;
3 import java.util.Set;
4 import java.util.HashSet;
5 import java.util.Scanner;
6
7 public class Hangman {
8     private static String[] hangman = new String[]{
9         " =====  ",
10        "   /     |  ",
11        "   |      0  ",
12        "   |     -|-  ",
13        "   |      / \\" ,

```

¹en.wikipedia.org/wiki/Modular_programming

²sv.wikipedia.org/wiki/Idiom_%28programmering%29

```
14      " |      ",  
15      " |      ",  
16      " ===== RIP :(";  
17  
18  private static String renderHangman(int n){  
19      StringBuilder result = new StringBuilder();  
20      for (int i = 0; i < n; i++){  
21          result.append(hangman[i]);  
22          if (i < n - 1) {  
23              result.append("\n");  
24          }  
25      }  
26      return result.toString;  
27  }  
28  
29  private static String hideSecret(String secret,  
30                                  Set<Character> found){  
31      String result = "";  
32      for (int i = 0; i < secret.length(); i++) {  
33          if (found.contains(secret.charAt(i))) {  
34              result += secret.charAt(i);  
35          } else {  
36              result += '_';  
37          }  
38      }  
39      return result;  
40  }  
41  
42  private static boolean foundAll(String secret,  
43                                   Set<Character> found){  
44      boolean foundMissing = false;  
45      int i = 0;  
46      while (i < secret.length() && !foundMissing) {  
47          foundMissing = !found.contains(secret.charAt(i));  
48          i++;  
49      }  
50      return !foundMissing;  
51  }  
52  
53  private static char makeGuess(){  
54      Scanner scan = new Scanner(System.in);  
55      String guess = "";  
56      do {  
57          System.out.println("Gissa ett tecken: ");  
58          guess = scan.next();  
59      } while (guess.length() != 1);
```

```
60         return Character.toLowerCase(guess.charAt(0));
61     }
62
63     public static String download(String address, String coding){
64         String result = "lackalänga";
65         try {
66             URL url = new URL(address);
67             ArrayList<String> words = new ArrayList<String>();
68             Scanner scan = new Scanner(url.openStream(), coding);
69             while (scan.hasNext()) {
70                 words.add(scan.next());
71             }
72             int rnd = (int) (Math.random() * words.size());
73             result = words.get(rnd);
74         } catch (Exception e) {
75             System.out.println("Error: " + e);
76         }
77         return result;
78     }
79
80     public static void play(String secret){
81         Set<Character> found = new HashSet<Character>();
82         int bad = 0;
83         boolean won = false;
84         while (bad < hangman.length && !won){
85             System.out.println(renderHangman(bad));
86             System.out.print("\nFelgissningar: " + bad + "\t");
87             System.out.println(hideSecret(secret, found));
88             char guess = makeGuess();
89             if (secret.indexOf(guess) >= 0) {
90                 found.add(guess);
91             } else {
92                 bad++;
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    public static void main(String[] args){
```

```

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

Riktlinjer och tips för översättningen:

1. Skriv Scala-koden med en texteditor i en fil som heter `hangman1.scala` och kompilera med `scalac hangman1.scala` i terminalen; använd alltså *inte* en IDE, så som Eclipse eller IntelliJ, utan en ”vanlig” texteditor, t.ex. gedit.
2. Översätt i denna första deluppgift rad för rad så likt den ursprungliga Java-kodens utseende (syntax) som möjligt, med så få ändringar som möjligt. Du ska alltså ha kvar dessa Scalaovanligheter, även om det inte alls blir som man brukar skriva i Scala:
 - (a) långa indrag,
 - (b) onödiga semikolon,
 - (c) onödiga () ,
 - (d) onödiga {} ,
 - (e) onödiga `System.out`, och
 - (f) onödiga `return`.
3. Försök också i denna deluppgift göra så att betydelsen (semantiken) så långt som möjligt motsvarar den i Java, t.ex. genom att använda `var` överallt, även där man i Scala normalt använder `val`.
4. En Javaklass med bara statiska medlemmar motsvara ett singeltonobjekt i Scala, alltså en `object`-deklaration innehållande ”vanliga” medlemmar.
5. För att tydliggöra att du använder Javas `Set` och `HashSet` i din Scala-kod, använd följande import-satser i `hangman1.scala`, som därmed döper om dina importerade namn och gör så att de inte krockar med Scalas inbyggda `Set`. Denna form av import går inte att göra i Java.

```

import java.util.{Set => JSet};
import java.util.{HashSet => JHashSet};
```

6. Javas `i++` fungerar inte i Scala; man får istället skriva `i += 1` eller mindre vanliga `i = i + 1`.
7. Typparametrar i Java skrivs inom `<>` medan Scalas syntax för typparametrar använder `[]`.
8. Till skillnad från Java så har Scalas metoddeklarationer ett tilldelnings-tecken `=` efter returtypen, före kroppen.
9. Du kan ladda ner Java-koden till Hangman-klassen nedan från kursens

repo³. I samma bibliotek ligger även lösningarna till översättningen i Scala, men kolla *inte* på dessa förrän du gjort klart översättningarna och fått dem att kompilera och köra felfritt! Tanken är att du ska träna på att läsa felmeddelande från kompilatorn och åtgärda dem i en upprepad kompilera-testa-rätta-cykel.

- b) Skapa en ny fil `hangman2.scala` som till att börja med innehåller en kopia av din direkt-översatta Java-kod från föregående deluppgift. Omforma koden så att den blir mer som man brukar skriva i Scala, alltså mer Scala-idiomisk. Försök förenkla och förkorta så mycket du kan utan att göra avkall på läsbarheten.

Tips och riktlinjer:

1. Kalla Scala-objektet för `hangman`. När man använder ett Scalaobjekt som en modul (alltså en samling funktioner i en gemensam, avgränsad namnrymd) har man gärna liten begynnelsebokstav, i likhet med konventionen för paketnamn. Ett paket är ju också en slags modul och med en namngivningskonvention som är gemensam kan man senare, utan att behöva ändra koden som använder modulen, ändra från ett singelobjektet till ett paket och vice versa om man så önskar.
2. Gör alla metoder publiskt tillgängliga och låt även strängvektorn `hangman` vara publiskt tillgänglig. Deklarera `hangman` som en **val** och konstruera den med `Vector`. Eftersom `Vector` är oföränderlig och man inte kan ärva från singelobjekt och `hangman` är deklarerad med **val** finns inga speciella risker med att göra den konstanta vektorn publik om vi inte har något emot att annan kod kan läsa (och eventuellt göra sig beroende av) vår hänggubbetext.
3. I metoden `renderHangman` använd `take` och `mkString`.
4. I metoden `hideSecret` använd `map` i stället för en **for**-sats.
5. Det går att ersätta metoden `findAll` med det kärnfulla uttrycket `(secret forall found)` där `secret` är en sträng och `found` är en mängd av tecken (undersök gärna i REPL hur detta fungerar). Skippa därför den metoden helt och använd det kortare uttrycket direkt.
6. I metoden `makeGuess`, i stället för `Scanner`, använd `scala.io.StdIn.readLine`.
7. Om du vill träna på att använda rekursion i stället för imperativa loopar: Gör metoden `makeGuess` rekursiv i stället för att använda **do-while**.
8. I metoden `download`, i stället för `java.net.URL` och `java.util.ArrayList`, använd `scala.io.Source.fromURL(address, coding).getLines.toVector` och gör en lokal import av `scala.io.Source.fromURL` överst i det block där den används. Det går inte att ha lokala **import**-satser i Java.
9. Låt metoden `download` returnera en `Option[String]` som i fallet att nedladdningen misslyckas returnerar `None`.
10. I metoden `download`, i stället för **try-catch** använd `scala.util.Try` och dess smidiga metoder `recover` och `toOption`.
11. Om du vill träna på att använda rekursion i stället för imperativa loopar: Använd, i stället för **while**-satsen i metoden `play`, en lokal rekursiv

³github.com/lunduniversity/introprog/blob/master/compendium/examples/scalajava/Hangman.java

funktion med denna signatur:

```
def loop(found: Set[Char], bad: Int): (Int, Boolean)
```

Funktionen `loop` returnerar en 2-tupel med antalet felgissningar och **true** om man hittat alla bokstäver eller **false** om man blev hängd.

Uppgift 2. Översätta mellan klasser i Scala och klasser i Java. Klassen `Point` nedan är en modell av en punkt som kan sparas på begäran i en lista. Listan är privat för kompanjonsobjektet och kan skrivas ut med en metod `showSaved`. I koden används en `ArrayBuffer`, men i framtiden vill man, vid behov, kunna ändra från `ArrayBuffer` till en annan sekvenssamplingsimplementation, t.ex. `ListBuffer`, som uppfyller egenskaperna hos supertypen `Buffer`, men har andra prestandaegenskaper för olika operationer. Därför är attributet `saved` i kompanjonsobjektet deklarerat med den mer generella typen.

```

1 class Point(val x: Int, val y: Int, save: Boolean = false) {
2   import Point._
3
4   if (save) saved.prepend(this)
5
6   def this() = this(0, 0)
7
8   def distanceTo(that: Point) = distanceBetween(this, that)
9
10  override def toString = s"Point($x, $y)"
11 }
12
13 object Point {
14   import scala.collection.mutable.{ArrayBuffer, Buffer}
15
16   private val saved: Buffer[Point] = ArrayBuffer.empty
17
18   def distanceBetween(p1: Point, p2: Point) =
19     math.hypot(p1.x - p2.x, p1.y - p2.y)
20
21   def showSaved: Unit =
22     println(saved.mkString("Saved: ", ", ", "\n"))
23 }
```

- a) Översätt klassen `Point` ovan från Scala till Java. Vi ska i nästa deluppgift kompilera både Scala-programmet ovan och ditt motsvarande Java-program i terminalen och testa i REPL att klasserna har motsvarande funktionalitet.

Tips och riktslinjer:

1. För att namnen inte ska krocka i våra kommande tester, kalla Javatypen för `JPoint`.
2. Istället för Scalas `ArrayBuffer` och `Buffer`, använd Javas `ArrayList` och `List` som båda ligger i paketet `java.util`.

3. Undersök dokumentationen för `java.util.List` för att hitta en motsvarighet till `prepend` för att lägga till i början av listan.
 4. I stället för default-argumentet i Scalas primärkonstruktor, använd en extra Java-konstruktor.
 5. Det finns inga singelobjekt och inga kompanionsobjekt i Java; istället kan man använda statiska klassmedlemmar. Placera kompanionsobjektets medlemmars motsvarigheter *innuti* Java-klassen och gör dem till **static**-medlemmar.
 6. Kod i klasskroppen i Scalaklassen, så som if-satsen på rad 4, placeras i lämplig konstruktor i Javaklassen.
 7. Utskrifter med `print` och `println` behöver i Java föregås av `System.out`.
 8. Det finns inget nyckelord **override** i Java, men en s.k. annotering som ger samma kompilatorhjälp. Den skrivas med ett snabel-a och stor begynnelselbokstav, så här: `@Override` före metoddeklarationen.
 9. I Java används konventionen att börja getter-metoder med ordet `get`, t.ex. `getX()`.
 10. Det finns ingen motsvarighet till `mkString` för `List` så du behöver själv gå igenom listan och hämta elementreferenser för utskrift med en `for`-loop. Notera att efter sista elementet ska radbrytning göras i utskriften och att inget komma ska skrivas ut efter sista elementet.
 11. I Java behövs en ny **import**-deklaration om man vill importera ännu en typ från samma paket. Man kan även i Java använda asterisk `*`, (motsvarande `_` i Scala), för att importera allt i ett paket, men då får man med alla möjliga namn och det vill man kanske inte.
 12. Metoder i Java slutar med `()` om de saknar parametrar.
 13. Alla satser i Java slutar med lättglömda semikolon. (Efter att man i skrivit mycket Javakod och växlar till Scalakod är det svårt att vänja sig av med att skriva semikolon...)
- b) Starta REPL i samma bibliotek som du kompilerat kodfilerna. Testa så att klasserna `Point` och `JPoint` beter sig på samma vis enligt nedan. Skriv även testkod i REPL för att avläsa de attributvärden som har getters och undersök att allt funkar som det ska.

```
$ scalac Point.scala
$ javac JPoint.java
$ scala
scala> val (p, jp) = (new Point, new JPoint)
scala> p.distanceTo(new Point(3, 4))
scala> Point.showSaved
scala> jp.distanceTo(new JPoint(3, 4))
scala> JPoint.showSaved
scala> for (i <- 1 to 10) { new Point(i, i, true) }
scala> Point.showSaved
scala> for (i <- 1 to 10) { new JPoint(i, i, true) }
scala> JPoint.showSaved
```

- c) Översätt nedan Javaklass `JPerson` till en **case class** `Person` i Scala med motsvarande funktionalitet.

```
1  public class JPerson {  
2      private final String name;  
3      private final int age;  
4  
5      public JPerson(final String name, final int age){  
6          this.name = name;  
7          this.age = age;  
8      }  
9  
10     public JPerson(final String name){  
11         this(name, 0);  
12     }  
13  
14     public String getName() {  
15         return name;  
16     }  
17  
18     public int getAge() {  
19         return age;  
20     }  
21  
22     public boolean canEqual(Object other) {  
23         return (other instanceof JPerson);  
24     }  
25  
26     @Override public boolean equals(Object other){  
27         boolean result = false;  
28         if (other instanceof JPerson) {  
29             JPerson that = (JPerson) other;  
30             result = that.canEqual(this) &&  
31                 this.getName() == that.getName() &&  
32                 this.getAge() == that.getAge();  
33         }  
34         return result;  
35     }  
36  
37     @Override public int hashCode() {  
38         return name.hashCode() * 41 + age;  
39     }  
40  
41     @Override public String toString() {  
42         return "JPerson(" + name + ", " + age + ")";  
43     }  
44 }
```

-  d) Undersök i REPL vilken funktionalitet i Scala-case-klassen Person som in-

te är implementerad i Java-klassen `JPerson` ovan. Skriv upp namnen på några av case-klassens extra metoder samt deras signatur genom att för en `Person`-instans, och för kompansjonsobjektet `Person`, trycka på TAB-tangenten. Prova några av de extra metoderna i REPL och förklara vad de gör.

```

1 scala> val p = Person("Björn", 49)
2 scala> p.      // tryck TAB en gång
3 scala> Person. // tryck TAB en gång
4 scala> p.copy // tryck TAB en gång
5 scala> p.copy()
6 scala> p.copy(age = p.age + 1)
7 scala> Person.unapply(p)
```

Uppgift 3. Auto(un)boxing. I JVM måste typparametern för generiska klasser vara av referensstyp. I Scala löser kompilatorn detta åt oss så att vi ändå kan ha t.ex. `Int` som argument till en typparameter i Scala, medan man i Java *inte* direkt kan ha den primitiva typen `int` som typparameter till t.ex. `ArrayList`.

I Java och i den underliggande plattformen JVM används s.k. wrapperklasser för att lösa detta, t.ex. genom wrapper-klassen `Integer` som boxar den primitiva typen `int`. Java-kompilatorn har stöd för att automatiskt packa in värden av primitiv typ i sådana wrapper-klasser för att skapa referensstyper och kan även automatiskt packa upp dem.

- a) Studera hur Scala-kompilatorn låter oss arbeta med en `Cell[Int]` även om det underliggande JVM:ens körtidstyp (eng. *runtime type*) är en wrapperklass. Man kan se JVM-körtidstypen med metoderna `getClass` och `gettypeName` enligt nedan.

```

1 scala> class Cell[T](var value: T){
2     val typeName: String = value.getClass.getTypeName
3     override def toString = "Cell[" + typeName + "](" + value + ")"
4 }
5 scala> val c = new Cell[Int](42)
6 scala> c.value.getClass.getTypeName
```

- b) Vad är körtidstypen för `c.value` ovan? Förklara hur det kan komma sig trots att vi deklarerade med typargumentet `Int`?
- c) Studera dokumentationen för `java.lang.Integer`⁴ och testa i REPL några av *klassmetoderna* (de som är `static` och därmed kan anropas med punktnotation direkt på klassens namn utan `new`) och några av *instansmetoderna* (de som inte är `static`).

```

1 scala> Integer. //tryck TAB
2 scala> Integer.
3 scala> Integer.toBinaryString(42)
4 scala> Integer.valueOf(42)
5 scala> val i = new Integer(42)
6 scala> i. // tryck TAB
7 scala> i.toString
8 scala> i.compareTo // tryck TAB 2 gånger
```

⁴docs.oracle.com/javase/8/docs/api/java/lang/Integer.html

```
9  scala> i.compareTo(Integer.valueOf(42))
10 scala> i.compareTo(42) // varför fungerar detta?
```

-  d) Enligt dokumentationen⁵ tar instansmetoden `compareTo` i klassen `Integer` en `Integer` som parameter. Hur kan det då komma sig att sista raden ovan fungerar med en `Int`?
- e) Studera nedan Java-program och beskriv vad som kommer att skrivas ut *innan* du kompilerar och testkör.

```
1 import java.util.ArrayList;
2
3 public class Autoboxing {
4     public static void main(String[] args) {
5         ArrayList<Integer> xs = new ArrayList<Integer>();
6         for (int i = 0; i < 42; i++) {
7             xs.add(new Integer(i));
8         }
9         for (Integer x: xs) {
10            int y = x.intValue() * 10;
11            System.out.print(y + " ");
12        }
13        int pos = xs.size();
14        xs.add(pos, new Integer(0));
15        System.out.println("\n\n[0]: " + xs.get(0).intValue());
16        System.out.println("[ " + pos + "]: " + xs.get(pos));
17        if (xs.get(0) == xs.get(pos)) {
18            System.out.println("EQUAL");
19        } else {
20            System.out.println("NOT EQUAL");
21        }
22    }
23 }
```

- f) Ändra i programmet ovan så att autoboxing och autounboxing utnyttjas på alla ställen där så är möjligt. Utnyttja även att `toString`-metoden på `Integer` ger samma stränrepresentation som `int` vid utskrift. Fixa också så att du undviker *fallgropen* att i Java jämföra med referenslikhet i stället för att använda `equals`. Testa så att allt fungerar som det borde efter dina ändringar.
-  g) Antag att du råkar skriva `xs.add(0, pos)` på rad 14 i ditt program från föregående uppgift. Förklara hur autoboxingen stjälper dig i en *fallgrop* då.
-  h) Med ledning av de båda tidigare deluppgifterna: sammanfatta de två nämnda fallgroparna med autoboxing i Java i två generella punkter, så att du har nytt av att memorera dem inför din framtida Javakodning.

⁵docs.oracle.com/javase/8/docs/api/java/lang/Integer.html#compareTo-java.lang.Integer-

Uppgift 4. *JavaConverters*. Med `import scala.collection.JavaConverters._` får man i sina Scalaprogram tillgång till de smidiga metoderna `asJava` och `asScala` som översätter mellan motsvarande samlingar i resp språks standardbibliotek. Kör nedan i REPL och gör efterföljande deluppgifter.

```

1  scala> val sv = Vector(1,2,3)
2  scala> val ss = Set('a','b','c')
3  scala> val sm = Map("gurka" -> 42, "tomat" -> 0)
4  scala> val ja = new java.util.ArrayList[Int]
5  scala> ja.add(42)
6  scala> val js = new java.util.HashSet[Char]
7  scala> js.add('a')
8  scala> import scala.collection.JavaConverters._
```

- a) Till vilka typer konverteras Scalasamlingarna `Vector[Int]`, `Set[Char]` och `Map[String, Int]` om du anropar metoden `asJava` på dessa?
- b) Till vilka typer konverteras Javasamlingarna `ArrayList[Int]` och `HashSet[Char]` om du anropar metoden `asScala` på dessa? Blir det föränderliga eller oföränderliga motsvarigheter?
- c) Vad får resultatet för typ om du kör `toSet` på en samling av typen `mutable.Set`?
- d) Undersök hur du kan efter att du gjort `sm.asJava.asScala` anropa ytterligare en metod för att få tillbaka en oföränderlig `immutable.Map`.
- e) Läs mer i dokumentationen om `JavaConverters`⁶ och prova några fler konverteringar.

12.1.2 Extrauppgifter

Uppgift 5. Översätt nedan kod från Java till Scala. Skriv koden i en fil som heter `showInt.scala` och kalla Scala-objektet med `main`-metoden för `showInt`. Läs tipsen som följer efter koden innan du börjar.

```

1  import java.util.Scanner;
2
3  public class JShowInt {
4      private static Scanner scan = new Scanner(System.in);
5
6      public static void show(Object obj) {
7          System.out.println(obj);
8      }
9
10     public static void show(Object obj, String msg) {
11         System.out.println(msg + obj);
12     }
```

⁶docs.scala-lang.org/overviews/collections/conversions-between-java-and-scala-collections.html

```
13
14     public static String repeatChar(char ch, int n) {
15         StringBuilder sb = new StringBuilder();
16         for (int i = 0; i < n; i++) {
17             sb.append(ch);
18         }
19         return sb.toString();
20     }
21
22     public static String readLine(String prompt) {
23         System.out.print(prompt);
24         return scan.nextLine();
25     }
26
27     public static void showInt(int i) {
28         int leading = Integer.numberOfLeadingZeros(i);
29         String binaryString =
30             repeatChar('0', leading) + Integer.toBinaryString(i);
31         show(i, "Heltal: ");
32         show((char) i, "Tecken: ");
33         show(binaryString, "Binärt: ");
34         show(Integer.toHexString(i), "Hex : ");
35         show(Integer.toOctalString(i), "Oktalt: ");
36     }
37
38     public static void loop() {
39         boolean hasExploded = false;
40         while (!hasExploded) {
41             try {
42                 String s = readLine("Heltal annars pang: ");
43                 showInt(Integer.parseInt(s));
44             } catch (Throwable e){
45                 show(e);
46                 hasExploded = true;
47             }
48         }
49         show("PANG!");
50     }
51
52     public static void main(String[] args){
53         if (args.length == 0) {
54             loop();
55         } else {
56             for (String arg: args) {
57                 showInt(Integer.parseInt(arg));
58                 System.out.println();
59             }
60         }
61     }
62 }
```

```

59         }
60     }
61 }
62 }
```

Tips:

- En Javaklass med bara statiska medlemmar motsvaras av ett singeltonobjekt i Scala, alltså en **object**-deklaration. Scala har därför inte nyckelordet **static**.
- Typen **Object** i Java motsvaras av Scalas **Any**.
- Du kan använda Scalas möjlighet med default-argument (som saknas i Java) för att bara definiera en enda show-metod med en tom sträng som default msg-argument.
- I Scala har objekt av typen **Char** en metod **def * (n: Int): String** som skapar en sträng med tecknet repeterat **n** gånger. Men du kan ju välja att ändå implementera metoden **repeatChar** med **StringBuilder** som nedan om du vill träna på att översätta en **for**-loop från Java till Scala.
- I stället för **Scanner.nextLine** kan använda **scala.io.StdIn.readLine** som tar en prompt som parameter, men du kan också använda **Scanner** i Scala om du vill träna på det.
- I Java *måste* man använda nyckelordet **return** om metoden inte är en **void**-metod, medan man i Scala faktiskt *får* använda **return** även om man brukar undvika det och i stället utnyttja att satser i Scala också är uttryck.

Kompilera din Scala-kod och kör i terminalen och testa så att allt funkar. Vill du även kompilera Java-koden så finns den i kursens repo i filen
[compendium/examples/scalajava/JShowInt.java](#)

Uppgift 6. Studera och prova denna fallgrop med innehållslikhet:

github.com/bjornregnell/lth-eda016-2015/blob/master/lectures/examples/eclipse-ws/lecture-examples/src/week10/generics/TestPitfall3.java

12.1.3 Fördjupningsuppgifter

Uppgift 7. Studera fallgropar för hur man skriver en **equals**-metod i Java här:  www.artima.com/lejava/articles/equality.html och jämför med det fullständiga receptet för hur man skriver en välfungerande **equals** och **hashcode** i Scala här: www.artima.com/pins1ed/object-equality.html

- Vilka skillnader och likheter finns vid överskuggning av **equals** i Java respektive Scala, som ska ge en fungerande innehållstest för en hierarki med bastyper och subtyper?
- Vilka fallgropar är gemensamma för Java och Scala?

12.2 Grupplaboration: lthopoly-team

Mål

- Förstå hur autoboxing fungerar i Java
- Förstå vad statiska metoder och attribut innehåller
- Förstå skillnaden mellan primitiva typer och objekt i listor
- Känna till hur en byter mellan ArrayLists och Array
- Känna till hur en läser in från fil
- Känna till syntaxen för en for-sats i Java
- Förstå hur en skickar information mellan scala och Java

Förberedelser

- Gör övning scalajava i avsnitt 12.1.
- Läs igenom Bakgrunden, kodstrukturen och alla kommentarer i kodskelettet.
- Diskutera i din samarbetsgrupp hur ni ska dela upp koden mellan er i flera olika delar, som ni kan arbeta med var för sig. En sådan del kan vara en klass, en trait, ett objekt, ett paket, eller en funktion.
- Varje del ska ha en *huvudansvarig* individ.
- Arbetsfördelningen ska vara någorlunda jämt fördelad mellan gruppmedlemmarna.
- När ni redovisar er lösning ska ni börja med att redogöra för handledaren hur ni delat upp koden och vem som är huvudansvarig för vad.
- Den som är huvudansvarig för en viss del redovisar den delen.
- Grupplaborationer görs i huvudsak som hemuppgift. Salstiden används primärt för redovisning.

12.2.1 Bakgrund

I denna labb skall ni tillverka ett spel kallat Lthopoly, en variant av det välkända brädspelet Monopol med några simplifieringar. Varje spelare börjar med en summa pengar och förflyttar sig längs spelplanen. I början av en runda slår den aktiva spelaren en tärning och deras pjäs flyttas det antal steg som tärningen visar. Beroende på vilken av de tre möjliga ruttyperna spelaren hamnar på sker olika saker:

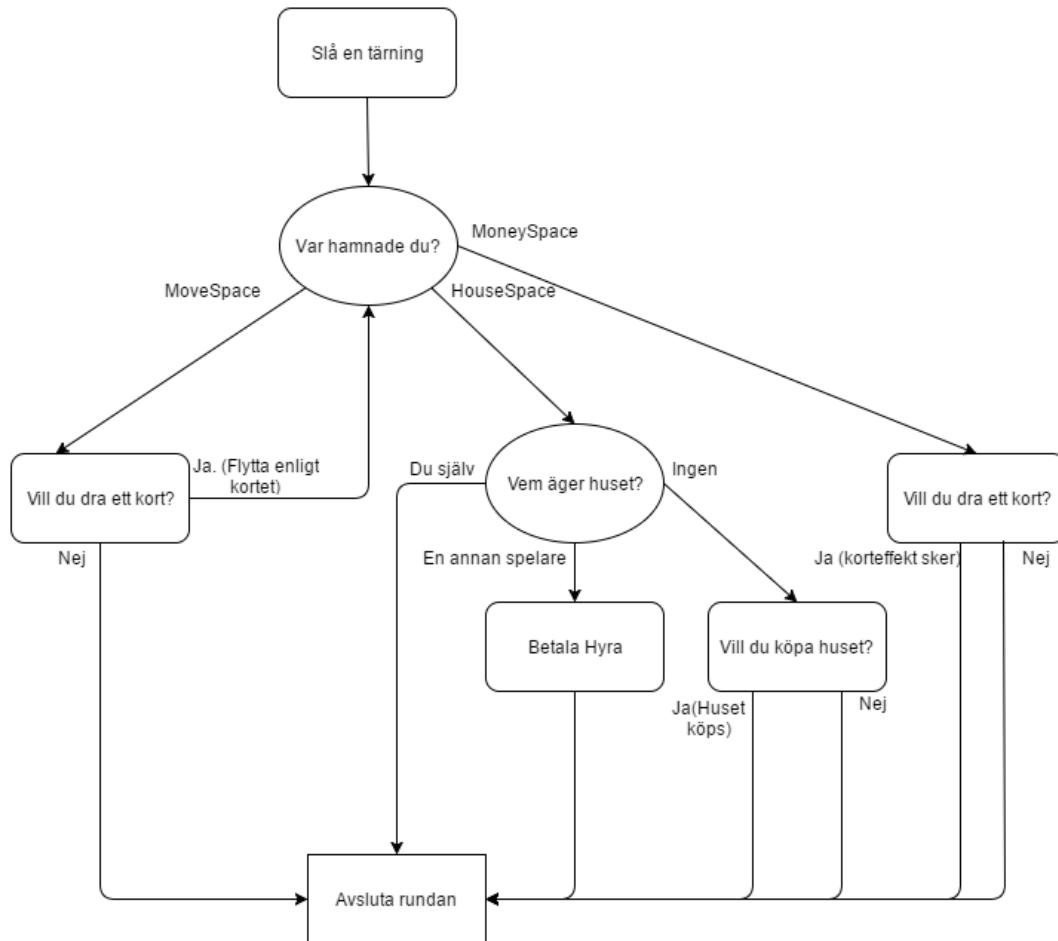
- MoveSpace: Om en spelare hamnar på denna ruta får den dra ett MoveCard, varpå spelaren förflyttas antingen framåt eller bakåt det antal steg som kortet anger.
- MoneySpace: Om en spelare hamnar på denna ruta får den dra ett MoneyCard, varpå spelaren antingen förlorar eller vinner pengar enligt beskrivningen på kortet.
- HouseSpace: Dessa rutor kan ägas utav spelare. Om en spelare förflyttas till denna rutan och ingen annan spelare äger den ges möjligheten att

köpa den. Äger en annan spelare rutan blir den nuvarande spelaren tvungen att betala hyra. Hyran är samma som inköpspriset.

Spelet ska uppfylla följande krav:

- Varje spelare måste alltid börja sin runda med att slå en tärning innan den utför någon annan spelhandling, d.v.s. någon annan handling som påverkar spelets tillstånd (exempelvis kan en alltid visa spelplanen eller avsluta spelet).
- Om någon spelare har mindre än 0 SEK kvar skall spelet sluta.
- Om någon spelare hamnar på en husruta som ägs av en annan spelare måste denne betala ägaren husets hyra i SEK. Om ingen äger huset ges spelaren möjlighet att köpa det för ett belopp motsvarande en hyra (förutsatt att den har råd).
- Om en spelare hamnar på en MoveSpace eller ett MoneySpace får spelaren möjligheten att dra ett kort. För MoveCard innebär detta en förflyttning (bakåt eller framåt) medan för MoneyCard en minskning eller ökning av pengar.
- Spelplanen skall vara cyklisk, d.v.s. att rutan direkt efter sista rutan är den första rutan på spelplanen.

Nedan visas ett förenklat flödesdiagram för en spelrunda.

**Figur 12.1:** Flödesdiagram för en spelrunda

Spelet avslutas när någon spelare får slut på pengar och då vinner spelaren med mest pengar.

12.2.2 Kodstruktur

Klassen Player representerar en spelare. Varje spelare måste känna till sitt saldo och sin position på brädet.

```

class Player

    /** Creates a new player*/
    public Player(String name, int money, int pos);

    /** Returns the players money*/
    public int getMoney() ;

    /** Adjusts the players money*/
    public void adjustMoney(int money);

    /** Returns the players position*/
    public int getPosition();

    /** Returns a string representation of the player*/
  
```

```

public String toString();

/** Sets the players position/
public void setPosition(int pos) ;

```

MoneyCard och MoveCard är två liknande klasser som representerar de kort som spelare kan dra. MoneyCard ska spara information om hur mycket pengar som ska tillföras respektive borttagas. MoveCard ska spara information om hur långt en spelare skall förflytta sig när kortet dras. Båda korten skall även innehålla en beskrivning utav varför detta händer. Informationen för dessa kort läses in från textfilerna moneycards.txt och movecards.txt.

```

class MoneyCard

/**Creates a new MoneyCard*/
public MoneyCard( String description, int money);

/**Returns the cards money adjustment value*/
public int getMoney();

/**Returns the description of why the money is adjusted*/
public String getDescription();

```

```

class MoveCard

/**Creates a new MoveCard*/
public MoveCard( String description, int positionAdjustment) ;

/**Returns the position adjustment*/
public int getPositionAdjustment();

/**Returns the description of why the position is adjusted*/
public String getDescription();

```

Klasserna MoneySpace och MoveSpace ska ärva från den abstrakta klassen BoardSpace. MoneySpace och MoveSpace ska använda sig av en array med respektive Cards.

```

class BoardSpace

/** Returns a array of int describing possible
 * game actions available while on this space*/
public abstract int[] getPossibleActions(GameBoard board);

/** Executes a game action available while on this space*/
public abstract void action(GameBoard board, int action);

/** Returns a string representation of this BoardSpace*/
public abstract String toString();

```

```

class MoneySpace

/** Returns an array of possible game actions permitted by this space */

```

```

public int[] getPossibleActions(GameBoard board);

/** Performs a MoneySpace-related action. */
public void action(GameBoard board, int action);

/** Returns a string representation of the MoneySpace */
public String toString();

```

class MoveSpace

```

/** Returns an array of possible game actions permitted by this space */
public int[] getPossibleActions(GameBoard board);

/** Performs a MoveSpace-related action. */
public void action(GameBoard board, int action);

/** Returns a string representation of the MoveSpace */
public String toString();

```

class HouseSpace

```

/** Returns an array of possible game actions permitted by this space */
public int[] getPossibleActions(GameBoard board);

/** Performs a HouseSpace-related action. */
public void action(GameBoard board, int action);

/** Returns a string representation of the HouseSpace with
 * the format "HouseName [Owner] (Rent)" */
public String toString();

```

Spelbrädets utseende bestäms av textfilen board.txt. Denna textfil specificerar både i vilken ordning de olika sorters rutorna kommer men även namn och hyra på HouseSpace. Klassen DocumentParser hanterar inläsning från fil och ska kunna läsa in MoneyCards, MoveCards samt hela spelplanen.

class DocumentParser

```

/**Returns a ArrayList of Boardspaces loaded from a file*/
public static ArrayList<BoardSpace> getBoard();

/**Returns a array of MoneyCards loaded from file*/
public static MoneyCard[] getMoneyCards();

/**Returns a array of MoveCards loaded from file*/
public static MoveCard[] getMoveCards();

```

Klassen GameBoard håller koll på spelets tillstånd. GameBoard kombinerar ovannämnda klasser för att bygga upp spelet. GameBoard har en metod getPossibleActions() som returnerar en lista över alla möjliga spelarhandlingar för den nuvarande spelaren. Denna används av main-metoden för att be användaren välja nästa handling. Olika sorters spelarhandlingar representeras

av statiska int-variabler i klassen GameBoard. Vid val av handling matar användaren in handlingens siffrvärdet i konsolen. Varje handling motsvaras då alltid av samma inmatningsvärdet för användaren.

```
class GameBoard

/** Creates a new board ready to play */
public GameBoard(List<Player> players);

/** Returns an int array containing possible game actions.
 * A game action can be any of the static constants in
 * GameBoard*/
public int[] getPossibleActions() ;

/** Checks whether the game is over or not */
public boolean isGameOver();

/** Returns the player with the most money */
public Player getRichestPlayer();

/** Returns a list of all players */
public List<Player> getPlayers();

/** Returns a list of all BoardSpaces */
public List<BoardSpace> getBoardSpaces();

/** Performs an action for the current player */
public void doAction(int action);

/** Returns the currently active player */
public Player getCurrentPlayer();

/** Returns the boardspace corresponding to the position
 * of the current player. */
public BoardSpace getCurrentBoardSpace();

/** Moves the currently active player adjustments spaces forward.
 * Negative adjustment moves the player backwards*/
public void moveCurrentPlayer(int adjustment);

/** Returns an ArrayList<Integer> where each element contains the total
 * sum of all players' money at the end of a round.
 * E.g. list.get(0) is the total amount of money in the game after the
 * first round. */
public ArrayList<Integer> getStatistics();

/** String Representation of the GameBoard */
public String toString() ;
```

Den visuella representationen av spelet sker via konsolfönstret med hjälp av klassen TextUI. TextUI är en färdigskriven klass med metoder som gör det enkelt att skriva ut spelplanen och en logg av spelhistoriken i konsolfönstret. När addToLog() anropas sparar den sitt argument i en lista och hela listan skrivs ut varje gång konsolen uppdateras via updateConsole-metoden. Utöver loggen skriver updateConsole även ut ett statusfönster med kortfattad infor-

mation om varje spelare. Om spelaren väljer att visa spelplanen ska metoden printBoard() istället anropas. Alla utskrifter som sker under spelets gång ska gå via TextUI.

```
1 =====
2
3 Oskar slog en 2:a!
4 Oskar drog ett kort: Jädrans! Studiebidraget har sänkts. Förlora 40 SEK
5 Oskar har avslutat sin runda.
6 Nästa spelare: Jonas
7 Jonas slog en 3:a!
8 Jonas drog ett kort: Det lönade sig att leva på nudlar! Inkassera 50 SEK
9 Jonas har avslutat sin runda.
10 Nästa spelare: Valthor
11 Valthor slog en 5:a!
12 Grattis, Valthor är nu den stolta ägaren av V-Huset
13 Valthor har avslutat sin runda.
14 Nästa spelare: Oskar
15 Oskar slog en 2:a!
16 Namn-----Position-----Pengar-----
17 Oskar*           Moroten och piskan(40)      260
18 Jonas            ChansRuta                  350
19 Valthor          V-Huset [Valthor](45)     255
20 -----
21 Välj ett alternativ:
22
23   3. Köp ett hus
24   5. Avsluta din runda
25   8. Visa standardvyn
26   9. Visa spelplanen
27   0. Avsluta Lthopoly
28
29 =====
```

Figur 12.2: Utskrift av standardvyn

```

1 Rutans Namn [Ägare] (Pris/Hyra) (Spelare, Pengar)*
2 -----
3 Studiecentrum(20)
4 A-huset(25)
5 ChansRuta
6 ChansRuta (Jonas,350)
7 Moroten och piskan(40) (Oskar,260)
8 V-Huset [Valthor](45) (Valthor,255)
9 RiskRuta
10 ChansRuta
11 LED-Cafe(70)
12 F-Huset(75)
13 ChansRuta
14 RiskRuta
15 Ideet(80)
16 ChansRuta
17 E-huset(100)
18 RiskRuta

```

Figur 12.3: Utskrift av spelplanen**Specification TextUI**

```

/** Prints an ASCII plot of the total amount
of money in the game as a function of the turn index*/
def plotStatistics(x: Buffer[Int]): Unit

/** Appends the String s to the end of the UI's event log */
def addToLog(s: String): Unit

/** Reprints the current state of the UI using the given
GameBoard to print the status bar*/
def updateConsole(board: GameBoard): Unit

/** Asks the user to select an option from a list of options
*
* @param options an Array of tuples of the form (choice, description)
*                 where choice is the number the user should enter to select
*                 the choice represented by description,
*                 e.g. (0, "End Game") allows the user to input 0 to end
*                 the game.
* @return         the selected choice
*/
def promptForInput(options: Array[(Int, String)]): Int

/** Prints the entire GameBoard */
def printBoard(board: GameBoard): Unit

```

Simuleringen av spelet sker i main-metoden i klassen Main, och skall implementeras i Scala (i uppgift 6).

Specification Main

```

package lthopoly

import lthopoly._
import scala.collection.JavaConverters._

object Main {
    def main(args: Array[String]): Unit = {

    }

    /**
     * Retrieves all possible actions from GameBoard and joins them with
     * a corresponding description String into tuples.
     * The tuples are then sent to the promptForInput method in TextUI.
     *
     * @return the user's choice as given by promptForInput.
     */
    def getAction(board: GameBoard): Int = ???

}

```

12.2.3 Obligatoriska uppgifter

Uppgift 1. All information om olika kort och om spelplanens upplägg finns i textfiler som måste läsas in med hjälp av metoderna i klassen DocumentParser. Textfilerna moneycards.txt och movecards.txt innehåller information om de olika korten som finns. Varje rad innehåller en beskrivning för kortet följd av ett värde separerat med semikolon. Dessa sparas i en vektor vid inläsning och när en spelare hamnar på en MoveSpace eller MoneySpace dras ett slumpmässigt kort från vektorn. Vektorn av kort måste alltså skickas med till motsvarande ruta när rut-objektet skapas . Filen board.txt innehåller spelplanen, men behandlas först i uppgift 3.

För att skapa ett File-objekt med informationen från filerna kan följande kod användas:

```

File f = new File(DocumentParser.class.
    getResource("/moneycards.txt").getFile());

```

Därefter kan ni använda er av ett scanner-objekt som får tillgång till Filen för att läsa in en rad i taget. Se textfilerna moneycards.txt, movecards.txt och board.txt i mappen resources för förståelse för hur inläsningen bör gå till för korten.

Ni kan nyttja metoden `String.split(String delimiter)` för att dela en sträng i en array av fält, där delimiter är den avgränsade strängen som används vid uppdelningen.

- Implementera klassen MoveCard.

- b) Implementera klassen MoneyCard.
- c) Implementera metoderna getMoneyCards() och getMoveCards() i DocumentParser. Metoderna ska klara av att läsa in ett variabelt antal kort.
- d) Implementera klassen Player.

Tips: För att konvertera mellan ArrayLists och arrays kan ArrayLists .toArray-metod användas enligt följande:

```
ArrayList<MyClass> list = new ArrayList<MyClass>();
MyClass[] arr = list.toArray(new MyClass[]{});
```

Uppgift 2. I denna uppgift skall de tre olika subklasserna till BoardSpace implementeras. Tänk på att MoveSpace och MoneySpace behöver tillgång till respektive kortlekar. För att skriva metoderna action och getPossibleActions kommer ni behöva nyttja att klassen GameBoard har statiska konstanterna som representerar de olika spelarvalen.

- a) Implementera en klass för varje typ av spelruta.

Obs! Än så länge kommer logiken inte fungera då inga metoder är implementerade i BoardGame, det går trots detta bra att anropa metoderna utan kompileringsfel (i väntan på att de implementeras).

Uppgift 3. Nu är det dags att implementera getBoard() i klassen DocumentParser. I denna metod skall ni läsa in från filen board.txt och nyttja de metoder ni redan skrivit för att nu kunna skapa MoveSpaces och MoneySpaces. Radernas ordning i filen bestämmer deras ordning på spelplanen. Varje rad börjar antingen med orden "Move", "Money" eller "House". House-rader följs dessutom av husets hyra och dess namn separerat av semikolon. Baserat på radens första ord skall ett motsvarande objekt konstrueras och läggas till i en ArrayList<BoardSpace> som slutligen returneras.

- a) Implementera getBoard().

Fundera på

- Behöver flera objekt skapas av varje ruttyp?

Uppgift 4. Implementera alla metoder utom getStatistics() i GameBoard (se specifikationen).

Tips:

- Ni kan använda privata hjälpmetoder för att underlätta implementeringen.
- Metoden printStatistics i klassen TextUI tar en vektor av int-värden som inparameter, vilket är opassande då det underlättar att lagra pengahistoriken i en ArrayList (eftersom dess storlek inte är bestämd). Det är därför

lämpligt att skriva en metod som flyttar över samtliga Integer-objekt från `ArrayList<Integers>` till en vektor av primitiva int-värden. Detta fungerar trots att de har olika typer p.g.a. autoboxing.

- Tänk på att spelarna skall kunna gå runt spelplanen ett obegränsat antal gånger.
- Glöm inte att alla utskrifter skall gå via `TextUI`.
- Se flödesdiagrammet för att få en överblick för vilka actions som är tillåtna vid en given tidpunkt.

Fundera på

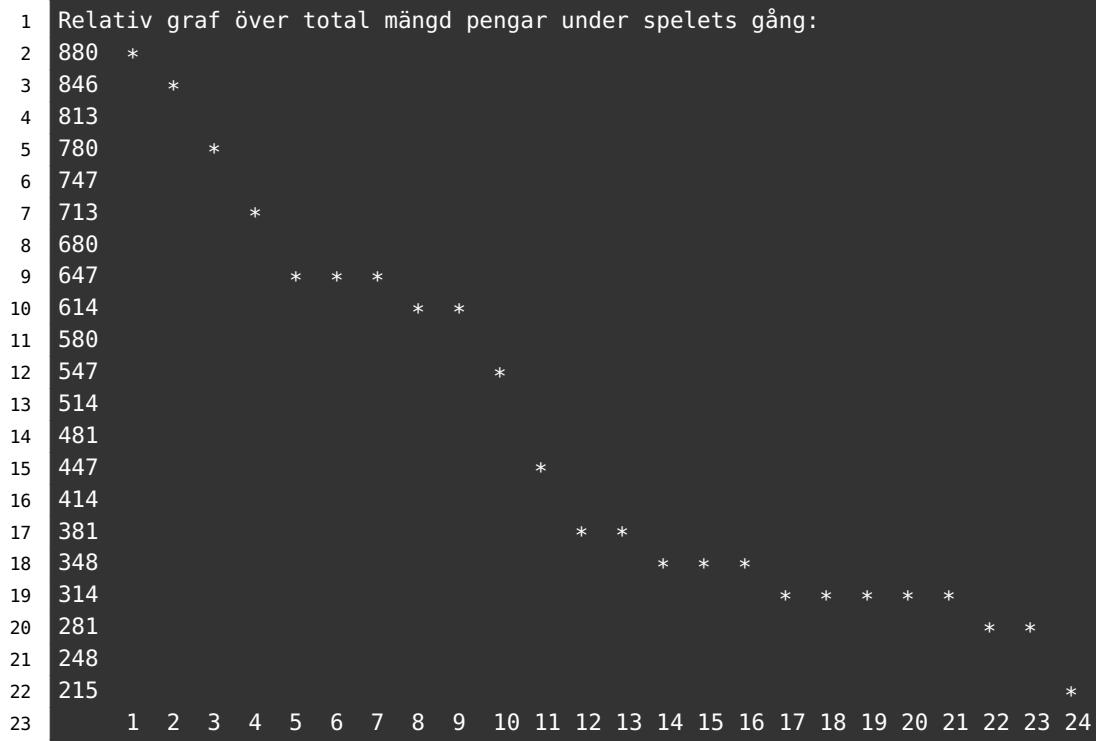
- Varför tar konstruktorn i `GameBoard` emot en `List<Player>` istället för en `ArrayList<Player>?`

Uppgift 5. Med spelplanen implementerad behövs en main-metod för att kunna starta spelet. I main-metoden skapas `GameBoard` samt alla `Spelare`. Spelet körs sedan som en loop där `GameBoard` tillfrågas vilka spelarhandlingar som är tillåtna för den nuvarande spelaren, erbjuder spelaren möjligheten att välja något av dessa alternativ, och matar sedan in spelarens val tillbaka till `GameBoard` som hanterar valet. `GameBoard` ska alltså hantera all spellogik internt. Spel-loopen skall köras tills dess att spelet är över enligt `GameBoard.isGameOver`.

- a) Implementera `getAction` i Scala. Metoden ska anropa `TextUI.promptForInput` med en lämplig lista av tupler för att begära input från användarna. Metoden skall nyttja de statiska variablerna från `GameBoard` för att ge en lämplig utskrift.
- b) Implementera main-metoden i Scala.

Uppgift 6. Vi skall nu lägga till möjligheten att se statistik över spelets monetära tillstånd. Metoden `getStatistics()` i `GameBoard` ska returnera en lista innehållande den totala summan av pengar i spelet i slutet av varje runda. Denna lista kan skickas vidare till metoden `TextUI.printStatistics` där den sedan skrivs ut i en vacker plot (se utskrift nedan).

- a) Implementera metoden `getStatistics()` i `GameBoard`.
- b) Utöka mainmetoden så att grafen skrivs ut efter spelets slut. Själva uppritringen sker med hjälp av den färdigskrivna metoden `plotStatistics` i `TextUI` som kräver en `Buffer[Int]` innehållande varje rundas totalsumma. Ett tips är att nyttja `scala.collection.JavaConverters` för att konvertera Javas datatyper till Scala.



Figur 12.4: Graf över spelets total mängd pengar som funktion av rundornas index

12.2.4 Frivilliga extrauppgifter

Uppgift 7. Utöka spelet med ny spelmekanik.

- Implementera funktionalitet för att varje spelare ska få extra pengar då den passerar första spelrutan.
- Implementera funktionalitet för att varje spelare som hamnar på en ruta de äger sedan tidigare har möjlighet att öka hyran för rutan ifall någon annan spelare skulle hamna på den.
- Implementera funktionalitet så att ägaren av ett hus måste betala en andel av dess värde varje gång de passerat första spelrutan..

Kapitel 13

Extra: design, api, trådar, webb

Begrepp som ingår i denna veckas studier:

- utvecklingsprocessen
- krav-design-implementation-test
- gränssnitt
- trait vs interface
- programmeringsgränssnitt (api)
- designexempel
- tråd
- jämlöpande exekvering
- icke-blockerande anrop
- callback
- `java.lang.Thread`
- `java.util.concurrent.atomic.AtomicInteger`
- `scala.concurrent.Future`
- kort om html+css+javascript+scala.js och webbprogrammering

13.1 Övning: threads

Mål

- Känna till vad en tråd är och kunna förklara begreppet jämlöpande exekvering.
- Känna till vad metoderna `run` och `start` gör i klassen `Thread`.
- Kunna skapa och starta en tråd med överskuggad `run`-metod.
- Kunna skapa ett enkelt program som från två trådar tävlar om att uppdatera en variabel och förklara varför beteendet kan bli oförutsägbart.
- Kunna använda en `Future` för att köra igång flera parallella beräkningar.
- Kunna registrera en callback på en `Future` med metoden `onComplete`.

Förberedelser

- Studera begreppen i kapitel 13.

13.1.1 Grunduppgifter

Uppgift 1. *Trådar:* Klassen `java.lang.Thread` används för att skapa **trådar** med jämlöpande exekvering (eng. *concurrent execution*). På så sätt kan man få olika koddelar att köra samtidigt.

Klassen `Thread` definierar en tom `run`-metod. Vill man att tråden ska göra något vettigt får man överskugga `run` med det man vill ska göras.

En tråd körs igång med metoden `start` och då anropas automatiskt `run`-metoden och tråden exekverar koden i `run` jämlöpande med övriga trådar. Om man anropar `run` direkt blir det *inte* jämlöpande exekvering.

- a) Skapa en tråd som gör något som tar lite tid och kör med `run` resp. `start`.

```

1 def zzz = { print("zzzzzz"); Thread.sleep(5000); println(" VAKEN!") }
2 zzz
3 val t2 = new Thread{ override def run = zzz }
4 t2.run
5 t2.run; println("Gomorron!")
6 t2.start; println("Gomorron!")
7 t2.start

```

- b) Vad händer om man anropar `start` mer än en gång på samma tråd?
- c) Skapa två trådar med överskuggade `run`-metoder och kör igång dem samtidigt enligt nedan. Vilken ordning skrivs hälsningarna ut efter rad 3 resp. rad 4 nedan? Förklara vad som händer.

```

1 val g = new Thread{ override def run = for (i <- 1 to 100) print("Gurka ") }
2 val t = new Thread{ override def run = for (i <- 1 to 100) print("Tomat ") }
3 g.run; t.run
4 g.start; t.start

```

- d) Använd `Thread.sleep` enligt nedan. Är beteendet helt förutsägbart (deterministiskt)? Förklara vad som händer. Du kan (om du kör Linux) avbryta REPL med `Ctrl+C`¹.

```

1 def ibland(block: => Unit) = new Thread {
2   override def run = while(true) { block; Thread.sleep(600) }
3 }.start
4 ibland(print("zzz ")); ibland(print("snark ")); ibland(println("hej!"))

```

Uppgift 2. *Jämlöpande variabeluppdatering.* Skriv klasserna `Bank` och `Kund` i en editor och klistica sedan in kodern i REPL.

```

class Bank {
  private var saldo = 0;
  def visaSaldo: Unit = println(s"saldo: $saldo")
  def sättIn: Unit = { saldo += 1 }
  def taUt: Unit    = { saldo -= 1 }
}

class Kund(bank: Bank) {
  def slösaSpara = {bank.taUt; Thread.sleep(1); bank.sättIn}
}

```

- a) Använd funktionen `ibland` från föregående uppgift och kör nedan rader i REPL. Resultatet av jämlöpande variabeluppdatering blir här heltokigt och leder till mycket upprörda bankkunder och -ägare. Förklara vad som händer.

```

1 val bank = new Bank
2 bank.visaSaldo
3 bank.sättIn
4 bank.visaSaldo
5 bank.taUt
6 bank.visaSaldo
7
8 val bamse = new Kund(bank)
9 val skutt = new Kund(bank)
10
11 bamse.slösaSpara
12 skutt.slösaSpara
13 bank.visaSaldo
14
15 def ofta(block: => Unit) = new Thread {
16   override def run = while(true) { block; Thread.sleep(1) }
17 }.start
18
19 ofta(bamse.slösaSpara); ofta(skutt.slösaSpara)
20
21 ibland(bank.visaSaldo)

```

¹stackoverflow.com/questions/6248884/can-i-stop-the-execution-of-an-infinite-loop-in-scala-repl

Uppgift 3. *Trådsäkra AtomicInteger.* Det finns stöd i JVM för att åstadkomma uppdateringar som inte kan avbrytas av andra trådar under pågående minnesskrivning. En operation som inte kan avbrytas kallas **atomär** (eng. *atomic*). Studera dokumentationen för `AtomicInteger`² och prova nedan kod. Förklara vad som händer.

Använd funktionerna ofta och ibland från tidigare uppgifter.

```
class SäkerBank {
    import java.util.concurrent.atomic.AtomicInteger
    private var saldo = new AtomicInteger
    def visaSaldo: Unit = println(s"saldo: ${saldo.get}")
    def sättIn: Unit = { saldo.incrementAndGet }
    def taUt: Unit   = { saldo.decrementAndGet }
}

class SäkerKund(bank: SäkerBank) {
    def slösaSpara = {bank.taUt; Thread.sleep(1); bank.sättIn}
}
```

```
1 val säkerBank = new SäkerBank
2 val farmor = new SäkerKund(säkerBank)
3 val vargen = new SäkerKund(säkerBank)
4
5 ofta(farmor.slösaSpara); ofta(vargen.slösaSpara)
6
7 ibland(säkerBank.visaSaldo)
```

Uppgift 4. Jämlöpande exekvering med `scala.concurrent.Future`. Att skapa och hålla reda på trådar kan bli ganska omständligt och knepigt att få rätt på. Med hjälp av `scala.concurrent.Future` kan man på ett enklare sätta skapa jämlöpande exekvering.

Bakgrund för kännedom: Med en Future skapas jämlöpande exekvering som ”under huven” använder ett ramverk som heter Akka³, skrivet i Scala och Java. Akka erbjuder automatisk multitrådning med s.k. trådpooler och möjliggör avancerad parallellprogrammering på en hög abstraktionsnivå, där man själv slipper skapa instanser av klassen Thread. I stället kan man helt enkelt placera sin kod inramat med `Future{ "körs parallellt" }` efter att man importerat det som behövs.

- a) För att skapa jämlöpande exekvering med Future behöver man först göra import enligt nedan; då skapas ett exekveringssammanhang med trådpooler redo för användning. Starta om REPL och studera felmeddelandet efter rad 1 nedan. Importera därefter enligt nedan. Vad har f för typ?

```
1 scala> concurrent.Future { Thread.sleep(1000); println("En sekund senare!") }
2 scala> import scala.concurrent...
3 scala> import ExecutionContext.Implicits.global
4 scala> val f = Future { Thread.sleep(1000); println("En sekund senare!") }
```

²docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicInteger.html

³<http://akka.io/>

- b) Skapa en procedur `printLater` enligt nedan som skriver ut argumentet efter slumpmässigt tid. Förklara vad som händer nedan.

```
1 scala> def printLater(a: Any): Unit =
2           Future { Thread.sleep((math.random * 10000).toInt); print(a + " ") }
3 scala> (1 to 42).foreach(i => printLater(i)); println("alla är igång!")
```

- c) Skapa enligt nedan en `Future` som räknar ut hur många siffror det är i ett väldigt stort tal. Med `onComplete` kan man ange vad som ska göras när den tunga beräkningen är färdig; detta kallas att ”registrera en callback”. Vilken returytpe har `big`? Hur många siffror har det stora talet? Vad har `r` för typ? Justera argumentet till `big` om du inte orkar vänta på resultatet...

```
1 scala> BigInt(10).pow(100)
2 scala> BigInt(10).pow(100).toString.size
3 scala> def big(n: Int) = Future { BigInt(n).pow(n).toString.size }
4 scala> big(1234567).onComplete{r => println(r + " siffror") }
```

- d) Den stora vinsten med `Future` är att man kan köra vidare under tiden, varför anropet av `Future` kallas **icke-blockerande** (eng. *non-blocking*). Det händer ibland att man ändå vill blockera exekveringen i väntan på ett resultat. Man kan då använda objektet `scala.concurrent.Await` och dess metod `result` enligt nedan. Använd `big` från föregående uppgift och gör en blockerande väntan på resultatet enligt nedan. Vad händer? Vad händer om du väntar för kort tid?

```
1 scala> import scala.concurrent.duration._
2 scala> Await.result(big(1234567), 20.seconds)
```

Uppgift 5. Använda `Future` för att göra flera saker samtidigt. I denna uppgift ska du ladda ner webbsidor parallellt med hjälp av `Future`, så att en nedladdning kan avslutas under tiden en annan dröjer.

- a) Koden för en minimal webbsida ser ut som nedan. Du kan beskåda sidan här: <http://fileadmin.cs.lth.se/pgk/mini.html> eller skriva in nedan kod i en fil som heter något som slutar på `.html` och öppna filen i din webbläsare.

```
<!DOCTYPE html>
<html>
<body>
  HELLO WORLD!
</body>
</html>
```

- b) För att simulera slöa webbservrar kan man ladda ner en sida via sajten <http://deelay.me/>. Ladda ner ovan sida med 2 sekunders födröjning: <http://deelay.me/2000/http://fileadmin.cs.lth.se/pgk/mini.html>
- c) Man kan ladda ner webbsidor med `scala.io.Source`. Vad händer nedan? Försök, med ledning av hur `delay` beräknas, uppskatta hur lång tid du måste

vänta i medeltal, i bästa fall, respektive värsta fall, innan du kan se första webbsidan i vektorn laddningar nedan?

```

1 scala> def ladda(url: String) = scala.io.Source.fromURL(url).getLines.toVector
2 scala> def slöladda(url: String) = {
3     val delay = (math.random * 1000 + 2000).toInt
4     val delaySite = s"http://deelay.me/$delay/"
5     ladda(delaySite+url)
6 }
7 scala> ladda("http://fileadmin.cs.lth.se/pgk/mini.html")
8 scala> def seg = slöladda("http://fileadmin.cs.lth.se/pgk/mini.html")
9 scala> val laddningar = Vector.fill(10)(seg)
10 scala> laddningar(0)

```

d) Innan vi kan köra igång en Future så måste vi, som visats i uppgift 4 importera den underliggande exekveringsmiljön som är redo att parallelisera ditt program i trådar utan att du själv måste skapa dem. Vad händer nedan?

```

1 scala> import scala.concurrent._
2 scala> import ExecutionContext.Implicits.global
3 scala> val f = Future{ seg }
4 scala> f    // kolla om den är klar annars prova igen senare
5 scala> f

```

e) Ladda indata utan att blockera (eng. *non-blocking input*). Förklara vad som händer nedan.

```

1 scala> val nonblock = Future{ Vector.fill(10)(seg) }
2 scala> nonblock // kolla igen senare om ej klar
3 scala> nonblock

```

f) Ladda indata separat i olika parallella trådar. Förklara vad som händer nedan. Kör uttrycket på rad 3 nedan upprepade gånger i snabb följd efter varandra med pil-upp+Enter i REPL.

```

1 scala> val para = Vector.fill(10)(Future{ seg })
2 scala> para
3 scala> para.map(_.isCompleted)
4 scala> para.map(_.isCompleted) // studera hur de blir färdiga en efter en
5 scala> para(0)

```

g) Registrera en callback med metoden `onComplete`. Förklara vad som händer nedan.

```

1 scala> val action = Vector.fill(10)(Future{ seg })
2 scala> action(0).onComplete(xs => println(s"ready:$xs"))
3 scala> // vänta tills laddning på plats 0 är klar

```

h) Registrera en callback för felhantering i händelse av undantag med metoden `onFailure`. Förklara vad som händer nedan.

```

1 scala> def lycka = { Thread.sleep(3000); println(":)") }
2 scala> def olycka = { Thread.sleep(3000); 42 / 0; lycka }
3 scala> Future{ lycka }.onFailure{ case e => println(s":($e)") }
4 scala> Future{ olycka }.onFailure{ case e => println(s":($e)") }

```

13.1.2 Extrauppgifter

Uppgift 6. Räkna ut stora primtal parallellt genom att använda nedan funktioner. Implementera `isPrime` enligt pseudokod från den engelska wikipediasidan om primtalstest⁴ med den s.k. ”naiva algoritmen”. Räkna ut 10 st slumpvisa primtal med 16 siffror vardera. Gör beräkningarna parallellt med hjälp av `Future`.

```
def isPrime(n: BigInt): Boolean = ???

def nextPrime(start: BigInt): BigInt = {
    var i = start
    while (!isPrime(i)) { i += 1 }
    i
}

def randomBigInt(nDigits: Int): BigInt = {
    def rndChar = ('0' + (math.random * 10).toInt).toChar
    val str = Array.fill(nDigits)(rndChar).mkString
    BigInt(str)
}
```



Uppgift 7. Svara på teorifrågor.

- a) Vad är en tråd?
- b) Hur skapar man en tråd med klassen `Thread`?
- c) Hur startar man en tråd?
- d) Vilka problem kan man råka ut för om man uppdaterar samma resurs i flera olika trådar?
- e) Vad innebär det att kod är *trådsäker*?
- f) Nämn några fördelar med att använda `Future` jämfört med att använda trådar direkt.

Uppgift 8. Läs om och testa klasserna `AtomicBoolean`, `AtomicDouble` och `AtomicReferens` för atomär uppdatering i paketet `java.util.concurrent.atomic`. Använd några av dessa tillsammans med `scala.concurrent.Future`.

⁴en.wikipedia.org/wiki/Primality_test

13.1.3 Fördjupningsuppgifter

Uppgift 9. Skapa din egen multitrådade webbserver.

- a) Skriv in⁵ nedan kod i en editor och spara i en fil med namn `webserver.scala` och kompilera och kör med `scala webserver.start` och beskriv vad som händer när du med din webbläsare surfar till adressen:

`http://localhost:8089/abbasillen`

```

1 package webserver
2
3 import java.net.{ServerSocket, Socket}
4 import java.io.OutputStream
5 import java.util.Scanner
6 import scala.util.Try
7
8 object html {
9   def page(body: String): String = //minimal web page
10  s"""<!DOCTYPE html>
11    |<html>
12    |<head><meta charset="UTF-8"><title>Min Sörver</title></head>
13    |<body>
14    |$body
15    |</body>
16    |</html>
17    """".stripMargin
18
19  def header(length: Int): String = //standardized header of reply
20  s"HTTP/1.0 200 OK\nContent-length: $length\n" +
21  "Content-type: text/html\n\n"
22 }
23
24
25 object start {
26   def handleRequest(cmd: String, uri: String, socket: Socket): Unit = {
27     val os = socket.getOutputStream
28     val response = html.page("Baklänges: " + uri.reverse)
29     os.write(html.header(response.size).getBytes("UTF-8"))
30     os.write(response.getBytes("UTF-8"))
31     os.close
32     socket.close
33   }
34
35   def serverLoop(server: ServerSocket): Unit = {
36     println(s"http://localhost:${server.getLocalPort}/hej")
37     while (true) {
38       Try {
39         var socket = server.accept // blocks thread until connect
40         val scan = new Scanner(socket.getInputStream, "UTF-8")
41         val (cmd, uri) = (scan.next, scan.next)
42         println(s"Request: $cmd $uri")
43         handleRequest(cmd, uri, socket)
44       }
45     }
46   }
47 }
```

⁵Eller ladda ner här: github.com/lunduniversity/introprog/blob/master/compendium/examples/simple-web-server/webserver.scala

```

44     }.recover{ case e: Throwable => s"Connection failed: $e" }
45   }
46 }
47
48 def main(args: Array[String]) {
49   val port = Try{ args(0).toInt }.getOrDefault(8089)
50   serverLoop(new ServerSocket(port))
51 }
52 }
```

- b) Du ska nu skapa en webbserver som gör något lite mer intressant. Den ska svara om du surfar till <http://localhost:8089/fib/13> med det 13:e Fibbonaci-talet⁶. Spara din webserver från föregående deluppgift under det nya namnet `fibserver.scala` och använd koden nedan och lägg till och ändra så att din server kan svara med Fibonaccital. Vi börjar med att räkna ut Fibonaccital i funktionen `compute.fib` nedan på ett onödigt processokrävande sätt med exponentiell tidskomplexitet så att webbservern verkligen får jobba, för att i senare deluppgifter implementera `compute.fib` med linjär tidskomplexitet och därmed undvika onödig planetuppvärmning.

```

object compute {
  def fib(n: BigInt): BigInt = {
    if (n < 0) 0 else
    if (n == 1 || n == 2) 1
    else fib(n - 1) + fib(n - 2)
  }
}

def fibResponse(num: String) = Try { num.toInt } match {
  case Success(n) => html.page(s"fib($n) == " + compute.fib(n))
  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
}
```

Kör i terminalen med `scala fibserver.start` och beskriv vad som händer i din webbläsare när du surfar till servern.

⁶<https://sv.wikipedia.org/wiki/Fibonaccital>

- c) Surfa efter flera stora Fibonacci-tal samtidigt i olika flikar i din browser. Hur märks det att servern bara kör i en enda tråd?
- d) Gör din server multitrådad med hjälp av den nya server-loopen nedan.

```
import scala.concurrent._
import ExecutionContext.Implicits.global

def serverLoop(server: ServerSocket): Unit = {
  println(s"http://localhost:${server.getLocalPort}/hej")
  while (true) {
    Try {
      var socket = server.accept // blocks thread until connect
      val scan = new Scanner(socket.getInputStream, "UTF-8")
      val (cmd, uri) = (scan.next, scan.next)
      println(s"Request: $cmd $uri")
      Future { handleRequest(cmd, uri, socket) }.onFailure {
        case e => println(s"Request failed: $e")
      }
    }.recover{ case e: Throwable => s"Connection failed: $e" }
  }
}
```

- e) Surfa efter flera stora Fibonacci-tal samtidigt i olika flikar i din browser. Hur märks det att servern är multitrådad?
- f) Det är onödigt att räkna ut samma Fibonacci-tal flera gånger. Med hjälp av en cache i form av en föränderlig Map kan du spara undan redan uträknade värden. Det funkar dock inte med en vanlig `scala.collection.mutable.Map` i vår multitrådade webbserver, eftersom den inte är **trådsäker** (eng. *thread-safe*). Med trådosäkra föränderliga datastrukturer blir det samma besvärliga beteende som i uppgift 2.

Du ska i stället använda `java.util.concurrent.ConcurrentHashMap`. Sök upp dokumentationen för `ConcurrentHashMap` och försök förstå koden nedan. Hur fungerar metoderna `containsKey`, `put` och `get`?

```
object compute {
  import java.util.concurrent.ConcurrentHashMap
  val memcache = new ConcurrentHashMap[BigInt, BigInt]

  def fib(n: BigInt): BigInt =
    if (memcache.containsKey(n)) {
      println("CACHE HIT!!! no need to compute: " + n)
      memcache.get(n)
    } else {
      println("cache miss :( must compute fib: " + n)
      val f = fastFib(n)
      memcache.put(n, f)
      f
    }

  private def fastFib(n: BigInt): BigInt = {
```

```

if (n < 0) 0 else
if (n == 1 || n == 2) 1
else fib(n - 1) + fib(n - 2)
}
}

```

- g) Använd ovan `fib`-objekt i en ny version av din webserver. Spara den i en ny kodfil med namnet `fibserver-memcached.scala`. Undersök hur snabbt det går med stora Fibonaccital med den nya varianten. Hur stora tal kan du räkna ut? Kan servern fortsätta efter överflödad stack? Förklara varför.
- h) Nu när vi kan få väldigt stora Fibonacci-tal kan det vara användbart att stoppa in radbrytningar på webbsidan. Html-taggen `</br>` ger en radbrytning.

```

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

```

Använd den rekursiva funktionen ovan för att pilla in radbrytningstaggar på var n :te position i långa strängar. Testa hur det ser ut på webbsidan med ovan funktion när din server svarar med väldigt stora tal.

- i) Vi ska nu använda det större heap-minnet i stället för stack-minnet och därmed inte begränsas av stackens max-storlek. Skriv om `fastFib` så att den använder en `while`-sats i stället för ett rekursivt anrop. Denna uppgift är ganska klurig, men om du kör fast kan du snegla i lösningarna i Appendix för inspiration.

Hur stora tal klarar din server nu? Vad händer med servern när minnet tar slut? Hur kan du skydda servern så att den inte kan hänga sig?

Uppgift 10. Utöka din server med fler beräkningsintensiva funktioner. Exempelvis primtalsberäkningar eller beräkningar av valfritt antal decimaler av π eller e . Utnyttja gärna det du lärt dig i matematiken om summor och serieutvecklingar.

Uppgift 11. Läs mer om Future och jämlöpande exekvering i Scala här: alvinalexander.com/scala/future-example-scala-cookbook-oncomplete-callback

Uppgift 12. Läs mer om jämlöpande exekvering och multitrådade program i Java här: www.tutorialspoint.com/java/java_multithreading.htm

När man skriver program med jämlöpande exekvering finns det många fallgrupper; det kan bli kapplöpning (eng. *race conditions*) om gemensamma resurser och dödläge (eng. *deadlock*) där inget händer för att trådar väntar på varandra. Mer om detta i senare kurser.



Uppgift 13. Studera dokumentationen i `scala.concurrent`.

- a) Studera dokumentationen för `scala.concurrent.Future`⁷. Hur samverkar Future med Try och Option? Vilka vanliga samlingsmetoder känner du igen?
- b) Studera dokumentationen för `scala.concurrent.duration.Duration`⁸. Vilka tidsenheter kan användas?
- c) Vid import av `scala.concurrent.duration._` dekoreras de numeriska klasserna med metoder för att skapa instanser av klassen Duration. Detta möjligörs med hjälp av klassen `scala.concurrent.duration.DurationConversions`. Studera dess dokumentation och testa att i REPL skapa några tidsperioder med metoderna på Int.

Uppgift 14. Fördjupa dig inom webbteknologi.

- a) Lär dig om HTML här: <http://www.w3schools.com/html/>
- b) Lär dig om Javascript här: <http://www.w3schools.com/js/>
- c) Lär dig om CSS här: <http://www.w3schools.com/css/>
- d) Lär dig om Scala.JS här: <http://www.scala-js.org/>

⁷<http://www.scala-lang.org/files/archive/api/current/#scala.concurrent.Future>

⁸www.scala-lang.org/api/current/#scala.concurrent.duration.Duration

13.2 Projektuppgift: life

Mål

- Kunna använda matriser som en datastruktur.
- Kunna separera modell från vy med hjälp av *Model-View* uppdelning.
- Känna till grundläggande cellulära automata.

Förberedelser

- Läs igenom laborationen.
- Bekanta dig med kodskelettet.
- Börja skriva konstruktorn för `ArrayMatrix2D` som handleds i uppgift 1a.

13.2.1 Bakgrund

Spelet Life (även kallat *Conway's Game of Life* efter skaparen och matematikern John Horton Conway) simulerar en koloni av encelliga organismer som lever, förökar sig och dör på ett rutnät (även kallat bräde). Varje enskild cells överlevnad beror på dess omgivning, vilket kan beskrivas med några enkla regler. Detta är gemensamt för alla cellulära automater. Spelet går ut på att simulera flera generationer av någon uppsättning celler, en såkallad cellkoloni, startkonfiguration eller frö (eng. *seed*). Många sådana cellkolonier kan i Life verka bete sig 'levande' och spelet har därifrån fått sitt namn.⁹ Spelet har inga medvetna spelare (ett så kallat 'zero-player game') och om reglerna följs så kommer slutresultatet fullständigt bero på startkonfigurationen.

I denna laboration ska vi implementera en datastruktur för brädet samt reglerna för Life.

I extrauppgifterna kommer det även finnas möjlighet att utforska två andra exempel på cellulära automater.

För mer om Game of Life, se Wikipedia:

- Engelska: https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
- Svenska: https://sv.wikipedia.org/wiki/Game_of_Life

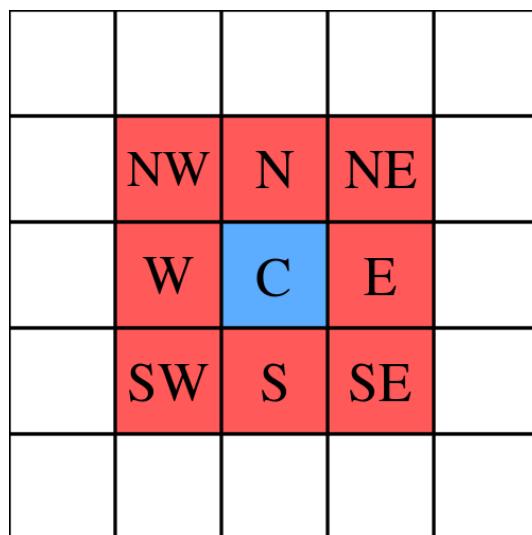
13.2.2 Reglerna i Life

Varje cells tillstånd i nästa generation bestäms av följande regler:

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

⁹Detta är ett exempel på s.k. 'emergence' och 'self-organization'

Med grannar menas i Life de s.k. Moore grannarna, detta är helt enkelt de närmsta omgivande cellerna vertikalt, horisontalt och diagonalt.



Figur 13.1: Moore grannskapet består av nio celler: mittcellen samt de åtta omringande cellerna.¹⁰

13.2.3 Beskrivning av Workspace

I workspaceet finner ni bl.a. paketen `models`, `rules` och `views`. I labben kommer ni främst ändra i `models` paketet men ni kommer även implementera reglerna för Life i `rules/LifeRule.scala`. Paketet `views` kommer med färdigimplementerad kod för två vyer `CellularConsoleView` samt `CellularGuiView`.

Utöver paketen finner ni i samma mapp några körbara Scala-filer som kan användas för att starta upp ett användargränssnitt som ritar upp brädet och låter användaren ‘spela’ spelet. Den första av dessa filerna ni ska titta i och senare köra är `life.scala`.

Läs igenom de två filerna i ‘models’ paketet noga, det är där ni kommer spendera den mesta av tiden.

13.2.4 Obligatoriska uppgifter

Uppgift 1. Skapa en modell som kan visas i vyer som implementerar `CellularView2D`.

Spelbrädet består av en matris med n rader och m kolumner (n och m brukar ibland modelleras som ∞ , men vi kommer begränsa oss för enkelhetens skull)

Varje cell i matrisen kan vid varje tidpunkt (i varje generation) ha ett av två tillstånd: levande eller död. Dessa kommer vi för enkelhetens skull att

¹⁰Källa: https://commons.wikimedia.org/wiki/File:Moore_neighborhood_with_cardinal_directions.svg

representera som 1 eller 0, respektive. Detta för att göra det enklare att räkna ihop hur många levande grannar en cell har.

- a) Implementera konstruktorn i kompanjonsobjektet för ArrayMatrix2D.

För att skapa en matris ska vi använda oss av Scalas inbyggda datastruktur `Array`. Denna datastruktur har en användbar konstruktur `ofDim[T](n1: Int, n2: Int)` som vi ska använda för att skapa *rows* stycken arrayer av storlek *cols* inuti en annan array, detta är vad vi kommer använda för att lagra datan i vår matris.

Ett exempel av resultatet från `Array.ofDim[Int](3, 3)` är följande ekvivalenta kod:

```
Array(
  Array(0,0,0),
  Array(0,0,0),
  Array(0,0,0)
)
```

- b) Implementera metoderna i ArrayMatrix2D.

Fortsätt nu med att implementera de övriga metoderna i `ArrayMatrix2D`. Läs igenom kommentarerna för de oimplementerade metoderna i traitet `Matrix2D` för kommentarer om vad som ska göras.

- c) Testa implementationen.

För att nu testa implementationen kan man slumpa tillstånden på brädet med hjälp av metoden `randomize` som finns på alla objekt som ärver traitet `Matrix2D`, med den kan man enkelt göra följande:

```
// Slumpar varje cell i matrisen 'matrix' till antingen 0 eller 1
matrix.randomize(2)
```

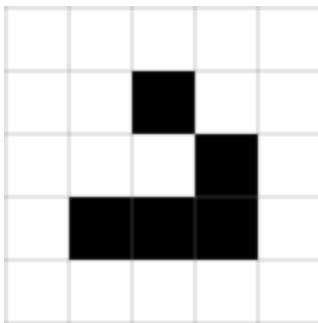
För att nu se till att allt har blivit rätt kan vi nu visa upp matrisen i terminalen med hjälp av `CellularConsoleView`. Ett exempel på detta finnes i `life_console.scala`.

```
// Skriver ut brädet i konsolen
CellularConsoleView.display(matrix)
```

Men detta testar inte för alla möjliga fel, så för säkerhets skull kan vi även testa att placera ut en s.k. glider i brädet med hjälp av metoden `place` som finns implementerad i det ärvda traitet `Matrix2D`.

```
// Placerar ut en glider med sitt övre vänstra hörn
// i positionen (3, 1), dvs på rad 4 och kolumn 2 (då vi noll-indexerar)
matrix.place(entities.glider, 3, 1)
```

Testa nu att rita ut brädet igen (utan `randomize` denna gången) för att se till att glidern hamnade rätt både med avseende på orientering och position, den ska vara roterad precis som på bilden.



Figur 13.2: En såkallad *glider*. Vanligt förekommande varelse i Life.

Uppgift 2. Implementera Life-regeln med hjälp av traitet Rule.

Nu har vi vår datastruktur för brädet på plats så nu är det dags att faktiskt implementera reglerna för Life. För att göra detta ska vi implementera objektet `LifeRule` som ärver traitet `Rule` vilket är ett interface som cellulära automaters regler kan implementera. Men först ska vi implementera några hjälpsamma funktioner, så att våran kod i `LifeRule` senare blir mer lättläst.

- Implementera `mooreNeighborsPositions` samt `mooreNeighborsStates` i `Matrix2D`.

Då Life-regeln förlitar sig på Moore-grannskapet som tidigare nämnts så behöver vi ett smidigt sätt att få ut grannarna för en viss cell, detta är vad funktionerna `mooreNeighborsPositions` samt `mooreNeighborsStates` ska göra.

När `mooreNeighborsStates` implementeras är det viktigt att ta hänsyn till om grannarna faktiskt finns (då vårat bräde är av begränsad storlek). För detta bör man använda `isWithinMatrix` metoden som tidigare implementerades i `ArrayMatrix2D`.

Börja med att implementera `mooreNeighborsPositions` och använd sedan den för att implementera `mooreNeighborsStates`.

- Implementera `apply` i ett nytt objekt `LifeRule` som implementerar `Rule`. Här nedan finnes specifikationen för traitet `Rule`, den innehåller endast en metod `apply` vars uppgift är att ta ett bräde samt en position på brädet och returnera vad denna position ska innehålla för värde i nästa generation.

Specification Rule

```
// apply tar en matris samt en position i matrisen (row, col)
// och applicerar regeln på den positionen
def apply(m: ArrayMatrix2D, row: Int, col: Int): Int
```

Reglerna för Life finner ni ovan i avsnitt 13.2.2.

- Testa implementationen.

För att nu i praktiken applicera våran regel på brädet ska vi använda oss av metoden `applyRule` som återfinns i `ArrayMatrix2D`.

Om man har problem med att sin regel inte beter sig som förväntat så man returnera antalet grannar istället för cellens levande/död tillstånd. Efter att applicera hela s.k. `NeighborsRule` kan man visa upp resultatet med

CellularConsoleView för att se om programmet räknar grannar korrekt.

Uppgift 3. Skapa en ny modell som ‘wrappar’ i kanterna.

Just nu har vi ett udda beteende i modellen, nämligen att alla celler utanför brädet i praktiken räknas som döda. För att få ett lite mer intressant beteende så vill vi nu göra så att om exempelvis en glider åker in i höger vägg ska den komma ut ur vänster vägg. Detta kan göras med några små modifikationer till ArrayMatrix2D.

- a) Ändra metoden `isWithinMatrix` så att alla positioner är giltiga.

Då funktionen `mooreNeighborsStates` i `Matrix2D` använder sig av funktionen `isWithinMatrix` för att avgöra vilka celler som är inom brädet så måste vi se till att `isWithinMatrix` inte hindrar grann-funktionen från att hämta celler ‘utanför’ brädet, detta kan vi enkelt göra genom att kommentera ut hela den tidigare koden och istället alltid returnera `true`. Om vi nu försöker köra programmet kommer vi nu se att vi får `ArrayIndexOutOfBoundsException`, detta då vi ännu inte gjort så att hämtningar eller tilldelningar ‘utanför’ brädet wrappar, vilket vi nu ska göra.

- b) Ändra metoderna `get` samt `set` så att hämtningar respektive tilldelningar utanför brädet wrappar.

För att åstadkomma wrappning så måste vi göra att alla hämtningar och tilldelningar utanför brädet går ‘runt’. Detta kan enklast göras med hjälp av modulo-operatorn.

```
val wrapped_row = row % rows
```

Men denna lösning tar inte hänsyn till negativa tal, vilket kan uppkomma när man t.ex. försöker hämta grannarna för cellen i det övre vänstra hörnet på brädet, dvs. position $(0, 0)$. För att enkelt lösa detta kan man förskjuta hela `row` med `rows` så att man istället får följande

```
val wrapped_row = (rows + row) % rows
```

Vi kan använda oss av denna kunskap för att i metoderna `get` samt `set` se till att operationerna wrappar. Testa sedan din implementation genom att låta en glider glida in i en vägg.

13.2.5 Frivilliga extrauppgifter

Inom cellulära automata finns det många intressanta ting att utforska, för de nyfikna så finns det här nedan några extrauppgifter som den intresserade läsaren uppmanas implementera då resultaten kan vara djupt tillfredsställande. De kan göras i valfri ordning.

Uppgift 4. Implementera spara och ladda

När man leker runt med cellulära automater vill man ofta spara sina bräden så att man senare enkelt kan ladda in dem igen. För att göra detta ska

vi i denna extrauppgift definiera ett dataformat som ska innehålla all data som behövs för att återskapa vår `ArrayMatrix2D`.

a) Spara brädets tillstånd. Tillståndet ska sparas till ett format som både är lätt att spara/exportera och ladda/importera. Det vi behöver spara är följande:

- Brädets storlek, dvs antalet rader och kolumner
- Hur många olika värden en cell på brädet kan anta
- Matrisen i sig, dvs alla cellers värden

Det är metoden `ArrayMatrix2D.toFileFormat` som ska implementeras.

Vi föreslår att ni börjar filen med en rad innehållande de två värdena i punkt ett, samt värdet i punkt två. Separera dem med mellanslag, komma, eller en emoji (på egen risk). Därefter är den enklaste vägen att skriva ut matrisen rad för rad där varje cell skrivs ut som en etta eller nolla. Man kan göra detta utan separator för bräden där celler bara antar värden i intervallet 0-9, men det fungerar inte längre om en cell kan anta andra värden (såsom i uppgiften med cykliska cellulära automater nedan) då man inte längre kan lita på att en siffra motsvarar en cell. För att lösa denna begränsning kan man separera sina värden med något tecken precis som för första raden.

Testa att spara genom att gå in i menyn i användargränssnittet och klicka "Save...". Öppna den sparade filen med en texteditor för att verifiera att innehållet ser korrekt ut.

b) Ladda in det sparade tillståndet.

Implementera metoden `ArrayMatrix2D.fromFileFormat` för att läsa in det sparade tillståndet.

Testa sedan att ladda genom att gå in i menyn i användargränssnittet och klicka "Load...". Brädet ska nu se ut precis som det gjorde när det sparades.

Uppgift 5. Implementera andra regler för cellulära automata.

Det finns massor med regler för cellulära automata med sina egna intressanta beteenden och tillstånd. Gör den eller de du tycker verkar mest intressant!

Fler regler finnes här: https://en.wikipedia.org/wiki/Category:Cellular_automaton_rules

Nedan följer några roliga exempel som valts ut och anses lämpliga.

a) Implementera regeln för cykliska cellulära automater.

Denna typ av automata kallas cyklisk just för att det finns N möjliga tillstånd och när tillståndet $N - 1$ nås så är ' nästa' tillstånd 0. Detta beteendet kan beskrivas med modulo-operatorn: $T_{\text{nästa}} = (T_{\text{nuvarande}} + 1) \% N$

Regeln för att en cell byter tillstånd ges av att om en granne till den aktuella cellen har tillståndet exakt ett över cellens tillstånd så får cellen sin grannes tillstånd. Det vill säga om en granne har tillståndet $T_{\text{nästa}}$ så får även mittcellen det värdet.

För att få intressant beteende brukar man initialisera hela brädet så att varje cell får ett slumpmätt tillstånd.

En bra förberedelse för att implementera denna cellulära automat är att läsa Wikipedia-artikeln: https://en.wikipedia.org/wiki/Cyclic_cellular_automaton

b) Implementera regeln för Wireworld.

Wireworld är annorlunda från andra cellulära automata då man i Wireworld designar 'kretsar' inte helt olika de som finns i moderna datorer. På grund av detta är majoriteten av celler vanligtvis fast i ett dött tillstånd (isolatorer).

I Wireworld kan man skapa komponenter såsom dioder och transistorer. Med dessa bygga logiska grindar och därmed hela datorer (som dock blir väldigt långsamma i jämförelse med datorerna de körs på).

En bra förberedelse för att implementera Wireworld är att läsa Wikipedia-artikeln: <https://en.wikipedia.org/wiki/Wireworld>



Figur 13.3: En enkel dator implementerad i Wireworld¹¹

13.2.6 Extra läsning

Intressanta mönster i Life

- <https://en.wikipedia.org/wiki/Spacefiller>
- https://en.wikipedia.org/wiki/Spaceship_%28cellular_automaton%29

Intressanta automater

- https://en.wikipedia.org/wiki/Codd's_cellular_automaton

¹¹Källa: <http://www.quinapalus.com/wi-index.html>

- https://en.wikipedia.org/wiki/Von_Neumann_universal_constructor

Intressanta resurser

- Eric Weisstein's treasure trove of The Game Of Life: <http://www.ericweisstein.com/encyclopedias/life/topics/>

13.3 Projektuppgift: bank

13.3.1 Fokus

- Kunna implementera ett helt program efter given specifikation
- Kunna sätta samman olika delar från olika moduler
- Förstå hur Java-klasser kan användas i Scala
- Förstå och bedöma när immutable/mutable såväl som var/val bör användas i större sammanhang
- Kunna använda sig av kompanjons-objekt
- Kunna läsa och skriva till fil
- Kunna söka i olika datastrukturer på olika sätt

13.3.2 Bakgrund

I detta projekt ska du skriva ett program som håller reda på bankkonton och kunder i en bank. Programmet ska även hålla reda på bankens nuvarande tillstånd, såväl som föregående. Tillstånden ska vid varje tillståndsförändring skrivas till fil så att utifall banken skulle krascha finns alla transaktioner som genomförts sparade, banken kan således återställas.

Programmet ska vara helt textbaserat, man ska alltså interagera med programmet via konsollen där en meny skrivs ut och input görs via tangentbordet.

Du ska skriva hela programmet själv, med andra ord ges ingen färdig kod. Programmet ska dock följa de specifikationer som ges i uppgiften, såväl som de objektorienterade principer du lärt dig i kursen.

13.3.3 Krav

Kraven för bankapplikationen återfinns här nedan. För att bli godkänd på denna uppgift måste samtliga krav uppfyllas:

- Programmet ska ha följande menyval:
 - 1. Hitta konton för en viss kontoinnehavare med angivet ID.
 - 2. Söka efter kontoinnehavare på (del av) namn.
 - 3. Sätta in pengar på ett konto.
 - 4. Ta ut pengar på ett konto.
 - 5. Överföra pengar mellan två olika konton.
 - 6. Skapa ett nytt konto.
 - 7. Ta bort ett befintligt konto.
 - 8. Skriv ut bankens alla konton, sorterade i bokstavsordning efter innehavare.
 - 9. Återställa banken till ett tidigare tillstånd för ett givet datum.
För simplicitet får alla transaktioner genomförda efter det datum banken återställts till permanent kasseras.

- 10. Avsluta.
- Programmet ska skapa ett nytt tillstånd med tidsstämpel och spara gamla tillstånd varje gång då:
 - Pengar sätts in på ett konto
 - Pengar tas ut från ett konto.
 - Pengar överförs mellan två konton.
 - Ett konto skapas.
 - Ett konto tas bort.
- Då bankens tillstånd förändras ska detta skrivas till fil.
- Då banken startas upp ska transaktionshistoriken läsas in så att banken laddar senaste sparade tillståndet.
- Inga utskrifter eller inläsningar får göras i klasserna Customer, BankAccount, Bank, State eller Transaction. Allt som berör användargränssnittet ska ske i BankApplication. Det är tillåtet att använda valfritt antal hjälpmetoder och hjälpklasser i klassen BankApplication.
- Alla metoder och attribut ska ha lämpliga åtkomsträttigheter.
- Valet av val/var och immutable/mutable måste vara lämpliga.
- Din indata måste ge samma resultat som i exemplen (som kommer komma i framtiden) i bilagan.
- Rimlig felhantering ska finnas, det är alltså önskvärt att programmet inte kraschar då man matar in felaktig input, utan istället säger till användaren att input är ogiltig.
- Programdesignen ska följa de specifikationer som är angivna nedan.
- Det räcker med att banken ska kunna hantera heltal, men dessa ska ske med klassen BigInt.
- Kontonummer ska genereras i klassen BankAccount, dessa ska vara unika för varje konto. Vid en tillståndsförändringar ska dessa återställas, detta betyder att om en återställning tar bort ett konto så ska detta kontonummer återigen bli tillgängligt.

13.3.4 Design

Nedan följer specifikationerna för de olika klasserna bankapplikationen måste innehålla:

Specification Customer

```
/*
 * Describes a customer of a bank with provided name and id.
 */
class Customer(val name: String, val id: Int) = {
  override def toString(): String = ???
```

Specification BankAccount

```
/*
 * Creates a new bank account for the customer provided.
 * The account is given a unique account number and initially
 * got a balance of 0 kr.
 */
class BankAccount(val holder: Customer) = {

  /**
   * Deposits provided amount on this account.
   */
  def deposit(amount: Int): Unit = ???

  /**
   * Returns the balance of this account.
   */
  def getBalance: Int = ???

  /**
   * Withdraws provided amount from this account, if there
   * is enough money on the account. Returns true if the
   * transaction was successfull, otherwise false.
   */
  def withdraw(amount: Int): Boolean = ???
```

Specification BankEvent

```
/*
 * Describes an event happening in the bank.
 */
abstract class BankEvent {
  /**
   * Output format for the transaction.
   */
  def write: String
```

Specification Bank

```
/*
 * Creates a new bank with no accounts and no state.
```

```

/*
class Bank() = {

    /**
     * Adds a new account in the bank.
     * The account number generates for the account is returned.
     */
    def addAccount(name: String, id: Int): Int = ???

    /**
     * Removes the bank account with provided account number,
     * returns true if successfull, otherwise false is returned.
     */
    def removeAccount(accountNbr: Int): Boolean = ???

    /**
     * Returns a list with every bank account in the bank.
     * The returned list is sorted in alphabetical order based
     * on customer name.
     */
    def getAllAccounts(): ArrayBuffer[BankAccount] = ???

    /**
     * Returns the account holding provided account number.
     * If no such account exists null is returned.
     */
    def findByName(accountNbr: Int): BankAccount = ???

    /**
     * Returns a list with every account belonging to the customer
     * with provided id.
     */
    def findAccountsForHolder(id: Int): ArrayBuffer[BankAccount] = ???

    /**
     * Returns a list with all customers which names matches
     * with provided name pattern.
     */
    def findByName(namePattern: String): ArrayBuffer[Customer] = ???

    /**
     * Executes a transaction in the bank.
     * Returns a string with information whether the
     * transaction was successful or failed.
     */
    def doEvent(transaction: Transaction): String = ???

    /**
     * Resets the bank to the state with time-stamp corresponding to the
     * provided date. If the date provided doesn't correspond exactly to
     * any time-stamp then the nearest time-stamp with a date previous
     * to the provided date is used instead.
     * Returns a string with information whether the transaction was
     * successful or failed.
     */
    def returnToState(returnDate: Date): String = ???
}

```

Specification State

```
/**  
 * Describes a bankstate.  
 * The queue log consists of a lists with all transactions  
 * made paired together with all dates corresponding to  
 * those transactions.  
 */  
class State(val log: Queue[(Transaction, Date)])
```

För att använda tidsstämplar ska klassen Date som finns bifogat i kursens workspace användas. Detta är en enkel wrapper av Java.time.

13.3.5 Tips

- För att representera tillstånden är det viktigt att alla händelser som förändrar tillståndet representeras av ett BankEvent.
- För att skriva till fil på ett enkelt sätt kan man t.ex. använda sig av klassen Files som finns tillgänglig i java.nio.file. För att undvika portabilitetsproblem kan man då använda sig av ett bestämt Charset, t.ex. UTF_8, som finns tillgänglig i java.nio.charset.StandardCharsets.UTF_8.
- För att läsa ifrån en fil kan man t.ex. använda sig av klassen Source som finns tillgänglig i scala.io.Source.
- Var nogrann med att testerna klarar alla tänkbara fall, och tänk på att fler fall än dem som givits i exempel kan förekomma vid rättnings.

13.3.6 Obligatoriska uppgifter

Uppgift 1. Implementera klassen Customer.

Uppgift 2. Implementera klassen BankAccount.

Uppgift 3. Implementera den BankEvent klass som skapar ett nytt konto.

Uppgift 4. Skapa en ny klass BankApplication.

- a) Klassen BankApplication ska innehålla main-metoden. Det kan vara bra att innan man fortsätter se till att denna klass skriver ut menyn korrekt och kan ta input från tangentbordet som motsvarar de menyval som finns.

Uppgift 5. Implementera klassen Bank.

- a) Implementera menyval 6 och 8. Testa noga.

- b) Implementera tillståndsfunktionaliteten. Varje nytt BankEvent ska ge upphov till ett nytt tillstånd och gamla tillstånd ska sparas som historik till det nya tillståndet.
- c) Implementera alla andra menyval, förutom menyval 9. Implementera även de klasser som förlängar BankEvent utefter att de behövs för nya menyval. Testa de nya menyvalen noga efterhand som du implementerar dem, i synnerhet så att tillståndsförändringarna fungerar korrekt. Gör de utökningar du anser behövs.

Uppgift 6. Implementera menyval 9. När man försöker återställa banken till ett datum ska den senaste BankEvent genomförd före detta datum hämtas, med andra ord ska alla BankEvent med tidsstämpel efter återställningsdatumet kasseras permanent. Testa noga. Det är viktigt att denna funktionalitet fungerar bra innan man går vidare.

Uppgift 7. Implementera säkerhetskopiering av tillstånden.

- a) Implementera utskriften till fil då ett nytt tillstånd skapas, utskriften ska ske omedelbart. Banken ska ej behöva avslutas för att utskriften ska hamna på fil, om så vore fallet kan information fortfarande gå förlorad om banken kraschar.

I repozitoriet för denna projekt uppgift finns en sparfil bifogad, för bekvämlighet finns ett utdrag av denna fil infogad nedanför. Inläsning och utskrift ska ske med dess format:

```
2016 3 7 10 6 N 850127 Fredrik
2016 3 7 10 28 D 1000 16500
2016 3 9 10 52 W 1000 3900
2016 3 9 11 8 N 900318 Casper
2016 3 9 16 28 D 1001 6500
2016 4 1 10 11 W 1001 1900
2016 4 1 11 19 W 1001 2000
2016 4 2 16 33 N 651002 Björn
2016 4 2 16 46 D 1002 25000
2016 4 3 10 11 T 1002 1000 4000
```

Formen är alltså:

År Månad Dag Timme Minut BankEventTag Parametrar

De olika klasserna av BankEvent representeras med följande bokstav:

- D - Deposit
- W - Withdraw
- T - Transfer
- N - NewAccount

- E - DeletedAccount
- b) Implementera inläsningen från fil då banken startas.

13.3.7 Frivilliga extrauppgifter

Gör först klart projektets obligatoriska delar. Därefter kan du, om du vill, utöka ditt program enligt följande.

Uppgift 8. Skriv en eller flera av klasserna Customer, BankAccount och State i Java istället och använd istället för din Scala versionen.

Uppgift 9. Implementera ett nytt menyalternativ som skriver ut all kontohistorik för en given person. I historiken ska typ av transaktion med tillhörande parametrar, dåvarande saldo vid transaktionen synas såväl som datumet för transaktionen synas.

13.3.8 Exempel på köring av programmet

Nedan visas möjliga exempel på köring av programmet. Data som matas in av användaren är markerad i fetstil. Ditt program måste inte se identiskt ut, men den övergripande strukturen såväl som resultatet av körningen ska vara densamma. När exemplet börjar förutsätts det att banken inte har några konton.

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton
 6. Skapa nytt konto
 7. Radera existerande konto
 8. Skriv ut alla konton i banken
 9. Återställ banken till ett tidigare datum
 10. Avsluta

Val: **6**

Namn: **Adam Asson**

Id: **6707071234**

Nytt konto skapat med kontonummer: 1001

10:03:0 CET 14 / 5 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton

6. Skapa nytt konto
7. Radera existerande konto
8. Skriv ut alla konton i banken
9. Återställ banken till ett tidigare datum
10. Avsluta

Val: **1**

Id: **6707071234**

Adam Asson, id 6707071234

10:04:0 CET 14 / 5 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton
 6. Skapa nytt konto
 7. Radera existerande konto
 8. Skriv ut alla konton i banken
 9. Återställ banken till ett tidigare datum
 10. Avsluta

Val: **6**

Namn: **Berit Besson**

Id: **8505255678**

Nytt konto skapat med kontonummer: 1001

10:12:0 CET 14 / 5 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton
 6. Skapa nytt konto
 7. Radera existerande konto
 8. Skriv ut alla konton i banken
 9. Återställ banken till ett tidigare datum
 10. Avsluta

Val: **8**

Konto 1000 (Adam Asson, id 850127) 0 kr

Konto 1001 (Berit Besson, id 900318) 0 kr

10:13:0 CET 14 / 5 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn

3. Sätt in pengar
4. Ta ut pengar
5. Överför pengar mellan konton
6. Skapa nytt konto
7. Radera existerande konto
8. Skriv ut alla konton i banken
9. Återställ banken till ett tidigare datum
10. Avsluta

Val: **2**

Namn: **adam**

Adam Asson, id 6707071234

10:15:0 CET 14 / 5 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton
 6. Skapa nytt konto
 7. Radera existerande konto
 8. Skriv ut alla konton i banken
 9. Återställ banken till ett tidigare datum
 10. Avsluta

Val: **6**

Namn: **Berit Besson**

Id: **8505255678**

Nytt konto skapat med kontonummer: 1002

13:56:0 CET 14 / 5 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton
 6. Skapa nytt konto
 7. Radera existerande konto
 8. Skriv ut alla konton i banken
 9. Återställ banken till ett tidigare datum
 10. Avsluta

Val: **2**

Namn: **erit**

Konto 1001 (Berit Besson, id 900318) 0 kr

Konto 1002 (Berit Besson, id 900318) 0 kr

14:01:0 CET 14 / 5 - 2016

-
- 1. Hitta ett konto för en given kund
 - 2. Sök efter en kund utifrån (del av) angivet namn
 - 3. Sätt in pengar
 - 4. Ta ut pengar
 - 5. Överför pengar mellan konton
 - 6. Skapa nytt konto
 - 7. Radera existerande konto
 - 8. Skriv ut alla konton i banken
 - 9. Återställ banken till ett tidigare datum
 - 10. Avsluta

Val: **3**

Kontonummer: **1000**

Summa: **5000**

Transaktionen lyckades.

14:36:0 CET 14 / 5 - 2016

-
- 1. Hitta ett konto för en given kund
 - 2. Sök efter en kund utifrån (del av) angivet namn
 - 3. Sätt in pengar
 - 4. Ta ut pengar
 - 5. Överför pengar mellan konton
 - 6. Skapa nytt konto
 - 7. Radera existerande konto
 - 8. Skriv ut alla konton i banken
 - 9. Återställ banken till ett tidigare datum
 - 10. Avsluta

Val: **5**

Kontonummer att överföra ifrån: **1000**

Kontonummer att överföra till: **1001**

Summa: **1000**

Transaktionen lyckades.

14:37:0 CET 14 / 5 - 2016

-
- 1. Hitta ett konto för en given kund
 - 2. Sök efter en kund utifrån (del av) angivet namn
 - 3. Sätt in pengar
 - 4. Ta ut pengar
 - 5. Överför pengar mellan konton
 - 6. Skapa nytt konto
 - 7. Radera existerande konto
 - 8. Skriv ut alla konton i banken
 - 9. Återställ banken till ett tidigare datum
 - 10. Avsluta

Val: **8**

Konto 1000 (Adam Asson, id 850127) 4000 kr

Konto 1001 (Berit Besson, id 900318) 1000 kr

Konto 1002 (Berit Besson, id 900318) 0 kr

14:52:0 CET 14 / 5 - 2016

-
- 1. Hitta ett konto för en given kund
 - 2. Sök efter en kund utifrån (del av) angivet namn
 - 3. Sätt in pengar
 - 4. Ta ut pengar
 - 5. Överför pengar mellan konton
 - 6. Skapa nytt konto
 - 7. Radera existerande konto
 - 8. Skriv ut alla konton i banken
 - 9. Återställ banken till ett tidigare datum
 - 10. Avsluta

Val: **2**

Ange konto att radera: : **1002**

Transaktionen lyckades.

14:01:0 CET 14 / 5 - 2016

-
- 1. Hitta ett konto för en given kund
 - 2. Sök efter en kund utifrån (del av) angivet namn
 - 3. Sätt in pengar
 - 4. Ta ut pengar
 - 5. Överför pengar mellan konton
 - 6. Skapa nytt konto
 - 7. Radera existerande konto
 - 8. Skriv ut alla konton i banken
 - 9. Återställ banken till ett tidigare datum
 - 10. Avsluta

Val: **7**

Namn: **erit**

Konto 1001 (Berit Besson, id 900318) 0 kr

14:01:0 CET 14 / 5 - 2016

-
- 1. Hitta ett konto för en given kund
 - 2. Sök efter en kund utifrån (del av) angivet namn
 - 3. Sätt in pengar
 - 4. Ta ut pengar
 - 5. Överför pengar mellan konton
 - 6. Skapa nytt konto
 - 7. Radera existerande konto
 - 8. Skriv ut alla konton i banken
 - 9. Återställ banken till ett tidigare datum

10. Avsluta

Val: **9**

Vilket datum vill du återställa banken till?

År: **2016**

Månad: **5**

Datum (dag): **14**

Timme: **10**

Minut: **5**

Banken återställd.

15:00:0 CET 14 / 5 - 2016

1. Hitta ett konto för en given kund
2. Sök efter en kund utifrån (del av) angivet namn
3. Sätt in pengar
4. Ta ut pengar
5. Överför pengar mellan konton
6. Skapa nytt konto
7. Radera existerande konto
8. Skriv ut alla konton i banken
9. Återställ banken till ett tidigare datum
10. Avsluta

Val: **8**

Konto 1000 (Adam Asson, id 850127) 0 kr

15:01:0 CET 14 / 5 - 2016

1. Hitta ett konto för en given kund
2. Sök efter en kund utifrån (del av) angivet namn
3. Sätt in pengar
4. Ta ut pengar
5. Överför pengar mellan konton
6. Skapa nytt konto
7. Radera existerande konto
8. Skriv ut alla konton i banken
9. Återställ banken till ett tidigare datum
10. Avsluta

Val: **3**

Kontonummer: **1001**

Summa: **5000**

Transaktionen misslyckades. Inget sådant konto hittades.

15:06:0 CET 14 / 5 - 2016

1. Hitta ett konto för en given kund
2. Sök efter en kund utifrån (del av) angivet namn
3. Sätt in pengar
4. Ta ut pengar

5. Överför pengar mellan konton
6. Skapa nytt konto
7. Radera existerande konto
8. Skriv ut alla konton i banken
9. Återställ banken till ett tidigare datum
10. Avsluta

Val: **3**

Kontonummer: **1001**

Summa: **5000**

Transaktionen misslyckades. O tillräckligt saldo.

15:23:0 CET 14 / 5 - 2016

13.4 Projektuppgift: tictactoe

Mål

- Implementera ett helt program efter specifikation.
- Få en inblick i hur rekursion kan användas, utöver svans-rekursion.
- Bli introducerad till spelteori och hur man kan uttrycka optimal strategi för spelet tictactoe.
- Träna på att använda abstrakta klasser.
- Kunna byta mellan representationer av en spelplan.

13.4.1 Bakgrund

I detta projektet ska du implementera din egen version av spelet tic-tac-toe (eller som vi på svenska kallar det, tre i rad)! Du kommer börja med att implementera en version där du kan spela mot en kursare och sen gå vidare till att implementera en datorspelare som lägger sin pjäs slumpmässigt och till slut en som inte kan förlora!

13.4.2 Regler

Om du känner dig säker på hur reglerna i tic-tac-toe funkar kan du skippa detta.¹²

- Spelplanen består av ett rutnät av storlek 3x3.
- Det finns två spelare: x och o.
- Spelarna placerar ut en pjäs var i växlande ordning där x börjar.
- Spelet tar slut om en spelare har fått antingen en rad, diagonal eller kolumn ifylld av sin spelpjäs eller om spelplanen är fylld.

Notera att pjäserna INTE får flyttas när de väl ligger på spelplanen.

13.4.3 Teori

Representationen är vald till en endimensionell vektor av typen Int av storlek 9 där element [0,2]¹³ representerar den första raden [3,5] andra och [6,8] den tredje. Anledningen till detta är att vi vill ha en representation så att spelaren kan svara vilket drag den vill göra med ett heltalet. Varje element i vektorn ska kunna representera en tom plats, en plats allokerad av x och en plats allokerad av o. Detta innebär att en vektor av typen Boolean inte räcker till. Istället väjs den (kanske lite minnesöverflödiga) typen Int. Vi har valt representationen där 0 representerar tom plats, 1 representerar x och -1 representerar o. Denna

¹²en.wikipedia.org/wiki/Tic-tac-toe

¹³Med beteckningen [x,y] menas alla heltalet från x till y, dvs: x, x+1, x+2, ..., y-1, y. [0,2] = 0,1,2

representation är dels smidig för vår framtida OptimalPlayer och även för att avgöra om spelare x eller o har vunnit. Man kan exempelvis summera en rad och kolla om radens summa är 3, då har x vunnit eller -3, då har o vunnit.

13.4.4 Design

Den här uppgiften kommer innehålla lite färre klasser och mindre objektorienterade utmaningar, men istället lite klurigare algoritmiska utmaningar. Vi skall dessutom använda rekursion till vår fördel, vilket för den ovane brukar vara krångligt, därför hamnar huvudfokus här. Programmet har följande klasstruktur: Game - i sin mainmetod frågar den användaren hur spelen skall gå till och med vilka spelare. När detta är specificerat anropas den rekursiva metoden play. play spelar ett spel mellan två spelare tills någon vinner eller det blir oavgjort. Kodskriften för Game ser ut på följande vis:

```
Specification Game

/*
 * Asks the user what kind of players should play against each other.
 * Creates the players that the user chooses via System.in
 * Asks the user if the board should be drawn.
 * If the board should not be drawn, ask the user for n,
 * the amount of games that should be played between the two players.
 * Else n = 1.
 * Call play n times with the two players, an empty board, depth = 0,
 * and drawing true/false dependent on if the board should be printed or not.
 * Save the results from play.
 * Present the results after n games have been played.
 */
def main(args: Array[String]): Unit = ???

/*
 * p1,p2 are the players that should play the game
 * depth models amount of moves done by both players
 * drawing is true if draw should be called before each move.
 *(1) If drawing: call draw(game)
 *(2) If game is won by any of the players, or depth == 9,
 * return 1 if p1 wins, -1 if p2 wins and 0 if there is a draw.
 *(3) Else: Asks player 1 for its move if depth%2 == 0, else ask player 2.
 * update game according to the move p1 or p2 does.
 *(5) return play(p1,p2,game,depth+1,drawing)
 *if someone wins with the current move,
 * it will be detected by (2) in this call of play
 */
def play(p1: Player, p2: Player, game: Array[Int],
         depth: Int, drawing:Boolean): Int = ???

/*
 * Given an Array[Int] game of size 9,
 * print the 3x3-board that the array represents.
 * [0,2] is the 1st, [3,5] is the 2nd and [6,8] is the 3rd row.
 * -1 should be represented by 'o', 0 with '.' and 1 with 'x'.
 * in particular [0,1,-1,0,0,1,0,1,-1] should print:
 * .xo
```

```

* ..x
* .xo
*/
def draw(game:Array[Int]):Unit = ???
```

Player - är en abstrakt klass kommer innehålla gemensam logik för players. För att inte göra våra players beroende av Game har vi valt att lägga en metod gameWon i Player. Valet av placering av denna metod kan definitivt diskuteras, man skulle kunna tänka sig att denna metod borde ligga i ett util-objekt eller liknande, men för tillfället är det bara denna metod som man skulle vilja ha i ett sådant objekt vilket gör det klumpigt. Anledningen till att den istället hamnat hos Player är att både klassen OptimalPlayer och FastOptimalPlayer, som extenderar Player kommer vilja ha tillgång till denna metod.

Players viktiga funktion är dess move-metod, det är denna metod som kommer skilja sig för olika players. I den första uppgiftern skall ett tal (draget) läsas in i HumanPlayers move-metod, medan i nästkommande uppgift skall ett random drag väljas i RandomPlayers move-metod. Kodsklettet för Player ser ut på följande sätt:

Specification Player

```

abstract class Player(name: String) {

    /**
     * abstract method, should not be implemented here,
     * but required by extensions of Player.
     * returns an int p in the interval [0,8] where game(p) == 0,
     * the index of the move this player does.
     */
    def move(game: Array[Int], depth: Int): Int;

    /**
     * returns true if there exists a row, column or diagonal,
     * where all the elements are equal to who.
     */
    def gameWon(game: Array[Int], who:Int): Boolean =  ???

    // returns a String with information about the player.
    override def toString(): String = ???
}
```

13.4.5 Obligatoriska uppgifter

Uppgift 1. Implementera ett fungerande spel genom att utöka kodskeletten i klasserna Player, HumanPlayer och Game.

- Implementera metoden gameWon i klassen Player som testar huruvida spelaren who vunnit spelet.
- Implementera HumanPlayers toString-metod
- Implementera HumanPlayers move-metod.

- d) Implementera en version av Game. Börja med att alltid spela ett spel och alltid rita spelplanen. main, draw och play behöver implementeras. All funktionalitet i main behöver ännu inte finnas.¹⁴

Uppgift 2. RandomPlayer

- Skapa en ny utökning av Player (kopiera HumanPlayer, och byt namn till RandomPlayer) där move istället för att läsa från System.in väljer ett random giltigt drag.
- Ändra Game så att användaren tillåts stänga av ritfunktionen och i så fall tillåts välja antalet spel.
- Vad är sannolikheterna för att x vinner, o vinner och att det blir oavgjort om två RandP spelar mot varandra?

Hamnar man i närheten av dessa resultat tror vi på er RandP.

- $P(x \text{ vinner}) = 0.586$
- $P(o \text{ vinner}) = 0.288$
- $P(\text{lika}) = 0.126$

- Varför är det större sannolikhet för x att vinna än o?

Uppgift 3. OptimalPlayer

Betrakta den givna funktionen eval

```
/** 
 * returns 1 if there is a guaranteed strategy for who to win
 * returns 0 if there is a guaranteed strategy for who to draw
 * returns -1 if the opponent can force a win,
 * no matter what who does.
 * This is done by min,max-evaluation.
 * Find the move that gives the opponent the worst possible
 * position (min) and return -min, this is our max.
 * depth is the amount of empty cells in game.
 * who is 1 if it's this players turn to make a move,
 * -1 if it's the opponents turn to make a move.
 * From move, eval should be called with who = -1.
 */
def eval(game: Array[Int], depth: Int, who: Int): Int = {
  if(gameWon(game, -who)) return -1
  if(depth == 9) return 0
  var min = 1
  for(i <- 0 until 9) {
    if(game(i) == 0) {
      game(i) = who
      if(depth > 0) {
        val result = eval(game, depth - 1, -who)
        if(result < min) min = result
      }
      game(i) = 0
    }
  }
}
```

¹⁴Notera att man behöver invertera spelplanen om den ska skickas till spelare två (alternativt låta spelaren hålla reda på om den är x eller o). Förslagsvis lösas detta med en extra metod invGame som skapar en ny array med omvänta tecken till orginalarrayen.

```

val score = eval(game,depth+1,-who)
if(score<min){
    min = score
}
game(i) = 0
}
}
-min
}

```

`eval` avgör om du är i en vinnande, förlorande eller oavgjord situation, givet att båda spelare spelar optimalt. Det som nog är svårast att förstå är varför vi retunerar `-min` på slutet. `min` sparar det sämsta värdet som vår motståndare kan få givet våra möjliga drag. Vi observerar att vi är i precis omvänt situation jämfört med vår motståndare. Om vår motståndare definitivt vinner förlorar vi definitivt, om det blir oavgjort för vår motståndare blir det också oavgjort för oss, och om vår motståndare definitivt förlorar, då vinner vi. Vi representerade ju vinst med 1, oavgjort med 0 och förlust med -1. Det är alltså härifrån minustecknet kommer ifrån. Vill man läsa mer om detta kan man kolla in wikipedias artikel ¹⁵ om minmax-evaluering. Vi tar helt enkelt det draget som är sämst för vår motståndare.

- Implementera `move`-metoden till `OptimalPlayer` genom att kalla på `eval`.
- Låt två `OptimalPlayer` spela mot varandra. Det skall alltid bli oavgjort.
- Testa att spela mot din `OptimalPlayer` med en `HumanPlayer`. Kan du spela lika? Kan du vinna?
- Vad händer om du sätter en `RandomPlayer` mot `OptimalPlayer`? Blir det någonsin oavgjort, hur ofta? Blir det någon skillad man byter vem som får spela först?

Uppgift 4. Utöka säkerheten och isoleringen av ditt program

I nuläget finns det förmodligen ett problem med din nuvarande implementation, och det är att du skickar iväg en mutable datastruktur till en `Player` som utifrån den mutable datan skall göra ett drag. Tänk om en elak programmerare bestämmer sig för att ändra på spelplanen i sin egna `players` `move`-metod. Då skulle man i princip kunna fuska. För att lösa detta kan man från Game istället för att ge ifrån sig den egna representationen av spelplanen göra en kopia och ge kopian till `Playern`.

Uppgift 5. Snabbare `OptimalPlayer`.

Om du låter en `OptimalPlayer` spela mot en `RandomPlayer` 1000 gånger lär det ta ganska lång tid. Det behöver det inte göra. När `OptimalPlayer` bestämmer vilket drag den skall göra första gången går den ju igenom alla andra möjliga drag man kan komma till. Det visar sig att det inte finns så många unika spelbräden. Färre än $3^9 < 20000$.

¹⁵<https://en.wikipedia.org/wiki/Minimax>

Skapar vi istället ett uppslagsverk (Map) som innehåller ett värde för varje spelbräde kommer vi kunna spela mycket snabbare när väl vår Map är genererad. Skillnaden i snabbhet för vårt program blir alltså att vi behöver göra ett uppslag i en Map, jämfört med ungefär $9! > 300000$ funktionsanrop för ett drag på ett tomt spelbräde.

Det räcker med att modifera evalueringsmetoden något för att bygga mapen. Sedan anropa denna med ett tomt bräde och så har vi vår HashMap och vår optimala spelare är nu supersnabb!

Man får dock vara lite klurig, en `Array[Int]` går inte att använda som nyckel i en Map i java, då den inte har en implementerad `hashCode`-metod. I scala är det tillåtet, men mapningen kommer att vara på objektlikhet och inte innehållslikhet, vilket vi i det här fallet inte vill. Enklast, som säker fungerar i både java och scala är att göra om vår array till en sträng, genom att lägga värdena i arrayen efter varandra i strängen. Vi kan göra en privat metod `hash(Array[Int]):String` som konkatenerar värdena i arrayen. `hash([0,0,0,1,1,0,-1,-1,0])` skall alltså retunera "000110-1-10".

- Skapa en ny subklass till Player: `FastOptimalPlayer`.
- Skapa och implementera en privat metod `hash(Array[Int]):String`
- Implementera en konstruktur som skapar och genererar en Map.

Mapen kan initieras på följande vis:

```
val boardCache = scala.collection.mutable.Map.empty[String, Int].
```

För att sedan generera Mapen krävs en motsvarande metod som eval-metoden i `OptimalPlayer`. Denna kan tas rakt av, men med två små modifieringar:

- Vid start: Om det redan finns ett key-value-par i Mapen: returnera värdet från Mapen.
- Vid slut: Innan du retunerar måste ett key-value-par läggas in i Mapen som vi genererar.

Denna metod bör sedan anropas i slutet av konstruktorn med ett tomt board.

- Låt move-metoden göra en Map-lookup med hjälp av hash-metoden och din Map.
- Testa att låta en `FastOptimalPlayer` spela mot en `RandomPlayer`. Du märker nog att skapandet av en `FastOptimalPlayer` kommer ta lite tid, typ en halv sekund. Sedan skall det dock gå jättesnabbt när spelarna spelar. 100000 spel skall gå utan problem på någon sekund, vilket borde gå på tiotals minuter för den gamla `OptimalPlayer`.

13.5 Projektuppgift: imageprocessing

13.5.1 Bakgrund

En digital bild består av ett rutnät (en matris) av pixlar. Varje pixel har en färg, och om man har många pixlar flyter de samman för ögat så att de tillsammans skapar en bild.

Det finns olika system för hur man färgsätter de olika pixlarna. T.ex. så används CMYK-systemet (cyan, magenta, gul, svart) vid blandning av färg som ska tryckas på papper eller annat material. På en dator dockanväntas vanligtvis RGB-systemet. RGB-systemet har tre grundfärgar: röd, grön och blå. Mättnaden av varje grundfärg anges av ett heltal som vi i fortsättningen förutsätter ligger i intervallet [0, 255]. 0 anger "ingen färg" och 255 anger "maximal färg". Man kan därmed representera $256 \times 256 \times 256 = 16\,777\,216$ olika färgnyanser. Man kan också representera gråskalar; det gör man med färger som har samma värde på alla tre grundfärgerna: (0, 0, 0) är helt svart, (255, 255, 255) är helt vitt.

13.5.2 Uppgiften

Du ska skriva ett program där du implementerar olika filter som ska manipulera en given bild på ett flertal olika sätt. Filterklasserna ska härva från en abstrakt `ImageFilter`-klass som är skriven i Java. `ImageFilter`-klassen hittar du i cslib.

Följande beskriver `ImageFilter`-klassen.

```
abstract class ImageFilter

/*
 * Skapar ett filterobjekt med ett givet namn och antalet argument.
 */
protected ImageFilter(String name, int nbrOfArgs);

/*
 * Tar reda på filtrets namn.
 */
public String getName();

/*
 * Tar reda på antalet argument filtret behöver till apply-metoden.
 */
public int getNumberOfArguments();

/*
 * Filtrerar bilden i matrisen inPixels och returnerar resultatet i
 * en ny matris. Utnyttjar eventuellt värdena i args.
 */
public abstract Color[][] apply(Color[][] inPixels, double[] args);

*/
```

```

 * Beräknar intensiteten hos alla pixlarna i pixels,
 * returnerar resultatet i en ny matris.
 */
protected short[][] computeIntensity(Color[][] pixels):

/**
 * Faltar punkten p[i][j] med faltningskärnan kernel.
 *
 * @param p      matris med talvärdet
 * @param i      radindex får den aktuella punkten
 * @param j      kolonnindex får den aktuella punkten
 * @param kernel faltningskärnan, en 3x3-matris
 * @param weight summan av elementen i kernel
 * @return     resultatet av faltningen
 */
protected short convolve(short[][] p, int i, int j,
    short[][] kernel, int weight);

```

Utöver filterklasserna ska du även skapa ett program där du kan välja ett variabelt antal filter och sedan applicera dessa på en bild. För att åstadkomma detta ska du implementera klasserna `FilterChooser`, som hanterar val av filter, och `FilterList` som representerar vilka filter som ska användas. Klasserna har följande specifikationer:

Specification FilterList

```

class FilterList = ???

/** Adds a filter to the FilterList */
def addFilter(filter: ImageFilter): Unit = ???

/** Applies all the filters on the given Image and draws it in SimpleWindow */
def applyFilters(image: Image, sw: SimpleWindow): Unit = ???

```

Specification FilterChooser

```

/** Creates a FilterChooser with all the available filters */
class FilterChooser(filters: Array[ImageFilter]) = ???

/** Shows which filters are available and lets the user choose filters
 * until an escape sequence has been given and returns a FilterList which
 * contain the chosen filters
 * Example:
 * 0. för Blått-filter
 * 2. för Kontrast-filter
 * 3. för Gauss-filter
 * 4. för Sobel-filter
 * 5. om du inte vill ha fler filter
 */
def chooseFilters(): FilterList = ???

```

Till din hjälp får du en `Image`-klass som representerar en bild samt ett `ImageUI` som hjälper dig att ladda in en JPEG bild.

Specification Image

```
class Image(val image: BufferedImage);

/** Returns a matrix of Color-objects that represents an image */
def getColorMatrix: Array[Array[Color]];

/** Updates the image in accordance with the given Color-matrix */
def updateImage(pixels: Array[Array[Color]]): Unit;
```

Specification ImageUI

```
object ImageUI;

/** Returns a chosen image from the images folder.
 * Prints:
 *
 * Välj en av följande bilder genom att mata in en siffra
 *
 * 0. boy.jpg
 * 1. car.jpg
 * 2. duck.jpg
 * 3. facade.jpg
 * 4. jay.jpg
 * 5. moon.jpg
 * 6. obidos.jpg
 * 7. sgrada.jpg
 * 8. shuttle.jpg
 * Ditt val:
 */
def getImage: BufferedImage;
```

Uppgift 1. Blåfilter. Skriv en klass BlueFilter som skapar en blå version av bilden. Det vill säga skapa ett filter där varje pixel bara innehåller den blå komponenten. Testa filtret genom att skapa ett ImageProcessing-object som ska innehålla en main-metod (ImageProcessing ska användas och utökas i senare uppgifter). Använd ImageUI för att välja en bild på följande sätt:

```
val im = new Image(ImageUI.getImage)
```

Använd SimpleWindow samt image attributet från Image-objektet för att visa bilden.

Uppgift 2. Inverteringsfilter. Skriv en klass InvertFilter som inverterar en bild dvs skapar en ”negativ” kopia av bilden. Ljusa färger ska alltså bli mörka och mörka färger ska bli ljusa. Fundera över vad som kan menas med en inverterad eller negativ kopia: de nya RGB-värdena är inte ett dividerat med de gamla värdena (då skulle de nya värdena kunna bli flyttal) och inte de gamla värdena med ombytt tecken (då skulle de nya värdena bli negativa).

Uppgift 3. Gråskalningsfilter. Skriv en klass GrayScaleFilter som gör om bilden till en gråskalebild. Använd ImageFilters computeIntensity metod för att bestämma vilken intensitet varje pixel ska ha. Om intensiteten i en

pixel till exempel är 105 så ska ett nytt Color-objekt med värdena (105, 105, 105) skapas.

Uppgift 4. Krypteringsfilter. Skriv en klass XORCryptFilter som krypterar bilden med xor-operatorn \wedge . Denna operator gör binär xor mellan bitarna i ett heltal. Exempelvis ger $8 \wedge 127$ värdet 119. Om man gör xor igen med 127, alltså $119 \wedge 127$, får man tillbaka värdet 8. Varje pixel krypteras genom att använda xor-operatoren med ursprungsvärdena för rött, grönt och blått tillsammans med ett slumpmässigt heltalsvärde som genereras av Scalas Random klass. Använd `paramValue` för att ge Random-objektet ett seed. På så sätt kan du återskapa bilden genom att applicera krypteringsfiltret igen, med samma `paramValue`, på den numera krypterade bilden.

Uppgift 5. Gaussfiltrer. Gaussfiltrering är ett exempel på så kallad faltningsfiltrering. Filtreringen bygger på att man modifierar varje bildpunkt genom att titta på punkten och omgivande punkter.

För detta utnyttjar man en så kallad faltningskärna K som är en liten kvadratisk heltalsmatris. Man placerar K över varje element i intensitetsmatrisen och multiplicerar varje element i K med motsvarande element i intensitetsmatrisen. Man summerar produkterna och dividerar summan med summan av elementen i K för att få det nya värdet på intensiteten i punkten. Divisionen med summan gör man för att de nya intensiteterna ska hamna i rätt intervall.

Exempel:

$$\text{intensity} = \begin{pmatrix} 5 & 4 & 2 & 8 & \dots \\ 4 & 3 & 4 & 9 & \dots \\ 9 & 8 & 7 & 7 & \dots \\ 8 & 6 & 6 & 5 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \quad K = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Här är summan av elementen i K $1 + 1 + 4 + 1 + 1 = 8$. För att räkna ut det nya värdet på intensiteten i punkten med index (1)(1) (det nuvarande värdet är 3) beräknar man:

$$\text{newintensity} = \frac{0 \cdot 5 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 4 + 4 \cdot 3 + 1 \cdot 4 + 0 \cdot 9 + 1 \cdot 8 + 0 \cdot 7}{8} = \frac{32}{8} = 4$$

Man fortsätter med att flytta K ett steg åt höger och beräknar på motsvarande sätt ett nytt värde för elementet med index (1)(2) (där det nuvarande värdet är 4 och det nya värdet blir 5). Därefter gör man på samma sätt för alla element utom för "ramen" dvs elementen i matrisens ytterkanter.

Skriv en klass GaussFilters som implementerar denna algoritm. Varje färg ska behandlas separat. Gör på följande sätt:

1. Bilda tre short-matriner och lagra pixlarnas red-, green- och blue-komponenter i matriserna.
2. Utför faltningen av de tre komponenterna för varje element och lagra ett nytt Color-objekt i `outPixels` för varje punkt.

3. Elementen i ramen behandlas inte, men i `outPixels` måste också dessa element få värden. Enklast är att flytta över dessa element oförändrade från `inPixels` till `outPixels`. Man kan också sätta dem till `Color.WHITE`, men då kommer den filtrerade bilden att se något mindre ut.

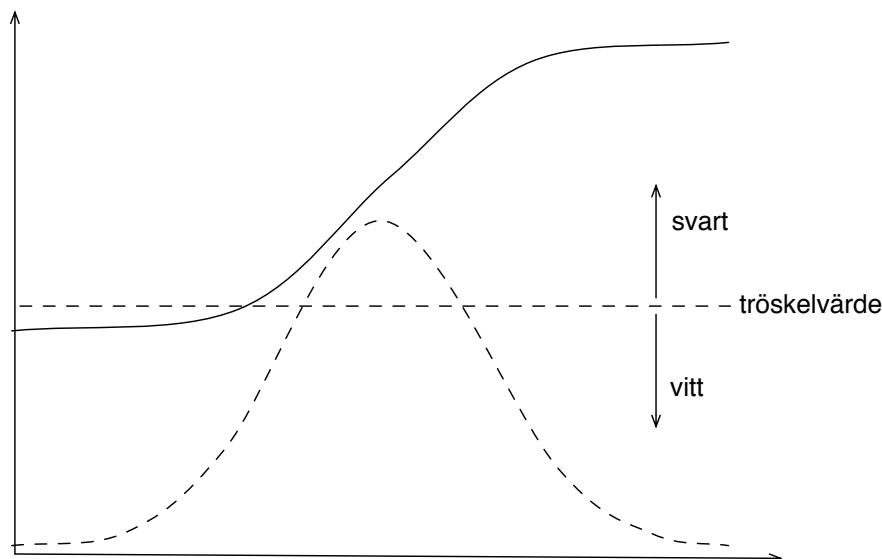
Använd `ImageFilters.convolve`-metod för att utföra faltningen. Metoden behöver en faltningssmatris, `kernel`, som input och ska anropas med red-, green- och blue-matrisen. Faltningssmatrisen kan vara ett attribut i klassen och ska ha följande utseende:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Det kan vara intressant att prova med andra värden än 4 i mitten av faltningssmatrisen. Med värdet 0 får man en större utjämning eftersom man då inte alls tar hänsyn till den aktuella pixelns värde. Mata in detta värde i Parameter-rutan.

Anmärkning: det kan ibland vara svårt att se någon skillnad mellan den filtrerade bilden och originalbilden. Om man vill ha en riktigt suddig bild så måste man använda en större matris som faltningsskärna.

Uppgift 6. Sobelfiltrer. Sobelfiltrering är, precis som Gaussfiltrering, en typ av faltningssfiltrering. Med Sobelfiltrering får man dock motsatt effekt i jämförelse med Gaussfiltrering, dvs man förstärker konturer i en bild. I princip deriverar man bilden i x- och y-led och sammansätter resultatet.



Figur 13.4: En funktion (heldragen linje) och dess derivata (streckad linje).

I figur 13.4 visas en funktion f (heldragen linje) och funktionens derivata f' (streckad linje). Vi ser att där funktionen gör ett ”hopp” så får derivatan

ett stort värde. Om funktionen representerar intensiteten hos pixlarna längs en linje i x-led eller y-led så motsvarar ”hoppen” en kontur i bilden. Om man sedan bestämmer sig för att pixlar där derivatans värde överstiger ett visst tröskelvärde ska vara svarta och andra pixlar vita så får man en bild med bara konturer.

Nu är ju intensiteten hos pixlarna inte en kontinuerlig funktion som man kan derivera enligt vanliga matematiska regler. Men man kan approximera derivatan, till exempel med följande formel:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

(Om man här låter h gå mot noll så får man definitionen av derivatan.) Uttryckt i Scala och matrisen `intensity` så får man:

```
val derivative = (intensity(i)(j+1) - intensity(i)(j-1)) / 2
```

Allt detta kan man uttrycka med hjälp av faltning.

1. Beräkna intensitetsmatrisen med metoden `computeIntensity`.
2. Falta varje punkt i intensitetsmatrisen med två kärnor:

$$X_SOBEL = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} Y_SOBEL = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Använd metoden `convolve` med vikten 1. Koefficienterna i matrisen `X_SOBEL` uttrycker derivering i x-led, i `Y_SOBEL` faltning i y-led. För att förklara varför koefficienterna ibland är 1 och ibland 2 måste man studera den bakomliggande teorin noggrant, men det gör vi inte här.

3. Om resultaten av faltningen i en punkt betecknas med `sx` och `sy` så får man en indikator på närvaron av en kontur med `math.abs(sx) + math.abs(sy)`. Absolutbelopp behöver man eftersom man har negativa koefficienter i faltningsmatriserna.
4. Sätt pixeln till svart om indikatorn är större än tröskelvärdet, till vit annars. Tröskelvärdet bestäms av `paramValue`.

Skriv en klass `SobelFilter` som implementerar denna algoritm.



Figur 13.5: Exempel på en bild där ett Sobelfilter applicerats med ett parametervärdet på 150.

Uppgift 7. Implementera `FilterList` enligt specifikationerna ovan.

Uppgift 8. Implementera `FilterChooser` enligt specifikationerna ovan.

Uppgift 9. Knyt ihop allt i `ImageProcessing`-objektet som du skapade innan.
Utskrifterna ska se ut på följande sätt:

Välj en av följande bilder genom att mata in en siffra

- 0. boy.jpg
- 1. car.jpg
- 2. duck.jpg
- 3. facade.jpg
- 4. jay.jpg
- 5. moon.jpg
- 6. obidos.jpg
- 7. sgrada.jpg
- 8. shuttle.jpg

Ditt val: **1**

Bild car.jpg laddad

- 0. för Vanligt-filter
- 1. för Blått-filter
- 2. för Krypterat-filter
- 3. för Inverterat-filter
- 4. för Grått-filter
- 5. för Kontrast-filter
- 6. för Gauss-filter
- 7. för Sobel-filter

8. om du inte vill använda fler filter

Välj ett filter **1**

Välj ett filter **8**

Välja ny bild? (y/n) **n**

Tänk på att användaren kan mata in otillåtna värden. Detta ska hanteras på lämpligt sätt.

13.5.3 Frivilliga extrauppgifter

Uppgift 10. Kontrastfilter. Om man applicerar kontrastfiltrering på en färgbild så kommer bilden att konverteras till en gråskalebild. (Man kan naturligtvis förbättra kontrasten i en färgbild och få en färgbild som resultat. Då behandlar man de tre färgkanalerna var för sig.) Många bilder lider av alltför låg kontrast. Det beror på att bilden inte utnyttjar hela det tillgängliga området 0–255 för intensiteten. Man får en bild med bättre kontrast om man ”töjer ut” intervallet enligt följande formel (lineär interpolation):

```
val newIntensity = 255 * (intensity - 45) / (225 - 45)
```

Som synes kommer en punkt med intensiteten 45 att få den nya intensiteten 0 och en punkt med intensiteten 225 att få den nya intensiteten 255. Mellanliggande punkter sprids ut jämnt över intervallet [0, 255]. För punkter med en intensitet mindre än 45 sätter man den nya intensiteten till 0, för punkter med en intensitet större än 225 sätter man den nya intensiteten till 255. Vi kallar intervallet där de flesta pixlarna finns för [lowCut, highCut]. De punkter som har intensitet mindre än lowCut sätter man till 0, de som har intensitet större än highCut sätter man till 255. För de övriga punkterna interpolerar man med formeln ovan (45 ersätts med lowCut, 225 med highCut).

Det återstår nu att hitta lämpliga värden på lowCut och highCut. Detta är inte något som kan göras helt automatiskt, eftersom värdena beror på intensitetsfördelningen hos bildpunkterna. Man börjar med att beräkna bildens intensitetshistogram, dvs hur många punkter i bilden som har intensiteten 0, hur många som har intensiteten 1, . . . , till och med 255.

I de flesta bildbehandlingsprogram kan man sedan titta på histogrammet och interaktivt bestämma värdena på lowCut och highCut. Så ska vi dock inte göra här. I stället bestämmer vi oss för ett procenttal cutOff (som bestäms av paramValue) och beräknar lowCut så att cutOff procent av punkterna i bilden har en intensitet som är mindre än lowCut och highCut så att cutOff procent av punkterna har en intensitet som är större än highCut.

Exempel: antag att en bild innehåller 100 000 pixlar och att cutOff är 1.5. Beräkna bildens intensitetshistogram i en vektor

```
val histogram = Array[Int](256)
```

Beräkna lowCut så att $\text{histogram}(0) + \dots + \text{histogram}(\text{lowCut}) = 0.015 * 100000$ (så nära det går att komma, det blir troligen inte exakt likhet). Beräkna

highCut på liknande sätt.

Sammanfattning av algoritmen:

1. Beräkna intensiteten hos alla punkterna i bilden, lagra dem i en short-matris. Använd den färdigskrivna metoden `computeIntensity`.
2. Beräkna bildens intensitetshistogram.
3. Parametervärdet `paramValue` är det värde som ska användas som `cutOff`.
4. Beräkna `lowCut` och `highCut` enligt ovan.
5. Beräkna den nya intensiteten för varje pixel enligt interpolationsformeln och lagra de nya pixlarna i `outPixels`.

Skriv en klass `ContrastFilter` som implementerar algoritmen. I katalogen `images` kan bilden `moon.jpg` vara lämpliga att testa, eftersom den har låg kontrast. Anmärkning: om `cutOff` sätts = 0 så får man samma resultat av denna filtrering som man får av `GrayScaleFilter`. Detta kan man se genom att studera interpolationsformeln.

Uppgift 11. Eget filter. Skapa ett eget filter som utnyttjar att `apply`-metoden tar emot en array av värden. Till exempel så kan du skicka in en array med fem värden där de två första värdena representerar ett intensitetsintervall och de tre sista värdena representerar röd-, grönt- och blåkomponenterna till en färg som ska stoppas in där intensiteten hamnar utanför det givna intervallet. Ett annat alternativ kan vara att använda dig av metoder i `SimpleWindow` för att välja specifika pixlar på orginalbilden som sedan kan användas för att manipulera bilden i filtrets `apply`-metod. Valet är ditt!

Kapitel 14

Tentaträning

Före tentan:

1. Repetera övningar och labbar i kompendiet.
2. Läs igenom föreläsningsanteckningar.
3. Studera **snabbref mycket noga** så att du vet vad som är givet och var det står, så att du kan hitta det du behöver snabbt.
4. Skapa och **memorera** en personlig **checklista** med programmeringsfel du brukar göra, som även inkluderar småfel, så som glömda parenteser och semikolon, och annat som en kompilator/IDE normalt hittar.
5. Tänk igenom hur du ska disponera dina 5 timmar på tentan.
6. Gör den fiktiva extentan som om det vore **skarp läge**:
 - (a) Avsätt 5 ostörda timmar (stäng av telefon, dator etc).
 - (b) Inga hjälpmaterial. Bara snabbref.
 - (c) Förbered dryck och tilltugg.

På tentan:

1. Läs igenom **hela** tentan först.
Varför? Förstå helheten. Delarna hänger ihop.
2. Notera och begrunda specifika begrepp och definitioner.
Varför? Begreppen är avgörande för förståelsen av uppgiften.
3. Notera förenklingar, antaganden och specialfall.
Varför? Uppgiften blir mkt enklare om du inte behöver hantera dessa.
4. **Fråga** tentamensansvarig om du inte förstår uppgiften – speciellt om det finns misstänkta felaktigheter eller förmodat oavsiktliga oklarheter.
Varför? Det är inte lätt att konstruera en ”perfekt” tenta.
Du får fråga vad du vill, men det är inte säkert du får svar :)
5. Läs specifikationskommentarerna och metodssignaturerna i alla givna klass-specifikationer **mycket noga**.
Varför? Det är ett vanligt misstag att förbise de ledtrådar som ges där.
6. Återskapa din memorerade personliga checklista för vanliga fel som du brukar göra och avsätt tid till att gå igenom den på tentan.
7. Lämna in ett försök även om du vet att lösningen inte är fullständig.

Del III

Appendix

Appendix A

Terminalfönster

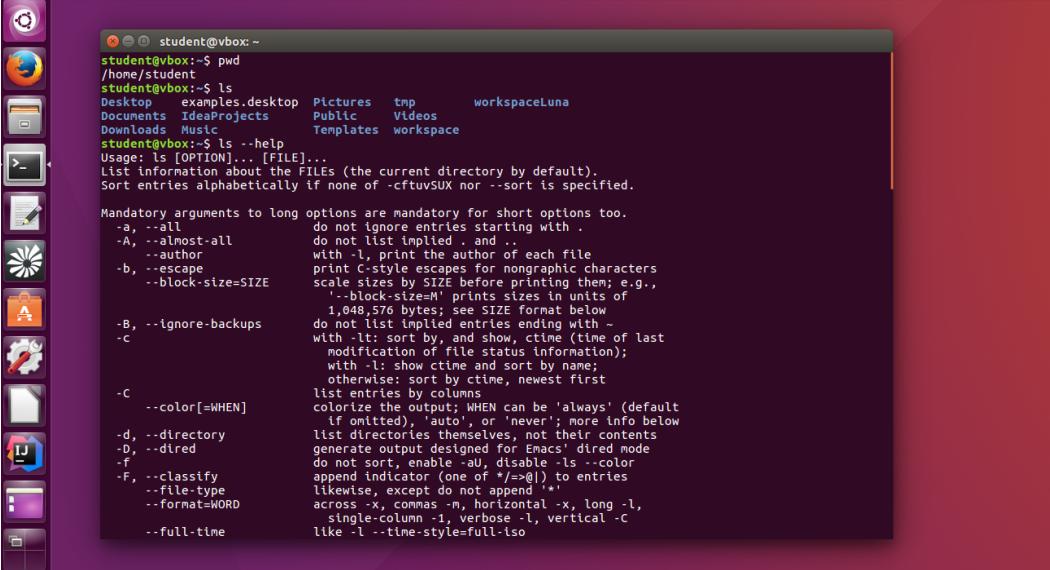
A.1 Vad är ett terminalfönster?

I ett terminalfönster kan man skriva kommandon som kör program och hanterar filer. När man programmerar använder man ofta terminalkommando för att kompilera och exekvera sina program.

Terminal i Linux

I Ubuntu trycker du lättast **Ctrl+Alt+T** eller sök efter ”terminal” i app-menyn. Då öppnas ett fönster med en blinkande markör som visar att det är redo att ta emot dina textkommando. Ett exempel på kommando är `ls` som skriver ut en lista med filer i det aktuella biblioteket, så som visas i fig. A.1.

Det som visas i ett terminalfönster sköts av ett **kommandoskal** (eng. *command shell*), som är redo att ta emot kommando efter en prompt som slutar med ett \$-tecken. När du skriver ett kommando och trycker Enter anropar



The screenshot shows a terminal window titled "student@vbox: ~". The user has run the command `ls`, which lists the contents of the current directory (~). The output includes:

```
student@vbox:~$ ls
Desktop  examples.desktop  Pictures  tmp      workspaceLuna
Documents  IdeaProjects    Public    Videos
Downloads  Music          Templates  workspace

student@vbox:~$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
-a, --all           do not ignore entries starting with .
-A, --almost-all   do not list implied . and ..
--author          with -l, print the author of each file
-b, --escape        print C-style escapes for nongraphic characters
--block-size=SIZE   scale sizes by SIZE before printing them; e.g.,
                   '-block-size=M' prints sizes in units of
                   1,048,576 bytes; see SIZE format below
-B, --ignore-backups  do not list implied entries ending with -
-c                with -lt: sort by, and show, ctime (time of last
                   modification of file status information);
                   with -l: show ctime and sort by name;
                   otherwise: sort by ctime, newest first
-C                list entries by columns
--color[=WHEN]     colorize the output; WHEN can be 'always' (default
                   if omitted), 'auto', or 'never'; more info below
-d, --directory    list directories themselves, not their contents
-D, --dired         generate output designed for Emacs' dired mode
-f                do not sort, enable -u; disable -ls --color
-F, --classify     append indicator (one of ">=@!") to entries
                   likewise, except do not append "*"
--file-type        across -x, commas -n, horizontal -x, long -l,
                   single-column -1, verbose -l, vertical -C
--format=WORD       like -l --time-style=full-ls0
--full-time
```

Figur A.1: Terminalfönster i Ubuntu öppnas med **Ctrl+Alt+T**.

kommandoskalet en kommandotolk som tolkar och utför dina kommandon. Om ett kommando inte kan tolkas, skrivs ett felmeddelande.

Det finns många användbara kortkommando, varav de viktigaste visas i tabell A.1. Det är bra om du lär dig dessa kortkommandon utantill så att ditt arbete i terminalen går snabbt och smidigt.

pil upp/ner	bläddra i kommandohistoriken
Tab	"auto-complete", fyll i resten baserat på vad du skrivit hittills
Tab Tab	två tryck på Tab listar flera alternativ, om så finnes
Ctrl+A	"ahead", flytta markören till början av raden
Ctrl+E	"end", flytta markören till slutet av raden
Ctrl+K	"kill", ta bort tecken från markören till radens slut
Ctrl+U	"undo", ta bort tecken från markören till början av raden
Ctrl+Y	"yank", sätt in det som senast togs bort
Ctrl+Z	"zleep", stoppa pågående process, skriv sedan bg för bakgrundskörning
Ctrl+L	rensa terminalfönstret
Ctrl+D	avsluta kommandoskalet

Tabell A.1: Viktiga kortkommandon i Linux terminalfönster.

Ctrl+C orsakar normalt ett avbrott av pågående process, men om du vill att Ctrl+C ska vara "Copy" som vanligt för att kopiera markerad text, kan du ställa om detta med terminalfönstrets meny "Edit → Keyboard Shortcuts", eller liknande.

PowerShell och Cmd i Microsoft Windows

Microsoft Windows är inte Linux-baserat, men i kommandotolken **Powershell** finns alias definierade för några vanliga Linux-kommandon, inkluderat ls, cd och pwd. Du startar Powershell t.ex. genom att trycka på Windows-knappen och skriva powershell. Du kan också, medan du bläddrar bland filer, klicka på filnamnsraden överst i filbläddraren och skriva powershell och tryck Enter; då startas Powershell i aktuellt bibliotek. Ändra gärna typsnitt och bakgrundsfärg med hjälp av fönstrets menyer, så att det blir lättare för dig att läsa vad som skrivs.

Det finns även i Windows den ursprungliga kommandotolken **Cmd** med helt andra kommandon. Till exempel skriver man i Cmd kommandot dir i stället för ls för att lista filer.

I Windows 10 du även köra Ubuntu-kommandoskalet, se <http://www.omgubuntu.co.uk/2016/08/enable-bash-windows-10-anniversary-update>

Terminal i Apple OS X / macOS

Apple OS X och macOS är Unix-baserade operativsystem. De flesta vanliga terminalkommandon som fungerar i Linux fungerar också under Apple OS X

och macOS. Du startar ett terminalfönster i Apples operativsystem genom att klicka på förstoringsglaset uppe till höger, skriva terminal, och trycka Enter.

A.2 Vad är en path/sökväg?

När du skriver ett kommando i terminalen, eller kör vilket program som helst på din dator, behöver operativsystemet identifiera i vilken fil programmets maskinkod ligger innan programmet kan köras.

Lokaliseringe av filer sker med hjälp av en **sökväg** (eng. *path*), som anger en position i filsystemet. Ofta betraktas filsystemet som ett upp-och-ned-vänt träd, och kallas därför även ”filträdet”. Den ”översta” positionen kallas ”rot” (eng. *root*) och betecknas med ett enkelt snedstreck /. Kataloger som ligger i kataloger utgör förgreningar i trädet. En sökväg pekar ut vägar genom trädet som behövs för att nå ”löven”, som utgörs av själva filerna.

Du kan se var ett program ligger i Linux med hjälp av kommandot which enligt nedan.¹ Listan med bibliotek i sövägen avskiljs med snedstreck.

```
$ which java  
/usr/lib/jvm/oracle_jdk8/bin/java  
$ which ls  
/bin/ls
```

En sökväg kan vara **absolut** eller **relativ**. En absolut sökväg utgår från roten och visar hela vägen från rot till destination, t.ex. /usr/bin/firefox, medan en relativ sökväg utgår från aktuellt bibliotek (där du ”står”) och börjar *inte* med ett snedstreck.

Alla operativsystem håller reda på en mängd olika sökvägar för att kunna hitta speciella filer i filträdet. Dessa sökvägar lagras i s.k. **miljövariabler** (eng. *environment variables*). Det finns en *speciell* miljövariabel som heter kort och gott **PATH**, i vilken alla sökvägar till de program finns, som ska vara tillgängliga för din användaridentitet direkt för exekvering genom sina filnamn, *utan* att man behöver ange absoluta sökvägar.

Du kan i Linux se vad som ligger i din PATH med kommandot echo \$PATH medan man i Windows Powershell skriver \$env.Path där det bara är första bokstaven som ska vara en versal. I Unix separeras biblioteken i sövägen med kolon, medan Windows använder semikolon.

Ibland kan du behöva uppdatera din PATH för att program som du installerat och ska bli allmänt tillgängliga. Detta görs på lite olika sätt i olika operativsystem, för Linux se t.ex. här: stackoverflow.com/questions/14637979/how-to-permanently-set-path-on-linux

När man anger sökvägar finns några tecken med speciell betydelse:

¹Skriv gcm ls i Windows Powershell för motsvarighet till which ls
Eller skriv New-Alias which get-command för tillgång till kommandot which i Powershell.
stackoverflow.com/questions/63805/equivalent-of-nix-which-command-in-powershell

- ~ "tilde", din hemkatalog
- / "slash", snedstreck anger filträdets rot om det finns i början av sökvägen, men utgör biblioteksavskiljare inuti sökvägen
- . en punkt anger aktuellt bibliotek, där du "står"
- .. två punkter anger ett steg "upp" i filträdet
- " omgärda en sökväg med citationstecken, först och sist, om den innehåller annat än engelska bokstäver, t.ex. blanktecken
- \ *backslash+blanktecken* används för att beteckna mellanslag i sökvägar som *inte* omgärdas av citationstecken

A.3 Några viktiga terminalkommando

I tabell A.2 finns en lista med några viktiga terminalkommando som är bra att lära sig utantill.

En introduktion till LTH:s datorer med exempel på hur du använder vanliga Linux-kommandon finns i denna skrift <http://www.ddg.lth.se/perf/unix/> som används i introduktionsveckan för nybörjare på dattateknikprogrammet vid LTH.

På sajten <http://ss64.com/> finns en mer omfattande lista med användbara terminalkommando och tillhörande förklaringar för för Linux (Bash), Windows (Powershell, Cmd) och Apple OS X.

<code>ls</code>	lista filer i aktuellt bibliotek (alltså där du "står")
<code>ls p</code>	lista filer i biblioteket <i>p</i>
<code>ls -A</code>	lista alla filer i aktuellt bibliotek, även gömda
<code>man ls</code>	manual för kommandot <code>ls</code> ; testa även <code>man</code> för andra kommandon!
<code>cd p</code>	"change directory", ändra aktuellt bibliotek till <i>p</i>
<code>pwd</code>	"print working directory", skriv ut sökväg för aktuellt bibliotek
<code>cp p1 p2</code>	"copy", kopiera filen med path <i>p1</i> till en ny fil kallad <i>p2</i>
<code>mv p1 p2</code>	"move", byt namn på filen <i>p1</i> till <i>p2</i>
<code>rm p</code>	"remove", ta bort filen <i>p</i>
<code>rm -r p</code>	"remove recursive", ta bort biblioteket <i>p</i> med allt innehåll; var försiktig!
<code>mkdir p</code>	"make dir", skapa ett nytt bibliotek <i>p</i>
<code>cat p1 p2</code>	"concatenate", skriv ut hela innehållet i en eller flera filer <i>p1 p2 etc.</i>
<code>less p</code>	skriv ut innehållet i filen <i>p</i> , en skärm i taget
<code>wget url</code>	ladda ner <i>url</i> , t.ex. <code>wget http://cs.lth.se/pgk/ws -o ws.zip</code>
<code>unzip p</code>	packa upp <i>p</i> , t.ex. <code>unzip ws.zip</code>

Tabell A.2: Några viktiga terminalkommando i Linux. Med *p*, *p1*, *p2*, etc. avses en absolut eller relativ sökväg (eng. *path*), se avsnitt A.2.

Appendix B

Editera

B.1 Vad är en editor?

En editor används för att redigera programkod. Det finns många olika editorer att välja på. Erfarna utvecklare lägger ofta mycket energi på att lära sig att använda favoriteeditorns kortkommandon och specialfunktioner, eftersom detta påverkar stort hur snabbt kodredigeringen kan göras.

En bra editor har **syntaxfärgning** för språket du använder, så att olika delar av koden visas i olika färger. Då går det mycket lättare att läsa och hitta i koden.

Nedan listas några viktiga funktioner som man använder många gånger dagligen när man kodar:

- **Navigera.** Det finns flera olika sätt att flytta markören och bläddra genom koden. Alla editorer erbjuder sökmöjligheter, och de flesta editorer har även mer avancerade sökfunktioner där kodmönster kan identifieras och multipla sökträffar markeras över flera kodfiler.
- **Markera.** Att markera kod kan göras på många sätt: med piltaxonger+Shift, med olika speciella menyalternativ, med mus + dubbelklick eller trippelklick, etc. I vissa editorer finns även möjlighet att ha multipla markörer så att flera rader kan editeras samtidigt.
- **Kopiera.** Genom Copy-Paste slipper du du skriva samma sak många gånger. Kortkommandona Ctrl+C för Copy och Ctrl+V för Paste sitter i fingrarna efter ett tag. Man ska dock vara medveten om att det lätt blir fel när man kopierar en stor del som sedan ska ändras lite; många Copy-Paste-buggar kommer av att man inte är tillräckligt noggrann och ofta är det bättre att skriva från grunden i stället för att kopiera så att du hinner tänka efter medan du skriver.
- **Klipp ut.** Genom Ctrl+X för Cut och Ctrl+V för Paste, kan du lätt flytta kod. Att skriva kod är en stegvis process där man gör många förändringar under resans gång för att förbättra och vidareutveckla koden. Att flytta på kod för att skapa en bättre struktur är mycket vanligt.

- **Formatering.** Med indragningar, radbrytningar och nästlade block i flera nivåer får koden struktur. Många editorer kan hjälpa till med detta och har speciella kortkommandon för att ändra indragningsnivå inåt eller utåt.
- **Parentesmatchning.** Olika former av parenteser, ({ [] }), behöver matchas för att koden ska fungera; annars går kompilatorn ofta helt vilse och konstiga felmeddelanden kan peka på helt fel plats i koden. En bra kodeditor kan hjälpa dig att markera vilka parentespar som hör ihop så att du undviker att spendera för mycket tid med att leta efter en parentes som saknas eller är står i vägen.

I en integrerad utvecklingsmiljö, en s.k. IDE, (se appendix D) finns en inbyggd editor som, tack vare ett mer intimt samarbete med kompilatorn, kan erbjuda ännu fler avancerade funktioner som hjälper dig i kodarbetet. Men även när du lärt dig använda en IDE kommer du fortfarande ha stor nytta av en ”vanlig” editor. Ofta har man flera terminalfönster igång, tillsammans med flera editorfönster och en IDE.

B.2 Välj editor

I tabell B.1 visas en lista med några populära editorer. Det är en stor fördel om din favoriteeditor finns på flera plattformar så att du har nytta av dina förvärvade färdigheter när du behöver växla mellan olika operativsystem.

Om du inte vet vilken du ska välja, börja med *gedit*, som inte är så avancerad, men ändå lätt att komma igång med. När du sedan är redo att investera din lärotid i en mer avancerad editor rekommenderas *Atom*, eftersom den är öppen, gratis och finns för Linux, Windows och macOS.

Det är också bra att lära sig åtminstone de mest basala kommandona i editorn *vim* eftersom denna editor kan köras direkt i terminalen, även vid fjärrinloggning, och finns förinstallerad i de flesta Linux-system.

<i>Editor</i>	<i>Beskrivning</i>
Gedit	öppen, fri och gratis; lätt att lära men inte så avancerad; finns för Linux, Windows & macOS; är förinstallerad på LTH:s Linux-datorer och startas med kommandot <code>gedit</code> i ett terminalfönster https://wiki.gnome.org/Apps/Gedit
Atom	öppen, fri och gratis; finns för Linux, Windows, & macOS; är förinstallerad på LTH:s Linux-datorer och startas med kommandot <code>atom</code> i ett terminalfönster; öppenkällkodssprojektet startades nyligen av GitHub och därför är Atom-editorn ännu inte lika mogen som övriga i denna lista https://atom.io/ Installera Ensime som ger kraftfullt Scala-stöd i Atom: http://ensime.github.io/editors/atom
Vim	öppen, fri och gratis; lång historik, hög inlärningströskel; finns för Linux, Windows, & Mac; är förinstallerad på LTH:s Linux-datorer och startas med kommandot <code>vim</code> i ett terminalfönster http://www.vim.org/
Emacs	öppen, fri och gratis; lång historik, hög inlärningströskel; finns för Linux, Windows, & Mac; är förinstallerad på LTH:s Linux-datorer och startas med kommandot <code>emacs</code> i ett terminalfönster http://www.gnu.org/software/emacs/
Sublime Text	sluten källkod; gratis att prova på, men programmet föreslår då och då att du köper en licens; finns för Windows, Mac, Linux. http://www.sublimetext.com/3
Notepad++	öppen, fri och gratis; finns endast för Windows; https://notepad-plus-plus.org/
Textwrangler	sluten källkod, gratis; lätt att lära men inte så avancerad; finns endast för macOS http://www.barebones.com/products/textwrangler/

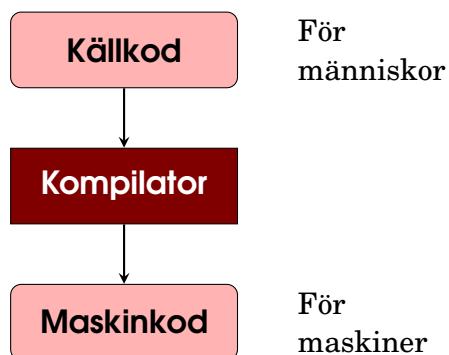
Tabell B.1: Några populära editorer. Om du inte vet vilken du ska välja, börja med att installera Gedit.

Appendix C

Kompilera och exekvera

C.1 Vad är en kompilator?

En **kompilator** (eng. *compiler*) är ett program som läser programtext och översätter den till exekverbar maskinkod, så som visas i figur C.1. Programtexten som kompileras kallas källkod och utgörs av text som följer reglerna för ett programmeringsspråk, till exempel Scala eller Java.



Figur C.1: En kompilator översätter från källkod till maskinkod.

Vissa kompilatorer genererar kod som kan köras av en processor direkt, medan andra kompilatorer genererar ett mellanformat som tolkas under exekveringen. Det senare är fallet med Java och Scala, vilket möjliggör att programmet kan kompileras en gång för alla plattformar och sedan kan programmet köras på all de processorer till vilka det finns en s.k. virtuell maskin för Java (eng. *Java Virtual Machine, JVM*). Den kod som genereras av en kompilator för JVM kallas **bytekod**.

Om kompileringen inte lyckas skriver kompilatorn ut ett felmeddelande och ingen maskinkod genereras. Det är inte lätt att bygga en kompilator som ger bra felmeddelanden i alla lägen, men felmeddelandet ger oftast goda ledtrådar till felorsaken efter att man lärt sig tolka det programmeringsspråksspecifika vokabulär som kompilatorn använder.

Även om programmet kompilerar utan felmeddelande och genererar exekverbar maskinkod, är det vanligt att programmet ändå inte fungerar som det

är tänkt. Ibland är det mycket svårt att lista ut vad problemet beror på och man kan behöva göra omfattande undersökningar av vad som händer under körningen, genom att t.ex. skriva ut olika variablers värden eller på annat sätt ändra koden och se vad som händer. Denna process kallas felsökning eller avlusning (eng. *debugging*), och är en väsentlig del av all systemutveckling.

En uttömmande testning av ett större program, som kör programmets *alla* möjliga exekveringsvägar, är i praktiken omöjlig att genomföra inom rimlig tid, då antalet kombinationsmöjligheter växer mycket snabbt med storleken på programmet. Därför är kompilatorn ett mycket viktigt hjälpmittel. Med hjälp av den analys och de kontroller som görs av kompilatorn kan många buggar, som annars vore mycket svåra att hitta, undvikas och åtgärdas i kompileringsfasen, redan *innan* man exekverar programmet.

C.2 Java JDK

Scala, Java och flera andra språk använder Java-plattformen som exekveringsmiljö. Om man inte bara vill köra program som andra har utvecklat, utan även utveckla egna program som fungerar i denna miljö, behöver man installera Java Development Kit (JDK). Detta utvecklingspaket innehåller flera delar, bland annat:

- Kompilatorn `javac` kompilerar Java-program till bytekod som lagras i klassfiler med filnamnsändelsen `.class`.
- Exekveringsmiljön Java Runtime Environment (JRE) med kommandot `java` som drar igång den virtuella javamaskinen (Java Virtual Machine) som kan ladda och exekvera bytekod lagrade i klassfiler.
- Programmet `jar` som packar ihop många sammanhörande klassfiler till en enda jar-fil som lätt kan distribueras via nätet och sedan köras med `java`-kommandot på alla maskiner med JRE.
- Programmet `javap` som läser klassfiler och skriver ut vad de innehåller i ett format som kan läsas av människor (ett sådant program kallas disassembler).
- I JDK ingår också en mycket stor mängd färdiga programbibliotek med stöd för nätverkskommunikation, filhantering, grafik, kryptering och en massa annat som behövs när man bygger moderna system.

Du kan läsa mer om Java och dess historik här:

[https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))

C.2.1 Kontrollera om du har JDK installerat

Öppna ett terminalfönster (se appendix A) och skriv (observera det avslutande c:et i `javac`):

```
javac -version
```

Då ska ungefär följande skrivas ut (där siffran 101 kan vara något annat):

```
javac 1.8.0_101
```

Om utskriften säger att javac saknas, installera JDK enl. nedan.

Du kanske redan har enbart Java Runtime Environment (JRE) installerad, men inte JDK. Då saknar du javakompilatorn javac m.m. och behöver installera JDK, se nedan. Du kan kolla om du har JRE genom att skriva `java -version` (alltså utan c efter java). Eller så har du redan JDK installerad men inte rätt bibliotek i din PATH; se vidare nedan ang. uppdatering av PATH.

C.2.2 Installerar JDK

Det finns flera JDK-distributioner att välja mellan, varav Oracle JDK och Azul Zulu OpenJDK är två exempel. Oracle JDK har störst spridning och är förinstallerad på LTH:s datorer. För att installera JDK på din egen dator behöver du gå igenom flera steg, varav vissa behöver anpassas efter det operativsystem du kör, enligt nedan. På kurshemsidan under "Verktyg" finns kompletterande instruktioner: <http://cs.lth.se/pgk/verktyg>

Din användaridentitet behöver ha administratörsrättigheter för att du ska kunna genomföra installationen.

Linux

För Ubuntu: läs igenom och följ sedan dessa instruktioner noga:

www.webupd8.org/2012/09/install-oracle-java-8-in-ubuntu-via-ppa.html

För andra Linux-distributioner, kör detta i terminalen (funkar även i Ubuntu, men du får med detta kommando inte Oracles aningen snabbare JVM):
`sudo apt-get install openjdk-8-jdk`

Windows/macOS

1. Installera senaste JDK från Oracle. Om du inte har installerat JDK förr på din dator så be gärna någon kurskamrat med erfarenhet av detta att assistera dig medan du följer stegen nedan.
 - (a) Surfa till Oracles hemsida för Java SE här:
<http://www.oracle.com/technetwork/java/javase/downloads/>
 - (b) Klicka på rubriken "Java SE 8u101 / 8u102" och på nästa sida klicka på knappen "Accept License Agreement" i listan under rubriken "Java SE Development Kit 8u101". (Siffrorna 101 eller 102 kan vara annorlunda om senare versioner tillkommit.)
 - (c) Välj rätt version av operativsystem (Windows x64 eller Mac OS X). Det är viktigt att du väljer x64, d.v.s 64-bitarsvarianten som gäller för alla moderna datorer.

- (d) Klicka på länken och en stor fil kommer laddas ner till din dator.
 (e) Installera när filen laddats färdigt.
2. Uppdatera PATH, så att du får tillgång till alla kommando i terminalen:
- För Windows görs detta enklast genom att ladda ner och sedan köra denna fil genom att dubbelklicka på den:
github.com/lunduniversity/introprog/raw/master/tools/windows-jdk-set-path.bat
 - För macOS, läs här:
docs.oracle.com/javase/8/docs/technotes/guides/install/mac_jdk.html
TODO!!! finn ut bästa rådet att sätta path på mac
 - Om något krånglar, be om hjälp. Om du behöver mer detaljer om PATH-uppdatering för java, läs här: java.com/sv/download/help/path.xml
 Om du kör engelska menyer byt sv mot en i adressen ovan. Du kan ta reda på vilken katalog som ska läggas in sist i din PATH genom att bläddra bland dina systemfiler och undersöka var JDK har installerats; i Windows antagligen något liknande detta (kolla exakt vilket versionsnummer du har): C:\Program Files\Java\jdk1.8.0_101\bin
3. Starta om datorn. Det är först efter att en ny användarinloggning initierats, som PATH-tilldelningen får effekt.
4. Kontrollera att javac fungerar enligt avsnitt C.2.1.

C.3 Scala

Scala använder JDK som exekveringsmiljö, men erbjuder ytterligare verktyg specifika för Scala. I utvecklingspaketet för Scala ingår bl.a. kompilatorn scalac och även ett interaktivt kommandoskal kallat Scala REPL där du kan testa din Scala-kod rad för rad och se vad som händer direkt.

De flesta av kursens övningar görs i Scala REPL, medan laborationerna kräver kompilering av lite större program.

Du hittar mer om Scalas historik och annan bakgrundsinformation här: [en.wikipedia.org/wiki/Scala_\(programming_language\)](https://en.wikipedia.org/wiki/Scala_(programming_language))

C.3.1 Installera Scala

Scala finns förinstallerat på LTH:s datorer. Du installerar Scala-kompilatorn och den interaktiva kodexperimentmiljön Scala REPL på din egen dator enligt nedan.

1. Kontrollera att du har JDK installerad enligt avsnitt C.2.1 och installera vid behov enligt avsnitt C.2.2.
2. Surfa till denna hemsida för nedladdning av Scala 2.11.8:
<http://scala-lang.org/download/2.11.8.html>

3. Klicka på ”Download” av den variant som är relevant för ditt operativsystem och spara filen:
 - (a) **Linux Ubuntu:** Filen heter `scala-2.11.8.deb` och installeras genom att dubbelklicka på filen eller via terminalkommandot:
`sudo apt install ~/Downloads/scala-2.11.8.deb`
Anpassa sökvägen ovan efter var du sparade filen.
 - (b) **Windows:** Filen heter `scala-2.11.8.msi` och installationen startas med ett dubbelklick. Följ instruktionerna. Installationsprogrammet uppdaterar även din PATH åt dig och kommandot `scala` bör fungera efter omstart.
 - (c) **macOS:** Filen heter `scala-2.11.8.tgz` och kan packas upp på lämpligt ställe med terminalkommandot `tar -xvf scala-2.11.8.tgz` och sedan är det underkatalogen `bin` som ska inkluderas i din PATH.
TODO!!! klura ut säkraste rådet för PATH-uppdatering på mac.

Kontrollera, efter ev. omstart, att terminalkommandot `scala` nu kan användas för att starta Scala REPL på din dator:

```
$ scala
Welcome to Scala 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_101).
Type in expressions for evaluation. Or try :help.

scala> val msg = "hej"
msg: String = hej

scala> println(msg)
hej

scala>
```

C.3.2 Scala Read-Evaluate-Print-Loop (REPL)

För många språk, t.ex. Scala och Python, finns det ett interaktivt program ämnat för terminalen som gör det möjligt att exekvera enstaka programrader och direkt se effekten. Ett sådant program kallas *Read-Evaluate-Print-Loop* (REPL), eftersom det läser och tolkar en rad i taget. Resultatet av evalueringen av din kod skrivs ut i terminalen och därefter är kommandoskalet redo för nästa kodrad.

Kursens övningar bygger till stor del på att du använder Scala REPL för att undersöka principer och begrepp som ingår i kursen genom dina egna kodexperiment. Även när du på labbarna utvecklar större program med en editor och en IDE, är det bra att ha Scala REPL till hands. Då kan du klistica in delar av programmet du håller på att utveckla i Scala REPL och stegvis utveckla delprogram, som till slut fungerar så som du vill.

I Scala REPL får du se typinformation för variabler och metoder, vilket är till stor hjälp när man försöker lista ut vad en kodrad innehåller. Genom att öva upp din förmåga att dra nytta av Scala REPL, kommer din produktivitet öka.

Du startar Scala REPL med kommandot `scala` och skriver Scala-kod efter prompten `scala>` och kompilering+exekvering sker när du trycker Enter.

```
$ scala
Welcome to Scala 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_101).
Type in expressions for evaluation. Or try :help.

scala> def inc(x: Int) = x + 1
inc: (x: Int)Int

scala> inc(inc(inc(1)))
res8: Int = 4

scala>
```

Med kommandot `:paste` försätter du Scala REPL i inklistringsläge (eng. *paste mode*) och du kan då med `Ctrl+V` (eller `Ctrl+Shift+V`, eller eventuellt högerklick med musen, beroende på hur ditt terminalprogram är inställt och vilket operativsystem du kör) klistra in större sjok av kod. När du med `Ctrl+D` avslutar inklistringsläget och Scala REPL tolkar alla raderna på en gång. Kommandot `:paste` kan förkortas till `:pa`, så som visas nedan. Koden mellan raderna som börjar med `//` klistrades in av användaren efter att ha kopierats från en editor i ett annat fönster.

```
scala> :pa
// Entering paste mode (ctrl-D to finish)

case class Point3D(val pos: (Int, Int, Int))
object Point3D {
  def apply(x: Int, y: Int) = new Point3D(x,y,0)
}

// Exiting paste mode, now interpreting.

defined class Point3D
defined object Point3D

scala> Point3D(1,2)
res6: Point3D = Point3D((1,2,0))

scala>
```

Många av de kortkommandon som fungerar i terminalens kommandoskal, fungerar också i Scala REPL. Gå gärna igenom listan i tabell A.1 på sidan 336, och testa vad som händer i Scala REPL. Om du tränar upp din fingerfärdighet med dessa kortkommandon, går ditt arbete i Scala REPL väsentligt snabbare.

Med kommandot `:help` får du se en lista med specialkommandon för Scala REPL, inklusive de som återfinns i tabell C.1 på sidan 349.

<i>Kommando</i>	<i>Förk.</i>	<i>Beskrivning</i>
:help	:he	visa lista med kommando och förklaringar
:paste	:pa	växla till inklistringsläge (eng. <i>paste mode</i>)
:paste <i>path</i>	:pa <i>path</i>	klistra in en hel fil, t.ex. :pa util/mio.scala
:quit	:q	avsluta Scala REPL
:require <i>path</i>	:req <i>path</i>	jar-fil till classpath, t.ex. :req lib/cslib.jar
:type	:t	visa typ med -v för ”verbose”, t.ex. :t -v 42.0
:warnings	:w	visa beskrivning av ev. varningar

Tabell C.1: Några vanliga kommandon i Scala REPL.

Appendix D

Integrerad utvecklingsmiljö

D.1 Vad är en integrerad utvecklingsmiljö?

En integrerad utvecklingsmiljö (eng. *integrated development environment, IDE*) samlar ett flertal verktyg, inklusive en avancerad **editor** (se appendix B), för att skapa, köra och testa program. Det finns flera utvecklingsmiljöer att välja mellan, som kan användas för både Scala och Java.

En IDE ger stöd för **kodkomplettering** (eng. *code completion*) där tillgängliga metoder visas i en lista och resten av ett namn kan fyllas i automatiskt efter att du skrivit de första bokstäverna i namnet. En IDE kan hjälpa dig med formatering och även skapa skelettkod utifrån **kodmallar** (eng. *code templates*). Med **felindikering** (eng. *error highlighting*) får du understrykning av vissa fel direkt i koden och ibland kan du även få hjälp med förslag på åtgärder för att rätta till enkla fel. Funktioner för **avlusning** (eng. *debugging*) hjälper dig att felsöka medan du kör din kod. Med funktioner för **omstrukturering** (eng. *refactoring*) av kod får du hjälp av editorn i samarbete med kompilatorn att göra omfattande strukturförändringar i många kodfiler samtidigt, t.ex. namnbyten med hänsyn taget till språkets synlighetsregler.

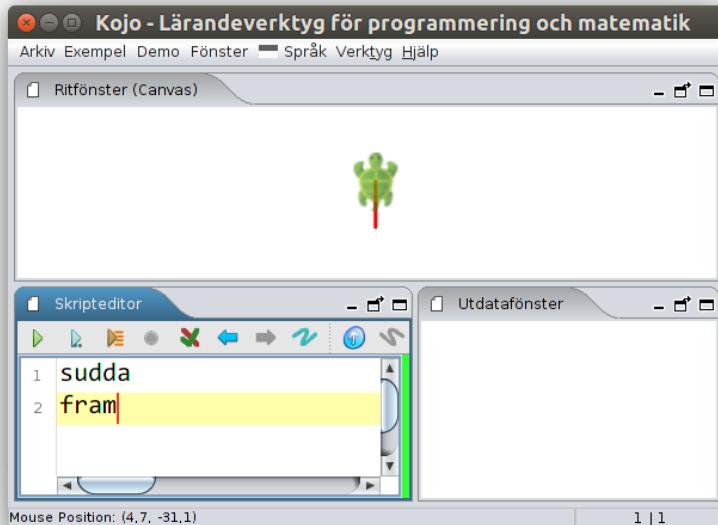
Alla dessa avancerade funktioner kan öka produktiviteten avsevärt, men samtidigt tar de tid att lära sig och en IDE kan kräva mycket datorkraft och viss väntetid jämfört med en vanlig, fristående editor. I början kan all funktionalitet upplevas som överväldigande och det kan vara svårt att hitta i alla menyer och inställningar. Ska man bara skriva ett litet, enkelt program, eller göra några mindre ändringar, är det många som föredrar en fristående, snabbstartad kodeditor före en fullfjädrad, tungrodd IDE. Å andra sidan kan en IDE med kodkomplettering vara till stor hjälp när man ska lära sig ett nytt api och experimentera med en okänd kodmassa.

I kursen använder vi flera utvecklingsmiljöer. På första labben använder vi Kojo (avsnitt D.2) som är en IDE speciellt anpassad på nybörjare. I laborationerna senare i kursen kan du välja att använda någon av de professionella utvecklingsmiljöerna Eclipse (avsnitt D.3) eller IntelliJ (avsnitt D.4). Om du inte vet vilken du ska välja, börja med att prova Eclipse.

D.2 Kojo

Kojo¹ är en integrerad utvecklingsmiljö för Scala som är speciellt anpassad för nybörjare i programmering. Kojo används i LTH:s Science Center Vattenhallen för utbildning av grundskolelärare i programmering och vid skolbesök och annan besöksverksamhet, i vilken lärare och studenter vid LTH arbetar som handledare. Kojo är fri öppenkällkod och utvecklingen leds av Lalit Pant.

Kursens första laboration genomförs med hjälp av Kojo, men Kojo kan med fördel användas som komplement till Scala REPL och annan IDE under hela kursens gång. Medan Scala REPL lämpar sig för korta kodsnuttar, och en fullfjädrad, professionell IDE har funktioner för att hantera riktigt stora programmeringsprojekt, passar Kojo bra för mellanstora program. I Kojo finns även lättillgängliga bibliotek som gör tröskeln lägre att programmera rörlig grafik och enkla spel.



Figur D.1: Den nybörjarvänliga utvecklingsmiljön Kojo för Scala på svenska.

D.2.1 Installera Kojo

Kojo är förinstallerat på LTH:s datorer och körs igång med kommandot `kojo`. För instruktioner om hur du installerar Kojo på din egen dator se här: lth.se/programvara/installera

Kojo kräver att java finns på din dator. Eftersom du behöver tillgång till JDK i kursen, är det lika bra att installera hela JDK direkt (och inte bara JRE, så som beskrivs å länken ovan); se vidare hur du gör detta i avsnitt C.2.2.

¹[en.wikipedia.org/wiki/Kojo_\(programming_language\)](https://en.wikipedia.org/wiki/Kojo_(programming_language))

D.2.2 Använda Kojo

När du startar Kojo första gången, välj ”Svenska” i språkmenyn och starta om Kojo. Därefter fungerar grafikfunktionerna på svenska enligt tabell D.1. När du startat om Kojo inställt på Svenska ser programmet ut ungefär som i figur D.1 på sidan [352](#).

Det finns ett antal användbara kortkommando som du hittar i menyerna i Kojo. Undersök speciellt Ctrl+Alt+Mellanslag som ger autokomplettering baserat på det du börjat skriva.

Tabell D.1: Några av sköldpaddans funktioner. Se även lth.se/programmera

Svenska / Engelska	Vad händer?
sudda clear	Ritfönstret suddas
fram forward(25)	Paddan går framåt 25 steg.
fram(100) forward(100)	Paddan går framåt 100 steg.
höger right(90)	Paddan vrider sig 90 grader åt höger.
höger(45) right(45)	Paddan vrider sig 45 grader åt höger.
vänster left(90)	Paddan vrider sig 90 grader åt vänster.
vänster(45) left(45)	Paddan vrider sig 45 grader åt vänster.
hoppa hop	Paddan hoppar 25 steg utan att rita.
hoppa(100) hop(100)	Paddan hoppar 100 steg utan att rita.
hoppaTill(100, 200) jumpTo(100, 200)	Paddan hoppar till läget (100, 200) utan att rita.
gåTill(100, 200) moveTo(100, 200)	Paddan vrider sig och går till läget (100, 200).
hem home	Paddan går tillbaka till utgångsläget (0, 0).
öster setHeading(0)	Paddan vrider sig så att nosen pekar åt höger.
väster setHeading(180)	Paddan vrider sig så att nosen pekar åt vänster.
norr setHeading(90)	Paddan vrider sig så att nosen pekar uppåt.

söder	Paddan vrider sig så att nosen pekar neråt.
setHeading(-90)	
mot(100,200)	Paddan vrider sig så att nosen pekar mot läget
towards(100, 200)	(100, 200)
sättVinkel(90)	Paddan vrider nosen till vinkeln 90 grader.
setHeading(90)	
vinkel	
heading	Ger vinkelvärdet dit paddans nos pekar.
sakta(5000)	
setAnimationDelay(5000)	Gör så att paddan ritar jättesakta.
suddaUtdata	
clearOutput	Utdatafönstret suddas.
utdata("hej")	
println("hej")	Skriver texten hej i utdatafönstret.
val t = indata("Skriv")	Väntar på inmatning efter ledtexten Skriv och
val t = readln("Skriv:")	sparar den inmatade texten i t.
textstorlek(100)	Paddan skriver med jättestor text nästa gång
setPenFontSize(100)	du gör skriv.
båge(100, 90)	Paddan ritar en båge med radie 100 och vinkel
arc(100, 90)	90.
cirkel(100)	Paddan ritar en cirkel med radie 100.
circle(radie)	
synlig	
visible	Paddan blir synlig.
osynlig	
invisible	Paddan blir osynlig.
läge.x	
position.x	Ger paddans x-läge
läge.y	
position.y	Ger paddans y-läge
pennaNer	
penDown	Sätter ner paddans penna så att den ritar när
pennaUpp	den går.
penUp	Sänker paddans penna så att den INTE ritar
pennanÄrNere	när den går.
penIsDown	Kollar om pennan är nere eller inte.
färg(rosa)	
setPenColor(pink)	Sätter pennans färg till rosa.
fyll(lila)	
setFillColor(purple)	Sätter ifyllnadsfärgen till lila.

fyll(genomskinlig)	Gör så att paddan inte fyller i något när den ritar.
setFillColor(noColor)	
bredd(20)	Gör så att pennan får bredden 20.
setPenThickness(20)	
sparaStil	Sparar pennans färg, bredd och fyllfärg.
saveStyle	
laddaStil	Laddar tidigare sparad färg, bredd och fyllfärg.
restoreStyle	
sparaLägeRiktning	Sparar pennans läge och riktning
savePosHe	
laddaLägeRiktning	Laddar tidigare sparad riktning och läge
restorePosHe	
siktePå	Sätter på siktet.
beamsOn	
sikteAv	Stänger av siktet.
beamsOff	
bakgrund(svart)	Bakgrundsfärgen blir svart.
setBackground(black)	
bakgrund2(grön,gul)	Bakgrund med övergång från grönt till gult.
setBackgroundV(green, yellow)	
upprepa(4){fram; höger}	Paddan går fram och svänger höger 4 gånger.
repeat(4){forward; right}	
avrunda(3.99)	Avrundar 3.99 till 4.0
slumptal(100)	Ger ett slumptal mellan 0 och 99.
slumptalMedDecimaler(100)	Ger ett slumptal mellan 0 och 99.99999999
systemtid	Ger nuvarande systemklocka i sekunder.
räknaTill(5000)	Kollar hur lång tid det tar för din dator att räkna till 5000.

Scala-koden för den svenska paddans api finns här:

bitbucket.org/lalit_pant/kojo/src/tip/src/main/scala/net/kogics/kojo/lite/i18n/svInit.scala

D.3 Eclipse och ScalaIDE

Eclipse² är en professionell IDE som stödjer många olika programmeringsspråk. Eclipse är skriven i Java och bygger vidare på ett utvecklingsprojekt som initierades av IBM. Eclipse är ett fritt och öppet projekt som numera kontrolleras av en oberoende stiftelse.

Till Eclipse finns en insticksmodul (eng. *plug-in*) som kallas ScalaIDE och erbjuder stöd för Scala med tillhörande standardbibliotek.

Eclipse är en omfattande och avancerad programmeringsmiljö med många funktioner och inställningar. Det finns även en omfattande uppsättning insticksmoduler och tilläggsprogram som underlättar utveckling av t.ex. webbprogram, databaser och mycket annat.

I detta avsnitt ges länkar till installation samt tips om hur du kommer igång med att använda Eclipse och ScalaIDE. Det går ganska snabbt att lära sig grunderna, men det kräver en viss ansträngning att lära sig de mer avancerade funktionerna. Det finns omfattande resurser på nätet som hjälper dig vidare.

D.3.1 Installera Eclipse Mars och ScalaIDE

Eclipse med ScalaIDE är förinstallerat på LTH:s datorer och startas med kommandot `scalaide` i ett terminalfönster.

ScalaIDE fungerar med Eclipse-versionerna *Luna* och *Mars* (men i skrivande stund fungerar ScalaIDE ännu *inte* med den allra senaste versionen kallad *Neon*).

För att installera ScalaIDE på din egen dator, följ nedan instruktioner:

1. Kontrollera enligt avsnitt C.2.1 att du har java installerat och installera vid behov JDK enligt avsnitt C.2.2.
2. Installera Eclipse version **Mars**, varianten för **Java Developers** som återfinns på denna sida:
<https://www.eclipse.org/downloads/packages/release/Mars/2>
 som är den *andra* varianten i listan (alltså inte Java EE). Följ dessa steg:
 - (a) Klicka på den **64-bit**-variant som passar ditt operativsystem.
 - (b) Filen som laddas ner heter något som liknar (beroende på OS):
`eclipse-java-mars-2-win32-x86_64.zip`
 Det kan ta lång tid att ladda ner filen som är på ca 170MB. Om du klickar på *"select a mirror"* kan du välja en svensk sajt för att ladda ner snabbare.
 - (c) Dubbelklicka på filen för att packa upp den, vilket kan ta många minuter. Du får, när uppackningen är klar, ett bibliotek med en fungerande Eclipse-installation som du kan placera var du vill. Kör du Windows, lägg den förslagsvis här:
`C:\eclipse\eclipse-java-mars-2-win32-x86_64`

²[en.wikipedia.org/wiki/Eclipse_\(software\)](https://en.wikipedia.org/wiki/Eclipse_(software))

- (d) för Ubuntu Linux finns kompletterande installationsanvisningar här, som ger dig en ikon i app-menyn m.m.:
<http://askubuntu.com/questions/26632/how-to-install-eclipse>

3. Installera Scala IDE inifrån³ Eclipse enligt nedan steg:

- (a) Starta Eclipse, t.ex. genom att köra igång den exekverbara filen som ligger i underbiblioteket **eclipse**, i Windows heter den **eclipse.exe** medan den exekverbara filen i Linux heter **eclipse** utan filändelse.
- (b) Välj i frågerutan som dyker upp, någon plats för *workspace* (kvittar vilken just nu, kan ändras senare).
- (c) Klicka på menyn *Help* → *Install new software*.
- (d) Klicka på *Add*-knappen till höger och skriv:
"ScalaIDE for Scala 2.11" i *Name*-fältet och ange denna adress i *Location*-fältet:
<http://download.scala-ide.org/sdk/lithium/e44/scala211/stable/site>
och klicka *OK*.
- (e) Du får nu upp en lista med alternativ. Kryssa för alternativet
 Scala IDE for Eclipse
och klicka *Next* och sedan *Next* igen och acceptera licensvillkoren och klicka *Finish*.
- (f) Låt installationen ta sin tid och starta sedan om Eclipse när installationen är färdig.
- (g) När Eclipse är igång igen visas en dialog som föreslår att du ska köra *Setup Diagnostics*. Gör detta och välj *Use recommended default settings*. Ändra även i filen **eclipse.ini** för höja den övre minnesgränsen. Det gör du genom att ändra på den rad i filen som börjar med **-Xmx**. Hur mycket du ska tillåta som max beror på hur mycket minne du har, men ge minst 1 gigabyte för smidig körning, genom att skriva så här på relevant rad i filen **eclipse.ini**:
-Xmx1G
- (h) Kompletterande information finns här, inklusive en video som visar installationsproceduren och hur man kommer igång med ett "hello world"-program:
<http://scala-ide.org/download/current.html>

I nästa avsnitt beskrivs några rekommenderade anpassningar som du kan göra bland de omfattande inställningsmöjligheterna för Eclipse.

³Det finns på ScalaIDE-hemsidan möjlighet att ladda ner en Eclipse-variant med färdiginstallerad ScalaIDE-plugin, men då får du i skrivande stund den gamla versionen Eclipse *Luna*, varför du rekommenderas att, enligt instruktionerna här, själv installera ScalaIDE inifrån Eclipse *Mars*, som är den senaste Eclipse-versionen för vilken ScalaIDE fungerar.

D.3.2 Anpassa Eclipse och ScalaIDE

Förutom maxminneshöjningen i filen `eclipse.ini`, som finns i installationskatalogen för Eclipse, till minst `-Xmx1G` (se föregående avsnitt), är det bra att göra några ytterligare anpassningar av Eclipse och ScalaIDE för att få en snabbare och smidigare utvecklingsmiljö. Du hittar inställningarna i menyn `Window → Preferences → ...` uppe till höger i Eclipse-fönstret.

1. *Window → Preferences → General →*
Markera **Show Heap Status** så får du se minnesanvändningen i en liten ruta i nederdelen av fönstret, vilket hjälper dig att upptäcka om minnesbegränsningen i filen `eclipse.ini` är en flaskhals vid stora projekt och många öppna fönster. Klicka sedan **Apply** längst ner.
2. *Window → Preferences → General → Editors → Perspective →*
Markera **Scala** i listan med perspektiv och klicka på knappen **Make default** till höger och sedan på knappen **Apply** längst ner.
3. *Window → Preferences → General → Editors → TextEditors →*
Markera **Insert spaces for tabs** så att du slipper specialtecken som kan tolkas olika av olika editorer. Klicka sedan **Apply** längst ner.
4. *Window → Preferences → General → Editors → TextEditors → Spelling →* Avmarkera **Enable spell checking** för att slippa att svenska namn och svenska kommentarer markeras som felstavade. Om du senare jobbar med ett projekt helt på engelska, kan du med fördel markera denna kryssruta igen. Klicka sedan **Apply** längst ner.
5. *Window → Preferences → General → Editors → Webbrowser →*
Markera **Use external web browser** för att köra din vanliga webbläsare när du klickar på länkar. Klicka sedan **Apply** längst ner.
6. *Window → Preferences → Scala → Compile →*
I fliken **Standard** markera dessa kryssrutor för att få extra varningar:
 deprecation varnar vid användning av föråldrad kod som snart utgår
 feature påminner om import vid användning av avancerad kod
 unchecked ger tips vid speciella problem med generiska typer
och klicka sedan på knappen **Apply** längst ner.
7. *Window → Preferences → Java → Compiler → Errors / Warnings →*
Veckla ut listan **Potential programming problems** och sätt **Resource leak** till alternativet **Ignore**, så slipper du varningar vid användning Scanner i Java. Klicka sedan **Apply** längst ner.

Ovan anpassningar är rekommenderade men inte nödvändiga och du kan gärna välja att göra andra anpassningar som passar just dig. Skriv då gärna ner vilken inställning du ändrat, så att du hittar tillbaka om du ångrar dig.

Du hittar tips om fler inställningar för att anpassa ScalaIDE här:

<http://scala-ide.org/docs/current-user-doc/advancedsetup>

D.3.3 Använda Eclipse och ScalaIDE

Ett grundläggande koncept i Eclipse är **workspace**. Ett workspace utgör ett arbetsområde kopplat till en katalog i ditt filsystem där du kan arbeta med ett eller flera **projekt**. Ett projekt innehåller i sin tur dina källkodsfiler och klassfiler etc. i en specifik katalogstruktur som Eclipse skapar när du editerar, kompilerar och kör dina projekt.

Starta och välja workspace

När du startar Eclipse måste du välja vilket workspace du vill använda innan du kommer vidare. När du kör igång Eclipse första gången, klicka OK enligt det förslag som ges. Du kan senare växla workspace genom menyn *File → Switch Workspace*. Om katalogen du anger inte redan finns, kommer den att skapas och initieras med de filer Eclipse behöver.

I figur D.2 visas välkomstfliken i Eclipse med sina länkar till funktionsöversikt och olika handledningar. Stäng välkomstfliken genom att klicka på flikens kryss eller på ikonen *Workbench*. Då kommer du vidare till den normala arbetsytan i Eclipse. Du kan få tillbaka välkomstfliken igen via menyn *Help → Welcome*.



Figur D.2: Välkomstfliken för Eclipse, som nås via menyn *Help → Welcome*. Gå vidare genom att klicka på *Workbench*.

Välja perspektiv och visa olika vyer

Eclipse-fönstret kan innehålla många underfönster i olika flikar, så kallade **views** eller **vyer**, som kan arrangeras på olika vis efter hur du vill ha dem.



Figur D.3: Arbetsytan i Eclipse. Du kan växla mellan Scala- och Java-perspektivet genom att klicka på perspektivvalsknappen.

Vilka vyer som syns och hur de placeras beror på vilket s.k. **perspective** som är aktivt. Figur D.3 visar arbetsytan med olika vyer i Java-perspektivet.

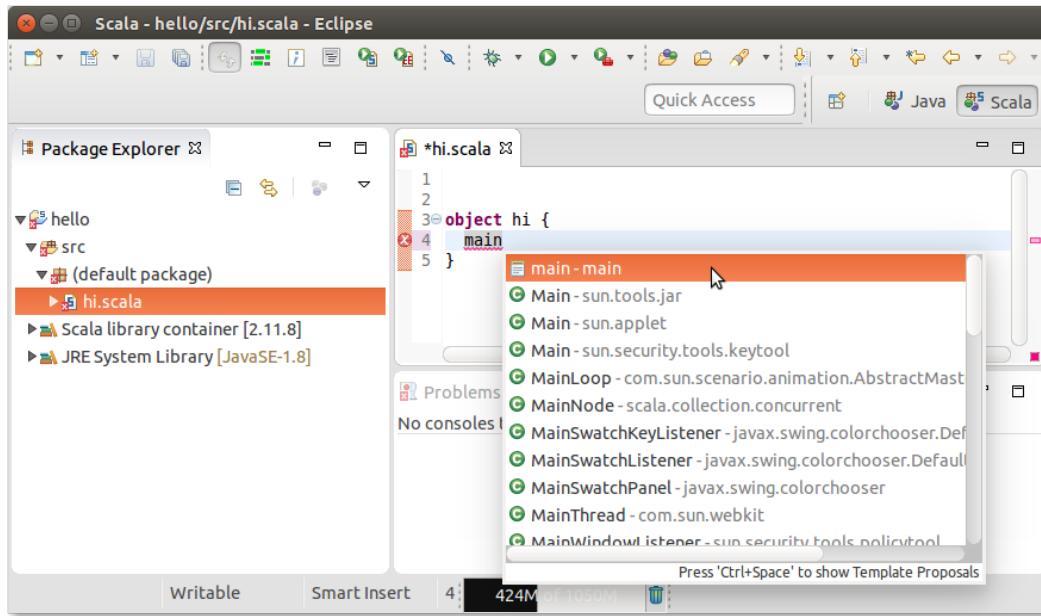
Du kan byta till Scala-perspektivet genom att trycka på eller genom menyn *Window* → *Perspective* → *Open Perspective* → *Other...* → *Scala*. Du kan anpassa inställningarna så att Scala blir *default perspective*, se steg 2 i avsnitt D.3.2 på sidan 358.

Stäng vyerna *Task List* och *Outline* om du vill ha mer plats till de övriga vyerna för paketnavigering, editering och utdata. Du kan öppna stängda vyer igen genom menyn *Window* → *Show View*.

Hello World

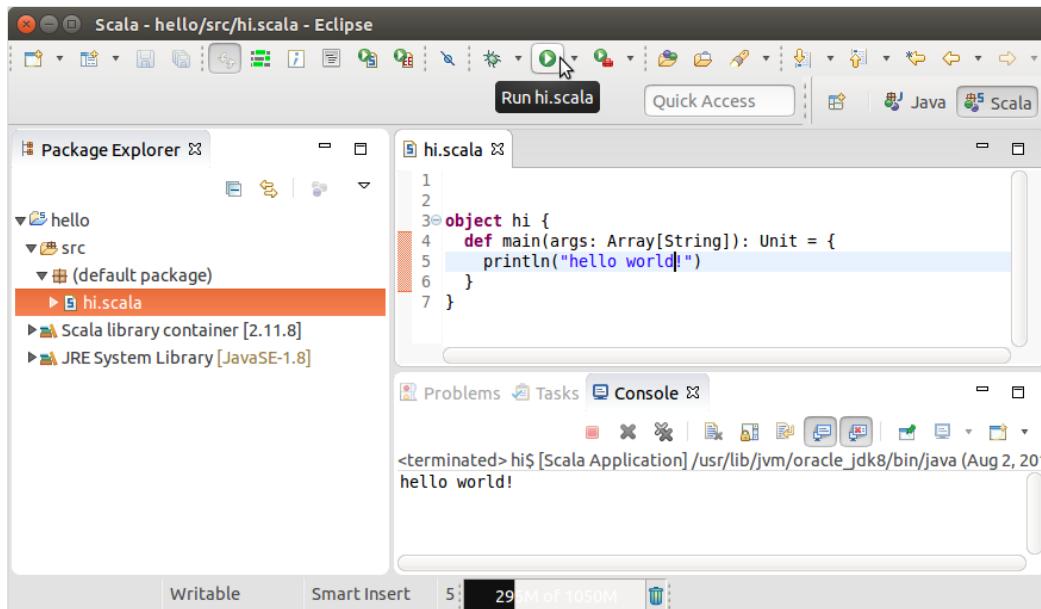
Efter att du öppnat Eclipse med ScalaIDE i ett tomt workspace och valt Scala-perspektivet enligt föregående avsnitt, kan du skapa ditt första projekt med ett "Hello World"-program enligt stegen nedan.

1. Högerklicka i *Package Explorer* och välj *New* → *Scala Project*, varefter en dialogruta visas.
2. Fyll i namnet `hello` i fältet *Project Name* och klicka **Finish**.
3. Högerklicka igen i *Package Explorer* och välj *New* → *Scala Object*, varefter en ny dialogruta visas.
4. Fyll i namnet `hi` i fältet *Project Name* och klicka **Finish**.
5. Du får nu i editorvyn ett kodskellet med **object** `hi`.



Figur D.4: Aktivera kodkomplettering med Ctrl+Mellanslag efter ordet main.

6. Börja skriv `main` som visas i figur D.4 och tryck Ctrl+Mellanslag för att aktivera kodkomplettering (eng. *code completion*). Då får du upp en lista med alternativ. Välj det översta alternatiivet `main` varefter ett kodskellet med en `main`-metod klistras in automatiskt i din kod.
7. Fyll i lämplig utskriftstext i ett `println`-anrop så att din `main`-metod blir så som visas i editorfliken i figur D.5.



Figur D.5: Skriv klart `main`-metoden och kör ditt program med play-knappen.

8. Kör ditt program genom att trycka på den gröna play-knappen, som

muspekaren i figur D.5 pekar på. Du kan också trycka F11 för att köra igång din app, efter att du vid första körningen i dialogen *Select Preferred Launcher* markerat **Use configuration specific settings** och valt alternativet *Scala Application (new debugger) Launcher*.

Ladda ner kursens workspace och importera i Eclipse

Det finns en zip-fil med ett workspace med projekt för flera av kursens laborationer som du kan ladda ner och importera i Eclipse. Följ stegen nedan.

1. Ladda ner kursens workspace här: <http://cs.lth.se/pgk/ws>
2. Packa upp filen på lämpligt ställe.
3. Starta Eclipse med ScalaIDE-plugin (se startinstruktioner på sidan 359).
4. Växla workspace till biblioteket du nyss packade upp, ungefär som i figur D.6 och klicka **OK**.



Figur D.6: Öppna kursens workspace genom att bläddra till biblioteket där du packade upp filen som du laddat ned från: <http://cs.lth.se/pgk/ws>

5. Stäng välkomstfliken för att komma vidare till workbench (se figur D.2 på sidan 359). Det ser då ut ungefär som i figur D.3 på sidan 360. Det syns ännu inget i *Package Explorer* då vi ännu inte importerat något projekt.
6. Innan du går vidare, säkerställ att du har Scala-perspektivet aktiverast. Du kan växla till Scala-perspektivet genom att trycka på  eller genom menyn *Window → Perspective → Open Perspective → Other... → Scala*. Du kan anpassa inställningarna så att Scala blir *default perspective*, se steg 2 i avsnitt D.3.2 på sidan 358.

7. Högerklicka i *Package Explorer* och välj *Import...*, se Fig. D.7, eller välj menyn *File → Import....*

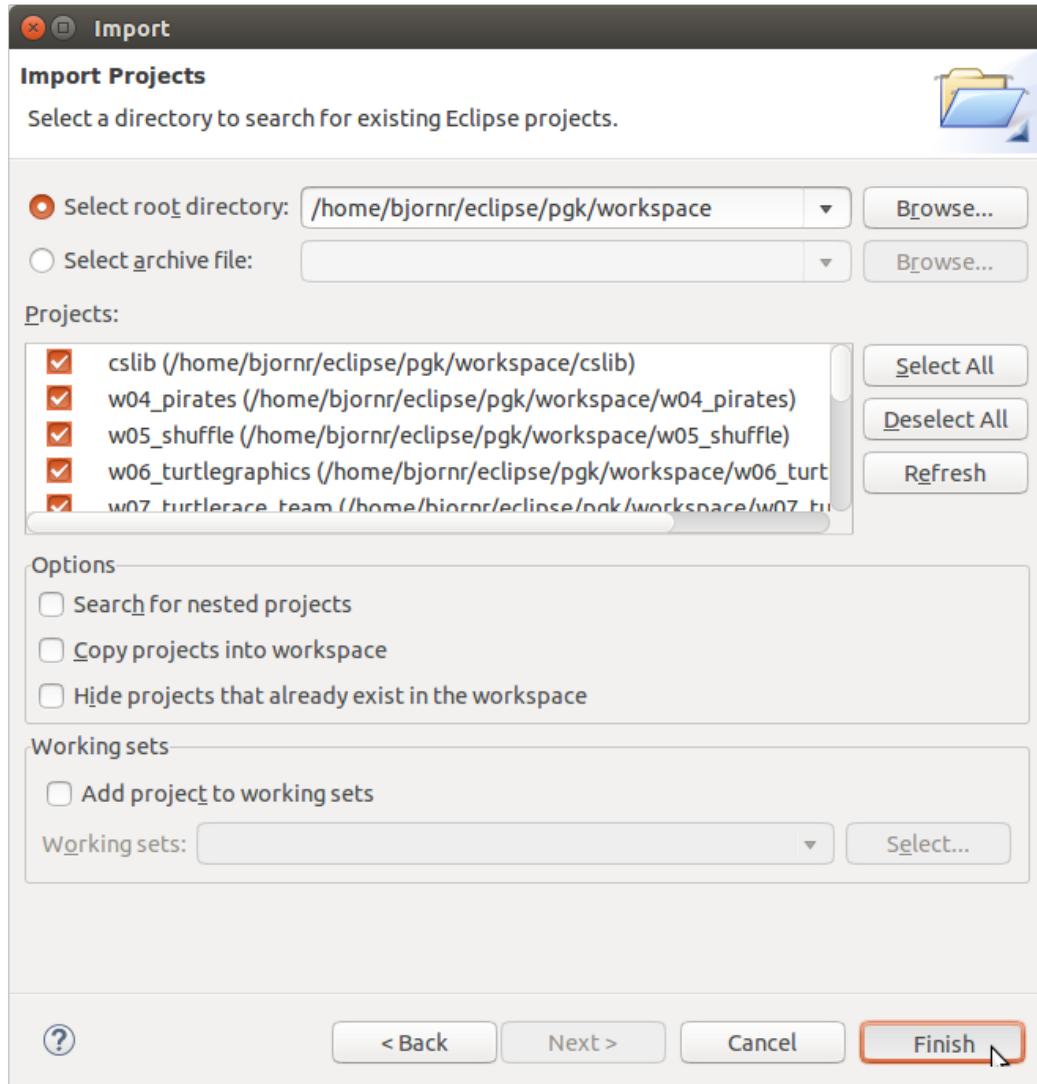


Figur D.7: Välj *Import*-menyn för att importera existerande projekt.

8. Nu öppnas *Import*-dialogen som visas i figur D.8. Öppna mappen *General*, markera **Existing Projects into Workspace** och klicka **Next**.



Figur D.8: Välj att importera existerande projekt under *General*.



Figur D.9: Välj **Select Root Directory** och klicka **Browse**.

9. Nu kommer ytterligare ett dialogfönster som visas i figur D.9. Med **Select Root Directory** markerad kan du klicka **Browse** för att ange workspace-mappen i ännu en dialog där du bara ska trycka **Ok** utan att välja underbibliotek till workspace. När det är klart ska det se ut som i figur D.9 där alla Eclipse-projekt **cslib**, **w04_pirates**, etc. är markerade. Klicka sedan **Finish**.
10. Följ "Hello World"-instruktionerna på sidan 360 och skapa programmet som visas i figur D.10, genom att veckla ut projektet **w04_pirates**, markera och högerklicka på paketet **priates**, och välja *New → Scala Object*.
11. Om du får problem, fråga någon som känner till Eclipse om hjälp. Det finns även mycket hjälp på nätet, se till exempel:
stackoverflow.com/questions/8522149/eclipse-not-recognizing-scala-code



Figur D.10: Skapa ett *New → Scala Object* med kod enligt bilden.

D.4 IntelliJ IDEA

IntelliJ IDEA⁴ är en professionell IDE som stödjer många olika programmeringsspråk. IntelliJ är skriven i Java och utvecklas av det tjeckiska företaget JetBrains.

IntelliJ IDEA finns i två varianter: en gratis gemenskapsvariant med öppenkällkodslicens (eng. *Community edition*), samt en betalvariant med sluten källkod och support-tjänster.

Till IntelliJ IDEA finns en insticksmodul (eng. *plug-in*) som erbjuder stöd för Scala med tillhörande standardbibliotek..

IntelliJ IDEA är en omfattande och avancerad programmeringsmiljö med många funktioner och inställningar. Det finns även en omfattande uppsättning insticksmoduler och tilläggsprogram som underlättar utveckling av t.ex. mobilappar, webbprogram, databaser och mycket annat.

I detta avsnitt ges länkar till installation samt tips om hur du kommer igång med att använda IntelliJ IDEA med Scala. Det går ganska snabbt att lära sig grunderna, men det kräver en viss ansträngning att lära sig de mer avancerade funktionerna. Det finns omfattande resurser på nätet som hjälper dig vidare.

Google tillkännagav 2013 att företaget övergår från Eclipse till IntelliJ som den officiellt understödda utvecklingsmiljön för Android och 2014 lanserades utvecklingsmiljön AndroidStudio⁵ som bygger vidare på IntelliJ.

D.4.1 Installera IntelliJ med Scala-plugin

IntelliJ med Scala-plug-in är förinstallerat på LTH:s datorer och startas med kommandot `idea` i ett terminalfönster.

- För Ubuntu Linux finns ett färdigt paket som du kan installera med dessa kommandon i terminalen:

```
sudo add-apt-repository ppa:mmk2410/intellij-idea-community
sudo apt-get update
```

Mer information om denna ppa finns här:

<https://launchpad.net/~mmk2410/+archive/ubuntu/intellij-idea-community>

- För Windows och Mac: ladda ner och kör installationsfil för ditt operativsystem för den öppna varianten kallad **Community** här:
<https://www.jetbrains.com/idea/download/>
Följ instruktionerna som ges av installationsprogrammet.

D.4.2 Anpassa IntelliJ

Första gången du kör igång IntelliJ får du ett antal frågor om vilka anpassningar du vill göra. Följ instruktionerna steg för steg enligt nedan.

⁴en.wikipedia.org/wiki/IntelliJ_IDEA

⁵en.wikipedia.org/wiki/Android_Studio

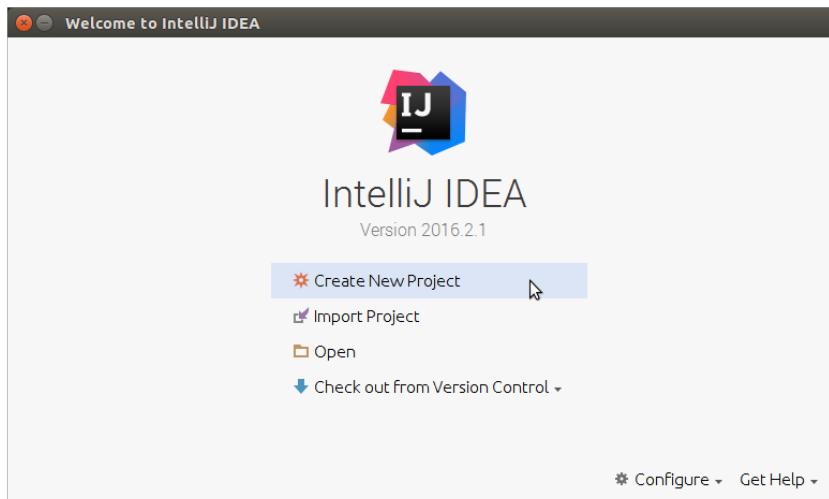
1. **UI Theme.** Denna dialog gäller utseende på gränssnittet. Det tema som kallas *Dracula* är en populär variant med nedtonade färger anpassade för att vara skonsamma mot ögonen. Klicka **Next** när du valt tema.
2. **Default plugins.** Denna dialog gäller inställningar av befintliga insticksmoduler. Dessa inställningar fungerar bra som de är. Klicka **Next**.
3. **Featured plugins.** I rutan för **Plugin for Scala language support** Klicka **Install** och låt installationen av Scala fullbordas.
4. Klicka därefter **Start using IntelliJ IDEA**.
5. I välkomstfönstrets nedre hörn, välj *Configure → Settings* och överväg om du vill göra följande lämpliga men ej nödvändiga inställningar.
 - (a) I fliken *Editor → General* markera **Change font size (Zoom) with Ctrl+Mouse Wheel** för att lätt kunna ändra textstorlek i editorn. Klicka **Apply** nere till höger.
 - (b) I fliken *Editor → Inspections* och välj *Spelling* i högra listan. Avmarkera **Typo** för att undvika att svenska ord blir markerade som felstavade. Klicka **Apply** nere till höger.
 - (c) I fliken *Editor → File and Code Templates* och under fliken *Files* i högra listan: för varje Scala-filtyp (Scala Class, Scala Trait, Scala Object, ...) ta bort de initiala raderna i mallen som börjar med # för att slippa onödiga kommentarer i koden när du skapar nya filer. Klicka **Apply** nere till höger.

Du kan också göra ovan och liknande anpassningar senare genom menyn *File → Settings...*

D.4.3 Använda IntelliJ

Skapa ett nytt projekt

När du startar IntelliJ IDEA utan förvalt projekt visas välkomstskärmen i figur D.11. Klicka på *Create New Project*, varefter dialogen i figur D.13 visas. Följ stegen enligt nedan.



Figur D.11: Välkomstfönstret för IntelliJ IDEA.

1. I dialogen **New Project** ska du ge projektet ett namn och välja körmiljö för ditt projekt. Ge projektet namnet `hello` enligt figur D.13.
2. Välj sedan att skapa ett Scala-projekt genom att markera **Scala** enligt figur D.12 och klicka **Next**.



Figur D.12: Välj att skapa ett Scala-projekt.

3. Välj sedan **Project SDK** genom att klicka på **New...**, välja *JDK* och sedan veckla ut fliken *JVM* och välja `Oracle_jdk8` och klicka **OK** enligt figur D.13.
4. Välj sedan **Scala SDK** genom att klicka på **Create...**, markera raden med *System 2.11.8* och klicka **OK** enligt figur D.13.

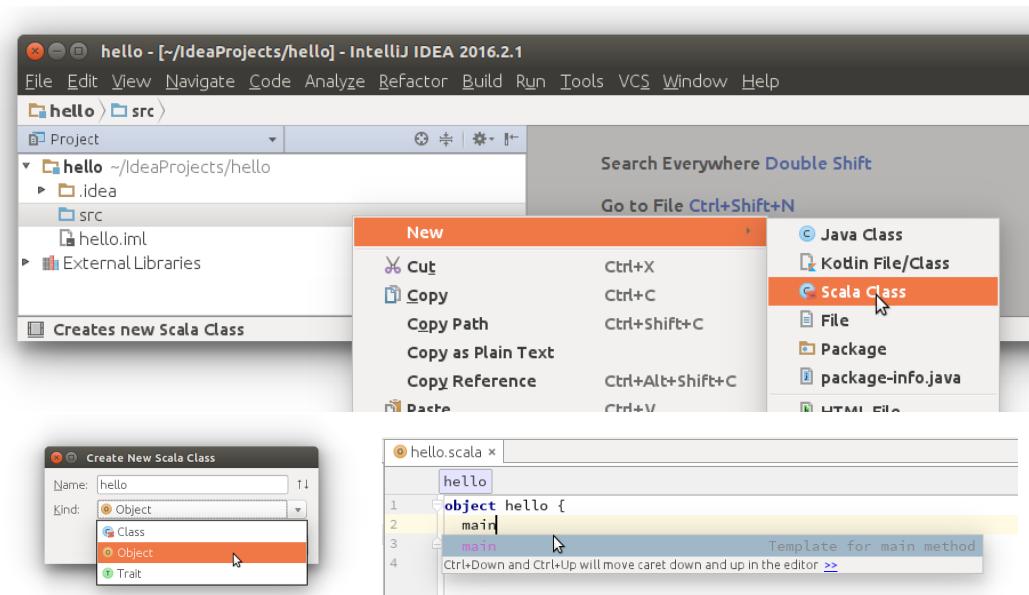
5. Avsluta med att klicka **Finish** i dialogen **New Project**.



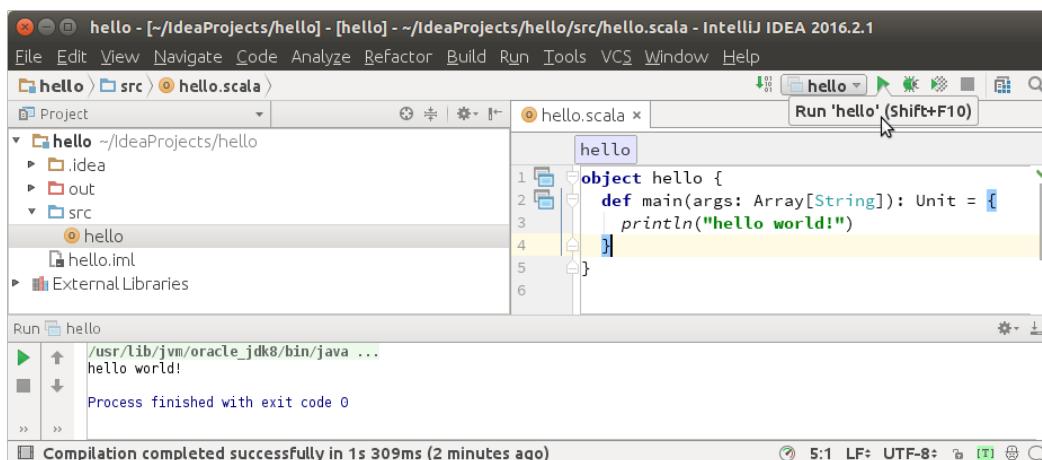
Figur D.13: Namnge ditt projekt och ställ in körmiljön för JVM och Scala genom att klicka på **New...** och **Create...**

6. Du får nu ett projektfönster som liknar det i figur D.14 på sidan 370.
7. Veckla ut ditt projekt och högerklicka på `src` och välj *New → Scala Class*. Välj sedan **Object** i dialogen **Create New Scala Class** och klicka **OK**, enligt figur D.14.
8. Du får nu upp ett editorfönster med koden för objektet `hello`. Skriv ordet `main` inuti objektet och tryck TAB för att aktivera kodkomplettering. En mall för `main`-metoden klistras då in i objektet.
9. Skriv kod så att det ser ut som i editorfönstret i figur D.15 på sidan 370.
10. Kör igång ditt program genom att klicka på play-knappen eller genom att trycka Shift+F10. Om play-knappen är initialt är grå i stället för grön, välj menyn *Run → Run....*

Mer information om hur du använder Scala-plugin för IntelliJ finns här:
confluence.jetbrains.com/display/SCA/Scala+Plugin+for+IntelliJ+IDEA



Figur D.14: Högerklicka på mappen `src` och välj *New* → *Scala Class* och skapa ett nytt Scala-objekt med `main`-metod. Aktivera kodkomplettering i editorn efter ordet `main` med TAB.



Figur D.15: Kör ditt program med play-knappen eller *Run* → *Run....*

Ladda ner kursens workspace och importera i IntelliJ IDEA

Det finns en zip-fil med ett workspace med projekt för flera av kursens laborationer som du kan ladda ner och importera i Eclipse. Följ stegen nedan.

1. Ladda ner kursens workspace här: <http://cs.lth.se/pgk/ws>
2. Packa upp filen på lämpligt ställe.
3. Starta IntelliJ. Om du redan har ett projekt igång välj menyn *File* → *Close project* så kommer du tillbaka till välkomstfönstret. Välj **Import Project** så som visas i figuren **D.16**.

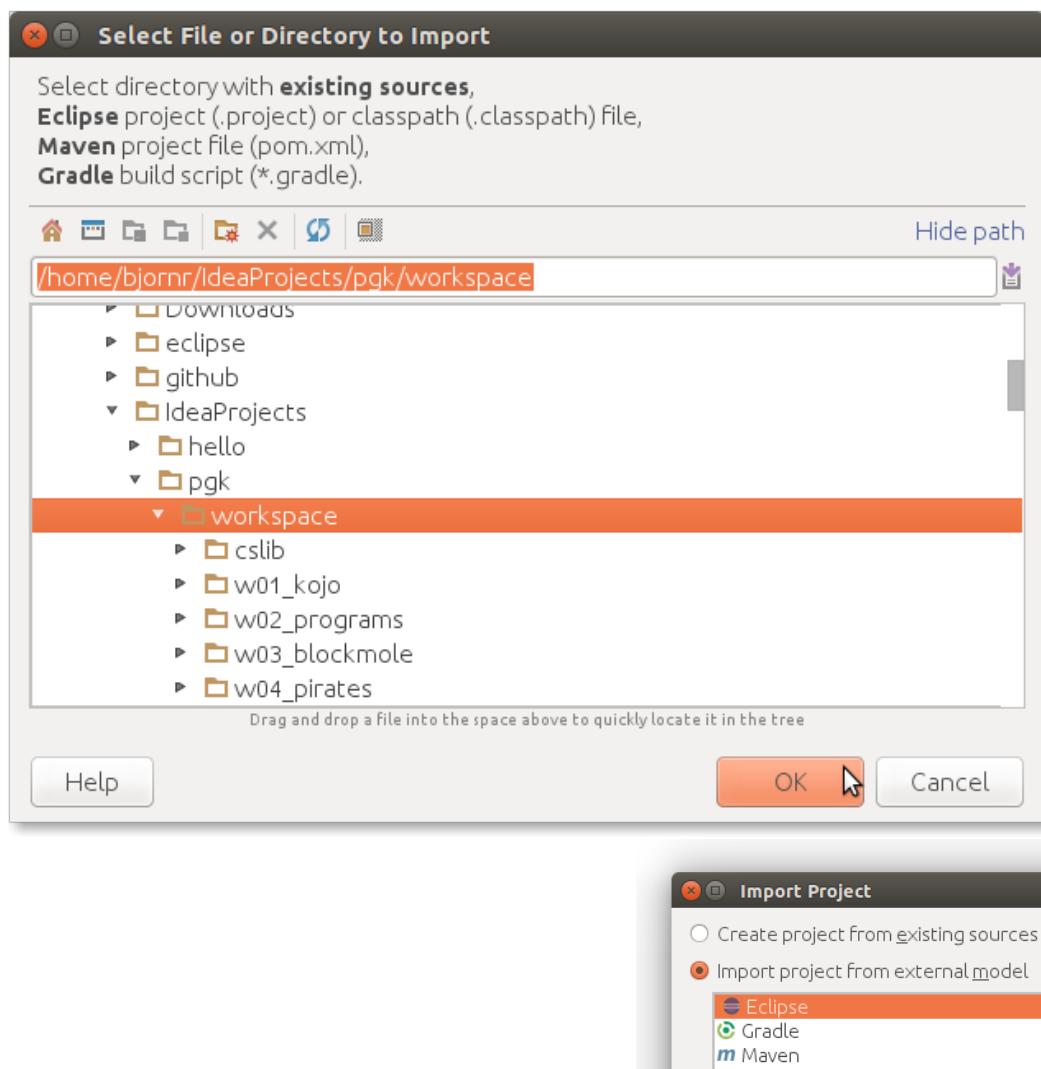


Figur D.16: Välj *Import Project* efter att du stängt ev. öppna projekt.

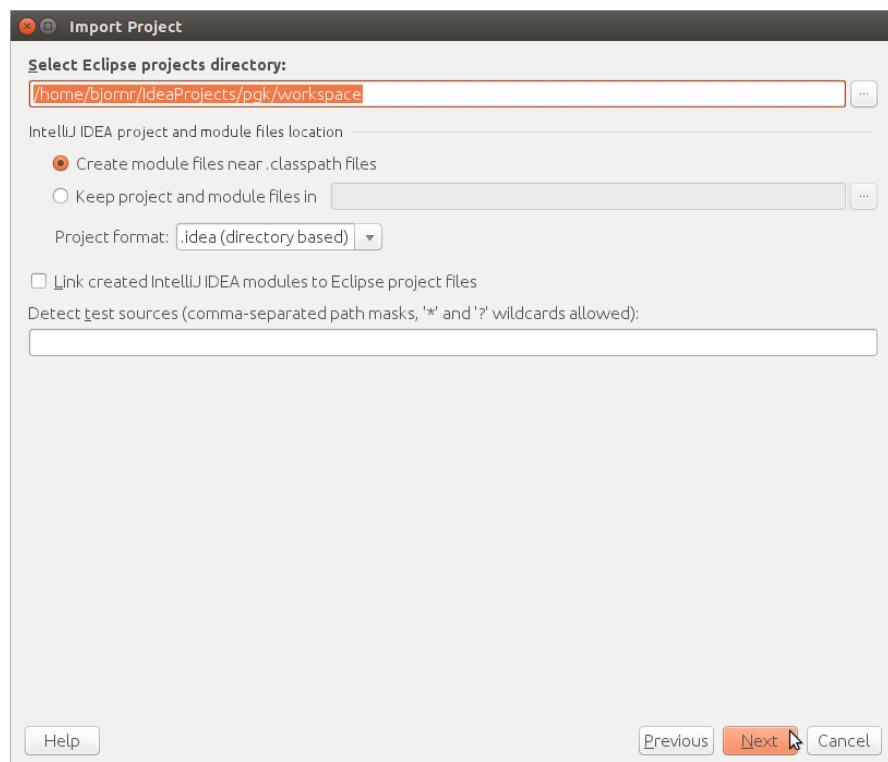
4. Bläddra dit du packat upp workspace och markera denna folder i likhet med figur **D.17** och klicka **OK**. I den efterföljande dialogen välj **Eclipse** och klicka **Next**.
5. Klicka **Next** igen i dialogen som liknar figur **D.18** på sidan **373**, där mappen du valt är förvalt.
6. Klicka **Next** igen enligt figur **D.19** på sidan **373**. Alla tillgängliga Eclipse-projekt ska vara markerade.
7. Klicka **Finish** enligt figur **D.19** med förifylld text oförändrad.
8. Bläddra fram filen `PirateSpeech.scala` och öppna den med ett dubbelklick. Klicka på länken **Setup Scala SDK** uppe till höger enligt figur **D.21** på sidan **374**. I efterföljande dialog kontrollera att `scala-sdk-2.11.8` är förvalt och klicka **OK**.

- Lägg till testutskrift enligt rad 7 i figur D.22 på sidan 375. Testkör genom att välja menyn *Run* → *Run..* eller tryck Alt+Shift+F10 och sedan välja **PirateSpeech**. Kontrollera att utskriften i utskriftsfönstret ser ut som förväntat.

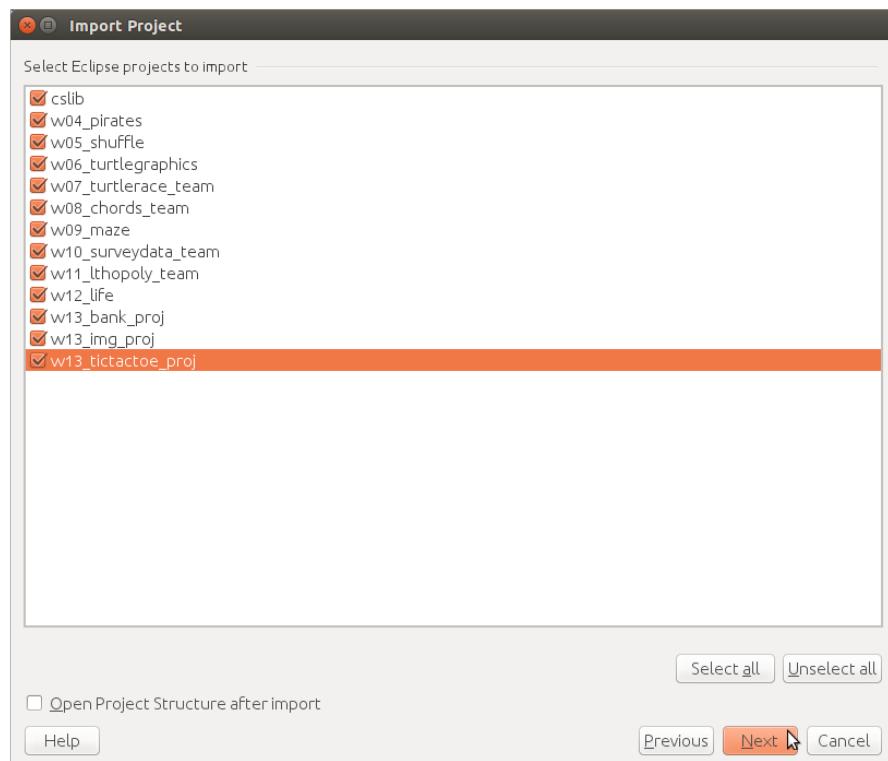
Om du får problem på vägen, be någon med erfarenhet av IntelliJ om hjälp.



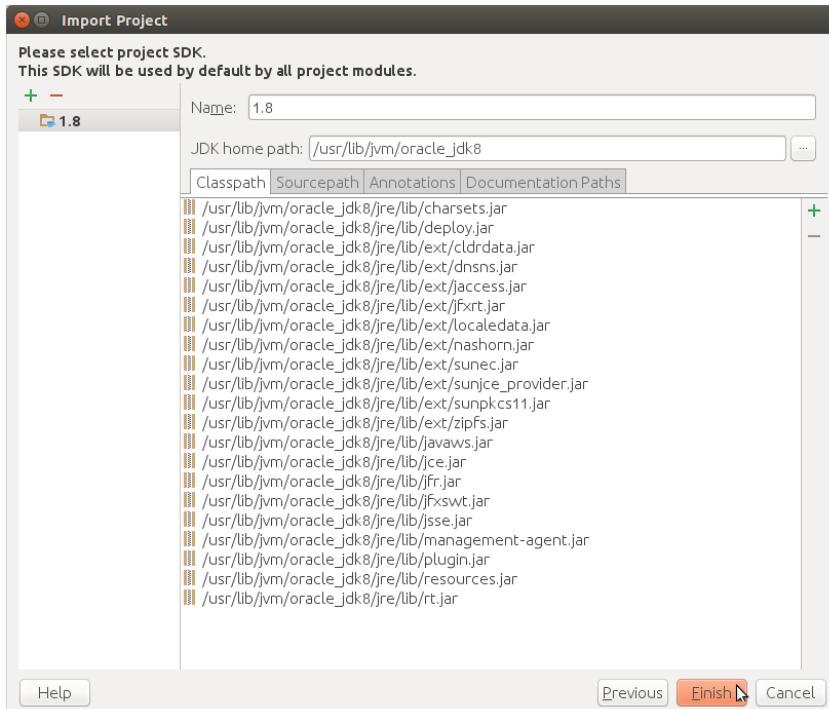
Figur D.17: Markera den upp-packade workspace-mappen från zip-filen som du laddat ner från: <http://cs.lth.se/pgk/ws> och välj **Eclipse**-import.



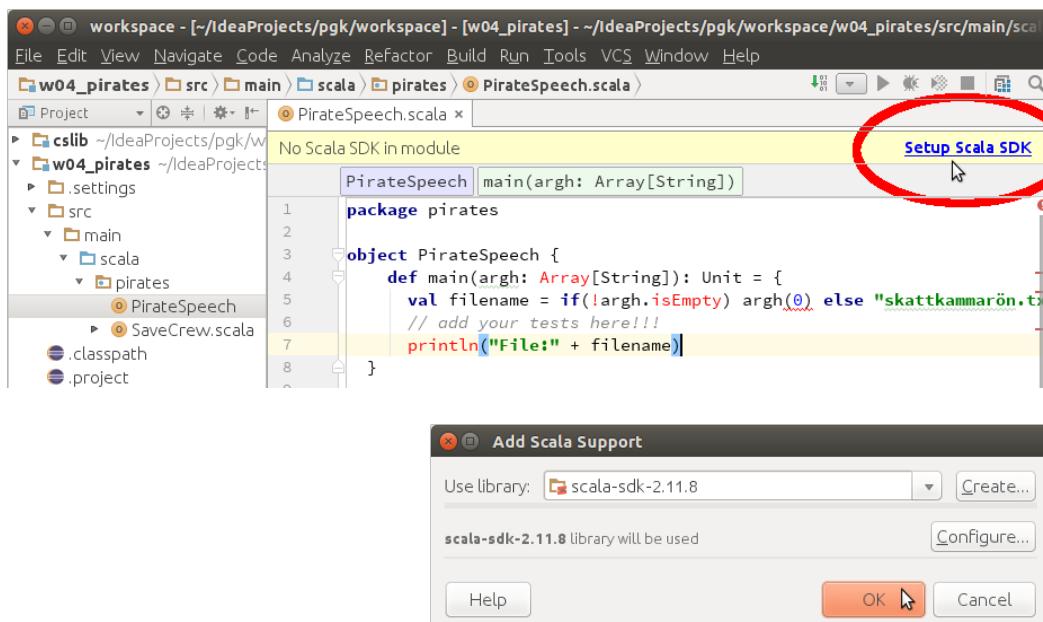
Figur D.18: Klicka **Next** med förvalda alternativ oförändrade.



Figur D.19: Klicka **Next** med alla projekt markerade.



Figur D.20: Klicka **Finish** med förifyllda fält oförändrade.



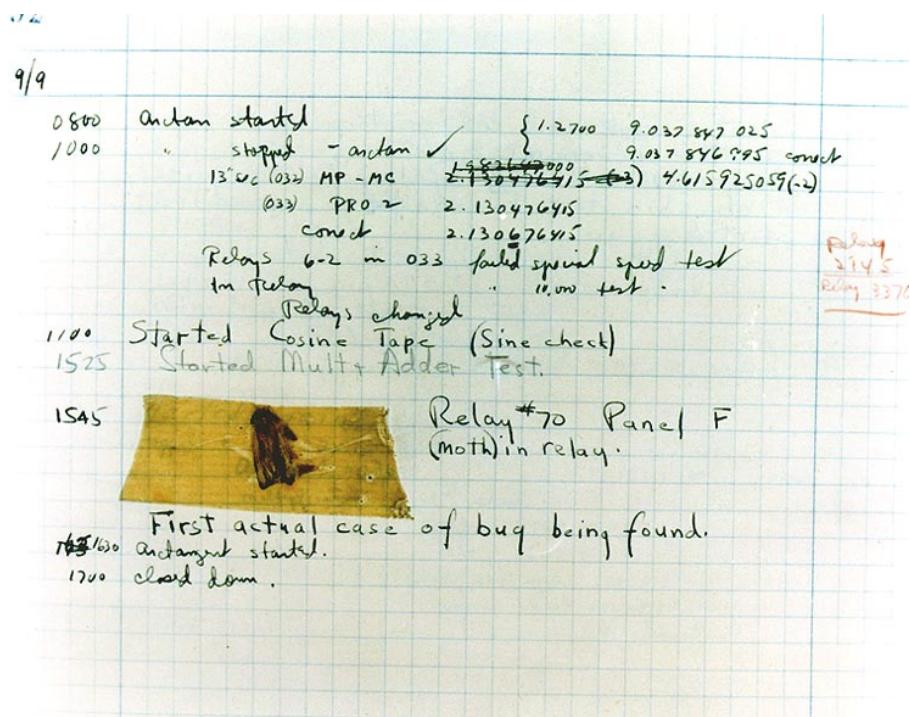
Figur D.21: Bläddra fram `PirateSpeech.scala` i projektet `w04_pirates` och klicka på länken **Setup Scala SDK** och klicka **OK** i efterföljande dialog.



Figur D.22: Lägg till utskriften i bilden ovan på rad7. Testkör genom att välja menyn *Run* → *Run..* (eller trycka Alt+Shift+F10) och sedan välja *PirateSpeech*. Observera utskriften i utskriftsfönstret.

Appendix E

Fixa buggar



Figur E.1: Den första dokumenterade buggen hittades 9 september 1947 i en Mark II Aiken Relay Calculator av Grace Hopper.¹

E.1 Vad är en bugg?

En bugg, även kallad lus (eng. *bug*), är en felaktighet som kan göra så att ett program inte beter sig som det är tänkt, och kan innehålla oönskad utdata, att programmet kraschar, eller till och med ond bråd död.²

¹commons.wikimedia.org/w/index.php?curid=165211 Courtesy of the Naval Surface Warfare Center, Dahlgren, VA., 1988. - U.S. Naval Historical Center Online Library Photograph NH 96566-KN, Public Domain.

²www.theguardian.com/technology/2016/jul/01/tesla-driver-killed-autopilot-self-driving-car-harry-potter

Ursprunget till ordets användning i programmeringssammanhang är något oklar, men kan härledas till engelskans *bug* som betyder insekt eller småkryp. Man brukar berätta att vid en felsökning av ett program som körde i en tidig dator byggd med elektromekaniska reläer, uppdagades en död nattfjäril ihjälklämd mellan drivankaret och spolen i ett relä, som orsakade att programmet inte kunde exekveras korrekt.

E.1.1 Olika sorters fel

När man ska lära sig mer om fel i programvarubaserade system, och hur de kan åtgärdas, är det viktigt att noga skilja på **misstag** (eng. *mistake*), **felorsak** (eng. *fault*) och **felyttring** (eng. *failure*). Med ”misstag” menar vi här ett fel som begås av människor (utvecklare, systemadministratörer, operatörer, användare, etc.) medan de skapar och använder ett programvarusystem.

Det kan bli fel i olika delar av processen:

- **Kravfel** uppstår medan man tänker ut vad systemet ska göra och då misstar sig angående användarnas behov och önskemål.
- **Designfel** uppkommer när man utformar systemets struktur på ett dåligt sätt.
- **Implementeringsfel** begås när man programmerar och skriver felaktiga kodrader.
- **Testfel** förekommer vid provkörning av systemet då testkoden är felaktig och därfor ger falskt alarm om ”fel”, trots att beteendet egentligen är korrekt.
- **Operatörsfel** sker när systemet lämnas över till de, som ska installera och köra systemet i skarp produktion, och där systemdriften (eng. *operations*, ”ops”) sköts på ett sätt som får problematiska konsekvenser.
- **Användarfel** händer då användarna ger felaktig indata, eventuellt i strid med riktlinjerna för hur systemet ska användas, som system inte klarar att hantera korrekt, varpå mer eller mindre allvarliga felbeteenden hos systemet följer.

I olika delar av utvecklingsprocessen kan alltså misstag begås som, antingen omedelbart, eller någon gång i framtiden, kan orsaka fel. Men det är inte säkert att ett fel någonsin kommer att märkas. Kanske kommer de felaktiga kodraderna, som *skulle* kunna orsaka ett fel, aldrig att exekveras. Eller så kommer ingen användare att någonsin vilja använda systemet så som stipuleras av (onödiga) krav. Det är alltså först när fel *yttrar* sig vid exekvering som misstag märks.

Fel kan också kategoriseras utifrån *hur* de upptäcks i utvecklingsprocessen. Man brukar skilja på fel upptäckta vid granskning, kompileringsfel och exekveringsfel, som diskuteras nedan:

- Fel upptäckta vid **granskning**. Ett effektivt sätt att upptäcka fel är att mäniskor noga läser igenom sin egen, och andras kod och försöker leta efter möjliga problem och brister. Man blir ofta ”hemmabland” när det gäller ens egen kod. Därför kan någon annans, oberoende granskning med ”nya, friska” ögon vara mycket fruktbar. I samband med kodgranskning kan man med fördel försöka bedöma huruvida koden är lätt att läsa, lätt att ändra i eller om koden har andra viktiga kvaliteter som har betydelse för den framtida utvecklingen av koden. Ofta hittar man vid granskning även enkla programmeringsmisstag, så som felaktiga villkor och loop-räknare som inte räknas upp på rätt sätt etc.
- **Kompileringsfel** uppkommer under kompilering och upptäcks tack vare kontroller som sker av kompilatorn.

Vid kompileringsfel får man också ofta av kompilatorn reda på *var* i koden det är fel och *varför* det är fel, så att sökandet efter felorsaken och åtgärdandet av misstaget underlättas. Men ibland är felmeddelanden från kompilatorn missvisande och pekar på helt fel ställe i koden, så det gäller att inte alltid lita blint på det kompilatorn skriver. Dessutom är felmeddelanden från kompilatorn ofta uttryckta i termer av språkets syntaktiska och semantiska regler och det tar tid att lära sig tolka kompilatorers felmeddelanden. Att skapa kompilatorer som ger bra felmeddelande är ett svårt problem som studeras inom den datavetenskapliga disciplinen *kompilatorteknik*, vilken du kan lära mer om i kurser på avancerad nivå.

Olika programmeringsspråk erbjuder olika stora möjligheter att göra kontroller vid kompileringstid. En kompilator för ett språk med ett avancerat typsystem, som till exempel Scala, ger förhållandevis stora möjligheter att identifiera fel redan under kompileringen, medan man med ett språk med ett svagare typsystem, till exempel Javascript, får förlita sig på prestandahämmande kontroller som kompilatorn genererar i maskinkoden eller som du själv väljer att lägga in i källkoden för säkerhets skull.

- **Exekveringsfel**, även kallat körtidsfel (eng. *runtime error*), sker medan programmet körs. Det kan kräva viss, specifik indata under specifika exekveringsomständigheter (en viss processor, en viss minnesstorlek, en viss nätverkskapacitet etc.) för att ett exekveringsfel ska yttra sig. När ett exekveringsfel väl yttrar sig, kan olika saker hända:

- **Exekveringen ger oönskat resultat.** Det är inte säkert att ett exekveringsfel avbryter exekveringen; det är vanligt att felet ”bara” resulterar i inkorrekt utdata eller på annat sätt ger dålig kvalitet. För att upptäcka detta innan systemet sätts i drift, är det allmän praxis att man skriver noga uttänkta **testfall** och analyserar **testresultat** från exekveringen av testfallen i detalj genom att undersöka utdata i jämförelse med önskat resultat eller med vad som anses vara en tillräckligt hög kvalitetsnivå.

- **Exekveringen hänger sig** (eng. *hang*). Ibland yttrar sig fel genom att inget alls ser ut att hända under exekveringen, vilket kan beror på t.ex.:

- * en **oändlig loop**, som aldrig blir färdig,
- * att det går **väldigt långsamt** eftersom bearbetningen av indata tar orimligt lång tid,
- * att programmet **väntar på indata** som aldrig kommer,
- * att olika jämlöpande delar av programmet väntar på varandra så att ett **dödläge** (eng. *deadlock*) uppstår.

När exekveringen hänger sig och man inte orkar vänta längre på att något ska hända, är det bara att brutalt avbryta exekveringen genom något lämpligt kommando som erbjuds i din körmiljö.³ I värsta fall får man stänga av strömmen.

- **Exekveringen kraschar** (eng. *crash*). Ibland blir det ett plötsligt tvärstopp och exekveringen avbryts med ett körtidsfelmeddelande. Detta kan bero på t.ex.:

- * att **minnet är slut**, antingen är det parameterminnet för funktionsanrop (eng. *stack memory*) som tagit slut eller så är minnet för allokering av objekt som skapas under programmets gång (eng. *heap memory*) fullt,
- * misstaget att försöka referera en **null-referens** som inte refererar till något objekt, utan har värdet **null**, vilket resulterar i *null pointer exception*,
- * att ett s.k. **undantag** har ”kastats” (eng. *throw exception*) genom att den som skrivit programmet medvetet kodat så att ett oönskat feltillstånd ska orsaka en krasch, om inte undantaget ”fångas” (eng. *catch*) och hanteras av omgivande kod.

När systemet kraschar får man en lista med den aktuella kedjan av funktionsanrop i en **stackspårning** (eng. *stack trace*). Man kan också begära en utskrift av hela innehållet i minnet vid kraschen (eng. *memory dump*), men en sådan kan vara svår att tolka.

När systemet ger oönskade resultat, hänger sig eller kraschar, får man försöka återskapa exekveringsfelet i en omkörning och, med hjälp av instrumentering eller en debugger, försöka lista ut vad som händer precis *innan* exekveringsfelet uppstår, se avsnitt [E.5](#).

I kursen *Programvarutestning* (eng. *Software Testing*) lär du dig mer om systematiska metoder för att testa system så att fel kan förebyggas, identifieras och åtgärdas.

Bugg eller feature?

När ett (eventuellt) fel upptäcks, kan det vara på sin plats att först ställa sig några grundläggande frågor:

³kill -9 <pid>, Ctrl+C, Ctrl+Shift+C, Ctrl+Z eller något annat beroende körmiljö.

- Är detta verkligen ett ”fel” eller är det egentligen ett avsett beteende? Det är inte alltid självklart om det är en bugg eller en medvetet skapad systemegenskap/funktion (eng. *feature*).
- Är det kanske testfallet som har felaktig testkod, medan koden som testas egentligen fungerar alldelvis utmärkt? Sådan problem kan vara speciellt svåra att lösa, då man ofta letar på fel ställe efter orsaken.
- Om buggen rör någon kvalitetsegenskap hos systemet kan man fråga sig: Var går egentligen gränsen för ”fel”? Är detta bra nog givet vad det kostar att förbättra kvaliteten? Kvalitetskrav berör egenskaper hos ett program som kan uttryckas på en glidande skala, där något kan vara mer eller mindre *bra* eller *dåligt* ur olika synvinklar. Sådana krav leder ofta till viktiga men svåra avvägningsbeslut under design och implementation. Dessutom kan testresultat bli svårbedömda och det kan finna olika åsikter om huruvida ett eventuellt fel är en bugg eller inte.

Här är några exempel på kvalitetskrav:

- **Prestandakrav** (eng. *performance requirements*) avser hur snabbt och effektivt programmet ska arbeta under olika omständigheter.
- **Kapacitetskrav** (eng. *capacity requirements*) avser hur mycket data systemet ska klara av under olika omständigheter.
- **Användbarhetskrav⁴** (eng. *usability requirements*) avser krav på hur lättanvänt systemet ska vara för en given användarkategori.

I kurserna *Kravhantering* (eng. *Software Requirements Engineering*) lär du dig mer om att identifiera, specificera och följa upp kvalitetskrav.

Felärendehanteringsverktyg

Det är allmän praxis i industriell systemutveckling att använda sig av ett felärendehanteringsverktyg (eng. *issue tracker*) så att samarbetande utvecklare får stöd i att hålla reda på alla uppkomna fel och problem (eng. *issue*). Många av de populära kodlagringsplatserna som finns på nätet, så som GitLab, GitHub och BitBucket (se avsnitt H.3), erbjuder felärendehanteringsfunktioner. Dessa kan till exempel vara:

- hantering och sammanställning av alla olika ärendetillstånd, så att man kan se vilka issues som är i tillstånden *Open* eller *Closed*,
- tillordning av ärende till specifika personer som ska åtgärda problemet,
- gradering av ärende i olika allvarlighetsgrader,
- meddelandegenerering till inblandade personer när ett ärende kommenteras eller ändrar tillstånd.

⁴sv.wikipedia.org/wiki/Användbarhet

E.2 Att förebygga fel

Även om det nästan är oundvikligt att inte låta buggar slinka in i koden allteftersom den blir mer och mer komplex, är det ändå viktigt att lägga stor möda vid att försöka undvika att så sker. Det är ofta mycket bättre investerad tid att jobba med buggförebyggande åtgärder medan du skapar koden, än att jaga buggar som skulle ha kunna undvikas med allmän noggrannhet och stramare disciplin i kodningen. Nedan sammanfattas några åtgärder som kan hjälpa till att minska mängden fel.

- **Skapa begriplig kod.** Grunden för att undvika buggar är anstränga sig att skriva begriplig kod som är lätt att läsa. Detta är en ständig kamp; kodens komplexitet växer för varje tillägg och med jämna mellanrum behövs omstruktureringar (eng. *refactoring*) för att bibehålla en god struktur som underlättar begripligheten och gör utvidgningar lättare.
- **Tänk ut bra namn.** En viktig pusselbit för att skapa begriplig kod är att tänka ut bra namn. Detta kan vara förvånansvärt svårt och kan kräva mycket diskussioner och tankemöda. Om du inser att ett namn är illa valt är det förmodligen värt jobbet att omstrukturera koden och införa ett bättre namn, speciellt om andra ännu inte vant sig alltför mycket vid begreppet.
- **Kontrollera parametrar och variabler.** Ofta vet känner man till vilka villkor som måste gälla för olika variablers värden. Till exempel vet man ofta att en viss funktionsparameter av heltalstyp inte får vara negativ. Då kan man säkerställa detta genom att lägga in kontroller av att villkoret är uppfyllt. Vid villkor som gäller parametrar, brukar man i Scala anropa `require`, till exempel: `require(x >= 0, "x must be positive")`. Det finns också en metod `assert` som fungerar på samma vis⁵; medan `require` används för att kontrollerar parametrar, brukar `assert` användas för att kontrollera generella villkor som ska gälla, till exempel `assert(x + y > n, "overflow")`. Fördelen med att lägga in kontroller av villkor är att villkorsbrott upptäcks direkt och felsökningen blir lättare.
- **Kontrollera typer.** Med *typannotationer* får du hjälp av kompilatorn att kontrollera dina hypoteser om vilka typer olika värden har. I Scala kan du nästan var du vill i ett uttryck lägga till ett kolon och en typ för att begära att kompilatorn kontrollerar typen. Till exempel kan du skriva `(xs + f(42)) : Set[Int]` för att säkerställa att uttrycket `xs + f(42)` verkligen ger en mängd med heltal. Även om du sällan i Scala behöver ange typer explicit, tack vare kompilatorns typinferens, bidrar det till läsbarheten och skapar säkrare kod om du på lämpliga ställen ändå anger de typer som du förväntar dig, speciellt vid i komplicerade uttryck

⁵stackoverflow.com/questions/26140757/what-to-choose-between-require-and-assert-in-scala

eller långa kedjor av metodenrop, och när metoders returtyper inte är uppenbara. Dessutom kan kompilatorn ibland undvika att gå vilse i speciellt svåra typhärleddningar, om du hjälper den på traven med explicita typannoteringar.

- **Hantera saknade värden.** Det är mycket vanligt att man måste hantera situationer där ett värde saknas, inte kan beräknas, eller inte finns tillgängligt av andra orsaker. Man kan hålla reda på att ett värde saknas genom att representera detta med speciella värden, t.ex. `-1` eller `null`. Men den strategin leder mycket lätt till buggar, då man lätt glömmer att på andra ställen i koden kontrollera dessa speciella värden. Med sådana speciella värden får man heller ingen hjälp av kompilatorn att upptäcka att man missat att ta hand om dem. Om man istället hanterar eventuellt saknade värden med `Option` (se kapitel 8), så får man hjälp vid kompileringstid och slipper exekveringsfel och besvärlig felsökning. Det blir dessutom väldigt tydligt för alla som läser din kod, inklusive du själv, att ett värde kan saknas.
- **Hantera undantag.** När undantag uppstår, t.ex. att en fil inte kan läsas eller det blir division med noll, avbryts exekveringen och programmets användare kan inte använda programmet längre, vilket i värsta fall kan få ödesdigra konsekvenser. Därför vill man hantera undantags situationer på ett sådant sätt att programmet blir robust och inte kraschar. Detta kan man med fördel göra genom att kapsla in undantaget i ett värde av typen `Try`, se kapitel 8. I likhet med `Option` för saknade värden, blir det tydligt i koden att ett värde av typen `Try` kan innehålla ett lyckat resultat (`Success`), eller så fallerar beräkningen (`Failure`) med en inkapslad, förhindrad krasch.
- **Granska kod.** Det är allmän praxis i industriell programvaruutveckling att gör kodgranskningar, vid vilka en grupp mäniskor noga studerar någon annans kod och ger kommentarer och identifierar potentiella problem. Ofta har man en checklista att utgå ifrån medan man läser koden, som innehåller punkter man vill kontrollera speciellt, t.ex. begriplighet, namngivning, kontroller av parametrar, hantering av saknade värden och undantag, etc. Många organisationer har en överenskommen kodningsstandard med riktlinjer för kodens utseende och stil som alla ska följa om inte speciella skäl finns. Att sådana stilriktslinjer följs kan kontrolleras genom granskningar. Det finns också verktygsstöd för att göra sådana kontroller. Ett exempel på kodningsriktlinjer för Scala finns på den officiella dokumentationssajten⁶.
- **Testa kod.** Det är allmän praxis i industriell programvaruutveckling att genomföra tester på flera olika nivåer. Man kombinerar ofta **enhetstest** (eng. *unit test*) av enskilda delar av koden, med **funktionstest** (eng. *feature test*) för att se så att indata i en tänkt användningssituation ger

⁶<http://docs.scala-lang.org/style/scaladoc.html>

önskat resultat, och **systemtest** (eng. *system test*) för att se att alla delar fungerar tillsammans under realistiska omständigheter.

- **Lär av användarnas upplevelser.** När koden sätts i produktion finns möjlighet att lära sig genom återkoppling från användare. Hur systemet används och hur användarna upplever det att använda systemet är mycket viktig information när man ska besluta om hu koden bäst ska utveckla vidare. Från användarna kan man få reda på både okända buggar och få briljanta idéer till nya värdefulla funktioner. En mjukvaruutvecklande organisations innovationsförmåga beror i stor utsträckning på dess förmåga att kontinuerligt leverera kod som får allt fler funktioner som användarna gillar, utan att för många irriterande eller ödesdigra buggar.

E.3 Vad är debugging?

När en felyttring identifierats, t.ex. genom testning eller slutanvändare rapporterar om problem, vidtar sökandet efter den bakomliggande felorsaken, så att vi förstår *varför* det blev fel och sedan kan *åtgärda* misstaget. Denna process kallas **avlusning** (eng. *debugging*).

E.3.1 Hur hitta felorsaken?

Första steget i avlusningsprocessen är att hitta den bakomliggande felorsaken. Detta kan vara mycket svårt, speciellt om systemet är stort och komplicerat.

När du stirrar dig blind på koden utan att hitta felorsaken, kan det bero på att du har en felaktig hypotes om vad koden egentligen gör. Du är övertygad om att en viss sak händer, men *egentligen* är det *inte* det du *tror* händer som *verkligen* händer. Exempelvis kanske du antar att en räknare räknas upp i en loop, men i själva verket saknas uppräkningen. Om du oreflekterat accepterar ditt felaktiga antagande, är det stor risk att du letar på fel ställe i koden.

Följande åtgärder är ofta lämpliga när man jagar buggar:

- **Återskapa buggen med ett minimalt testfall.** När du upptäckt en felyttring är det viktigt att kunna återskapa felet, så att felsökningen kan vid exekvering av den koden körs precis *innan* buggen uppstår. Allra bäst är det om du kan skapa ett **minimalt testfall** där precis den minimala indata och de enskilda förutsättningar nedtecknas, som ska gälla för att buggen ska uppstå. Beskrivningen av det minimala testfallet är första pusselbiten i det detektivarbete som vidtar under felsökningen.
- **Formulera och verifiera hypoteser om buggen.** En grundläggande princip vid felsökning är att uttryckligen formulera hypoteser som du har om vad som sker i systemet medan buggen uppstår och sedan *Verifiera* att de verkligen stämmer, genom olika undersökningar av det exekverande systemet. Du ska alltså tydligt beskriva hur du tror att koden fungerar och sedan med olika former av instrumentering, t.ex. genom utskrifter i

terminalen av variablers värden, kontrollera att så verkligen är fallet. Detta kan göras med instrumentering enligt nedan.

- **Instrumentering med utskrifter, ”print-debugging”.**

För att verifiera din hypoteser om vad som leder fram till buggen, behöver du kontrollera vad som händer. Det kan du göra genom att på välväldiga ställen ligga in `println`-utskrifter i koden där värden på intressanta variabler skrivs ut. Det kan behövas lite klurighet för att hitta precis rätt utskrifter; om man skriver ut allt som händer i alla loopar drunknar man i all information, men skriver man ut för lite förbiser man kanske den falsifierade hypotesen och får ingen hjälp att knäcka bugg-gåtan.

Du kan även använda en avlusare (eng. *debugger*), som normalt ingår i en integrerad utvecklingsmiljö, för att instrumentera din kod. Se vidare i avsnitt [E.5](#) om hur du använder avlusarna i Eclipse och IntelliJ IDEA.

E.4 Åtgärda fel

Ofta är det det svåraste att *hitta* buggen, medan själva buggrättningen visar sig trivial. Har du, till exempel, väl hittat den saknade uppräkningen av din loop-variabel är det uppenbart vad du ska göra.

Men ibland är det riktigt knepigt att åtgärda felet. Nedan sammanfattas några av de situationer som kan uppstå, som gör att felsökningen blir extra svår.

- Kanske är själva algoritmen i grunden felränt och en helt ny algoritm behöver konstrueras. Att skapa nya algoritmer från grunden kan visa sig mycket svårt i en del fall. I fortsättningskurser får du lära dig mer om algoritmkonstruktionens ädla konst.
- Kanske algoritmen fungerar för olika normalfall, medan ovanliga undantagsfall inte hanteras korrekt. Att på ett bra sätt hantera alla upptänkliga fall kan visa sig väldigt knepigt. Tyvärr är det ofta undantagsfall i kombination med buggar som öppnar för säkerhetssluckor redo att utnyttjas av elaka hackare för att krascha systemet eller smitta ner det med virus.
- Kanske är problemet i sig väldigt svårt att lösa på ett korrekt sätt. Algoritmen kan vara riktigt knepig med många villkor, loopar och nästlade datastrukturer. Blir det fel i en sådan algoritm kan det ta lång tid att få ändringar att fungera och alla villkor, loopar och nästlade datastrukturer att passa ihop igen efter felsökningen.
- Medan man rättar en bug kan man råka att, av misstag, skapa nya buggar. Risken för detta är speciellt stor om koden är komplex. Ibland låter man till och med bli att åtgärda ett fel om systemet ändå fungerar hjälpligt i andra avseenden och risken är för stor att nya buggar skapas. Då behöver systemet strukturerats om så att det blir lättare att ändra i.

- Kanske växer exekveringstiden exponentiellt med datamängden. Det kan då i praktiken vara omöjligt att skriva ett program som i alla lägen blir färdigt inom rimlig tid. Då får man försöka tänka ut kluriga genvägar till suboptimala lösningar som ändå duger, vilket ibland kräver mycket avancerad programmeringsteknik.

Det finns ingen allenarådande snabbfix att ta till när man stöter på svåra fel. Att bli en produktiv systemutvecklare, som framgångsrikt redar ut allehanda buggar, handlar i stor utsträckning om att kombinera en bred allmänbildning inom datavetenskap med ett livslångt lärande, där varje bugg du hittar och åtgärdar ger dig nya kunskaper och erfarenheter inför framtiden. Även om din bugg är irriterande, försök se den som en ny chans till ökad lärdom!

E.5 Använda en debugger

Med en professionell integrerade utvecklingsmiljö kommer ofta en avancerad debugger, som kan hjälpa dig att följa exekveringen och se vad som händer medan koden kör. Normalt ingår dessa funktioner i en debugger:

- **Sätta brytpunkter.** Med hjälp av debuggern kan du sätta så kallade *brytpunkter* på speciella ställen i koden. Detta görs ofta genom att du markerar en kodrad i marginalen varpå en brytpunktssymbol placeras där. När exekveringen når brytpunkten avbryts exekveringen och du kan stega dig vidare eller inspektera variablers värden vid brytpunkten.
- **Stegad exekvering.** När du nått en brytpunkt kan du med hjälp av debuggern stega dig fram genom koden rad för rad och se vad som händer. Om du kommer till ett funktionsanrop kan du välja att stega in i koden som implementerar funktionen eller bara köra funktionen i ett svep och stega till nästa rad som kommer efter funktionsanropet. Det kan kräva många omkörningar från en viss brytpunkt, innan man hittar vilka funktioner som inte verkar relevanta alls för buggen och bara kan stegas över, eller vilka funktioner som utgör gåtans lösning och som du vill stega in i och undersöka närmare. Stegar man djup ner i funktionsanropskedjan, går man lätt vilse och får börja om. (Det går inte att stega bakåt...)
- **Inspektera variabler.** Medan du stegar dig fram i koden kan du inspektera variablers värden. I ett område på skärmen presenterar debuggern både enkla värden så som heltal och strängar, men även datastrukturer, så som vektorer och listor, kan inspekteras genom att debuggern låter dig bläddra bland arrayer och objektreferenser. Ett program kan ha väldigt många variabler och djupa strukturer av objekt som refererar till nya objekt. Det är ofta ett knepigt detektivarbete att försöka lista ut hur olika variabelvärdet relaterar till orsaken bakom buggen som du letar efter. Du behöver ofta växla mellan att läsa koden, stega dig fram, sätta nya brytpunkter och inspektera variabler för att förstå vad som händer.

E.5.1 Debuggern i Eclipse med ScalaIDE

Sätta brytpunkter i Eclipse

TODO!!!

Stegad exekvering i Eclipse

TODO!!!

Inspektera variabler i Eclipse

TODO!!!

E.5.2 Debuggern i IntelliJ IDEA med Scala-plugin

Sätta brytpunkter i IntelliJ

TODO!!!

Stegad exekvering i IntelliJ

TODO!!!

Inspektera variabler i IntelliJ

TODO!!!

Appendix F

Dokumentation

Dokumentation hjälper andra att använda din kod, men underlättar även för dig själv när du vid ett senare tillfälle ska erinra dig hur den fungerar och hur du ska använda och bygga vidare på din kod. Modern systemutveckling baseras ofta på öppen källkod och färdiga api (eng. *application programming interface*), där kvaliteten på dokumentationen är avgörande för hur lätt det är att komma igång med att använda koden.

Nedan listas exempel på olika typer av dokumentation¹:

- **Kravdokumentation** beskriver det övergripande målet med mjukvaran, samt funktionella krav och kvalitetskrav som uppfylls av systemet.
- **Designdokumentation** beskriver arkitekturen, hur koden är organiserad i moduler, och den interna systemstrukturen t.ex. i form av klasser, objekt och deras relation.
- **Slutanvändardokumentation** kan t.ex. vara manualer för användning av systemet och installationsanvisningar.
- **Teknisk dokumentation** kan t.ex. vara api-dokumentation som beskriver vilka funktioner som ingår i ett programbibliotek. Sådan dokumentation genereras ofta med hjälp av ett **dokumentationsverktyg** (se avsnitt F.1). Andra typer av teknisk dokumentation är instruktioner om hur man bygger koden med eventuellt tillhörande byggverktygskonfigurationsfiler; ofta beskrivs byggförfarandet steg för steg i en textfil med namnet README. (Läs mer om byggverktyg i appendix G.)

Det är en stor utmaning att hålla dokumentationen uppdaterad allteftersom koden utvecklas. Även om man får hjälp att generera en navigerbar sajt av ett dokumentationsverktyg, måste själva *innehållet* i de manuellt författade dokumentationskommentarerna vara i överensstämmelse med den aktuella versionen av koden. Uppdateras koden, måste man alltså vara noga med att uppdatera dokumentationskommentarerna, annars uppstår stor förvirring.

Detta problem är så pass allvarligt att man ska tänka sig noga för hur man kan formulera dokumentationskommentarerna på ett framtidssäkert sätt, och

¹en.wikipedia.org/wiki/Software_documentation

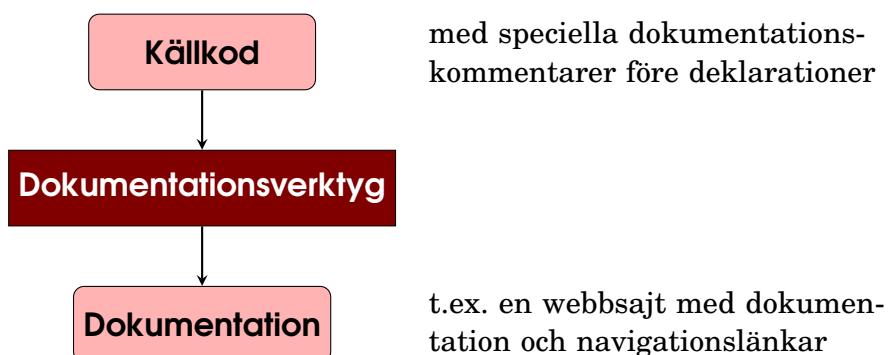
hur omfattande de ska vara i förhållande till den framtida arbetsinsatsen med att hålla dem uppdaterade. Desto mer omfattande kommentarer desto mer jobb att hålla dem uppdaterade.

Det är i praktiken svårt att uppnå en optimal balans mellan bra och många kommentarer som *hjälper* användaren, och å andra sidan svårunderhållna och föråldrade kommentarer som *stjälper* användare.

F.1 Vad gör ett dokumentationsverktyg?

Ett dokumentationsverktyg genererar teknisk dokumentation av koden baserat på speciella **dokumentationskommentarer** som skrivs i koden omedelbart före deklarationer av det som ska dokumenteras. Dessa dokumentationskommentarer skrivs enligt en speciell syntax som dokumentationsverktyget kan tolka.

Utdata från ett dokumentationsverktyg utgörs typiskt av en webbsajt med ändamålsenlig formatering och navigationslänkar, se figur F.1.



Figur F.1: Ett dokumentationsverktyg läser koden och dokumentationskommentarer och genererar dokumentation, t.ex. i form av en webbsajt.

F.2 scaladoc

Med Scala-installationen följer dokumentationsverktyget scaladoc, som genererar en webbsajt med ändamålsenlig layout och specialfunktioner för att söka, filtrera och navigera i dokumentationen.

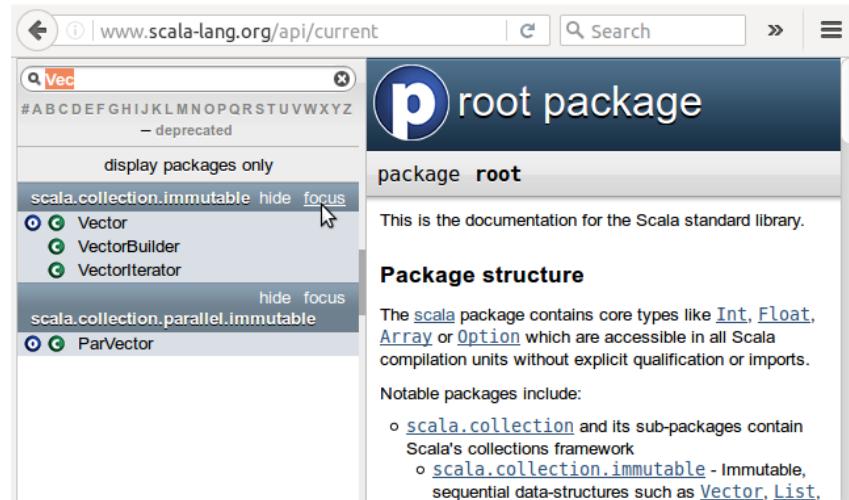
Dokumentationen av stora bibliotek kan bli omfattande och det krävs träning i att använda dokumentationssajter för att få maximal nytta av dem. I efterföljande avsnitt beskrivs först hur du använder dokumentation som är genererad med scaladoc. Därefter visas hur du själv kan generera dokumentation för din egen kod.

F.2.1 Använda dokumentation från scaladoc

Dokumentationen av Scalas standardbibliotek är genererad med scaladoc och att navigera i denna ger bra träning i hur man använder avancerad api-

dokumentation. Du hittar dokumentationen för Scalas standardbibliotek här: <http://scala-lang.org/api/current>

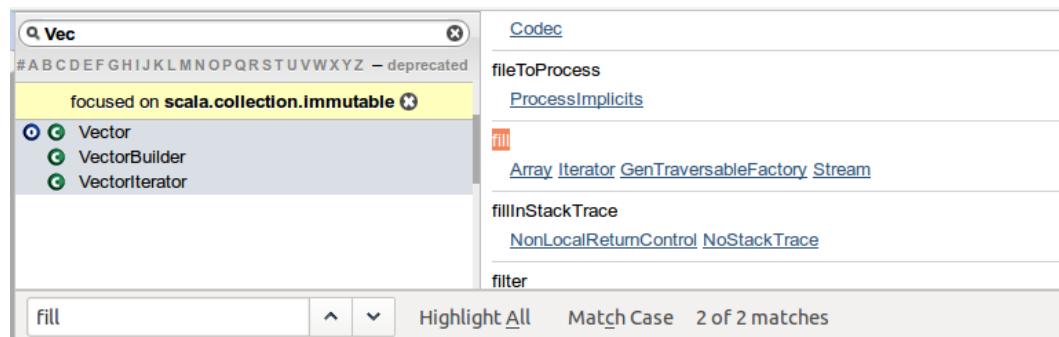
När du surfar dit möts du av dokumentationen för *root package*, som ger en översikt av olika paket i standardbiblioteket. I sökrutan uppe till vänster kan du skriva början på namnet på klasser, traits, eller objekt som du letar efter, så som visas i figur F.2.



Figur F.2: scaladoc.

Om du är speciellt intresserad av, t.ex., paketet `scala.collection.immutable`, kan du klicka **focus** för att begränsas visningen till att endast innehålla typerna i detta paket.

Om du söker efter typen där en viss metod är implementerad, men inte vet riktigt i vilken klass den finns, kan du klicka på bokstaven som metodnamnet börjar på i listan med bokstäver under den övre vänstra sökrutan. Då får du en lista med allt möjligt som börjar på F, så som visas i figur F.3. Sök i listan med din webbläsares sökfunktion (Ctrl+F) efter "fill", så hittar du alla typer som implementerar metoden `fill`.



Figur F.3: scaladoc.

Om du klickar vidare, i detta exempel på länken till klassen `Array`, kan du sedan klicka på länken till källkoden i `array.scala` för att se implementatio-

nen på GitHub; sök på sidan med din webbläsares sökfunktion Ctrl+F efter ”def fill”.

Om du klickar på den typ du är intresserad av, t.ex. klassen Vector, får du upp en sida med en mängd information, inklusive alla metoder, även ärvda. Figur F.4 visar hur en sökning bland alla metoder i Vector initieras i sökrutan nere till höger, vilken kommer att visa metoder som börjar på ”rev”, här skrollas fönstret till metoden reverse längre ner på sidan.

Genom att klicka på det gröna cirkeln med bokstaven C överst i dokumentationsfönstret, växlar du till vyn för kompanjonsobjektet där du hittar alla fabriksmetoder för Vector, även ärvda, t.ex. fill som du kan söka efter i sökrutan.



Figur F.4: scaladoc.

F.2.2 Skriva dokumentationskommentarer för scaladoc

Verktyget scaladoc läser kommentarer som börjar med `/**` och slutar med `*/` och associeras till efterföljande deklaration. Notera de dubbla asteriskerna. Alla rader som följer efter `/**` ska, enligt konventionen för Scalas dokumentationskommentarer, börja med en asterisk `*` med indrag med flera blanksteg så att den hamnar under *andra* asterisken i öppningskommentaren, som nedan:

```
/** Först kommer en sammanfattning på en enda rad.
 *
 * Sedan kommer eventuellt en mer detaljerad beskrivning,
 * som kan vara flera rader lång.
 */
```

Dokumentationskommentaren slutar med `*/` rakt under asterisk-kolumnen.

I figur F.5 på sidan 394 visas exempel på dokumentationskommentarer. Annoteringen `@param` i början på en rad ger en speciell kommentar angående

parametrar. Annoteringen @return i början av en rad ger en speciell kommentar angående vad som returneras vid metodenrapp.

F.2.3 Generera dokumentation med scaladoc

Du genererar en dokumentationssajt med terminalkommandot scaladoc följt av en eller flera källkodsfiler. Med optionen -d anger du i vilket bibliotek sajten ska sparas. Du visar sajten genom att öppna filen index.html i en webbläsare. Nedan visas hur dokumentationen genereras för källkodsfilen i figur F.5.

```
$ scaladoc mio.scala -d apidoc  
$ firefox apidoc/index.html
```

I figur F.6 på sidan 395 visas delar av en webbsida som genererats utifrån koden i figur F.5 på sidan 394. För de publika metoder där ingen dokumentationskommentar finns, visas ändå metodens signatur med parametrar, parametertyper, och returtyp. Medlemmar som deklareras **private** visas inte, men om man klickar på knappen **All** bredvid rubriken **Visibility** visas medlemmar som är deklarerade **protected**.

Om du klickar på symbolen ► till vänster om metodsituren, ändras den till symbolen ▼ som indikerar att den mer detaljerade beskrivningen av parametrar etc. har vecklats ut (i den mån detaljerade kommentarer finns).

Om du vill ha övergripande dokumentation om ett paket x, ges det speciella objektet **package object** x en dokumentationskommentar med sådan information. Ofta innehåller **package object** medlemmar som man vill ska bli synliga vid import av paketet, så som variabler, metoder och implicita medlemmar som inte har någon annan naturlig hemvist.

F.2.4 Lära mer om scaladoc

- En video med tips om hur du söker och navigerar i scaladoc-dokumentation:
<http://docs.scala-lang.org/overviews/scaladoc/interface.html>
- Riktlinjer för hur du skriver dokumentationskommentarer:
<http://docs.scala-lang.org/style/scaladoc.html>
- Länksida till mer detaljerade beskrivningar:
<https://wiki.scala-lang.org/display/SW/Writing+Documentation> inkluderande bland annat:
 - En beskrivning av syntaxen för formatering:
<https://wiki.scala-lang.org/display/SW/Syntax>
 - En beskrivning av speciella annoteringar, t.ex. @param:
<https://wiki.scala-lang.org/display/SW/Tags+and+Annotations>
- Kör kommandot scaladoc -help för att se användbara optioner.
- sbt doc är ett smidigt sätt att generera api-dokumentation. Läs mer om sbt och api-dokumentation här:
<http://www.scala-sbt.org/0.13/docs/Howto-Scaladoc.html>

```

1 // file mio.scala at https://github.com/lunduniversity/introprog/tree/master/util/
2
3 import scala.language.~
4 import java.nio.file.{Path, Paths, Files}
5 import java.nio.charset.StandardCharsets.UTF_8
6 import scala.collection.JavaConverters.~
7
8 /** An object with many useful input/output methods.
9  *
10 * Compile it with `scalac mio.scala` to put it on your classpath,
11 * or just paste the whole source file in the REPL using for example
12 * `:pa util/mio.scala` where `util` is the path to `mio.scala`
13 */
14 object mio {
15
16   /** Load a text file as a Vector of strings, one per line.
17   *
18   * @param fileName the name of the text file to load
19   * @return a vector with lines of the loaded text file
20   */
21   def load(fileName: String): Vector[String] =
22     io.Source.fromFile(fileName).getLines.toVector
23
24   /** Prints each line of a file. */
25   def cat(fileName: String): Unit = load(fileName).foreach(println)
26
27   def listFiles(dir: String): Vector[Path] =
28     Files.list(Paths.get(dir)).toArray.map(_.asInstanceOf[Path]).toVector
29
30
31   def ls: Unit = listFiles("").foreach(println)
32
33   def ls(dir: String): Unit = listFiles(dir).foreach(println)
34
35   def currentDir: Path = Paths.get("").toAbsolutePath
36
37   def pwd: Unit = println(currentDir)
38
39   def save(data: String,
40           fileName: String = "untitled.txt"): Unit = {
41     println("Saving to file: " + Paths.get(fileName).toAbsolutePath)
42     Files.write(Paths.get(fileName), data.getBytes(UTF_8))
43   }
44
45   def isDir(name: String): Boolean = (new java.io.File(name)).isDirectory
46
47   private lazy val console = new jline.console.ConsoleReader
48
49   def readln(prompt: String): String = console.readLine(prompt)
50
51   def readln: String = console.readLine("")
52
53   /** Run `ls` if args is empty or else run ls on each dir in args. */
54   def main(args: Array[String]): Unit =
55     if (args.isEmpty) ls else args.foreach(ls)
56
57 }
```

Figur F.5: Dokumentationskommentarer som kan läsas av scaladoc för att generera en dokumentations-webbsajt. Sådana kommentarer börjar med snedstreck och dubbla asterisker, se bl.a. raderna 8–13 ovan.

The screenshot shows a Scaladoc-generated web page for the `mio` object. At the top, there is a logo with a blue circle containing a white 'O' and the word 'mio'. To the right, a link to 'Related Doc: package root' is visible. Below the header, the title 'object mio' is displayed, followed by a brief description: 'An object with many useful input/output methods.' A note says to 'Compile it with scalac mio.scala to put it on your classpath, or just paste the whole source file in the REPL using for example :pa util/mio.scala where util is the path to mio.scala'. A section titled 'Linear Supertypes' is shown. Below this is a search bar and a filter interface with tabs for 'Ordering' (set to 'Alphabetic'), 'Inherited' (set to 'mio'), and 'Visibility' (set to 'Public'). Buttons for 'Hide All' and 'Show All' are also present. The main content area is titled 'Value Members' and lists several methods:

- `def cat(fileName: String): Unit` - Prints each line of a file.
- `def currentDir: Path`
- `def isDir(name: String): Boolean`
- `def listFiles(dir: String): Vector[Path]`
- `def load(fileName: String): Vector[String]` - Loads a text file as a Vector of strings, one per line. Parameters: `fileName` (the name of the text file to load), `returns` (a vector with lines of the loaded text file). A copy icon is next to this method.
- `def ls(dir: String): Unit`
- `def ls: Unit`
- `def main(args: Array[String]): Unit` - Runs `ls` if `args` is empty or else runs `ls` on each dir in `args`.
- `def pwd: Unit`
- `def readLn: String`
- `def readLn(prompt: String): String`
- `def save(data: String, fileName: String = "untitled.txt"): Unit`

Figur F.6: Delar av en webbsida genererad med hjälp av scaladoc. Mer detaljerade beskrivningar kan i förekommande fall vecklas ut eller in om man växlar mellan ▶ och ▼.

Package	Description
cslib.examples	Programexempel.
cslib.fractal	Fraktaler (MandelbrotGUI).
cslib.images	Bildbehandling.
cslib.maze	Labyrint (Maze).
cslib.shapes	Geometriska figurer och lista av figurer (Shape och ShapeList).
cslib.square	Kvadrater (Square).
cslib.window	Lättanvänt stöd för att skapa ritfönster.

Figur F.7: Delar av en webbsida genererad med hjälp av javadoc.

F.3 javadoc

Med Java JDK följer dokumentationsverktyget javadoc, som utifrån dokumentationskommentarer i Java-kod genererar en webbsajt med navigationslänkar. Webbsidor genererade med javadoc erbjuder inte samma funktioner för sökning och filtrering som scaladoc, men det fungerar bra hitta det man söker om navigationslänkarna används tillsammans med webbläsarens inbyggda sökfunktion (Ctrl+F).

F.3.1 Använda dokumentation genererad med javadoc

I figur F.7 visas exempel på javadoc för biblioteket cslib. Om du klickar på ett paket kan du navigera till en översikt av innehållet i paketet. Om du klickar på en klass får du en översikt av klassens medlemmar, så som visas i F.8. Om du t.ex. klickar på ett metodnamn får du se mer detaljerade kommentarer.

Ramarna till vänster på webbsidorna innehåller länkar till paket och klasser. Om du klickar på länken *All Classes* överst till vänster för du en lista med navigationslänkar till alla tillgängliga klasser. De gulmarkerade rubrikerna visar vilken vy som är aktiv och navigationslänkar skrivs med blå text.

F.3.2 Skriva dokumentationskommentarer för javadoc

Kommentarer för javadoc och scaladoc ser ganska lika ut, även om det finns några skillnader. Det finns t.ex. inte lika många styrtecken för layouten i javadoc som i scaladoc, och konventionen i Java är fyra blankstegs indrag och att fortsättningsrader i dokumentationskommentarer börjar asterisken under *första* asterisken i öppningskommentaren.

Nedan visas delar av javadoc-kommentarerna för klassen SimpleWindow och dess konstruktör:

The screenshot shows a Java API documentation page. On the left, there's a sidebar with navigation links for 'All Classes' and 'Packages'. Under 'Packages', several classes are listed: cslib.examples, cslib.fractal, cslib.images, cslib.maze, cslib.shapes, cslib.square, and cslib.window. Below this, another 'All Classes' section lists: ImageFilter, MandelbrotGUI, Maze, Shape, ShapeList, SimpleWindow (which is highlighted with a cursor), SimpleWindowTest, SimpleWindowTest, Sprite, SpriteTest, and Square.

The main content area has two tabs: 'Constructor Summary' and 'Method Summary'. The 'Constructor Summary' tab is active, showing one constructor: `SimpleWindow(int width, int height, java.lang.String title)`. A brief description follows: 'Creates a window and makes it visible.' The 'Method Summary' tab is also present.

Under 'Method Summary', there's a table with four columns: 'All Methods', 'Static Methods', 'Instance Methods', and 'Concrete Methods'. The 'Instance Methods' column is active. It contains three methods:

Modifier and Type	Method and Description
void	<code>addSprite(Sprite sprite)</code> Adds a sprite to the window.
void	<code>clear()</code> Clears the window.
void	<code>close()</code> Closes the window.

Figur F.8: Delar av en webbsida med klassdokumentation genererad med hjälp av javadoc.

```
package cslib.window;

/** A simple window to draw in */
public class SimpleWindow {
    /**
     * Creates a window and makes it visible.
     *
     * @param width    the width of the window
     * @param height   the height of the window
     * @param title    the title of the window
     */
    public SimpleWindow(int width, int height, String title) {
        ...
    }
}
```

Annoteringen `@param` i början på en rad ger en speciell kommentar angående en parameter. Vid dokumentation av metoder kan annoteringen `@return` användas i början av en rad för att skapa en speciell kommentar angående vad som returneras.

Övergripande dokumentation om innehållet i ett paket läggs i en textfil i paketets katalog med namnet `package-info.java`, se till exempel här:
github.com/lunduniversity/introprog/tree/master/workspace/cslib/src/main/java/cslib/window

Du kan läsa mer om hur man skriver javadoc-kommentarer här:
www.oracle.com/technetwork/java/javase/documentation/index-137868.html

F.3.3 Generera dokumentationskommentarer för javadoc

Om du står i den katalog där din källkod finns, kan du med nedan kommando i terminalen gå igenom alla paket och underpaket och generera javadoc-webbsidor i katalogen doc. Du kan därefter öppna dokumentationen i en webbläsare.

```
$ javadoc -d doc -encoding UTF-8 -charset UTF-8 -sourcepath . -subpackages . *
$ firefox doc/index.html
```

Ett smidigt sätt att generera både scaladoc och javadoc är att använda sbt; det är bara att skriva `sbt doc` i terminalen så genereras alla dokumentation för både Scala och Java i den katalog som sbt meddelar i sin resultatutskrift.

Om du lägger in nedan i `settings` i din `build.sbt` fungerar även svenska bokstäver och andra specialtecken på alla plattformar.

```
javacOptions in (Compile, doc) ++= Seq(
  "-encoding", "UTF-8",
  "-charset", "UTF-8",
  "-docencoding", "UTF-8")
```

Du kan också använda din IDE för att köra javadoc. I Eclipse, använd menyn *Project → Generate Javadoc...*, medan du i IntelliJ hittar motsvarande i menyn *Tools → Generate Javadoc...*

Appendix G

Byggverktyg

G.1 Vad gör ett byggverktyg?

Ett **byggverktyg** (eng. *build tool*) används för att

- ladda ner,
- kompilera,
- testköra,
- paketiera och
- distribuera

programvara. Ett stort utvecklingsprojekt kan innehålla många hundra kodfiler och under utvecklingens gång vill man kontinuerligt testköra systemet för att kontrollera att allt fortfarande fungerar; även den kod som inte ändrats, men som kanske ändå påverkas av ändringen. Ett byggverktyg används för att *automatisera* denna process.

Ett viktigt begrepp i byggsammanhang är **beroende** (eng. *dependency*). Om koden X behöver annan kod Y för att fungera, sägs kod X ha ett beroende till kod Y.

I konfigurationsfiler, som är skrivna i ett format som byggverktyget kan läsa, specificeras de beroenden som finns mellan olika koddelar. Byggverktyget analyserar dessa beroenden och, baserat på ändringstidsmarkeringar för kodfilerna, avgör byggverktyget vilken delmängd av kodfilerna som behöver **omkompileras** efter en ändring. Detta snabbar upp kompileringen avsevärt jämfört med en total omkompilering från grunden, som för ett stort projekt kan ta många minuter eller till och med timmar. Efter omkompilering av det som ändrats, kan byggverktyget instrueras att köra igenom testprogram och rapportera om testernas utfall, men även ladda upp körbbara programpaket till t.ex. en webbserver.

En vanlig typ av beroende är färdiga programbibliotek som utnyttjas av systemet under utveckling, vilket i praktiken ofta innebär att en sökväg till en den kompilerade koden för programbiblioteket behöver göras tillgänglig.

I JVM-sammanhang innebär detta att sökvägen till alla nödvändiga jar-filer behöver finnas på sökvägslistan kallad **classpath**.

Många byggverktyg kan utföra så kallad **beroendeupplösning** (eng. *dependency resolution*), vilket innebär att nätverket av beroenden analyseras och rätt uppsättning programpaket görs tillgänglig under bygget. Detta kan även innebära att programpaket som är tillgängliga via nätet automatiskt laddas ned inför bygget, t.ex. via lagringsplatser för öppen källkod.

Även om man bara har ett litet kodprojekt med några få kodfiler, är det ändå smidigt att använda ett byggverktyg. Man kan nämligen göra så att byggverktyget är aktivt i bakgrunden och, så fort man sparar en ändring av koden, gör omkompileering och rapporterar eventuella kompileringsfel.

Det är klokt att kompilera om ofta, helst vid varje liten ändring, och rätta eventuella fel *innan* nya ändringar görs, eftersom det är mycket lättare att klura ut ett enskilt problem efter en mindre ändring, än att åtgärda en massa svåra följdfel, som beror på en sekvens av omfattande ändringar, där misstaget begicks någon gång långt tidigare.

En integrerad utvecklingsmiljö, så som Eclipse eller IntelliJ IDEA, bygger om koden kontinuerligt och kan ofta kommunicera med flera olika typer av byggverktyg för att i samklang med dessa automatisera byggprocessen.

Det finns många olika byggverktyg. Några allmänt kända byggverktyg med öppen källkod listas nedan, tillsammans med namnen på deras konfigurationsfiler så att du ska känna igen vilket byggverktyg som används i kodprojekt som du stöter på, t.ex. på GitHub.

- **sbt** . Även kallad *Scala Build Tool*. Användas för att bygga Java- och Scala-program i samexistens, men även för att automatisera en mängd andra saker. Byggverktyget är utvecklat i Scala och konfigurationsfilerna, som heter `build.sbt`, och innehåller Scala-kod som styr byggprocessen.
- **make**. Detta anrika byggverktyg har varit med ända sedan 1970-talet och används fortfarande för att bygga många system under Linux, coh är populärt vid utveckling med programspråken C och C++. En konfigurationsfil för `make` heter `Makefile` och har en egen, speciell syntax.
- **Apache ant**. Detta byggverktyg är utvecklat i Java som ett alternativ till `make` och används fortfarande i många Java-projekt, även om Maven och Gradle (se nedan) är vanligare numera. Konfigurationsfilerna heter `build.xml` och skrivs i det standardiserade språket XML enligt speciella regler.
- **Apache Maven**, `mvn` är också skriven i Java och är en efterföljare till `ant`. Maven används av många Java-utvecklare. Konfigurationsfilerna heter `pom.xml` och innehåller en s.k. projektobjektmodell specificerad i XML enligt speciella regler.
- **gradle** bygger vidare på idéerna från `ant` och `maven` och är skrivet i Java och Groovy. Konfigurationsfilerna skrivs i Groovy och heter `build.gradle`.

G.2 Byggverktyget sbt

Byggverktyget sbt är skrivet i Scala och är det mest populära byggverktyget bland Scala-utvecklare. Med sbt kan du skriva byggkonfigurationsfiler i Scala och även styra byggprocessen via ett interaktivt kommandoskal i terminalfönstret. Med inkrementell (stegvis) kompilering och parallelköring av byggprocessens olika delar, kan den snabbas upp avsevärt.

G.2.1 Installera sbt

sbt finns förinstallerat på LTH:s datorer och körs igång med kommandot sbt i terminalen.

Om du vill installera sbt på din egen dator, säkerställ först att du har java på din dator med terminalkommandot `java -version`. Om java saknas, följ instruktionerna i avsnitt C.2.2 på sidan 345. Följ sedan instruktionerna här för att installera sbt : <http://www.scala-sbt.org/download.html>

- **Linux.** Om du surfar till ovan sida från en Linux-dator syns några terminalkommando som du använder för att installera sbt i terminalen.
- **Windows.** Om du surfar till ovan sida från en Windows-dator visas en länk till en `.msi`-fil. Ladda ner och dubbelklicka på den.
- **macOS.** Följ instruktionerna under rubriken *Manual Installation*.

När du kör sbt första gången kommer ytterligare filer att laddas ner och installeras och delar av denna process kan ta lång tid. Ha tålamod och avbryt inte körningen, även om inget speciellt ser ut att hända på ett bra tag.

G.2.2 Använda sbt

sbt är konstruerat för att klara mycket stora projekt, men det är enkelt att använda sbt också om du bara har ett litet projekt med någon enstaka kodfil. Med sbt installerat, är det bara att skriva

```
$ sbt run
```

i terminalen i det bibliotek där dina kodfiler ligger. sbt letar då upp och kompilerar alla de `.scala`-filer som ligger i biblioteket och, om det bara finns ett objekt med main-metod, kör sbt igång denna main-metod direkt, förutsatt att kompileringen kan avslutas utan fel. Även `.java`-filer kompileras automatiskt om de ligger i samma bibliotek.

Om du enbart skriver sbt körs det interaktiva kommandoskalet igång, där du kan köra kommando så som `compile` och `run`. Om du skriver ett ~ före kommandot `run`, enligt nedan kommer sbt vara aktivt i bakgrunden medan du redigerar och så fort du sparar en ändring kommer omkompileeringen av ändrade kodfiler ske, varefter main-metoden exekveras om kompileringen lyckades.

```
$ sbt
[info] Set current project to hello (in build file:/home/bjornr/hello/)
> ~run
[info] Running hello
Hello, World!
[success] Total time: 0 s, completed Aug 9, 2016 9:50:16 PM
1. Waiting for source changes... (press enter to interrupt)
[info] Compiling 1 Scala source to /home/bjornr/hello/target/scala-2.10/classes
[info] Running hello
Hello again, World!
[success] Total time: 1 s, completed Aug 9, 2016 9:50:45 PM
2. Waiting for source changes... (press enter to interrupt)
```

I ovan körning gör sbt en omkompilering, efter att en ändring av utskriftssträngen sparats.

```
// in file hello.scala

object hello {
    def main(args: Array[String]): Unit = {
        println("Hello again, World!") // added 'again' then Ctrl+S
    }
}
```

Katalogstruktur

Om man har kod i underkataloger förutsätter sbt att du följer en viss, specifik katalogstruktur. Denna katalogstruktur används även av andra byggverktyg, så som Maven, och fungerar även i många utvecklingsmiljöer så som Eclipse och IntelliJ.

Det blir också mindre rörligt och lättare för alla att hitta i projektets kataloger om dina kodfiler placeras i en given struktur som är allmänt accepterad. Placera därför gärna dina kodfiler i underkataloger enligt strukturen som visas i figur G.1.

Lägg enligt denna struktur dina .scala-filer i underkatalogen `src/main/scala/` och dina .java-filer i underkatalogen `src/main/java/`. Om du lägger kod i biblioteken `src/test/scala/` respektive `src/test/java/` kommer denna kod köras när du skriver sbt -kommandot test. Om du lägger filer i underkatalogen `src/main/resources/` kommer dessa att paketeras med i jar-filen som skapas när du kör sbt -kommandot package.

Om du använder t.ex. `package x.y.z;` i din Java-kod, måste även strukturer på underkataloger matcha och kodfilen alltså ligga i `src/main/java/x/y/z/`.

I Scala är det egentligen inte nödvändigt att koden ligger i samma bibliotek som de kompilerade .class-filerna, men det kan vara bra att följa paketstrukturen även för Scala-källkoden; speciellt om du senare vill kunna köra din kod med Eclipse, som kräver denna överensstämmelse mellan paket och källkodskataloger, inte bara för Java, utan även för Scala.

```

src/
  main/
    resources/
      <files to include in main jar here>
    scala/
      <main Scala sources>
    java/
      <main Java sources>
  test/
    resources
      <files to include in test jar here>
    scala/
      <test Scala sources>
    java/
      <test Java sources>

```

Figur G.1: Katalogstrukturen i ett sbt -projekt. Bara de kataloger som har något innehåll behöver finnas.

Konfigurera dina byggen i filen build.sbt

Om du vill göra inställningar och även hjälpa andra att kunna återskapa dina byggen, så skapa en konfigurationsfil med namnet `build.sbt` och placera den i projektets baskatalog. Figur G.2 visar en enkel byggkonfigurationsfil. Där väljer du namn på ditt projekt, sätter ett versionsnummer på ditt bygge, samt specificerar vilken version av Scala-kompilatorn du använder. Det senare är viktigt för att andra ska kunna bygga din kod under samma förutsättningar som du.

```

lazy val root = (project in file(".")).

  settings(
    name := "hello",
    version := "1.0",
    scalaVersion := "2.11.8"
  )

```

Figur G.2: En enkel konfigurationsfil för sbt som innehåller det som kallas en *build definition* i sbt-terminologin. Filen ska ha namnet `build.sbt` och vara placerad i projektets baskatalog.

Du kan läsa mer om alla möjligheter med sbt och hur man skapar mer avancerade byggkonfigurationsfiler här:

<http://www.scala-sbt.org/0.13/docs/>

Du hittar ett exempel på en avancerad byggdefinition i kursens repo, som har många aggregerade underprojekt, bl.a. för att bygga detta kompendium med pdflatex. I byggdefinitionen instrueras även sbt att bygga kursens workspace,

samt att generera de speciell projektfiler som Eclipse+ScalaIDE kräver med en sbt -plugin. Filen finns här:

<https://github.com/lunduniversity/introprog/blob/master/build.sbt>

Lägga till beroenden

I filen build.sbt kan man lägga till s.k. beroenden till jar-filer med kod. Det finns på nätplatsen *Maven Central* en mycket omfattande koddatabas, som är sökbar här <http://search.maven.org>, med en massa användbara öppenkällkodsprojekt. Du kan be sbt att ladda ner den färdigkompilerade koden till vilket som helst av projekten på *Maven Central*. och automatiskt lägga till jar-filen till classpath så att koden blir tillgänglig direkt i ditt program.

Till exempel kan du lägga till paketet jline¹ som gör det möjligt att göra terminalinläsning från tangentbordet bara genom att lägga till denna rad i din build.sbt där "2.14.2" anger en specifik version.

```
libraryDependencies += "jline" % "jline" % "2.14.2"
```

Du kan läsa mer om hur man kan hantera beroenden med sbt här:
<http://www.scala-sbt.org/0.13/docs/Library-Dependencies.html>

¹<https://github.com/jline/jline2>

Appendix H

Versionshantering och kodlagring

H.1 Vad är versionshantering?

Versionshantering¹ (eng. *version control eller revision control*) av mjukvara innebär att hålla koll på olika versioner av koden i ett utvecklingsprojekt allteftersom koden ändras. Versionshantering är en deldisciplin inom **konfigurationshantering** (eng. *software configuration management*) som inbegriper allt i processen för att identifiera, besluta, genomföra och följa upp ändringar.

En viktig del av versionshantering är att *lägra* olika versioner av koden allt eftersom den utvecklas, så att tidigare versioner kan *återskapas* vid behov. Ett bra verktygsstöd och en väldefinierad arbetsprocess för versionshanteringen, som alla i utvecklingsprojektet följer, möjliggör att flera utvecklare kan *arbeta parallellt* med att sammanfoga (eng. *merge*) varandras tillägg och ändringar i den gemensamma kodbasen utan att det blir kaos och förvirring.

God versionshantering är helt avgörande för utvecklarnas produktivitet, speciellt för stora projekt med många utvecklare som jobbar parallellt mot en omfattande kodbas med många olika interna och externa komponenter. Men även ett litet projekt med en enda utvecklare kan ha god nytta av ett versionshanteringsverktyg och ett disciplinerat förfarande för att namnge versioner, t.ex. för att kunna återskapa tidigare versioner av projektets olika kodfiler när en ändring visar sig mindre lyckad.

Det finns flera olika modeller för hur kodlagringen sker:

- **lokal**; alla utvecklare jobbar i samma, lokala filsystem där alla olika versioner lagras.
- **centraliserad**; ett repozitorium (förk. repo), alltså en databas med koden, finns centralt på en server som alla jobbar mot med hjälp av en versionshanteringsklient.
- **distribuerad**; alla utvecklare har sitt eget lokala repo och varje utvecklare initierar enskilt delning av ändringar mellan olika repo.

¹en.wikipedia.org/wiki/Version_control

H.2 Versionshanteringsverktyget Git

Det finns många olika versionshanteringsverktyg² som använder olika modeller för kodlagring; lokal, centraliserad, distribuerad eller kombinationer därav. På senare tid har verktyget **Git**³ fått en stark ställning, speciellt i öppenkällkodsvärlden. Git utvecklades ursprungligen av Linus Torvalds för att versionshantera Linuxkärnan, men har växt till ett omfattande öppenkällkodsprojekt med stor spridning och många användare och bidragsgivare.

Git är skapat för **distribuerad** versionshantering där var och en kan jobba snabbt och smidigt i sitt eget lokala repo, utan att behöva vänta på att en klient ska synkronisera koden med ett centralt repo på en server över nätverket. Ändringar delas mellan repo på begäran av enskilda utvecklare.

Varje ny version av koden lagras som en avgränsad mängd ändringar sedan förra versionen, en s.k. **commit**⁴, och hanteras internt av Git i en lokal databas i katalogen .git som ligger överst i din projektkatalog. Genom olika kommandon i terminalen, eller via en klient med ett grafiskt användargränssnitt, kan din kod överföras till och från den lokala koddatabasen, alternativt delas med andra repos via nätet.

Det finns en välskriven bok kallad *"Pro Git"* som förklrar Git på djupet och är tillgänglig fritt här: <https://git-scm.com/book/en/v2>. Läs kapitel 1 och 2 så får du en bra grund attstå på.

Dessa termer är bra att kunna utantill innan du körs igång med Git:

- **repo** (*substantiv*: ett repositorium, *eng. a repository*) En koddatabas med ändringshistorik.
- **commit** (*substantiv*: en inlämning, *verb*: att lämna in). En avgränsad mängd nya ändringar lämnas in i det lokala repot. Repots ändringshistorik utgörs av sekvensen av alla inlämningar.
- **push** (*substantiv*: en leverans, *verb*: att leverera, att trycka upp). En eller flera inlämningar trycks upp till ett annat repo.
- **pull** (*substantiv*: en hämtning, *verb*: att hämta, att dra ner). En eller flera inlämningar dras ner från ett annat repo.
- **merge** (*substantiv*: en ihopslagning, *verb*: att sammanfoga). En eller flera inlämningar slås samman till en ny inlämning.
- **merge conflict** (*substantiv*: en sammanfogningskonflikt, *eng. a merge conflict*) Problem vid sammanfogning; ändringar kan inte enkelt sammanfogas på ett entydigt sätt.
- **pull request** (förk. PR, *substantiv*: en hämtningsbegäran, *verb*: att begära en hämtning). Utvecklare A ber en annan utvecklare B att hämta en eller flera inlämningar från A:s repo och sammanfoga med B:s repo.

²https://en.wikipedia.org/wiki/List_of_version_control_software

³[https://en.wikipedia.org/wiki/Git_\(software\)](https://en.wikipedia.org/wiki/Git_(software))

⁴På svenska kan t.ex. ”inlämning” användas, men låneordet commit är redan etablerat.

H.2.1 Installera git

Git finns förinstallerat på LTH:s Linuxdatorer. Du kan kolla om Git redan finns på din maskin genom att skriva `git help` i terminalen.

Det finns bra instruktioner om hur du installerar Git på din egen maskin här: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

Om du vill ha en Git-klient med grafiskt användargränssnitt finns det många att välja på, se här: <https://git-scm.com/downloads/guis>

Om du inte vet vilken du ska välja, prova GitKraken som är gratis (men stängd) och finns för alla plattformar: <https://www.gitkraken.com/>.

H.2.2 Anpassa Git

Innan du börjar använda git, konfigurera ditt användarnamn och din email med nedan terminalkommando, där du anger ditt användarnamn i stället för `fornamnefternamn` och din mejladress i stället för `mejladr@plats.se`:

```
$ git config --global user.name fornamnefternamn  
$ git config --global user.email mejladr@plats.se
```

Det är bra att välja *ett* användarnamn, för *alla* repo, även kodlagringsplatser på nätet; förslagsvis fornamnefternamn utan svenska tecken, så att du blir lätt att känna igen, speciellt om du jobbar med öppen källkod där ditt namn kommer associerat med alla de kodbidrag du gör under ditt yrkesliv.

Läs mer om hur du gör andra inställningar här, t.ex. hur du anger vilken editor som git startar när du ska skriva commit-beskrivningar:

<https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup>

H.2.3 Använda git

Nedan listas några vanliga terminalcommandon i Git.

- Skapa ett repo i en katalog:

```
$ cd myproject  
$ git init
```

- Se vilka filer som ändrats och ännu ej lämnats in:

```
$ git status  
$ git status -s
```

- Se vilka ändringar som gjorts i filer som ännu ej lämnats in:

```
$ git diff
```

- Se vilka inlämningar som finns i ändringshistoriken:

```
$ git log  
$ git log --oneline -5
```

- Lägg till filer som ska ingå i nästa inlämning och gör sedan inlämningen; ge inlämningen en bra beskrivning som förklarar vad inlämningen omfattar:

```
$ git add *.scala  
$ git commit -m 'initial project version'
```

- Ångra alla tillägg inför inlämning (ändringarna finns kvar och kan läggas till igen om du vill):

```
$ git reset
```

- Du kan skippa de senaste, ännu ej committade, ändringar i filen `filename`, och göra ”*undo*”, med kommandot `git checkout` på filen enligt nedan. Gör bara detta om du är helt säker på att du vill ångra dina senaste ändringar.
VARNING! Dina senaste ändringar i filen förloras för alltid; kan ej ångras!

```
$ git checkout filename
```

- Man vill förhindra versionshantering av vissa filer, t.ex. binärkodsfiler så som `.class`-filer och andra genererade filer. Detta gör du genom att skapa en fil med namnet `.gitignore` och lägga in filändelser enligt nedan syntax, där `**/` avser alla kataloger och underkataloger och `*` kan vara vilken början på ett filnamn som helst. Symbolen `#` föregår en kommentarsrad.

```
# this is my .gitignore  
  
# Java / Scala  
**/*.class  
  
# Sbt  
**/target
```

H.3 Kodlagringsplatser på nätet

Många utvecklare använder kodlagringsplatser på nätet ("i molnet"). (eng. *code hosting*) för att underlätta samarbete kring kod och för att dela med sig av öppen källkod. Det finns många olika kodlagringsplatser som kan användas gratis under vissa förutsättningar eller mot betalning med tillhörande extratjänster.

Nedan beskrivs några vanliga nätplatser för öppen och sluten kodlagring, som alla är Git-baserade:

- **GitHub**, <https://github.com>, är en av de mest populära kodlagringsplatserna för öppen källkod, men har även blivit en populär plats för jobbsökande utvecklare att visa upp sina kodarbetssprover för framtida arbetsgivare. GitHub är gratis att använda för dig som privatperson om du låter ditt repo vara öppet att läsa för alla. Det kostar pengar om du vill ha ett slutet repo. Många företag betalar GitHub för att lagra sin stängda kod med tilläggstjänster för att testa, bygga och driftsätta kod etc. Koden som styr själva kodlagringsplatsen GitHub är stängd, till skillnad från GitLab.
- **BitBucket**, <https://bitbucket.org>, är en populär kodlagringsplats både för öppen och stängd källkod och drivs av det australiensiska företaget Atlassian. Det är gratis för privatpersoner och små team att ha både öppna och slutna repon, men bara om det är få bidragsgivare. Kostnader tillkommer om antalet bidragsgivare kommer över en viss nivå. Universitetsanställda och studenter kan få mer gynnsamma villkor efter ansökan. Atlassian erbjuder en hel verktygssvit för att hantera buggar och samarbeta över nätet. BitBucket stödjer, förutom Git, även andra versionshanteringsverktyg.
- **GitLab**, <https://gitlab.com>, erbjuder gratis kodlagring för öppen källkod, men det är även gratis för privatpersoner och gemenskapsprojekt att ha stängda repo. Företag kan betala för stängd kodlagring med extratjänster för att testa, bygga och driftsätta kod etc. GitLab är i sig ett öppenkällkodspunkt och koden som styr kodlagringsplatsen är öppen och fri. Detta innebär att du själv kan ladda ner koden och starta en kodlagringsplats. LTH har en GitLab-baserad kodlagringsplats här: <https://git.cs.lth.se>

Använda kodlagringsplatser

Det är bra att registrera ditt användarnamn, förslagsvis fornamnefternamn som ett ord utan svenska tecken, på någon eller alla av ovan sajter, dels för att paxa ditt namn och dels för börja samarbeta med utvecklarvänner världen över. Om du inte vet vilken du ska välja, börja med <https://github.com>. Om du vill ha både öppna och slutna repon gratis, testa <https://gitlab.com>.

Med en Git-baserad kodlagringsplats för du möjlighet att synka ditt lokala repo mot en server på nätet med hjälp av git-kommandon i terminalen eller via en Git-klient med grafiskt användargränssnitt, se avsnitt [H.2.1](#).

Innan du börjar använda en kodlagringsplats är det bra att sätta sig in i begreppen nedan.

- **clone** (*substantiv*: en klon är kopia av ett (nätagrat) repo, *verb*: att klona, att skapa en kopia). Genom att klona ett repo som ligger på en nätagringsplats kan du bygga, undersöka och vidareutveckla koden lokalt på din dator. Om du har rättigheter att lämna in kod till det centrala orginalet kan du pusha dina commits direkt via terminalkommando eller Git-klient.
- **fork** (*substantiv*: en förgrening av ett helt repo, *verb*: att förgrena ett repo, att ”forka”). Genom att förgrena ett repo skapar du en kopia, normalt även den nätagrad på en kodlagringsplats, som du kan utveckla separat från orginalet. Det blir då möjligt för dig att lämna in ändringar och trycka upp dem, även om du inte har rättigheter att leverera (”pusha”) till orginalet. Gör en ändringsbegäran (Pull Request, PR) om du vill bidra med dina ändringar, så kan ägaren av orginalet sedan välja att sammfoga (”merga”) dina ändringar med orginalet. Många nätagringsplatser, så som GitHub, har en speciell knapp som du trycker på för att enkelt skapa en fork av ett repo under din användare.
- **upstream** (*preposition*: uppströms, *substantiv*: uppströmsrepo) Ett uppströmsrepo utgör orginal till ett förgrenat repo (en ”fork”).
 - Här beskrivs hur du länkar en förgrening uppströms:
<https://help.github.com/articles/configuring-a-remote-for-a-fork/>
 - Här beskrivs hur du synkar en förgrening uppströms:
<https://help.github.com/articles/syncing-a-fork/>

Om du vill bidra till ett öppenkällkodsprojekt, börja med att forka repot på kodlagringsplatsen och sedan klona repot till din lokala dator. Därefter kan du commita ändringar och pusha till din fork och slutligen gör en pull request från din fork till upstream. Läs om hur ett bidrag kan gå till i avsnitt [J.3](#).

Här följer några användbara kommandon:

- Skapa en lokal kopia av ett fjärran (eng. *remote*) repo; här visas hur du klonar kursens repo från GitHub:

```
$ git clone --depth 1 https://github.com/lunduniversity/introprog
```

- Dra ner nya inlämningar från ett fjärran repo:

```
$ git pull
```

- Trycka upp nya lokala inlämning till ett fjärran repo:

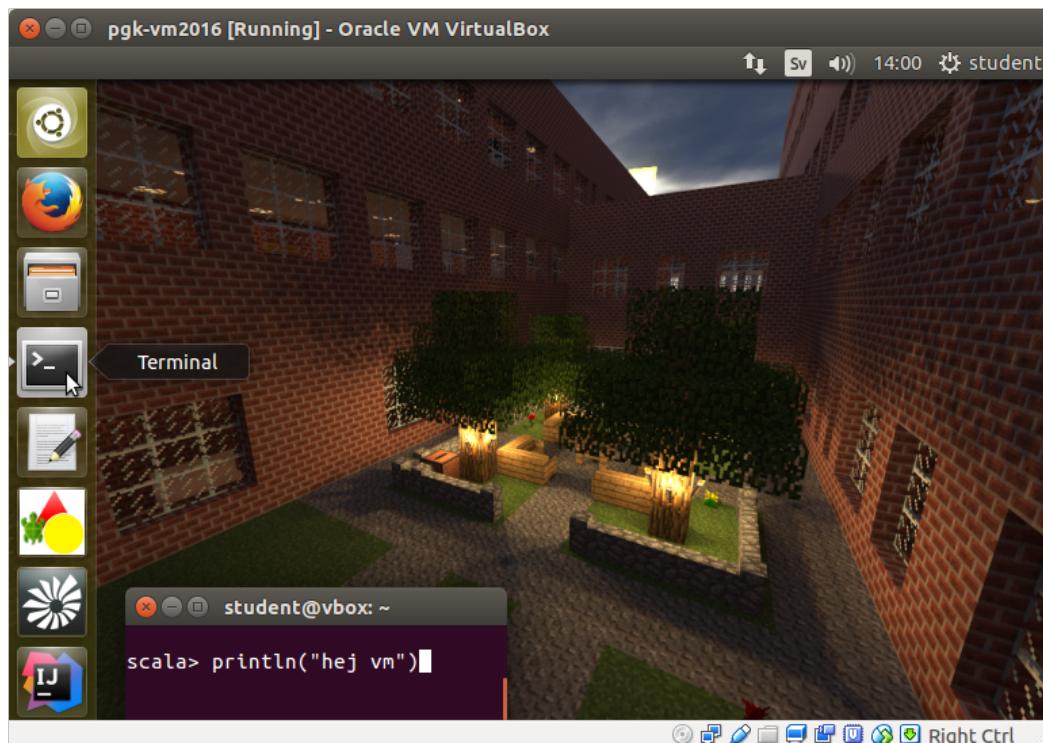
```
$ git push
```

Appendix I

Virtuell maskin

I.1 Vad är en virtuell maskin?

Du kan köra alla kursens verktyg i en så kallad **virtuell maskin** (förk. vm, eng. *virtual machine*). Detta är ett enkelt och säkert sätt köra ett annat operativsystem i en ”sandlåda” som inte påverkar din dators ursprungliga operativsystem. Figur I.1 visar kursens virtuella maskin med sin exklusiva bakgrundsbild. Exekveringen av en vm sker på en **värdator** (eng. *host*). I figur I.1 körs kursens vm i en Linux-värd med virtualiseringsapplikationen *VirtualBox*¹, som är öppen och gratis och även finns för Windows- och macOS-värdar.



Figur I.1: Den virtuella maskinen pgk-vm2016.

¹/en.wikipedia.org/wiki/VirtualBox

I.2 Vad innehåller kursens vm?

Kursens virtuell maskin har alla verktyg som du behöver förinstallerade. Vår vm kör Ubuntu 16.04.1 med fönstermiljön Unity, vilket är samma miljö som körs på Linuxdatorerna i E-huset på LTH.

Detta och mycket annat är förinstallerat i kursens vm:

- java och javac med JDK 8
- scala och scalac med Scala 2.11.8
- Kojo 2.4.09
- Eclipse Mars.2 med ScalaIDE 4.4.1
- IntelliJ IDEA 2016.2 med Scala-plugin
- gedit
- git
- sbt
- pdflatex (inkl. texlive-full, texlive-lang-europe, texworks, m.m.)
- amm 0.7.0, för Scala-skriptning, se <http://www.lihaoyi.com/Ammonite/>
- Alla skrivbordsappar som kommer med Unbuntu, t.ex. LibreOffice för ordbehandling och kalkylblad kompatibel med MS Word och MS Excel.
- Kursens repo i katalogen `~/git/lunduniversity/introprog/` inklusive `compendium/compenium.pdf` och kursens workspace.
- Den maximalt avskalade fönstermiljön <https://i3wm.org/> om du gillar att jobba effektivt med tangentbordskortkommandon. Logga först ut genom att klicka i systemmenyn längst upp till höger och välj sedan i3 i rullgardingsmenyn som trillar ner när du klickar på Ubuntu-symbolen ovanför lösenordsrutan på inloggningsskärmen.

I.3 Installera kursens vm

Det går lite längsammare att köra i en virtuell maskin jämfört med att köra direkt ”på metallen”, då det sker vissa översättningar och kontroller under virtualiseringprocessen som annars är onödiga. Och den virtuella maskinen behöver få en rejäl andel av din dators minne. Så för att köra en virtuell maskin utan att det ska bli segt behövs en ganska snabb processor, gärna över 2.5 GHz, och ganska mycket minne, gärna mer än 4GB.

Även om det går lite segt är en virtuell maskin ett utmärkt sätt att prova på Linux och Ubuntu. Eftersom man lätt kan spara undan en hel maskin är det ett bra sätt att experimentera med olika inställningar och installationer

utan att ens normala miljö påverkas. Och kör du terminalfönster och en enkel editor brukar svag prestanda och lite minne inte vara ett stort problem.²

Gör så här för att installera VirtualBox och köra kursens virtuella maskin:

1. Ladda ner VirtualBox v5 för ditt operativsystem här och installera:
<https://www.virtualbox.org/wiki/Downloads>
2. Ladda även ner ”VirtualBox Oracle VM VirtualBox Extension Pack” och installera enligt instruktionerna här:
<https://www.virtualbox.org/wiki/Downloads>
Om du stöter på problem eller undrar hur, fråga någon om hjälp.
3. Det kan hända att du får felmeddelande som innehåller något som liknar ”Intel VT-x” eller ”Hyper-V”, så som beskrivs här:
www.howtogeek.com/213795/how-to-enable-intel-vt-x-in-your-computers-bios-or-uefi-firmware
Då behöver du tillåta virtualiseringsfunktioner i BIOS på din dator. Om du inte vet hur du ska göra detta, be någon som vet om hjälp.
4. Ladda ner filen pgk-vm2016.ova här:
<http://fileadmin.cs.lth.se/pgk/pgk2016.ova>
OBS! Då filen är på nästan 10GB kan nedladdningen ta mycket lång tid, kanske flera timmar beroende på din internetuppkoppling. Har du problem med nedladdningstider kan du prova att ladda ner filen till ett USB-minne på skolans datorer, så att nedladdningen sker lokalt i E-huset.
5. Öppna VirtualBox och välj *File → Import appliance..* och bläddra till filen pgk-vm2016.ova och klicka **Next** och sedan **Import**. Själva importen kan ta lång tid, kanske flera tiotals minuter beroende på hur snabbt din dator läser från disk.
6. Markera maskinen **pgk-vm2016** och välj menyn *Machine → Settings...* (eller tryck Ctrl+S) och undersök inställningarna. Se speciellt under fliken **System** och **Motherboard** där det står hur mycket **Base memory** du tilldelar. Om du har gott om minne kan du med fördel öka minnet till 4096MB, speciellt om du tänker köra igång de tungkörda IDE-apparna Eclipse eller IntelliJ.
7. Starta maskinen **pgk-vm2016** med ett dubbelklick. Ha lite tålamod innan maskinen är igång. Du kan behöva justera skärmstorleken i värdmaskinsmenyn *View*.
8. Öppna ett terminalfönster och skriv `scala` och du är igång och kan börja göra övningarna i detta kompendium!

²Om du tycker det går alltför segt kan du istället installera Linux direkt på din dator jämsides ditt andra operativsystem – fråga någon som vet om hur man gör detta.

9. Du behöver inte logga in för att köra igång maskinen under användaren student, men du behöver lösenordet³ för att installera nya program.
10. Börja med att uppdatera mjukvaran på din virtuella maskin genom att köra dessa terminalkommando:

```
$ sudo apt-get update  
$ sudo apt-get upgrade  
$ sudo apt-get dist-upgrade
```

11. För att dra ner de senaste inlämningarna i kursrepo och uppdatera kompendiet och workspace, kör följande terminalkommando:

```
$ cd ~/git/lunduniversity/introprog  
$ git pull  
$ sbt build  
$ sbt eclipse
```

12. Om allt verkar fungera fint kan du nu prova att sätta på 3D-accelereringen för snabbare grafikrendering. Stäng maskinen genom att välja *Shutdown...* i systemmenyn. Ändra inställningar i menyn *Settings... → Display* genom att i fliken **Acceleration** under **Screen** markera **Enable 3D acceleration**. Stara maskinen. Om det fungerar så blir animeringar avsevärt snyggare och smidigare. Om det inte fungerar, stäng av maskinen med *Power off* och avmarkera **Enable 3D acceleration** igen.⁴

³pgkBytMig2016

⁴Du kan också prova att genomföra stegen som visas här, för att ominstallera vissa saker som kan ha uppdaterats sedan detta skrevs: <https://www.linuxbabe.com/virtualbox/how-to-install-virtualbox-guest-additions-on-ubuntu-step-by-step>

Appendix J

Hur bidra till kursmaterialet?

J.1 Bidrag är varmt välkomna!

Ett av huvudsyftena med att göra detta kursmaterial fritt och öppet är att möjliggöra bidrag från alla som är intresserade. Speciellt välkommet är bidrag från studenter som vill vara delaktiga i att utveckla undervisningen.

J.2 Instruktioner

J.2.1 Vad behövs för att kunna bidra?

Om du hittar ett problem, t.ex. ett enkelt stavfel, eller har något mer omfattande som borde förbättras, men ännu inte känner till eller har tillgång till de verktyg som beskriv nedan och som behövs för att göra bidrag, kontakta då någon som redan bidragit till materialet, så att någon annan kan implementera ditt förslag.

Innan du själv kan implementera ändringar direkt i materialet, behöver du känna till, och ha tillgång till, ett eller flera av följande verktyg (beroende på vad ändringen gäller):

- Latex: en.wikibooks.org/wiki/LaTeX
- Scala: [en.wikipedia.org/wiki/Scala_\(programming_language\)](https://en.wikipedia.org/wiki/Scala_(programming_language))
- git: [https://en.wikipedia.org/wiki/Git_\(software\)](https://en.wikipedia.org/wiki/Git_(software))
- GitHub: en.wikipedia.org/wiki/Github
- sbt: [en.wikipedia.org/wiki/SBT_\(software\)](https://en.wikipedia.org/wiki/SBT_(software))

Läs mer om hur du bidrar här:

github.com/lunduniversity/introprog#how-to-contribute-to-this-repo

J.2.2 Svenska eller engelska?

Vi blandar engelska och svenska enligt följande principer:

- Publika diskussioner, t.ex. i issues och pull requests på GitHub, sker på engelska. I en framtid kan delar av materialet komma att översättas till

engelska och då är det bra om även icke-engelskspråkiga kan förstå vad som har hänt. Alla ändringshändelser sparas och man kan söka och gå tillbaka i historiken.

- Kompendiet finns för närvarande bara på svenska eftersom kursen initialt endast ges för svenska språkiga studenter, men texten ska hjälpa läsaren att tillgodogöra sig motsvarande engelsk terminologi. Skriv därför mostvarande engelska begrepp (eng. *concept*) i parentes med hjälp av latex-kommandot \Eng{concept}.
- På övningar och föreläsningar är svenska variabelnamn ok. Svenska kan användas för att hjälpa den som håller på att lära sig att skilja på ord som vi själv hittar på och ord som finns i programmeringsspråket. Detta signalerar också att när man lär sig och experimenterar kan man hitta på tokrätta namn och använda svenska hur mycket man vill. Man lär sig genom att prova!
- Kod i labbar ska vara på engelska. Detta signalerar att när man kodar för att det ska bli något bestående, då kodar man på engelska.

J.3 Exempel

Som exempel på hur det går till i ett typiskt öppen-källkodsprojekt, beskrivs nedan vad som hände i ett verkligt fall: en dokumentationsuppdatering av Scala-dokumentationen efter att ett fel upptäckts. Detta exemplefall är ett typiskt scenario som illustrerar hur det kan gå till, och vad man kan behöva tänka på. Exemplet ger också länkar till och inblick i ett riktigt stort projekt med öppen källkod.

Scenario: *att göra ett bidrag vid upptäckt av problem*

”Jag fick till min stora glädje denna *Pull Request* (PR) accepterad till dokumentationssajten för Scala. Man kan se mitt bidrag här:

github.com/scala/scala.github.com/commit/7da81868ba4d74b87fe0b1

Att börja med att bidra till dokumentation är ofta en bra väg att komma in i ett open source-projekt, då det är en god chans att hjälpa till utan att det behöver kräva djup kompetens om koden i repot. Jag beskriver nedan vad som hände steg för steg då jag fick en riktig PR accepterad, som ett typiskt exempel på hur det ofta fungerar.

1. Jag tyckte dokumentationen för metoden `lengthCompare` på indexerbara samlingar på scala-lang.org/documentation var förvirrande. När jag provade i REPL blev det uppenbart att något var fel: antingen så var dokumentationen fel eller så funkade inte metoden som den skulle. Ojoj, kanske har jag upptäckt ett nytt fel? En chans att bidra!

2. Först sökte jag noga bland alla issues som ligger under fliken 'issues' på GitHub för att se om någon redan hittat detta problem. Om så vore fallet hade jag kunnat kommentera en sådan issue och skriva något till stöd för att den behöver fixas, eller allra helst att erbjuda mig att försöka fixa den. Men jag hittade ingen issue om detta...
3. Jag skapade därför ett nytt ärende genom att klicka på knappen *New issue* i webbgränssnittet på GitHub och här syns resultatet:
<https://github.com/scala/scala.github.com/issues/515#>

Jag tänkte noga på hur jag skulle formulera mig:

 - Titlen på issuen är extra viktig: den ska sammanfatta på en enda rad vad det hela rör sig om så att läsaren av rubriken förstår vad problemet är.
 - Jag jobbade sedan med att skriva en tydlig och detaljerad beskrivning av problemet och angav exakt vilken version det gällde. Det är bra att klistica in exempel från Scala REPL och andra testfallskörningar med indata och utdata om relevant. Det är viktigt att problemet går att hitta och återskapa av andra, därför behövs information om vilken version det gäller och ett minimalt testfall som renodlar problemet.
 - Det är bra att ställa frågor och komma med förslag för att öppna en diskussion om ärendet. Jag frågade speciellt om detta var ett dokumentationsproblem eller en bugg i koden.
 - OBS! Man ska inte öppna en issue innan man först kollat noga att det verkligen är något som bör åtgärdas och att det inte är en dubblett eller överlapp med andra issues: varje gång man öppnar ett ärende kommer det att generera arbete för andra även om ärendet inte ens till slut resulterade i någon åtgärd...
 - Om det är ett mer öppet, allmänt förslag, en förbättring eller en helt ny feature kan man också skapa en issue (det måste alltså inte vara en renodlad bugg). Är man osäker på om ärendet är relevant, är det bra att diskutera det i gemenskapens mejlforum först.
4. Jag fick snabbt kommentarer på min issue, vilket är kännetecknande för en väl fungerande community med alerta maintainers. Och när jag fick uppmuntran att bidra, så erbjöd jag mig att implementera förbättringen. Tänk på att alltid skriva i en saklig, kortfattad och trevlig ton!
5. Nästa steg är att "forka" repot på GitHub genom att helt enkelt klicka på *Fork* i webbgränssnittet. Jag fick då en egen kopia av repot under min egen användare på GitHub, där jag har rättigheter att ändra.
6. Därefter klonade jag repot till min lokala maskin med terminalkommandot `git clone https://...` (eller så kan man använda skrivbordsappen GitHub Desktop).
7. Sedan rättade jag problemet direkt i relevant fil i en editor på min dator, i detta fallet var filen i formatet Markdown (ett lättläst textformat som

man kan generera html från):

raw.githubusercontent.com/scala/scala.github.com/master/overviews/collections/seqs.md

8. När jag fixat problemet gjorde jag git add på filen och sedan git commit -m "välgrenomtänkt commit msg". Jag tänkte efter noga innan jag skrev första raden i commit-meddelandet så att det skulle vara både kort och kärnfullt. Men ändå glömde jag att inkludera issue-numret : (, se min kommentar till commiten, som jag tillfogade i efterhand, när jag till slut upptäckte min fadäs:
scala.github.com/commit/2624c305a8a6f24ea3398fe0fcbd0c72492bdd12#comments
9. Efter att jag gjort git commit så finns ändringen ännu så länge bara lokalt på min dator. Då gäller det att "pusha" till min fork på GitHub med git push (eller använda *Synch*-knappen i GitHub-desktop-appen).
10. Därefter skapade jag en PR genom att helt enkelt trycka på knappen *New pull request* på GitHub-sidan för min fork. Jag tänkte efter noga innan jag författade rubriken som beskriver denna PR. Hade denna ändring varit mer omfattande hade jag också behövt göra en detaljerad beskrivning av hur ändringen var implementerad för att underlätta granskningen av mitt förslag. Ni kan se denna (numera avlutade) PR här:
<https://github.com/scala/scala.github.com/pull/517>
11. När jag skapat en PR fick de som sköter repot ett automatiskt meddelande om denna nya PR och den efterföljande granskningsfasen inträdde. Den brukar sluta med att en eller flera andra personer kommenterar PR i webbgränssnittet med 'LGTM'. LGTM = "*Looks Good To Me*" och betyder ungefär "jag har kollat på detta nu och det verkar (vad jag kan bedöma) vara utmärkt och alltså redo för merge". Om det inte ser bra ut så förväntas granskaren föreslå vad som behöver förbättras i en saklig och trevlig ton.
12. När PR är granskad så kan en person, som har rättigheter att ändra, "merga" in PR på huvudgrenen, som ofta kallas *master*, i det centrala repot, som ofta kallas *upstream*.
13. Avslutningsvis kan ärendet stängas av de ansvariga för repot. Denna issue är nu markerad "Closed" och syns inte längre i listan med aktiva issues.

Puh! Sen var det klart :)"

Epilog: Om du i framtiden får chansen att göra fler bidrag är det viktigt att först uppdatera din fork mot upstream innan du gör några nya ändringar i din lokala kopia; annars är risken att din PR innehåller föråldrad information och därmed blir en merge onödigt krånglig. Detta kan man göra genom en knapp i GitHub Desktop eller genom att följa denna beskrivning: help.github.com/articles/syncing-a-fork/ Det är i allmänhet den som ändrar ansvar att se till att ändringar alltid sker i samklang med den mest aktuella versionen av upstream.