

Programmering, grundkurs

Kompendium

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

<http://cs.lth.se/pgk>

Editor: Björn Regnell

Contributors: Björn Regnell, Erik Bjäreholt, ..., and contributors

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

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

This manuscript is on-going work. Contributions are welcome!

Contact: bjorn.regnell@cs.lth.se

LICENCE: CC BY-SA 4.0

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

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

Copyright © Computer Science, LTH, Lund University. 2016. Lund. Sweden.

Framstegsprotokoll

Genomförda övningar

Till varje laboration hör en övning med uppgifter som utgör förberedelse inför labben. Du behöver minst behärska grundövningarna 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. Genom att du kryssar för nedan vilka övningar du har gjort, blir det lättare för handledaren att förstå vilka förkunskaper du har inför labben.

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 projektet 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 när du blivit godkänd på detta blad, som är ditt kvitto. Spara detta blad tills du fått slutbetyg i kursen.

Namn:

Namnteckning:

Lab	Datum gk	Handledares namnteckning
kojo
bugs
pirates
cards
turtlegraphics
turtlerace-team
chords-team
maze
surveydata-team
lthopoly-team
life
Projekt

Projektuppgift (välj en)

- () bank
- () imageprocessing
- () tictactoe
- () *egendefinerad*

Om egen, ge kort beskrivning:

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 läranderesan 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 kursrepot 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 i form av föreläsningsbilder med tillhörande anteckningar.

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 bejaka 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, bugggrättande och alla misslyckade försök som efter hårt arbete vänds till lyckade lösningar och/eller bestående lärdomar.

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

LTH, Lund 2016

Innehåll

Framstegsprotokoll	3
Förord	5
I Om kursen	9
Kursens arkitektur	11
Anvisningar	15
Samarbetsgrupper	15
Föreläsningar	15
Övningar	15
Laborationer	15
Resurstider	15
Kontrollskrivning	15
Tentamen	15
Hur bidra till kursmaterialet?	17
II Moduler	21
1 Introduktion	23
1.1 Vad är programmering?	24
1.2 Vad är en kompilator?	24
1.3 Vad består ett program av?	25
1.4 Exempel på programmeringsspråk	25
1.5 Varför Scala + Java som förstaspråk?	26
1.6 Hello world	26
1.7 Utvecklingscykeln	27
1.8 Utvecklingsverktyg	27
1.9 Övning: expressions	28
1.9.1 Grunduppgifter	28
1.9.2 Extrauppgifter: öva mer på grunderna	35
1.9.3 Fördjupningsuppgifter: avancerad nivå	36
1.10 Laboration: kojo	38
1.10.1 Obligatoriska uppgifter	38

1.10.2	Frivilliga extrauppgifter	44
2	Kodstrukturer	47
2.1	Övning: programs	48
2.1.1	Grunduppgifter	48
2.1.2	Extrauppgifter: öva mer på grunderna	57
2.1.3	Fördjupningsuppgifter: avancerad nivå	57
3	Funktioner, Objekt	59
3.1	Övning: functions	60
3.1.1	Grunduppgifter	60
3.1.2	Extrauppgifter: öva mer på grunderna	68
3.1.3	Fördjupningsuppgifter: avancerad nivå	69
3.2	Laboration: bugs	71
3.2.1	Obligatoriska uppgifter	71
3.2.2	Frivilliga extrauppgifter	71
4	Datastrukturer	73
4.0.3	Att göra denna vecka	74
4.1	Denna vecka: Fatta datastrukturer	74
4.1.1	Olika sorters datastrukturer	74
4.2	Olika sätt att skapa datastrukturer	74
4.2.1	Tupler	74
4.3	Vad är en tupel?	74
4.4	Övning: data	75
4.4.1	Grunduppgifter	75
4.4.2	Extrauppgifter: öva mer på grunderna	88
4.4.3	Fördjupningsuppgifter: avancerad nivå	89
4.5	Laboration: pirates	93
4.5.1	Obligatoriska uppgifter	93
4.5.2	Frivilliga extrauppgifter	93
5	Sekvensalgoritmer	95
5.1	Vad är en sekvensalgoritm?	96
5.2	Några indexerbara samlingar	96
5.3	Algoritm: SEQ-COPY	96
5.4	Övning: sequences	97
5.4.1	Grunduppgifter	97
5.4.2	Extrauppgifter: öva mer på grunderna	107
5.4.3	Fördjupningsuppgifter: avancerad nivå	109
5.5	Laboration: cards	112
5.5.1	Bakgrund	112
5.5.2	Obligatoriska uppgifter	112
5.5.3	Frivilliga extrauppgifter	113

6	Klasser	115
6.1	Vad är en klass?	116
6.2	Specifikationer av klasser i Scala	116
6.3	Specifikationer av klasser och objekt	117
6.4	Specifikationer av Java-klasser	118
6.5	Övning: classes	119
6.5.1	Grunduppgifter	119
6.5.2	Extrauppgifter: öva mer på grunderna	125
6.5.3	Fördjupningsuppgifter: avancerad nivå	125
6.6	Laboration: turtlegraphics	126
6.6.1	Bakgrund	126
6.6.2	Obligatoriska uppgifter	126
6.6.3	Frivilliga extrauppgifter	129
7	Arv	131
7.1	Övning: traits	132
7.1.1	Grunduppgifter	132
7.1.2	Extrauppgifter: öva mer på grunderna	135
7.1.3	Fördjupningsuppgifter: avancerad nivå	135
7.2	Laboration: turtlerace-team	136
7.2.1	Obligatoriska uppgifter	136
7.2.2	Frivilliga extrauppgifter	136
8	Mönster, Undantag	137
8.1	Övning: matching	138
8.1.1	Grunduppgifter	138
8.1.2	Extrauppgifter: öva mer på grunderna	138
8.1.3	Fördjupningsuppgifter: avancerad nivå	139
8.2	Laboration: chords-team	140
8.2.1	Obligatoriska uppgifter	140
8.2.2	Frivilliga extrauppgifter	140
9	Matriser, Typparametrar	141
9.1	Övning: matrices	142
9.1.1	Grunduppgifter	142
9.1.2	Extrauppgifter: öva mer på grunderna	143
9.1.3	Fördjupningsuppgifter: avancerad nivå	143
9.2	Laboration: maze	144
9.2.1	Bakgrund	144
9.2.2	Obligatoriska uppgifter	144
9.2.3	Frivilliga extrauppgifter	145
10	Sökning, Sortering	147
10.1	Övning: sorting	148
10.1.1	Grunduppgifter	148
10.1.2	Extrauppgifter: öva mer på grunderna	148
10.1.3	Fördjupningsuppgifter: avancerad nivå	148

10.2 Laboration: surveydata-team	149
10.2.1 Obligatoriska uppgifter	149
10.2.2 Frivilliga extrauppgifter	149
11 Scala och Java	151
11.1 Övning: scalajava	152
11.1.1 Grunduppgifter	152
11.1.2 Extrauppgifter: öva mer på grunderna	152
11.1.3 Fördjupningsuppgifter: avancerad nivå	152
11.2 Laboration: lthopoly-team	153
11.2.1 Bakgrund	153
11.2.2 Obligatoriska uppgifter	156
11.2.3 Frivilliga extrauppgifter	158
12 Trådar	159
12.1 Övning: threads	160
12.1.1 Grunduppgifter	160
12.1.2 Extrauppgifter: öva mer på grunderna	160
12.1.3 Fördjupningsuppgifter: avancerad nivå	160
12.2 Laboration: life	161
12.2.1 Bakgrund	161
12.2.2 Reglerna	161
12.2.3 Obligatoriska uppgifter	162
12.2.4 Frivilliga extrauppgifter	162
13 Design	165
13.1 Projektuppgift: bank	166
13.1.1 Bakgrund	166
13.1.2 Krav	166
13.1.3 Design	167
13.1.4 Obligatoriska uppgifter	169
13.1.5 Frivilliga extrauppgifter	170
13.2 Projektuppgift: tictactoe	171
13.2.1 Regler	171
13.2.2 Teori	171
13.2.3 Obligatoriska uppgifter	171
13.2.4 Frivilliga extrauppgifter	172
13.3 Projektuppgift: imageprocessing	173
13.3.1 Bakgrund	173
13.3.2 Uppgiften	173
13.3.3 Frivilliga extrauppgifter	177
14 Tentaträning	179

III	Appendix	181
A	Virtuell maskin	183
A.1	Vad är en virtuell maskin?	183
A.2	Installera kursens vm	183
A.3	Vad innehåller kursens vm?	184
B	Terminalfönster och kommandoskal	185
B.1	Vad är ett terminalfönster?	185
B.2	Några viktiga terminalkommando	185
C	Editera	187
C.1	Vad är en editor?	187
C.2	Välj editor	187
D	Kompilera och exekvera	189
D.1	Vad är en kompilator?	189
D.2	Java JDK	189
D.2.1	Installera Java JDK	189
D.3	Scala	189
D.3.1	Installera Scala-kompilatorn	189
D.4	Read-Evaluate-Print-Loop (REPL)	189
D.4.1	Scala REPL	189
E	Dokumentation	191
E.1	Vad gör ett dokumentationsverktyg?	191
E.2	scaladoc	191
E.3	javadoc	191
F	Integrerad utvecklingsmiljö	193
F.1	Vad är en IDE?	193
F.2	Kojo	193
F.2.1	Installera Kojo	193
F.2.2	Använda Kojo	193
F.3	Eclipse och ScalaIDE	193
F.3.1	Installera Eclipse och ScalaIDE	193
F.3.2	Använda Eclipse och ScalaIDE	193
G	Byggverktyg	195
G.1	Vad gör ett byggverktyg?	195
G.2	Byggverktyget sbt	195
G.2.1	Installera sbt	195
G.2.2	Använda sbt	195
H	Versionshantering och kodlagring	197
H.1	Vad är versionshantering?	197
H.2	Versionshanteringsverktyget git	197
H.2.1	Installera git	197

H.2.2	Använda git	197
H.3	Vad är nyttan med en kodlagringsplats?	197
H.4	Kodlagringsplatsen GitHub	197
H.4.1	Installera klienten för GitHub	197
H.4.2	Använda GitHub	197
H.5	Kodlagringsplatsen Atlassian BitBucket	197
H.5.1	Installera SourceTree	197
H.5.2	Använda SourceTree	197
I	Nyckelord	199
I.1	Vad är ett nyckelord ord?	199
I.2	Nyckelord i Scala	199
I.3	Nyckelord i Java	199
J	Lösningförslag till övningar	201
J.1	expressions	202
J.1.1	Grunduppgifter	202
J.1.2	Extrauppgifter: öva mer på grunderna	202
J.1.3	Fördjupningsuppgifter: avancerad nivå	202
J.2	programs	203
J.2.1	Grunduppgifter	203
J.2.2	Extrauppgifter: öva mer på grunderna	203
J.2.3	Fördjupningsuppgifter: avancerad nivå	203
J.3	functions	204
J.3.1	Grunduppgifter	204
J.3.2	Extrauppgifter: öva mer på grunderna	204
J.3.3	Fördjupningsuppgifter: avancerad nivå	204
J.4	data	205
J.4.1	Grunduppgifter	205
J.4.2	Extrauppgifter: öva mer på grunderna	205
J.4.3	Fördjupningsuppgifter: avancerad nivå	205
J.5	sequences	206
J.5.1	Grunduppgifter	206
J.5.2	Extrauppgifter: öva mer på grunderna	206
J.5.3	Fördjupningsuppgifter: avancerad nivå	206
J.6	classes	207
J.6.1	Grunduppgifter	207
J.6.2	Extrauppgifter: öva mer på grunderna	207
J.6.3	Fördjupningsuppgifter: avancerad nivå	207
J.7	traits	208
J.7.1	Grunduppgifter	208
J.7.2	Extrauppgifter: öva mer på grunderna	208
J.7.3	Fördjupningsuppgifter: avancerad nivå	208
J.8	matching	209
J.8.1	Grunduppgifter	209
J.8.2	Extrauppgifter: öva mer på grunderna	209
J.8.3	Fördjupningsuppgifter: avancerad nivå	209

J.9	matrices	210
J.9.1	Grunduppgifter	210
J.9.2	Extrauppgifter: öva mer på grunderna	210
J.9.3	Fördjupningsuppgifter: avancerad nivå	210
J.10	sorting	211
J.10.1	Grunduppgifter	211
J.10.2	Extrauppgifter: öva mer på grunderna	211
J.10.3	Fördjupningsuppgifter: avancerad nivå	211
J.11	scalajava	212
J.11.1	Grunduppgifter	212
J.11.2	Extrauppgifter: öva mer på grunderna	212
J.11.3	Fördjupningsuppgifter: avancerad nivå	212
J.12	threads	213
J.12.1	Grunduppgifter	213
J.12.2	Extrauppgifter: öva mer på grunderna	213
J.12.3	Fördjupningsuppgifter: avancerad nivå	213

K	Ordlista	215
----------	-----------------	------------

Del I

Om kursen

Kursens arkitektur

Veckoöversikt

<i>W</i>	<i>Modul</i>	<i>Övn</i>	<i>Lab</i>
W01	Introduktion	expressions	kojo
W02	Kodstrukturer	programs	–
W03	Funktioner, Objekt	functions	bugs
W04	Datastrukturer	data	pirates
W05	Sekvensalgoritmer	sequences	cards
W06	Klasser	classes	turtlegraphics
W07	Arv	traits	turtlerace-team
KS	KONTROLLSKRIVN.	–	–
W08	Mönster, Undantag	matching	chords-team
W09	Matriser, Typparametrar	matrices	maze
W10	Sökning, Sortering	sorting	surveydata-team
W11	Scala och Java	scalajava	lthopoly-team
W12	Trådar	threads	life
W13	Design	Uppsamling	Projekt
W14	Tentaträning	Extenta	–
T	TENTAMEN	–	–

Kursen består av ett antal moduler med tillhörande teori, övningar och laborationer. Genom att göra övningarna bearbetar du teorin och förbereder dig inför laborationerna. När du klarat av övningarna och laborationen i en modul är du redo att gå vidare till nästa modul.

Vad lär du dig?

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

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 föreläsningsanteckningar, övningar & laborationer
- Säljs på KFS
<http://www.kfsab.se/>

Rekommenderade böcker

För nybörjare:



För de som redan kodat en del:



Kursmoment — varför?

- **Föreläsningar:** skapa översikt, ge struktur, förklara teori, svara på frågor, motivera varför
- **Övningar:** bearbeta teorin med avgränsade problem, grundövningar för alla, extraövningar om du behöver öva mer, fördjupningsövningar om du vill gå vidare, **förberedelse** inför laborationerna
- **Laborationer:** lösa programmeringsproblem praktiskt, **obligatoriska** uppgifter; lösningar redovisas för handledare
- **Resurstider:** få hjälp med övningar och laborationsförberedelser av handledare, fråga vad du vill
- **Samarbetsgrupper:** grupplärande 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
- **Tenta:** Skriftlig tentamen utan hjälpmedel, förutom **snabbreferens**.

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 med **individuellt** med teori, övningar, labbförberedelser på **måndag-torsdag**
3. Kom till **resurstiderna** 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

Anvisningar

Samarbetsgrupper

Samarbetskontrakt

Föreläsningar

Övningar

Laborationer

Resurstider

Kontrollskrivning

Tentamen

Hur bidra till kursmaterialet?

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.

Instruktioner

Vad behöver jag 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 beskrivs 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_%28programming_language%29
- git: https://en.wikipedia.org/wiki/Git_%28software%29
- GitHub: en.wikipedia.org/wiki/Github
- sbt: en.wikipedia.org/wiki/SBT_%28software%29

Läs mer om hur du bidrar här:

github.com/lunduniversity/introprog#how-to-contribute-to-this-repo

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 svenskspråkiga studenter, men texten ska hjälpa läsaren att tillgodogöra sig motsvarande engelsk terminologi. Skriv därför motsvarande 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 tokroliga 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.

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 exempelfall 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 klistra 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 issuen inte ens till slut åtgärdas...
 - 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 "välgenomtänkt 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 avslutade) 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, "merge" in PR på huvudgrenen, som ofta kallas *master*, i det centrala repot, som ofta kallas *upstream*.
13. Avslutningsvis kan issuen stängas av de ansvariga för repot. Issuen 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 ändrars ansvar att se till att ändringar alltid sker i samklang med den mest aktuella versionen av upstream.

Del II

Moduler

Kapitel 1

Introduktion

Koncept du ska lära dig denna vecka:

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

1.1 Vad är programmering?

- Programmering innebär att ge instruktioner till en maskin.
- Ett **programmeringsspråk** används av människor för att skriva **källkod** 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.
- Ha picknick i Ada Lovelace-parken på Brunshög!
- sv.wikipedia.org/wiki/Programmering
- en.wikipedia.org/wiki/Computer_programming
- kartor.lund.se/wiki/lundanamn/index.php/Ada_Lovelace-parken



1.2 Vad är en kompilator?



Grace Hopper uppfann första kompilatorn 1952.

en.wikipedia.org/wiki/Grace_Hopper



1.3 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**
- **Deklaration**: 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

1.4 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](#)



1.5 Varför Scala + Java som förstaspråk?

- Varför Scala?
 - Enkel och enhetlig syntax => lätt att skriva
 - Enkel och enhetlig semantik => lätt att fatta
 - Kombinerar flera angreppssätt => 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 plattformar
 - Effektivitet: avancerad & mogen teknik ger snabba program
- Java och Scala fungerar utmärkt tillsammans
- Illustrera likheter och skillnader mellan olika språk
=> Djupare lärande

1.6 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!");  
  }  
}
```


1.7 Utvecklingscykeln

editera; kompilera; hitta fel och förbättringar; editera; kompilera; hitta fel och förbättringar; editera; kompilera; hitta fel och förbättringar; editera; kompilera; hitta fel och förbättringar; editera; kompilera; hitta fel och förbättringar; editera; kompilera; hitta fel och förbättringar; ...

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

1.8 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: från övn 2
 - Kompilera med **scalac** och **javac**: från övn 2
 - Integrerad utvecklingsmiljö (IDE)
 - * **Kojo**: från lab 1
 - * **Eclipse** med plugin **ScalaIDE**: från lab 3
 - **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!

1.9 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 deklarerar 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 booelska uttryck och värdena **true** och **false**, samt kunna förenkla booelska uttryck.
- ☐ Förstå skillnaden mellan heltalsdivision och flyttalsdivision, samt användning av rest vid heltalsdivision.
- ☐ Förstå precedensregler och användning av parenteser i uttryck.
- ☐ Kunna använda **if**-satser och **if**-uttryck.
- ☐ Kunna använda **for**-satser och **while**-satser.
- ☐ Kunna använda `math.random` för att generera slumptal i olika interval.

Förberedelser

- ☐ Studera teorin i kapitel 1.
- ☐ Du behöver en dator med Scala installerad, se appendix D.

1.9.1 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 2.11.7 (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")
```

- a) Vad händer?
- b) Skriv samma sats igen men "glöm bort" att skriva högerparentesen innan du trycker på *Enter*. Vad händer?
- c) 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"
```

- d) Evaluera uttrycket `res0 * 42` men byt ut `0`:an mot siffran efter `res` i utskriften från förra evalueringen. Vad har uttrycket för värde och typ?

```
scala> res0 * 42
```



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?

- a) 42
- b) 42L
- c) '*'
- d) "*"
- e) 42.0
- f) 42D
- g) 42d
- h) 42F
- i) 42f
- j) **true**
- k) **false**



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

```
scala> def p = { print("hej"); print("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 returtyp har println?

Uppgift 6. Vilken typ och vilket värde har följande uttryck?

- a) 1 + 41
- b) 1.0 + 41
- c) 42.toDouble
- d) (41 + 1).toDouble
- e) 1.042e42
- f) 42E6.toLong
- g) "gurk" + 'a'

- h) 'A'
- i) 'A'.toInt
- j) '0'.toInt
- k) '1'.toInt
- l) '9'.toInt
- m) ('A' + '0').toChar
- n) "?!%#".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) $42L + 2d$

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

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

Uppgift 9. *Heltalsdivision.* Vilket värde och vilken typ har följande uttryck?

- a) $42 / 2$
- b) $42 / 4$
- c) $42.0 / 4$
- d) $1 / 4$
- e) $1 \% 4$
- f) $45 \% 42$
- g) $42 \% 2$

Uppgift 10. *Heltalsomfång.* För var och en av heltalstyperna 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`.

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

-
- 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?
- 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. Booelska 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) 42.000000000000000001 == 42
- m) 42.0001 > 42
- n) 42.000000000000000001 > 42
- o) 42.0001 >= 42
- p) 42.000000000000000001 <= 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 = 42
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

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

```
1 scala> var x = 42
2 scala> x + 1
3 scala> x
4 scala> x = x + 1
5 scala> x
6 scala> x == x + 1
7 scala> val y = 42
8 scala> y = y + 1
9 scala> var z = {println("gurka"); 42}
10 scala> def w = {println("gurka"); 42}
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`? 

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 = 42
2 scala> var b = a + 42
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 variabelvärden. Med ett `s` framför en strängliteral får man hjälp av kompilatorn att, på ett typsäkert sätt, infoga variabelvärden i en sträng. Variablernas namn ska föregås med ett `$`-tecken, t.ex. `s"Hej $namn"`.

```
1 scala> val f = "Kim"
2 scala> val e = "Robinson"
3 scala> val tot = f.size + e.size
4 scala> println(s"Namnet '$f $e' har $tot 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 'Robinson' 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 kasta = if (math.random > 0.5) print(" krona") else print(" klave")
6 scala> kasta; kasta; kasta
```

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.

- 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 foreach.

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.

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>

- Vilken typ har värdet som returneras av funktionen `random`? 
- Vilket är det minsta respektive största värde som kan returneras? 
- Är `random` en *äkta* funktion (eng. *pure function*) i matematisk mening? 
- 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 10) println(math.random)
```

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`

1.9.2 Extrauppgifter: öva mer på grunderna

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`.)

1.9.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 26. Läs om modulatoräkning här en.wikipedia.org/wiki/Modulo_operation och undersök hur tecknet blir med olika tecken på divisor och dividend.

Uppgift 27. `Integer.toString`, `Integer.toHexString`

Uppgift 28. Typannoteringar.

Uppgift 29. `0x2a`

Uppgift 30. `i += 1; i *= 1; i /= 2`

Uppgift 31. `BigInt`, `BigDecimal`

Uppgift 32. Vad händer här?

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

Uppgift 33. Sök reda på dokumentationen i javadoc för klassen `java.lang.Math` i JDK 8. Tryck Ctrl+F i webbläsaren och sök efter förekomster av texten "overflow". Vad är "overflow"? Vilka metoder finns i `java.lang.Math` som hjälper dig att upptäcka om det blir overflow?

Uppgift 34. 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 35. 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.toString(Int.MaxValue + 1).size
```

```
Integer.toString((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 36. 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
```

1.10 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 lätta 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 succesivt bygga upp allt mer utvecklade program.

Förberedelser

- ☐ Gör övning expressions i kapitel 1.9.
- ☐ Läs 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 F.2.

1.10.1 Obligatoriska uppgifter

Uppgift 1. Sekvens.

a) Starta Kojo. Om du inte redan har svenska menyer: välj svenska i språkmenyn 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 F.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 lätta för en människa att läsa?

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

h) Rita en 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.

- 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.
- 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 att exekveras? Använd lämpligt argument till `sakta` för att du ska hinna studera exekveringen.
- 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) Om du anropar `sakta(0)`, hur många kvadratvarv hinner sköldpaddan rita på en sekund? Använd nedan program för att ta reda på ungefärligt antal varv per sekund.

```
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 loop-variabel.

```
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) Vad är fördelar och nackdelar med de olika sätten att loopa: `upprepa`, **while**, **for**, respektive `foreach`? Diskutera dina svar med en handledare. 

Uppgift 3. Abstraktion.

a) Använd en repetition för 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 satser. Vid vilka punkter i programmet sker ett ”hopp” i sekvensen i stället för att efterföljande sats att 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. Vilket av de två sätten (med och utan abstraktionen *kvadrat*) är lättast att läsa?
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.

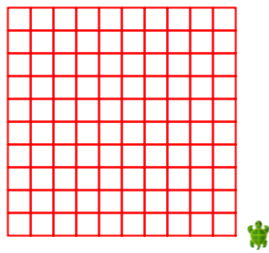
```
def kvadrat = ???
def stapel(n: Int) = ???

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?
-  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 editor-kortkommando. Notera skillnaderna. Vilket autoformateringsprogram gör programmet lättare att läsa? Vilka manuella formateringsåtgärder 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'); 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) Lägg till nedan kod i början av programmet och gör så att när man trycker på tangenten G så sätter man omväxlande på och av rutnätet.

```
var isGridOn = false
```

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

d) 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. Visa din lösning för en handledare.

1.10.2 Frivilliga extrauppgifter

Uppgift 6. *Tidmätning.* Hur snabb är din dator?

a) Skriv in koden nedan i Kojos editor och kör upprepade gånger med den gröna play-knappen. Hur lång tid tar det för din dator att räkna till 4.4 miljarder?¹

```
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 < 4400000000L) { i += 1 }
val t = timer.elapsedSeconds
println("Räknade till " + i + " på " + t + " sekunder.")
```

b) 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 rundor i while-loopen i föregående uppgift? (Det kan hända att din dator kan variera centralprocessorns klockfrekvens. Prova både medan du kör tidmätningen i Kojo och då

¹Det går att göra ungefär en heltalsaddition per klockcykel per kärna. Den första elektroniska datorn Eniac hade en klockfrekvens motsvarande 5kHz. Björn Regnells dator har en i7-4790K som turboklockar på 4.4 GHz.

www.extremetech.com/computing/185512-overclocking-intels-core-i7-4790k-can-devils-canyon-fix-haswells-low-clock-speeds/2

din dator ”vilar”. Vad är det för poäng med att en processor kan variera sin klockfrekvens?)

```
1 > lscpu | grep MHz
```

c) 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.

d) 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)
}
```

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

Uppgift 7. 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.

```
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           instanceof
9 getAlpha      getRGB            toString
10 getBlue      getRGBColorComponents
11
12 scala> c.getAlpha
13 res3: Int = 100
```



Uppgift 8. Ladda ner dessa pdf-kompendier och gör några uppgifter som du tycker verkar intressanta:

- "Uppdrag med Kojo" som kan laddas ner här:
fileadmin.cs.lth.se/cs/Personal/Bjorn_Regnell/uppdrag.pdf
- "Programming Fundamentals with Kojo" som kan laddas ner här:
wiki.kogics.net/kojo-codeactive-books

Kapitel 2

Kodstrukturer

Koncept du ska lära dig denna vecka:

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

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 heltalsvärden som resultat av enkla for-uttryck.
- ☐ Förstå skillnaden mellan en algoritm i pseudo-kod och dess implementation.
- ☐ 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 implementera och köra igång ett Java-program.
- ☐ Känna till några grundläggande syntaxskillnader mellan Scala och Java, speciellt variabeldeklarationer och indexering i Array.
- ☐ Förstå vad ett block är.
- ☐ Förstå vad en lokal variabel är.
- ☐ Förstå hur nästlade block påverkar namnsynlighet och namnöverskuggning.
- ☐ Förstå kopplingen mellan paketstruktur och klassfilstruktur.
- ☐ Kunna skapa en jar-fil.
- ☐ Kunna skapa dokumentation med scaladoc.

Förberedelser

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

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`

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("0rd(", ", ", ", ")")`
- 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. Algoritmen: SWAP.

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

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 med hjälp av en editor en fil med namn `hello-script.scala` 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  $sum \leftarrow 0$ 
2  $i \leftarrow 1$ 
3 while  $i \leq n$  do
4    $sum \leftarrow sum + 1$ 
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 en bugg i algoritmen. Vilken? Rätta buggen.

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 standardfunktioner 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. *Algorithm: 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 heltal
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?

- a) `val a = {1 + 1; 2 + 2; 3 + 3; 4 + 4}; println(a)`
- b) `val b = {1; 2; 3; {val b = 4; b + b; b + 1}}; println(b)`
- c) `{val a = 42; println(a)}`
- d) `{val a = 42}; println(a)`
- e) `{val a = 42; {val a = 43; println(a)}; println(a)}`
- f) `{var a = 42; {a = a + 1}; var a = 43}`
- g) `{var a = 42; {a = a + b; var b = 43}; println(a)}`
- h) `{var a = 42; {var b = 43; a = a + b}; println(a)}`
- i) `{var a = 42; {a = a + b; def b = 43}; println(a)}`
- j) `{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-java-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 paketedokumentationskommentar p2 */
package p2 {
  /** min paketedokumentationskommentar 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: öva mer på grunderna

2.1.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 17. `ArrayBuffer` vs `Vector` vs `Array` och metoden `append`

Uppgift 18. Läs om krullparenteser och vanliga parenteser på stack overflow: [what-is-the-formal-difference-in-scala-between-braces-and-parentheses-and-when](#)

Uppgift 19. 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 `%`-operatoren för att avgöra vem som vinner.

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

```
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  
    }  
  }  
}
```


Kapitel 3

Funktioner, Objekt

Koncept du ska lära dig denna vecka:

- | | |
|--|--|
| <input type="checkbox"/> definera funktion | <input type="checkbox"/> metod |
| <input type="checkbox"/> anropa funktion | <input type="checkbox"/> medlem |
| <input type="checkbox"/> parameter | <input type="checkbox"/> funktionsvärde |
| <input type="checkbox"/> returtyp | <input type="checkbox"/> funktionstyp |
| <input type="checkbox"/> värdeandrop | <input type="checkbox"/> äkta funktion |
| <input type="checkbox"/> namnanrop | <input type="checkbox"/> stegad funktion |
| <input type="checkbox"/> default-argument | <input type="checkbox"/> apply |
| <input type="checkbox"/> namngivna argument | <input type="checkbox"/> lazy val |
| <input type="checkbox"/> applicera funktion på alla element i en samling | <input type="checkbox"/> lokala funktioner |
| <input type="checkbox"/> procedur | <input type="checkbox"/> anonyma funktioner |
| <input type="checkbox"/> värdeanrop vs namnanrop | <input type="checkbox"/> lambda |
| <input type="checkbox"/> uppdelad parameterlista | <input type="checkbox"/> aktiveringspost |
| <input type="checkbox"/> skapa egen kontrollstruktur | <input type="checkbox"/> rekursion |
| <input type="checkbox"/> objekt | <input type="checkbox"/> basfall |
| <input type="checkbox"/> modul | <input type="checkbox"/> anropsstacken |
| <input type="checkbox"/> punktnotation | <input type="checkbox"/> objektheapen |
| <input type="checkbox"/> tillstånd | <input type="checkbox"/> 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 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 med hjälp av anropsstacken.
- ☐ Känna till svansrekursion och att svansrekursiva funktioner kan optimeras till loopar.

Förberedelser

- ☐ Studera teorin 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) Definera funktionen minska(x: Int): Int med returvärdet x - 1.
- h) Vad är värdet av uttrycket öka(minska(öka(öka(minska(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? Förklara vad som händer.


- a) diff(0, 100)
- b) diff(100, add(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.* Förklara 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", "Robinson")
4 scala> skrivNamn(förnamn = "Viktor", efternamn = "Oval")
5 scala> skrivNamn(efternamn = "Triangelsson", förnamn = "Stina")
```

- a) Förklara 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? Förklara 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(enhetsvärdet)`
- 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. Använd fördröjd evaluering och stegad funktion och skapa din egen loop-konstruktion.

```
1 scala> def upprepa(n: Int)(block: => Unit) = {
2     var i = 0
3     while (i < n) {block; i = i + 1}
4 }
5 scala> upprepa(10)(println("hej"))
6 scala> upprepa(1000){
7     val tärning = (math.random * 6 + 1).toInt
8     print(tärning + " ")
9 }
```

Förklara vad som händer ovan. (Det är så som upprepa i Kojo är definierad.)

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 funktionsvärde 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 räkna(x: Int, f: Int => Int) = f(x,y)
2 scala> def dubbla(x: Int) = 2 * x
3 scala> def halva(x: Int) = x / 2
4 scala> val d = räkna(42, dubbla)
5 scala> val h = räkna(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 braxer. 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.

```
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 objektets är 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) Förklara 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 slumpal i intervallet $[a, b]$. Anropa objektets `apply`-metod med `(1 to 100).foreach(println(???))`, där `???` ersätts först med punktnotation och sedan med funktionsapPLICERINGSSYNTAX.

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

- a) Förklara vad som händer här:

```
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? Förklara vad som händer.
-  b) Rita hur anropsstacken förändras under exekveringen av main-metoden.

Uppgift 17. *Lokala funktioner.* Skapa nedan program i en editor, kompilera 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? Förklara 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 `def öka(i: Int): Int = i + 1`, som har namnet `öka`.

Anonyma funktioner kallas även för *funktionslitteraler* 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 funktionsliterals 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: öva mer på grunderna

Uppgift 20. Visa anropsstacken genom att kasta undantag.

3.1.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 21. *Kolla bajtkoden.*

```
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 bajtkoden med `:javap plusxyz`. Vad skiljer bajtkoden mellan `plusxy` och `plusxyz`?



c) Läs om bajtkod här: en.wikipedia.org/wiki/Java_bytecode. Vad betyder den inledande bokstaven i additionsinstruktionen?

Uppgift 22. *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 23. *@tailrec-annotering.* Du kan be kompilatorn att ge felmeddelande om den inte kan optimera koden till en loop och därmed öka prestanda och undvika en överfull anropsstack (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(1000000L)
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: bugs

Mål

- ☐ Kunna definiera och anropa funktioner.
- ☐ Kunna definiera default-argument.
- ☐ Kunna ange parametervärden med parameternamn.

Förberedelser

- ☐ Att göra.

3.2.1 Obligatoriska uppgifter

Uppgift 1. En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.

3.2.2 Frivilliga extrauppgifter

Uppgift 2. En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.

Kapitel 4

Datastrukturer

Koncept du ska lära dig denna vecka:

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

4.0.3 Att göra denna vecka

4.1 Denna vecka: Fatta datastrukturer

- Läs teori
- Gör övning data
- Gör lab ???

4.1.1 Olika sorters datastrukturer

4.2 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 (operationer på data)
- Samlingar
 - speciella klasser som samlar data i element av **samma** typ
 - finns ofta *många* färdiga **bra-att-ha-metoder**

4.2.1 Tupler

4.3 Vad är en tupel?

`("hej", 42, math.Pi)` är en 3-tupel med typ: `(String, Int, Double)`

4.4 Övning: data

Mål

- ☐ Kunna skapa och använda tupler, som variabelvärden, parametrar och returvärden.
- ☐ 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 klassparametrar.
- ☐ Beskriva innebörden 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 teorin i kapitel 4.

4.4.1 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 understreckt 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 nOdd = (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. Poster av typen Konto håller reda på hur mycket pengar det finns på kontot och vilket kontonumret är:

```

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 initialvärden när objektet skaps med **new** kan placera attributen i en parameterlista till klassen. Kod som körs när objektet skapas och attributen 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 versus 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 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 därmed 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> p1.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)
```

- Vad händer egentligen på sista raden ovan?
- Omdefiniera `toString` så att den ger en sträng på formen `Punkt(1.0, 42.0)`.
- 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)
}
```

- Skriv satser som använder metoden `apply` i fabriksobjektet **object** `Pt` för att skapa flera olika punkter.
- Ge `apply`-metoden default-argument 0.0 för både `x` och `y` så att `Pt()` skapar en punkt i origo.
- 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.
- Skapa ett fabriksobjekt med en `apply`-metod som tar två heltalsparametrar och skapar en instans av klassen `Rational`.
- 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))
```

- 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 length: Double = math.hypot(x, y)
```

```

3     def add(p: Point): Point = Point(x + p.x, y + p.y)
4     def +(p: Point): Point = Point(x + p.x, y + p.y)
5 }

```

- a) Använd metoden `length` 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`. Vad får uttrycken `p3.x` resp. `p3.y` för värden?

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 kompilatorn kan härleda typen utifrån argumenten. Om man anger typparametrar explicit så hjälper kompilatorn 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. Förklara 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("Odlä", "och ät!")(vego)
12 scala> wrap("Odlä", "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 hjälp 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 hjälp 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. *Jämföra 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 dubletter. 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 förvånande ordning.

```
1 scala> val s = Set("Malmö", "Stockolm", "Göteborg", "Köpenhamn", "Oslo")
2 s: scala.collection.immutable.Set[String] =
3   Set(Oslo, Malmö, Köpenhamn, Stockolm, 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) intersect Set(2,3,4)`
- m) `Set(1,2,3) & Set(2,3,4)`
- n) `Set(1,2,3) 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))
```

- b) Vad har variabeln 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, get.

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

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

- 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.

4.4.2 Extrauppgifter: öva mer på grunderna

Uppgift 23. Träna mer på klass

```
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 24. Träna mer på mängd

- a) Keno-bollar.

4.4.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 25. *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 26. *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 27. *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 28. 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> :paste

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
}

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: 47.1852 ns
Average time: 41.64156 ns
Average time: 105.53986 ns
Average time: 41.91562 ns
Average time: 41.73559 ns
Average time: 63.17134 ns
Average time: 52.93756 ns
Average time: 41.58533 ns
Average time: 41.68017 ns
Average time: 60.18881 ns
Average time: 41.69867 ns
Average time: 41.60771 ns
Average time: 60.32759 ns
Average time: 41.62671 ns
Average time: 43.88916 ns
Average time: 70.47824 ns
Average time: 41.68801 ns
Average time: 41.67223 ns
Average time: 41.67262 ns
Average time: 102.84893 ns
res85: Double = 41.58533

scala> (for(i <- 1 to 20) yield time(n){v.take(10)}).min
Average time: 312.67005 ns
Average time: 88.60023 ns
Average time: 73.21829 ns
Average time: 92.148 ns
Average time: 91.01078 ns
Average time: 87.82874 ns
Average time: 74.04663 ns
Average time: 94.16038 ns
Average time: 88.4243 ns
```



```
Average time: 105.88971 ns
Average time: 98.85731 ns
Average time: 72.77369 ns
Average time: 97.04337 ns
Average time: 90.01969 ns
Average time: 88.11196 ns
Average time: 75.20191 ns
Average time: 93.72112 ns
Average time: 110.19777 ns
Average time: 132.4207 ns
Average time: 324.28702 ns
res86: Double = 72.77369

scala> (for(i <- 1 to 20) yield time(1000){l.takeRight(10)}).min
Average time: 247365.43 ns
Average time: 212801.958 ns
Average time: 212335.938 ns
Average time: 212313.427 ns
Average time: 212524.963 ns
Average time: 219525.627 ns
Average time: 223059.563 ns
Average time: 222426.504 ns
Average time: 221838.828 ns
Average time: 223268.567 ns
Average time: 222739.402 ns
Average time: 222685.229 ns
Average time: 223122.599 ns
Average time: 222683.921 ns
Average time: 222865.865 ns
Average time: 222889.118 ns
Average time: 223247.135 ns
Average time: 222016.82 ns
Average time: 223040.299 ns
Average time: 222624.613 ns
res87: Double = 212313.427

scala> (for(i <- 1 to 20) yield time(1000){v.takeRight(10)}).min
Average time: 2665.715 ns
Average time: 190634.043 ns
Average time: 773.111 ns
Average time: 509.008 ns
Average time: 519.04 ns
Average time: 418.172 ns
Average time: 365.54 ns
Average time: 409.016 ns
Average time: 353.115 ns
Average time: 503.679 ns
Average time: 421.369 ns
Average time: 388.685 ns
Average time: 461.725 ns
Average time: 390.791 ns
Average time: 381.83 ns
Average time: 309.667 ns
Average time: 372.09 ns
Average time: 312.254 ns
Average time: 323.925 ns
```

```
Average time: 310.261 ns  
res88: Double = 309.667
```

b) Varför går det olika snabbt olika körningar?

Uppgift 29. Studera skillnader i prestanda mellan olika samlingar här:
docs.scala-lang.org/overviews/collections/performance-characteristics.html
(Mer om detta i kommande kurser.)

Uppgift 30. Gör något rekursivt med en lista för att visa hur syntaxen kan se ut med cons.

4.5 Laboration: pirates

Mål

- ☐ Öva på att använda datastrukturerna tuple, lista, map.
- ☐ Att spara data till fil.
- ☐ Att läsa in data från fil.
- ☐ Att spara objekt till fil.
- ☐ Att läsa in objekt från en fil.

Förberedelser

- ☐ Gör övning 4.
- ☐ Öppna Scala IDE i Eclipse enligt instruktionerna XX.
- ☐ Skapa ett projekt och skapa ett **object** Hello med en main-metod enligt XY.
- ☐ Skriv ut en hälsning till terminalen med `println("...")` och testkör programmet genom att markera filnamnet i projektmenyn och trycka på den gröna pilen. Kontrollera att hälsningen skrivs ut!

4.5.1 Obligatoriska uppgifter

Efter en rad olyckliga omständigheter har du blivit pirat i 1700-talets Karibien. Nu behöver du undvika galgen, hitta en skatt och försöka förutse dina förädiska skeppskamraters nästa steg.

Uppgift 1. *Save your crew.*

a) Kung George är villig att benåda fem personer ur din besättning! Skapa en lista där personerna sparas med förnamn, efternamn och befattning genom att läsa in dem från terminalen (kallad Console i Eclipse). Inläsning kan göras med kodraden

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

Namnen och befattningen kan sparas som en tuppel som skapas med koden (förnamn, efternamn, befattning). En tom föränderlig lista med sådana tuppler kan skapas genom att ange typerna till listan:

```
var enlist = List[Tuple3[String, String, String]]()
```

Exempel

b) En underuppgift.

Uppgift 2. *Read the map.*

a) En underuppgift.

4.5.2 Frivilliga extrauppgifter

Uppgift 3. En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.

Kapitel 5

Sekvensalgoritmer

Koncept du ska lära dig denna vecka:

- | | |
|--|---|
| <input type="checkbox"/> sekvensalgoritm | <input type="checkbox"/> for-sats i Java |
| <input type="checkbox"/> algoritm: SEQ-COPY | <input type="checkbox"/> java.util.Scanner |
| <input type="checkbox"/> in-place vs copy | <input type="checkbox"/> scala.collection.mutable.ArrayBuffer |
| <input type="checkbox"/> algoritm: SEQ-REVERSE | <input type="checkbox"/> StringBuilder |
| <input type="checkbox"/> algoritm: SEQ-REGISTER | <input type="checkbox"/> java.util.Random |
| <input type="checkbox"/> sekvenser i Java vs Scala | <input type="checkbox"/> slumptalsfrö |

5.1 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 **föränderliga** indatasekvensen

5.2 Några indexerbara samlingar

- Oföränderliga:
 - Kan **ej** ändra elementreferenserna:
Scala: **Vector**, **List**
- Föränderliga: kan **ändra** elementreferenserna
 - Kan **ej ändra storlek** efter allokering:
Scala+Java: **Array**
 - Kan ändra storlek efter allokering:
Scala: **ArrayBuffer**
Java: **ArrayList**

5.3 Algoritm: SEQ-COPY

Indata : Heltalsarray xs

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

```

1  $n \leftarrow$  antalet element i  $xs$ 
2  $ys \leftarrow$  en ny array med plats för  $n$  element
3  $i \leftarrow 0$ 
4 while  $i < n$  do
5   |  $ys(i) \leftarrow xs(i)$ 
6   |  $i \leftarrow i + 1$ 
7 end
8 return  $ys$ 
```

5.4 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, omvändning, 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.
- ☐ Känna till på vilket sätt Array har en särställning i JVM.
- ☐ Kunna välja när det är lämpligt att använda Vector, Array och ArrayBuffer.
- ☐ Känna till att klassen Arrays har färdiga metoder för kopiering.
- ☐ Kunna implementera algoritmer som registrerar antalet förekomster av objekt i en sekvens som indexeras med antalet förekomster.
- ☐ Kunna generera sekvenser av pseudoslumptal med specificerat slump-talsfrö.
- ☐ 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 teorin i kapitel 5.

5.4.1 Grunduppgifter

Uppgift 1. *Variabelt antal argument.* Det går fint att deklarera en funktion som tar en argumentsekvens av godtycklig längd. Syntaxen består ev 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` skrivs 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? Förklara 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 allokera 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): Array[Int]` som implementerar nedan algoritm, *efter* att du rättar de **två buggarna** i algoritmens while-loop:

<p>Indata : Heltalsarray <code>xs</code> och heltalet <code>x</code></p> <p>Resultat: En ny array som är en kopia av <code>xs</code> men med <code>x</code> tillagt på slutet som extra element.</p> <pre> 1 $n \leftarrow$ antalet element i <code>xs</code> 2 <code>ys</code> \leftarrow en ny array med plats för $n + 1$ element 3 $i \leftarrow 0$ 4 while $i \leq n$ do 5 $ys(i) \leftarrow xs(i)$ 6 end 7 $ys(n) \leftarrow x$</pre>

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, 2)
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 Fibonaccital¹. 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 koddelar 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
```

¹sv.wikipedia.org/wiki/Fibonaccital

```

6 scala> ys
7 scala> xs
8 scala> zs



```

Uppgift 6. *Algoritim: SEQ-REVERSE-COPY.* Implementera nedan algoritm:

```

Indata : Heltalsarray  $xs$  och heltalet  $x$ 
Resultat: En ny heltalsarray med elementen i  $xs$  i omvänd 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  $ys$ 

```

- 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. 
- 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. 
- Definiera en funktion i REPL med namnet reverseCopy med din implementation i uppgift b.

Uppgift 7. *Algoritim: 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 algoritm som vänder bak-och-fram på en sträng.

```

Indata : En sträng  $s$  av typen String
Resultat: En ny sträng av typen String
1  $sb \leftarrow$  en ny StringBuilder som innehåller  $s$ 
2  $n \leftarrow$  antalet tecken i  $s$ 
3  $i \leftarrow 0$ 
4 for  $i \leftarrow 0$  to  $\frac{n}{2} - 1$  do
5    $temp \leftarrow sb(i)$ 
6    $sb(i) \leftarrow sb(n - i - 1)$ 
7    $sb(n - i - 1) \leftarrow temp$ 
8 end
9 return  $sb$  omvandlad till en String

```

- Implementera algoritmen ovan i en funktion med signaturen:
def reverseString(s: String): String

```

// Kod till facit:
def reverseString(s: String): String = {

```

```

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

```

b) 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`. Implementera en ny variant av `isPalindrome` som använder denna metod. Skriv först algoritmen på papper i pseudo-kod.

```

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

```

Uppgift 8. 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 jämna 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:

²sv.wikipedia.org/wiki/Palindrom

```

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 heltal 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)

```

// kod till facit:


```

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

```

}


Uppgift 9. *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]
```

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


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
```

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


Uppgift 11. *Deterministiska pseudoslumpslängdssekvenser 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 pseudoslumpslängd 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) Förklara vad som händer nedan?

```

1 import java.util.Random
2 val frö = 42L
3 val rnd = new Random(frö)
4 rnd.nextInt(10)
5 (1 to 100).foreach(print(rnd.nextInt(10)))
6 val rnd1 = new Random(frö)
7 val rnd2 = new Random(frö)
8 val rnd3 = new Random(System.nanoTime)
9 val rnd4 = new Random((math.random * Long.MaxValue).toLong)
10 def flip(r: Random) = if (r.nextInt(2) > 0) "krona" else "klave"
11 val xs = (1 to 100).map{i => (flip(rnd1), flip(rnd2), flip(rnd3), flip(rnd4))}
12 xs foreach println
13 xs.exists(q => q._1 != q._2)
14 xs.exists(q => q._1 != q._3)

```

- c) Nämn några sammanhang då det är användbart att kunna bestämma fröet till en slumpalssekvens. 
- 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 slumpalssekvensen om man skapar ett `Random`-objekt med hjälp av konstruktorn utan parameter?

Uppgift 12. Undersök om tärningskast är rektangelfördelade.

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.

```

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

```

Uppgift 13. Array och **for**-sats i Java.

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` skrivs 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`.

```
// 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. Läs 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 programmet fungerar med olika indata.

5.4.2 Extrauppgifter: öva mer på grunderna

Uppgift 15. *Algorithm: SEQ-INSERT-COPY.*

Indata : En sekvens xs av typen `Array[Int]` och heltalen 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  $n \leftarrow$  antalet element  $xs$ 
2  $ys \leftarrow$  en ny Array[Int] med plats för  $n + 1$  element
3 for  $i \leftarrow 0$  to  $pos - 1$  do
4   |  $ys(i) \leftarrow xs(i)$ 
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]
```

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

- b) 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 16. *Algorithm: 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 "försvinner".

- 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 17. Implementera funktionen `tärningsRegistrering` från uppgift 8 på nytt, men nu med en **for**-sats istället.

5.4.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 18. 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 19. 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 20. Studera vad metoden `java.util.Arrays.deepEquals` gör här:

[Arrays.html#deepEquals-java.lang.Object:A-java.lang.Object:A-](#)

Vad skiljer ovan metod från metoden `java.util.Arrays.equals`?

Uppgift 21. Keno-dragningar under ett år -> Registrering...

Uppgift 22. 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`³ och klassen `jline.console.ConsoleReader`⁴. 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 val scan = new java.util.Scanner(System.in)  
2 scan.next  
3 scan.nextInt  
4 val cr = new jline.console.ConsoleReader  
5 cr.readLine  
6 cr.readLine("> ")  
7 cr.readLine("Ange tal: ").toInt  
8 scala.util.Try{cr.readLine("Ange tal: ").toInt}.toOption
```

³ github.com/jline/jline2

⁴ jline.github.io/jline2/apidocs/reference/jline/console/ConsoleReader.html

- 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 klistra 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 = ???

  /** 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 kompilerar ungefär så här:

```
scalac -cp "lib/jline-2.10.jar" termutil.scala
```

5.5 Laboration: cards

Mål

- ☐ Att öva på att använda vektorer
- ☐ Att använda SHUFFLE-algoritmen för kortblandning
- ☐ Att räkna frekvenser

Förberedelser

- ☐ Läs igenom så att du förstår SHUFFLE-algoritmen.

5.5.1 Bakgrund

Denna labb kommer att handla mycket om kortblandning. Att blanda kort så att varje möjlig permutation är lika sannolik är ganska knepigt; om man inte blandar systematiskt kommer det leda till en skev fördelning. Ett bra sätt att blanda en kortlek på, förutsatt att man har en bra slumpgenerator, är 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. Ungefär på detta sätt fungerar den s.k. Fisher-Yates-algoritmen, ibland även kallad en Knuth-shuffle, här endast kallad SHUFFLE.

Indata: Array xs som ska blandas

```
1  $len \leftarrow$  antalet element i  $xs$ 
2 for  $i \leftarrow (len - 1)$  to 0 do
3    $r \leftarrow$  slumpstal mellan 0 och  $i$ 
4    $temp \leftarrow xs(i)$ 
5    $xs(i) \leftarrow xs(r)$ 
6    $xs(r) \leftarrow temp$ 
7 end
```

5.5.2 Obligatoriska uppgifter

Uppgift 1. Den första uppgiften är att implementera metoden `shuffle` i klassen `CardDeck`. Följ algoritmen noga, och använd `cards.length` för att få fram längden på kortleken.

- a) Implementera metoden.
- b) Kör `TestDeck` för att testa att blandningen är jämnt fördelad. Du bör få sex ungefär lika långa staplar.

Uppgift 2. Fyll i resten av klassen `CardDeck`, skriv kod för att skapa en array innehållande en 52-korts standardlek. Använd case-klasserna i `Cards`. Tänk på att en `for/yield`-sats inte nödvändigtvis ger en Array, men att alla samlingar kan omvandlas till en sådan med `toArray`.

Uppgift 3. Använd den färdiga CardDeck-klassen för att ta fram sannolikheterna för att kortkombinationerna “royal flush”, “straight flush” “straight” eller “flush” dyker upp bland 5 kort dragna från en blandad kortlek. Simulera detta genom att upprepade gånger blanda kortleken, dra 5 kort och registrera vilka kombinationer som uppstår.

5.5.3 Frivilliga extrauppgifter

Uppgift 4. Utöka simuleringen till att även ta reda på antalet fyrtal, triss, kåk, dubbelpar och enkelpar.

Kapitel 6

Klasser

Koncept du ska lära dig denna vecka:

- | | |
|--|--|
| <input type="checkbox"/> objektorientering | <input type="checkbox"/> getters och setters |
| <input type="checkbox"/> klass | <input type="checkbox"/> new |
| <input type="checkbox"/> Point | <input type="checkbox"/> null |
| <input type="checkbox"/> Square | <input type="checkbox"/> klassparameter |
| <input type="checkbox"/> inkapsling | <input type="checkbox"/> primär konstruktor |
| <input type="checkbox"/> accessregler | <input type="checkbox"/> objektfabriksmetod |
| <input type="checkbox"/> private | <input type="checkbox"/> referenslikhet vs strukturelikhet |
| <input type="checkbox"/> private[this] | <input type="checkbox"/> eq vs == |
| <input type="checkbox"/> kompanjonsobjekt | |

6.1 Vad är en klass?

- En mall för att skapa objekt.
- Objekt skapade med **new** Klassnamn 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).

6.2 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 ???
- Kommentarer utgör krav på implementationen.

6.3 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.

6.4 Specifikationer av Java-klasser

- Specificerar signaturer för konstruktörer och metoder.
- Kommentarer utgör krav på implementationen.
- Används flitigt på extensor i EDA016, EDA011, EDA017...
- Javaklass-specifikationerna 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;
```

6.5 Övning: classes

Mål

- ☐ Kunna deklarerera klasser med klassparametrar.
- ☐ Kunna skapa objekt med **new** och konstruktorargument.
- ☐ Förstå innebörden av referensvariabler och värdet **null**.
- ☐ Förstå innebörden av begreppen instans och referenslikhet.
- ☐ Kunna använda nyckelordet **private** för att styra synlighet i primärkonstruktor.
- ☐ Förstå i vilka sammanhang man kan ha nytta av en privat konstruktor.
- ☐ Kunna implementera en klass utifrån en specikation.
- ☐ 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 teorin i kapitel 6.

6.5.1 Grunduppgifter

Uppgift 1. Referensvariabler, **null** och **new**.

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 konstruktor.*

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 = new Gurka8(-42)
17 scala> val g = new 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 egendefinierade setters?

Uppgift 5. *Klassen Square.*

a) Implementera 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 objects with position and side. */
class Square(val x: Int, val y: Int, val side: Int) {
  /** The area of this Square */
  val area: Int = ???

  /** Moves this square to position (x + xd, 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. Förklara vad som händer.


```
1 scala> val (s1, s2) = (Square(0,0,2), Square(1, 10, 2))
2 scala> val s3 = s1.move(1,-5)
3 scala> s1 isEqualSizeAs s3
4 scala> s2 isEqualSizeAs s1
5 scala> s2.scale(math.Pi) isEqualSizeAs s2
6 scala> s2.scale(math.Pi) isEqualSizeAs s2.scale(math.Pi)
```

c) Gör primärkonstruktorn i klassen Square privat och säkerställ i fabriksobjektet att det inte går att skapa kvadrater med negativ sida. Använd require.

d) Lägg till ett privat attribut **var** numberOfMoves: Int i klassen Square som räknar hur många gånger en instans har flyttats.

e) Lägg till en privat variabel **var** created: Vector[Square] i kompanjonsobjektet som sparar alla kvadratobjekt som skapats.

f) Lägg till en metod **def** totalNumberOfMoves: Int i kompanjonsobjektet som räknar ut det totala antalet förflyttningar som skett.

g) Varför är det avgörande för korrektheten av beräkningen av totala antalet förflyttningar i föregående uppgift att konstruktorn i klassen Square är privat? 

h) Vad kallas en metod som gör **new** och returnerar referensen till det nyskapade objektet? 

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 byter ut metoden equals som anropas av == – detta ska vi 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) Förklara 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.

- Implementera klassen *Point* som en oföränderlig case-klass med heltalssattributen `x` och `y`.
- Lägg till metoden `distanceTo(that: Point): Double` som räknar ut avståndet till en annan punkt med hjälp av `math.hypot`.
- Lägg till metoden `distanceTo(x: Int, y: Int): Double` som räknar ut avståndet till koordinaterna `x` och `y` med hjälp av metoden i föregående deluppgift.
- Lägg till metoden `move(dx: Int, dy: Int): Point` som skapar en ny punkt på translaterad position enligt delta-koordinaterna `dx` och `dy`.
- Lägg till ett kompanjonsobjekt med medlemmen **val** `origin` som ger en punkt i origo.
- 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
```

1g) 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?

```
// kod till facit
case class Point(x: Int, y: Int) {
  def distanceTo(that: Point): Double = math.hypot(that.x - x, that.y - y)

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

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

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

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(dx:dy: (Int, Int)): Point = ???
}

object Point {
  /** A Point object at position (0, 0) */
  val origin: Point = ???
}
```

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

  val y: Int = p._2

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

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

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

object Point {
```

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



Uppgift 10. Vad behöver du ändra i klassen Square från deluppgift 5 för att den ska fungera med en Point med 2-tuple-position från deluppgift 9?

6.5.2 Extrauppgifter: öva mer på grunderna

Uppgift 11. *Klassen Frog*

a)

Specification Frog

```
class Frog(startX: Int, startY: Int) {

  def jump(dx: Int, dy: Int): Unit = ???

  def x: Int = ???

  def y: Int = ???

  def randomJump: Unit = ???

  def distanceToStart: Double = ???

  def distanceJumped: Double = ???

}
```

6.5.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 12.

6.6 Laboration: turtlegraphics

Mål

- ☐ Kunna skapa egna klasser.
- ☐ Förstå skillnaden mellan klasser och objekt.
- ☐ Förstå skillnaden mellan muterbara och omuterbara objekt.
- ☐ Känna till och kunna skapa egna equals-metoder.
- ☐ 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

- ☐ Att göra.

6.6.1 Bakgrund

Under den här laborationen ska du skriva en samling av klasser som tillsammans kan användas för att rita i en fönsterruta. För att ge dig en grund att stå på får du tillgång till den färdigskrivna klassen `SimpleWindow`. `SimpleWindow` är en fönsterruta som tillåter programmeraren att rita linjer med hjälp utav en penna". `SimpleWindow` håller koll på pennans nuvarande position och kan ombes att flytta pennan genom att antingen lyfta den eller genom att rita en rak linje till en ny position.

```
class SimpleWindow {  
  /** Constructs a new SimpleWindow */  
  def this(xSize : Int, ySize : Int, windowTitle : String) : Unit  
  
  /** Moves the pen to the given coordinates without  
    drawing a line. */  
  def moveTo(x : Int, y : Int) : Unit  
  
  /** Draws a line from the pen's current position  
    to the given coordinates. */  
  def lineTo(x : Int, y : Int) : Unit  
}
```

Med hjälp utav `SimpleWindow` kan vi nu skapa en `Turtle`-klass som fungerar likt den i `Kojo`.

6.6.2 Obligatoriska uppgifter

Uppgift 1. Skapa en klass `Point` för att beskriva en viss koordinat (x,y) i ett fönster. Klassen ska vara omuterbar - man ska alltså inte kunna ändra på en koordinat efter att den har skapats.

- Hur borde specifikationen för `Point` se ut? Vilka attribut bör den innehålla?
- Tänk efter när du väljer synlighetsnivå för klassens attribut. Bör klassen ha några metoder?
- I klassen `Point`, lägg till en `equals`-metod som kan avgöra om en viss `Point` är samma punkt som en annan.

```
// ...

/** Returns true if the other Point represents
    the same coordinates as this Point */
def equals(other : Point) : Boolean

// ...
```

- När `equals`-metoden körs i kodsnutten nedan, vilken `Point` kommer parametern `other` referera till?

```
1 scala> val a = new Point(1,1)
2 scala> val b = new Point(2,2)
3 scala> a.equals(b)
```

Uppgift 2. Skapa en klass `Turtle` enligt följande specifikation:

```
/**
 * 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.
 * @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) {
  /**
   * Moves the turtle to a new position without drawing a line.
   */
  def jumpTo(newPosition: Point) : Unit

  /**
   * 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 right.
 *
 * @param turnAngle The number of degrees to turn.
 */
def turnLeft(turnAngle: Double): Unit

/**
 * Turns the turtle to the left.
 *
 * @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
 * the turtle moves.
 */
def penDown(): Unit

/**
 * Lifts the turtle's pen, preventing it from drawing lines,
 * even when it moves.
 */
def penUp(): Unit
}
```

- Implementera klassen Turtle.
- Svara på följande frågor om Turtle: Vilka attribut finns i klassen, och vilken synlighetsnivå har de? Vilka konstruktorer finns?
- Är klassen muterbar eller omuterbar? Motivera! Hade man kunnat göra tvärtom?
- 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?

- e) Använd din Turtle för att rita en cirkel.
- f) Skapa ett par av Turtles i samma fönster som rör sig sammanvävt. Fungerar allt som tänkt?

6.6.3 Frivilliga extrauppgifter

Uppgift 3. Skapa en klass Rectangle

```
class Rectangle(position: Point, side: Int) {  
  def draw(turtle: Turtle): Unit  
  def scale(factor : Double): Rectangle  
}
```

Uppgift 4. TODO: Skapa en klass som tar en Rectangle som parameter och ritar ut en sammansatt figur av flera omskalade rektanglar.

Kapitel 7

Arv

Koncept du ska lära dig denna vecka:

- ☐ arv
- ☐ polymorfism
- ☐ asInstanceOf
- ☐ klasshierarkin i Scala: Any Any-Ref Object AnyVal Nothing Null
- ☐ referensklasser vs värdeklasser
- ☐ klasshierarkin i Scalas samlingar
- ☐ Shape som basklass till Point och Rectangle
- ☐ accessregler vid arv
- ☐ protected
- ☐ final
- ☐ abstrakt klass
- ☐ trait
- ☐ inmixning
- ☐ klass vs trait
- ☐ case-object
- ☐ typer med uppräknade värden

7.1 Övning: traits

Mål



Förberedelser

- ☐ Studera teorin 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 flera olika typer försöker kompilatorn härleda den mest specifika gemensamma typen. 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**. Attributet vikt i traiten Grönsak nedan initialiseras inte förrän konstruktörerna anropas när vi gör **new** på någon av de konkreta 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 värdet av 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. Värdet vikt sägs vara abstrakt eftersom det saknar utan definition – den bara har 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 ett block efter typens namn vid instansiering. Vad får anonymGrönsak nedan för typ och strängrepresentation?

```
1 val anonymGrönsak = new Grönsak { val vikt = 42 }
```

Uppgift 2. 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 Ånkor kan simma och oftast även flyga. Fågel i generell, kollektiv bemärkelse kallas på gammal skånska för Fyle.

```
trait Fyle {
  val läte: String
  def väsnas: Unit = print(läte)
  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"
  override def väsnas = print(läte * 2)
}

class Ånka extends Fyle with KanSimma with KanKanskeFlyga {
  val läte = "kvack"
  override def väsnas = print(läte * 4)
}
```

a) En flitig fågelskådare hittar 42 fåglar i en skog där fågelsorterna är lika sannolika, representerat av vektorn fyle nedan. Skriv ett uttryck som undersöker hur många av dessa som är flygkunniga Ånkor, 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(_._väsna)
4 scala> val antalFlygånkor: Int = ???

```

```

// kod till facit
fyle.filter(f => f.isInstanceOf[Ånka] && f.ärFlygkunnig).size

```

b) Om alla de fåglar som fågelskådaren 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 = ???

```

```

// kod till facit
val antalKrax: Int = fyle.filter(f => !f.ärSimkunnig).size * 2
val antalKvack: Int = fyle.filter(f => f.ärSimkunnig).size * 4

```

Uppgift 3. *Typtester med `isInstanceOf` och typkonvertering med `asInstanceOf`.* Gör nedan deklARATIONER och instansieringar.

```

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

```

- Rita en bild över vilka typer som ärver vilka.
- 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.isInstanceOf[Object]
13 scala> 42.isInstanceOf[Any]

```

- 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 4. Uppräknade värden.

a)

```
1 trait Färg
2 case object Spader extends Färg
3 case object Hjärter extends Färg
4 case object Ruter extends Färg
5 case object Klöver extends Färg
```

b)

```
1 trait Färg
2 object Färg {
3   val values: Vector[Färg] = Vector(Spader, Hjärter, Ruter, Klöver)
4 }
5 case object Spader extends Färg
6 case object Hjärter extends Färg
7 case object Ruter extends Färg
8 case object Klöver extends Färg
```

c)

```
1 abstract class Färg(val toInt: Int)
2 object Färg {
3   val values: Vector[Färg] = Vector(Spader, Hjärter, Ruter, Klöver)
4 }
5 case object Spader extends Färg(0)
6 case object Hjärter extends Färg(1)
7 case object Ruter extends Färg(2)
8 case object Klöver extends Färg(3)
```

7.1.2 Extrauppgifter: öva mer på grunderna

Uppgift 5.

7.1.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 6.

7.2 Laboration: turtlerace-team

Mål

- ☐ Kunna arv
- ☐ Kunna traits

Förberedelser

- ☐ Gör övning traits i kapitel ??.
- ☐ Läs på om och förstå arv

7.2.1 Obligatoriska uppgifter

Uppgift 1. Grov plan.

- a) Skapa RaceTurtle ärver från Turtle.
- b) Skapa RaceWindow ärver från SimpleWindow. RaceWindow ska vara snyggt! inte bara två sträck, utan kanske med någon form av startfält
- c) Skapa TurtleRace som kan köra ett turtle race.
- d) Skaka TurtleRaceTournament som kan köra en turneringen med x antal Turtles i varje race och vinnarna går vidare och kör flera race och tillsist har man en prispall med alla turtles

7.2.2 Frivilliga extrauppgifter

Uppgift 2. En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.

Kapitel 8

Mönster, Undantag

Koncept du ska lära dig denna vecka:

- | | |
|---|---|
| <input type="checkbox"/> mönstermatchning | <input type="checkbox"/> flatten |
| <input type="checkbox"/> match | <input type="checkbox"/> flatMap |
| <input type="checkbox"/> Option | <input type="checkbox"/> partiella funktioner |
| <input type="checkbox"/> try | <input type="checkbox"/> collect |
| <input type="checkbox"/> catch | <input type="checkbox"/> implementera equals utan arv |
| <input type="checkbox"/> Try | <input type="checkbox"/> Complex |
| <input type="checkbox"/> unapply | <input type="checkbox"/> implementera equals med arv |
| <input type="checkbox"/> sealed | <input type="checkbox"/> Shape |

8.1 Övning: matching

Mål



Förberedelser



8.1.1 Grunduppgifter

Uppgift 1.

a)

Uppgift 2. *Byta ut metoden equals* Om man byter ut metoden den befintliga equals kommer metode == att fungera annorlunda. Vi börjar studera den befintliga equals med referenslikhet.

```
1 scala> class Gurka(val vikt: Int)
2 scala> val g1, g2 = new Gurka(42)
3 scala> val g3 = new Gurka(42)
4 scala> g1 == g2
5 scala> g1 == g3
6 scala> g1.equals // tryck TAB två gånger
```

a) Om du trycker TAB två gånger efter ett metodnamn får du se metodens signatur. Vilken signatur har metoden equals?

b) Byt ut equals enligt nedan och förklara vad som händer.

```
1 scala> class Gurka(val vikt: Int) {
2     override def equals(other: Any): Boolean = other match {
3
4     }
5 scala> val g = new Gurka(42)
6 scala> g.equals // tryck TAB två gånger
```

Uppgift 3. *Klassen Complex och metoden equals.* Implementera klassen Complex som representerar ett komplext tal¹ med realdel och imaginärdel.

```
1 scala> class Complex(val re: Double, im: Double)
```

8.1.2 Extrauppgifter: öva mer på grunderna

Uppgift 4.

¹sv.wikipedia.org/wiki/Komplexa_tal

8.1.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 5.

8.2 Laboration: chords - team

Mål

- ☐ Kunna använda mönstermatchning
- ☐ Känna till exceptions
- ☐ Förstå try-catch-satser

Förberedelser

- ☐ Gör övning matching i kapitel ??.
- ☐ Läs om exceptions och felhantering.
- ☐ Bonus: ha tillgång till en dator där ni kan spela upp ljud.

8.2.1 Obligatoriska uppgifter

Uppgift 1. Grov plan

- a) Skapa Notes-klass
- b) Skapa Chord-klass
- c) Skapa GuitarChord-/UkuleleChord-klass som ärver från Chord och har olika tuning och olika antal strängar
- d) använd SimpleNotePlayer för att skapa ChordPlayer

8.2.2 Frivilliga extrauppgifter

Uppgift 2. Grov plan

- a) Använd ChordPlayer för att få datorn att spela hela låtar
- b) Använd NotePlayer för att få datorn att kunna plocka ackord (alltså spela en sträng i taget)

Kapitel 9

Matriser, Typparametrar

Koncept du ska lära dig denna vecka:

- | | |
|---|---|
| <input type="checkbox"/> matris | <input type="checkbox"/> generisk funktion |
| <input type="checkbox"/> nästlade for-satser | <input type="checkbox"/> generisk klass |
| <input type="checkbox"/> designexempel: Tre-i-rad | <input type="checkbox"/> matriser i Java vs Scala |

9.1 Övning: matrices

Mål



Förberedelser



9.1.1 Grunduppgifter

Uppgift 1. *case class StringMatix with heading using Vector[Vector[String]]*

Uppgift 2. *Dense Matrix of Double using private Array[Double]*

Uppgift 3. *Sparse Matrix of Double using private mutable.Map[(Int, Int), Double]*


Uppgift 4. *Generiska klasser.*

```
1 scala> class Cell[T](var value: T){
2     override def toString = "Cell(" + value + ")"
3 }
4 scala> val c = new Cell(42)
```

a) Lägg till en metod **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
```

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

b) Vad händer om du i stället för typparameternamnet U i concat använder namnet T? 

Uppgift 5. *Skapa en generisk, oföränderlig matrisklass.*

```
case class Matrix[T](data: Vector[Vector[T]]) {
    def apply(x: Int, y: Int): T = ???
    def get(x: Int, y: Int): Option[T] = ???
    def row(r: Int): Vector[T] = ???
    def col(c: Int): Vector[T] = ???
}
```

```
def updated(x: Int, y: Int, value: T): Matrix[T] = ???  
}  
object Matrix {  
  def empty[T]: Matrix[T] = new Matrix[T](Vector())  
  def ofRowSize[T](rowSize: Int)(elements: T*): Matrix[T] =  
    new Matrix(elements.toVector.grouped(rowSize).toVector)  
}
```

a)

9.1.2 Extrauppgifter: öva mer på grunderna

Uppgift 6.

9.1.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 7.

9.2 Laboration: maze

Mål

- ☐ Skapa och iterera över matriser med nästlade for-loopar.
- ☐ Använda sig av och förstå arv.
- ☐ Träna på villkor med if-satser.
- ☐ Få en inblick i algoritmer för att lösa problem så som att ta sig igenom en labyrinth (?)

Förberedelser

- ☐ Läs om matriser.
- ☐ Läs om arv.
- ☐ Läs om olika algoritmer för att ta sig igenom en labyrinth (https://en.wikipedia.org/wiki/Maze_solving)

9.2.1 Bakgrund

I denna laboration kommer du att få skapa labyrinth och sedan implementera algoritmer för att ta dig igenom dessa. En labyrinth är ett rum som har en ingång och en utgång. Ingången är i de här fallen alltid längst ner i labyrinthen, och utgången högst upp. Alla väggar är också parallella med antingen x-axeln eller y-axeln. Ett sätt att beskriva en sådan labyrinth i kod är med hjälp av en matris. Varje element i matrisen motsvarar då en ruta"(ordval? steg??) i labyrinthen. Om ett element på en viss plats i matrisen är TRUE betyder det att på motsvarande plats i labyrinthen finns en vägg, och om ett element är FALSE att det här i labyrinthen finns en gång.

Det finns många olika sätt att ta sig igenom en labyrinth men ett av de vanligaste och enklaste sätten att konstruera en algoritm är att hålla sin vänstra eller högra hand mot motsvarande vägg och följa väggen utan att släppa den med handen tills man når slutet av labyrinthen. Detta funkar för alla labyrinth där väggarna från ingången till utgången är sammankopplade.

Det finns även flertalet algoritmer för att hitta den kortaste vägen igenom en labyrinth. En av dessa är med hjälp av så kallad Breadth First Search. (Lägg till kort beskrivning av BFS!)

9.2.2 Obligatoriska uppgifter

Uppgift 1. I denna uppgift ska du implementera en metod för att rita upp en labyrinth i SimpleWindow.

a) Läs igenom klassen Maze och se till att du förstår det mesta. Vad Maze gör är att den läser in rader med strängar, antingen från en fil eller direkt som argument, där tecknet '#' representerar en vägg och '.' (eller vad för tecken ska väljas??) representerar en gång. Utifrån detta skapar Maze en Boolean-matris som representerar labyrinthen. Fråga om något är oklart!

b) Implementera metoden `DRAW` i `Maze` som ritar upp labyrinten i `SimpleWindow`. För att göra detta behöver du gå igenom matrisen i `Maze` och undersöka elementen på varje plats. Om ett element är `TRUE` så betyder det att det här ska finnas en vägg i labyrinten, om `FALSE` att det här ska finnas en gång. Ta hjälp av metoderna `buildWall` och `moveRight/moveUp` som finns implementerade i `Maze`.

Uppgift 2. En labbuppgiftsbeskrivning.

a) Skapa en ny klass `MazeMain` (namn?) med en `main`-metod där du skapar ett objekt av `Maze` och sedan anropar metoden `draw` på detta. Läs in labyrinten från någon av filerna `maze1.txt` - `maze5.txt` eller skapa en egen labyrint. Båda alternativen ska fungera. Kolla att din labyrint ser ut som du tänkt dig, och fixa annars till den.

Uppgift 3. I den här uppgiften ska du implementera en algoritm för att få en sköldpadda att ta sig genom en labyrint med hjälp av att alltid hålla i väggen med vänster hand (eller fot i det här fallet!).

- a) Skapa en ny klass `MazeTurtle` som ärver från klassen `Turtle`.
- b) Definiera i `MazeTurtle` en ny metod `walk`. Implementera sedan denna metod. I metoden ska sköldpaddan med hjälp av tekniken "hålla vänster hand i väggen" ta sig genom labyrinten, från början till slut. Varje steg motsvarar att flytta sig från en ruta till en annan i Boolean-matrisen i `Maze`.
- c) Lägg till kod i `MazeMain` som skapar en sköldpadda och sedan låter denna gå igenom labyrinten med metoden `walk`. Testa att din `MazeTurtle` fungerar som den ska! Sköldpaddan ska klara att ta sig genom alla labyrinter i filerna `maze1.txt` - `maze5.txt`.

9.2.3 Frivilliga extrauppgifter

(Lägg till pseudokod för BFS-algoritm för att hitta `shortest path`!)

Uppgift 4. I den här uppgiften ska du implementera en algoritm för att hitta den kortaste vägen genom en labyrint.

- a) Inspektera nedanstående pseudokod och försök förstå den. Fråga gärna om något är oklart!
- b) Lägg till en ny metod `walkShortestPath` i klassen `MazeTurtle` där sköldpaddan ska hitta och sedan gå den kortaste vägen genom en labyrint. Implementera denna metod med hjälp av den givna pseudokoden (eller på egen hand för den modiga/nyfikna!).
- c) Anropa metoden `walkShortestPath` i `MazeMain`! Jämför med algoritmen att hålla i väggen med vänster hand. Vilken sköldpadda är snabbast?

Kapitel 10

Sökning, Sortering

Koncept du ska lära dig denna vecka:

- | | |
|---|---|
| <input type="checkbox"/> compareTo på strängar | <input type="checkbox"/> sortering på plats |
| <input type="checkbox"/> trait Ordered[T] | <input type="checkbox"/> algoritm: INSERTION-SORT |
| <input type="checkbox"/> algoritm: LINEAR-SEARCH | <input type="checkbox"/> algoritm: SELECTION-SORT |
| <input type="checkbox"/> algoritm: BINARY-SEARCH | <input type="checkbox"/> mer om filer |
| <input type="checkbox"/> algoritmisk komplexitet | <input type="checkbox"/> serialisering |
| <input type="checkbox"/> sortering till ny vektor | |

10.1 Övning: sorting

Mål



Förberedelser



10.1.1 Grunduppgifter

Uppgift 1.

a)

10.1.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

10.1.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

10.2 Laboration: surveydata-team

Mål

- ☐ Att lära sig.

Förberedelser

- ☐ Att göra.

10.2.1 Obligatoriska uppgifter

Uppgift 1. En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.

10.2.2 Frivilliga extrauppgifter

Uppgift 2. En labbuppgiftsbeskrivning.

- a) En underuppgift.
- b) En underuppgift.

Kapitel 11

Scala och Java

Koncept du ska lära dig denna vecka:

- ☐ skillnader mellan Scala och Java
- ☐ klasser i Scala vs Java
- ☐ referensvariabler vs enkla värden i Java
- ☐ referenstilldelning 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
- ☐ enum i java ???

11.1 Övning: scalajava

Mål



Förberedelser



11.1.1 Grunduppgifter

Uppgift 1. *Autoboxing i JVM.* I JVM måste typparametern för generiska klasser vara av referenstyp. I Scala löser kompilatorn detta åt oss så att vi ändå kan ha t.ex. `Int` som argument till en typparameter i Scala. Men i Java och i den underliggande plattformen JVM, så måste s.k. wrapper-klasser användas, t.ex. `Integer` som boxar en **`int`**.

a) Studera hur Scala-kompilatorn låter oss arbeta med en `Cell[Int]` även om det inderliggande JVM:ens körtidstyp (eng. *runtime type*) är en wrapper-klass. 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(42)
6 scala> c.value.getClass.getSimpleName
```

b) Vad är JVM:ens körtidstyp för `c.value` ovan? Hur kan det komma sig?

c)

11.1.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

11.1.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

11.2 Laboration: lthopoly-team

Mål

- ☐ Förstå hur autoboxing fungerar i Java
- ☐ Förstå vad statiska metoder och attribut innebär
- ☐ Förstå skillnad mellan primitiva typer och objekt i listor
- ☐ Kunna byta mellan ArrayLists och Array
- ☐ Kunna hur man läser in från fil
- ☐ Kunna for-sats i Java

Förberedelser

- ☐ Scanner
- ☐ ArrayList
- ☐ Enum?
- ☐ Statiskt
- ☐ Autoboxing
- ☐ Arv

11.2.1 Bakgrund

I denna labb skall ni tillverka lthopoly, en variant av det välkända brädspellet Monopol med några simplifieringar. Spelet går ut på att spelarna i tur och ordning slår en tärning, varpå deras pjäs flyttas det antal steg som tärningen visar. Det finns tre typer av rutor som alla ärver BoardSpace, och har olika beteenden. Dessa tre typer kallas för MoveSpace, MoneySpace och HouseSpace.

Beroende på vilken av rutorna spelaren hamnar på sker olika saker. I fallet att man hamnar på MoveSpace får spelaren dra ett s.k. MoveCard, som antingen förflyttar spelaren framåt eller bakåt längs spelplanen. Skulle spelaren hamna på MoneySpace kan denne antingen vinna eller förlora pengar genom att dra ett MoneyCard. Skulle spelaren hamna på en HouseSpace får denne alternativet att köpa rutan, förutsatt att den inte köpts av någon annan spelare, om så är fallet måste spelaren som hamnade på rutan betala hyra till ägaren.

Den visuella representationen av spelet sker via konsolfönstret med hjälp av klassen TextUI. TextUI är en färdigskriven klass med statiska metoder som gör det möjligt att skriva ut två kolumner av text i konsolfönstret. Utskriften kan exempelvis se ut såhär:

```
1 Studiecentrum(20) (Valthor)
2 A-huset(25) (Jonas)
3 ChansRuta
4 ChansRuta
5 Moroten och piskan(40)
6 V-Huset(45)
7 RiskRuta (Oskar)
8 ChansRuta
9 LED-Cafe(70)           Oskar slog en 6:a!
```

```

10 F-Huset(75)           Flytta framåt 6 steg
11 ChansRuta             Oskar drog ett kort: Inkassera 50 SEK!
12 Ideet(80)             Oskar har avslutat sin runda.
13 ChansRuta             Nästa spelare: Jonas
14 E-huset(100)          Jonas slog en 1:a!
15 RiskRuta              Grattis, Jonas är nu den stolta ägaren av A-huset!

```

Notera att utskriftens två kolumner är oberoende av varandra och kan uppdateras separat. För att uppdatera UI:t och få input av användaren finns följande metoder:

Färdigimplementerat

```

class TextUI

    /**Clears the right column (without reprinting). */
    public static void emptyRightColumn();

    /** Clears the left column (without reprinting). */
    public static void emptyLeftColumn();

    /** Adds a new string to be printed in the right column
     * every time it updates. Does not reprint the UI. */
    public static void addToRightColumn(String event);

    /** Adds a new string to be printed in the left column
     * every time it updates. Does not reprint the UI. */
    public static void addToLeftColumn(String event);

    /** Prints a console plot of the given array. */
    public static void printStatistics(int[] moneyLog);

    /** Prints a large number of newlines, visually clearing the console. */
    public static void clearConsole();

    /** Resets the left column with the toString of the GameBoard
     * and reprints both columns.
     * @param board.
     */
    public static void updateConsole(GameBoard board);

    /**Prompts the user to select one of the given options.
     * @param options An ArrayList containing a list of
     * descriptions for the possible options.
     * @return the index of the option selected by the user.
     */
    public static int promptForInput(List<String> options);

```

Implementeras Själv

```

class Player

```



```
/**Beskriver en spelare med pengar money, namn name och position pos . */
public Player(int money, String name, int pos)

/**Ger en int med spelarens pengar. */
public int getMoney()

/** Ändrar spelarens pengar. */
public void setMoney(int money)

/** Ger en int med spelarens position. */
public int getPosition()

/** Ändrar spelarens position. */
public void setPosition(int pos)
```

class BoardSpace

```
/**Beskriver en generell ruta på spelplanen. */
public BoardSpace()

/* Ger en vektor av möjliga val för spelare som står på rutan. */
public abstract PlayerAction[] getPossibleActions(GameBoard board);

/* Genomför det val som skickas med. */
public abstract void action(GameBoard board, PlayerAction action);

/*Ger en strängrepresentation av rutan. */
public abstract String toString();
```

class GameBoard

```
/**Beskriver ett spelbräde med en vektor av spelarna players. */
public GameBoard(Player[] players)

/** Ger en vektor av möjliga val för spelaren vars tur det är. */
public PlayerAction[] getPossibleActions()

/** Ger sant eller falskt beroende på om spelet är slut eller ej.*/
public boolean isGameOver()

/** Hämtar den för närvarande rikaste spelaren. */
public Player getRichestPlayer()

/** Genomför valet action för den aktuella spelaren. */
public void doAction(PlayerAction action)

/** Ger spelaren vars tur det är. */
public Player getCurrentPlayer()

/** Ger rutan som den aktuella spelaren står på. */
public BoardSpace getCurrentBoardSpace()

/** Förflyttar nuvarande spelare. */
public void moveCurrentPlayer(int adjustment)
```

```
/** Ger en vektor med totala summan pengar för varje runda,  
 * där runda ett är index 0 och sista rundan är i sista elementet  
 * i vektorn. */  
public int[] getStatistics()  
  
/** Ger en strängrepresentation av spelbrädet. */  
public String toString()
```

class DocumentParser

```
/* Ger en representation av spelplanen i en lista. */  
public static ArrayList<BoardSpace> getBoard()
```

class MoneyCard

```
/* Beskriver ett pengakort. */  
public MoneyCard(String description, int money)
```

class MoveCard

```
/* Beskriver ett flyttkort. */  
public MoveCard(String description, int position)
```

Spelregler

- 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 en spelare väljer att ta ett kort skall det värde som påverkas av kortet justeras. För Riskkort innebär detta en förflyttning medan för Chanskort en ändring av pengar.

11.2.2 Obligatoriska uppgifter

Uppgift 1. Här skall ni skriva en statisk metod för inläsning av data från fil i klassen DocumentParser. Här får inläsningssättet väljas helt fritt så länge resultatet av inläsningen blir att man kan anropa getBoard() och få kort enligt den kortstruktur som implementeras. Även klasserna MoveCard och MoneyCard skall implementeras så att det går bra ihop med inläsningen.

Obs! Se textfilerna moneycards.txt, movecards.txt och board.txt för förståelse för hur inläsningen bör gå till för korten.

- a) Implementera klassen MoveCard.
- b) Implementera klassen MoneyCard.
- c) Implementera två statiska metoder för att hämta en vektor med MoveCards respektive MoneyCard från fil i DocumentParser.

d) Implementera klassen Player.

Uppgift 2. I denna uppgift skall tre olika typer av spelrutor implementeras, som alla ärver ifrån klassen BoardSpace. Ni kommer då behöva implementera de tre abstrakta metoderna i BoardSpace i respektive subklass. Här behöver subklasserna tillgång till spelbrädet då förändringen av speltillståndet sker till störren delen i metoden action för de olika rutorna. Rutan som representerar ett hus kommer dessutom behöva ett attribut som håller koll på vem som äger rutan, medan rutorna som representerar chans- och riskhändelserna kommer behöva tillgång till vektorer av de kort som läses in.

a) Implementera en klass för varje typ av spelruta.

Obs! Än så länge kommer logiken inte fungera då inga metoder är implementerade i BoardGame ännu, 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 läsa in MoveSpaces och MoneySpaces. Viktigt här är att ordningen i vilken de olika rutorna är representerade i textfilen spelar roll, då den utgör upplägget av spelplanen. Alltså, beroende på vilken ruta som läses in från textfilen skall motsvarande objekt som representerar denna ruta konstrueras och läggas till i en ArrayList<BoardSpace> som slutligen returneras.

a) Implementera getBoard().

Uppgift 4. För att kunna skriva ut och visa spelplanen använder sig klassen TextUI av metoden toString() i klassen GameBoard. Denna ger en textrepresentation av spelplanen. Övriga metoder i klassen GameBoard skall nu implementeras, nyttja att TextUI har metoder för att lägga till utskrifter i olika kolumner för utskrifter av händelser. Det skall inte förekomma någon direkt utskrift i GameBoard, alla utskrifter ska gå via TextUI. I slutet av varje spelares tur skall dessutom den totala summan av pengar hos alla spelare läggas till i en ArrayList, för att det efter spelets slut skall kunna visas statistik för den totala ekonomin.

a) Implementera GameBoard.

Tips!

- Privata hjälpmetoder kan hjälpa till att underlätta.
- 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 ints. Detta fungerar trots att de har olika typer p.g.a. autoboxing.

- Tänk på att spelarna skall kunna gå runt spelplanen obegränsat antal gånger.
- Tänk på att dela upp arbetet så att det sker en lämpligt arbetsfördelning inom gruppen, men samarbeta så att ni vet hur era olika implementationer interagerar.

11.2.3 Frivilliga extrauppgifter

Uppgift 5. Här är det dags att utöka spelet för att ge utökad funktionalitet som liknar det riktiga spelet.

- Implementera så att varje spelare får extra pengar då den passerar början av listan.
- Implementera så att om en spelare hamnar på en ruta de äger sedan tidigare så har de möjlighet att öka hyran för den ifall någon annan spelare skulle hamna på den.
- Implementera så att spelarna själva måste betala en femtedel värdet av sina hus varje runda (varje gång de passerar ett varv på brädet).

Kapitel 12

Trådar

Koncept du ska lära dig denna vecka:

- | | |
|-------------------------------------|---|
| <input type="checkbox"/> Thread | <input type="checkbox"/> (Javascript ???) |
| <input type="checkbox"/> Future | <input type="checkbox"/> (css ???) |
| <input type="checkbox"/> Duration | <input type="checkbox"/> Scala.js ??? |
| <input type="checkbox"/> Await | <input type="checkbox"/> Android ??? |
| <input type="checkbox"/> (HTML ???) | |

12.1 Övning: threads

Mål



Förberedelser



12.1.1 Grunduppgifter

Uppgift 1.

a)

12.1.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

12.1.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

12.2 Laboration: life

Mål

- ☐ Att lära sig om hur man kan använda matriser som en datastruktur.
- ☐ Att lära sig om hur man separerar beteende från vy med hjälp av Model-View uppdelningen.
- ☐ Att lära sig om grundläggande cellulära automata.
- ☐ Att lära sig om trådar, en grundläggande metod för att köra kod *samtidigt* som annan kod.

Förberedelser

- ☐ Läsa igenom laborationen.

12.2.1 Bakgrund

Life (även kallat *Conway's Game of Life* efter skaparen och matematikern John Horton Conway) är en cellulär automata som med enkla regler kan ge upphov till komplexa beteenden och har blivit känt som ett exempel på 'emergence' and 'self-organization'.

Spelet har noll medvetna spelare (ett så kallat 'zero-player game') där slutresultatet beror fullständigt på startkonfigurationen.

12.2.2 Reglerna

Reglerna i spelet är följande:

1. Spelbrädet består av en matris med n rader och m kolumner (n och m brukar ibland modelleras som ∞)
2. Varje cell i matrisen kan vid varje tidpunkt (varje generation) ha ett av två tillstånd: levande eller död
3. Varje cells tillstånd i nästa generation bestäms av följande regler:
 - (a) Om cellen är levande och har två eller tre grannar så lever den vidare, annars dör den.
 - (b) Om cellen är död och har exakt tre grannar så föds den och dess tillstånd ändras till levande, annars fortsätter den vara död.

För mer om Game of Life, se Wikipedia:

1. Engelska: https://en.wikipedia.org/wiki/Conway's_Game_of_Life
2. Svenska: https://sv.wikipedia.org/wiki/Game_of_Life

12.2.3 Obligatoriska uppgifter

Uppgift 1. Skapa en model som kan visas i vyn.

- a) En underuppgift.
- b) En underuppgift.

Uppgift 2. Implementera Life-regeln med hjälp av traitet Rule.

- a) En underuppgift.
- b) En underuppgift.

12.2.4 Frivilliga extrauppgifter

Uppgift 3. 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 kan finnas här: https://en.wikipedia.org/wiki/Category:Cellular_automaton_rules

Nedan följer några roliga exempel som valts ut och anses lämpliga.

- a) Implementera cyklisk cellulär automata.

Denna typ av automata kallar 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-operatoren: $T_{nästa} = T_{nuvarande} + 1 \% N$

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.

För att få intressant beteende brukar man initialisera hela brädet så att varje cell får ett slumpvalt tillstånd.

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

- b) Implementera Wireworld

Wireworld är ett lite annorlunda då man i Wireworld designar 'kretsar' inte helt olika de som finns i moderna datorer.

I Wireworld kan man skapa komponenter som fungerar som dioder samt transistorer, och med dessa bygga logiska grindar.

<https://en.wikipedia.org/wiki/Wireworld>

Uppgift 4. Implementera spara och ladda Svårighet: Medel

- a) Spara brädets tillstånd. Tillståndet ska sparas till ett format som både är lätt att spara/exportera och ladda/importera. Förslagsvis kan man använda formatet CSV (Comma Separated Values) eller helt enkelt bara skriva ut matrisen rad för rad där varje cell skrivs ut som en etta eller nolla.
- b) Ladda in det exporterade tillståndet. Implementera en metod för att läsa in det sparade tillståndet

Uppgift 5. Alternativ vy: Kör programmet i webbläsaren med Scala.js

Uppgift 6. Alternativ vy: Kör programmet på Android

Uppgift 7. Implementera brädet som en sparse-matris Svårighet: Medel

I den tidigare lösningen har vi allokerat en hel matris där bara en del av brädet vanligtvis är levande, en sådan matris kallas för en sparse matris (en matris där majoriteten av värdena är 0).

a) ???

Kapitel 13

Design

Koncept du ska lära dig denna vecka:



13.1 Projektuppgift: bank

Mål

- ☐ Kunna implementera ett helt program efter given specifikation
- ☐ 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

13.1.1 Bakgrund

I denna uppgiften 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 tillståndsförändringar skrivas till fil så att utifall banken skulle krascha finns alla transaktioner som genomförts sparade så att banken kan 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, men det ska dock följa de specifikationer som ges i uppgiften, såväl som de objektorienterade principer du lärt dig i kursen.

13.1.2 Krav

Kraven för bankapplikationen återfinns här nedan. För att bli godkänd på denna uppgift måste samtliga krav uppfyllas:

- Programmet ska ha följande funktioner tillgängliga via menyn:
 - Hitta konton för en viss kontoinnehavare.
 - Sök kontoinnehavare på (del av) namn.
 - Sätta in pengar på ett konto.
 - Ta ut pengar på ett konto.
 - Överföra pengar mellan två olika konton.
 - Skapa ett nytt konto.
 - Ta bort ett befintligt konto.
 - Skriv ut bankens alla konton.
 - Återställa banken till ett tidigare tillstånd vid ett givet datum.
 - Avsluta.
- Programmet ska skapa nytt tillstånd med tidsstämpel och spara gamla varje gång då:

- Pengar sätts in eller tas ut ifrån ett konto.
 - Pengar överförs mellan två konton.
 - Ett konto skapas.
 - Ett konto tas bort.
- Då ett tillstånd förändras ska detta skrivas till fil.
 - Programdesignen ska följa de specifikationer som är angivna nedan.
 - 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 antalet 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.

13.1.3 Design

Nedan följer specifikationerna för de olika klasserna bankapplikationen måste innehålla:

BankAccount

```
/**
 * Skaper ett nytt bankkonto åt innehavaren Customer.
 * Kontot tilldelas ett unikt kontonummer och innehåller
 * inledningsvis 0 kr.
 */
class BankAccount(val holder: Customer);

/**
 * Sätter in mängden amount pengar på detta konto.
 */
def deposit(amount: Int): Unit;

/**
 * Hämtar saldot på konton.
 */
def getBalance: Int;
```

```
/**
 * Tar ut mängden amount pengar från kontot sålänge
 * amount ej överstiger nuvarande saldo.
 * Returnerar true om transactionen lyckades, annars false.
 */
def withdraw(amount: Int): Boolean;
```

Bank

```
/**
 * Skapar en ny bank utan konton och tillstånd.
 */
class Bank();

/**
 * Skapar ett nytt konto på banken. Kontonumret som
 * genereras åt kontot returneras.
 */
def addAccount(name: String, id: Int): Int

/**
 * Returnerar kontoinnehavaren kopplat till det givna
 * id-numret, om ingen sådan kund finns returneras
 * null istället.
 */
def findHolder(id: Int): Customer;

/**
 * Ta bort kontot med kontonumret accountNbr och returnerar
 * true.
 * Om inget sådant konto existerar returneras false istället.
 */
def removeAccount(accountNbr: Int): Boolean;

/**
 * Returnerar en lista med samtliga bankkontot i banken.
 * Finns inga kontot returneras en tom lista.
 */
def getAllAccounts(): ArrayBuffer[BankAccount];

/**
 * Returnerar det bankkonto som har kontonummer accountNbr.
 * Finns inget sådant konto returneras null istället.
 */
def findByNumber(accountNbr: Int): BankAccount;

/**
```

```
    * Söker upp alla konton med en innehavaren vars namn
    * innehåller strängen namePattern och returnerar dessa
    * i en lista.
    */
    def findByName(namePattern: String): ArrayBuffer[Customer];

    /**
     * Genomför en transaktioner i banken och returnerar en sträng
     * med nödvändig information till konsolen.
     */
    def makeTransaction(transaction: Transaction): String;

    /**
     * Återställer banken till det tillstånd som gällde vid
     * tiden date.
     */
    def returnToState(returnDate: Date): String;
```

Transaction

```
    /**
     * Beskriver en transaktion som används av banken.
     */
    abstract class Transaction;
```

Customer

```
    /**
     * Beskriver en kund med namnet name och personnummer id.
     */
    class Customer(val name: String, val id: Int);
```

13.1.4 Obligatoriska uppgifter

Uppgift 1. Implementera klassen Customer

Uppgift 2. Implementera klassen BankAccount

Uppgift 3. Implementera de klasser som ska förlänga Transaction

Uppgift 4. Skapa klassen 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 menyval 6 och 8. Testa noga.

Uppgift 6. Implementera tillstånds funktionaliteten. Varje ny transaktion ska ge upphov till nytt tillstånd.

Uppgift 7. Implementera klassen Bank utefter de menyval du implementerar.

Uppgift 8. Implementera menyval 9. Testa noga.

```
/** A simple wrapper of Java.time */  
case class Date;
```

13.1.5 Frivilliga extrauppgifter

Uppgift 9. Ej bestämt ännu.

13.2 Projektuppgift: tictactoe

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!

13.2.1 Regler

Om du känner dig säker på hur reglerna i tic-tac-toe funkar kan du skippa detta.

- 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 spelpjäsa eller om spelplanen är fylld.

Notera att pjäserna INTE får flyttas när de väl ligger på spelplanen.

13.2.2 Teori

Representationen är vald till en endimensionell vektor av typen Int av storlek 9 där element [0,2](note: intervall mellan 0 och 2) 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 heltal. 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. Smidigast(note: detta kommer visa sig senare) är att välja representationen där 0 representerar tom plats, 1 representerar x och -1 representerar o.

13.2.3 Obligatoriska uppgifter

Uppgift 1. Implementera ett fungerande Spel.

- Implementera funktionen gameWon.
- Implementera en Human player.
- Implementera första version av Game.

Uppgift 2. Randomized player

- Implementera en spelare som väljer ett slumpmässigt giltigt drag.
- Ändra Game så att användaren tillåts stänga av ritfunktionen och tillåts spela många spel.

c) Vad är sannolikheterna för att x vinner, o vinner och att det blir oavgjort om två randomized players spelar mot varandra?

Hamnar man i närheten av dessa resultat tror vi på er randomized player.

- $P(x \text{ vinner}) = 0.586$
- $P(o \text{ vinner}) = 0.288$
- $P(\text{lika}) = 0.126$

d) Varför är det större sannolikhet för x att vinna än o?

Uppgift 3. Optimal Player

- Läs igenom eval-funktionen och Appendix om max-min-evaluering.
- Implementera Optimal Players move-funktion.
- testa att spela mot din Optimal player med en human player, kan du spela lika? Kan du vinna?
- Vad händer om du sätter en random player mot Optimal player? Blir det någonsin oavgjort, hur ofta?

13.2.4 Frivilliga extrauppgifter

Uppgift 4. Hashning.

- En underuppgift.
- En underuppgift.

13.3 Projektuppgift: imageprocessing

13.3.1 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-modellen (cyan, magenta, gul, svart) vid blandning av färg som ska tryckas på papper eller annat material. På en dator däremot används vanligtvis RGB-systemet. RGB-systemet har tre grundfärger: 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 gråskalor; det gör man med färger som har samma värde på alla tre grundfärgerna: (0, 0, 0) är helt svart, (255, 255, 255) är helt vitt.

13.3.2 Uppgiften

Du ska skriva ett program där du implementerar olika filter som ska manipulera en given bild på ett flertal olika sätt. Till ditt förfogande kommer du att få en abstrakt basklass `ImageFilter` samt en `ImageGUI`-klass. Båda dessa är skrivna i Java men all kod som redovisas ska vara skriven i Scala.

Följande beskriver `ImageFilter` klassen som ska ärvas från.

abstract class ImageFilter

```
/**
 * Skapar ett filterobjekt med ett givet namn.
 */
protected ImageFilter(String name);

/**
 * Tar reda på filtrets namn.
 */
public String getName();

/**
 * Filtrerar bilden i matrisen inPixels och returnerar
 * resultatet i en ny matris. Utnyttjar eventuellt
 * värdet av paramValue
 */
public abstract Color[][] apply(Color[][] inPixels,
                                double paramValue);

/**
 * 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 talvärden
 * @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);

```

Följande main metod ska användas och utökas där dina filter ska ges till ImageGUI.

```

import se.lth.cs.pt.images.{ImageFilter, ImageGUI}

object ImageProcessor {
  def main(args: Array[String]): Unit = {
    val filters = Array(new IdentityFilter("vanligt"))
    new ImageGUI(filters)
  }
}

```

Uppgift 1. Börja med att implementera klassen IdentityFilter som enbart skapar en kopia av bilden. Testa main metoden och få den att fungera.

Uppgift 2. Implementera ett blåfilter. Det vill säga skapa ett filter där varje pixelelement bara innehåller den blå komponenten.

Uppgift 3. Implementera ett inverteringsfilter som skapar en negativ kopia av bilden. Fundera över vad som kan menas med en inverterad eller negativ kopia: de nya RGBvärdena är inte ett dividerat med de gamla värdena (då skulle de nya värdena bli flyttal) och inte de gamla värdena med ombytt tecken (då skulle de nya värdena bli negativa).

Uppgift 4. Implementera ett filter som gör om bilden till en gråskalebild. Använd ImageFilters computeIntensity metod för att bestämma vilken intensitet varje bildelement ska ha. Om intensiteten i ett bildelement till exempel var 105 så ska ett nytt Color objekt med värdena (105, 105, 105) skapas.

Uppgift 5. Skapa ett filter som krypterar bilden med xor-operatorn. Varje bildelement krypteras genom att använda xor-operatorn med ursprungsvärdena för rött, grönt och blått tillsammans med ett slumpmässigt heltalsvärde som

genereras av Javas Random klass.

Uppgift 6. Förbättring av kontrasten i bild. Vi inskränker oss här till att förbättra kontrasten i gråskalebilder. 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):

$$newIntensity = 255 * (intensity - 45) / (225 - 45)$$

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. Detta är ett typiskt registreringsproblem som ska lösas enligt metoden i avsnitt 8.10 i läroboken.

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 vi matar in i Parameter-rutan i användargränssnittet) 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 `int[] histogram = new int[256]`. Beräkna lowCut så att `histogram[0] + histogram[1] + ... + histogram[lowCut] = 0.015 * 100000` (så nära det går att komma, det blir troligen inte exakt likhet). Beräkna highCut på liknande sätt. Sammanfattning av algoritmen:

1. Beräkna intensiteterna 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 nya intensiteter 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 7. Gaussfiltrering. Skriv en klass GaussFilter där ImageFilters convolve metod används. Varje färg ska behandlas separat. Gör på följande sätt:

1. Bilda tre short-matriser och lagra pixlarnas red-, green- och blue-komponenter i matriserna.
2. Utför faltningen av de tre komponenterna för varje element och lagra ett nytt Colorobjekt i outPixels för varje punkt.
3. Elementen i ramen behandlas inte, men i outPixels måste också dessa element få värden. Enklarest 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.

Metoden faltar punkten p[i][j] med faltningskärnan kernel och ska anropas med red-, green- och blue-matrisen. weight är summan av elementen i kernel. Faltningskärnan kan vara ett attribut i klassen och vara en matris med följande utseende:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Uppgift 8. Sobelfiltrering.

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 (här behöver man inte normera resultatet). Koefficienterna i matrisen X_SOBEL uttrycker derivering i x-led (ger vertikala konturer), i Y_SOBEL faltning i y-led (ger horisontella konturer). 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 s_x och s_y så får man en indikator på närvaron av en kontur med $\text{Math.abs}(s_x) + \text{Math.abs}(s_y)$. Absolutbelopp behöver man eftersom man har negativa koefficienter i faltningsmatriserna.
4. Sätt pixeln till svart om indikatorn är större än tröskelvärdet, till vit annars. Mata in tröskelvärdet i Parameter-rutan. Skriv en klass SobelFilter som implementerar denna algoritm.

13.3.3 Frivilliga extrauppgifter

Uppgift 9. Blurfilter?

- a) En underuppgift.

Kapitel 14

Tentaträning

Koncept du ska lära dig denna vecka:



Del III

Appendix

Appendix A

Virtuell maskin

A.1 Vad är en virtuell maskin?

Du kan köra alla kursens verktyg i en så kallad virtuell maskin (vm). Det är ett enkelt och säkert sätt att installera ett nytt operativsystem i en "sandlåda" som inte påverkar din dators ursprungliga operativsystem.

A.2 Installera kursens vm

Det finns en virtuell maskin förberedd med alla verktyg som du behöver förinstallerade. Gör så här:

1. Installera VirtualBox v5 här:
<https://www.virtualbox.org/wiki/Downloads>
2. Ladda ner filen vbox.zip här:
<http://fileadmin.cs.lth.se/pgk/vbox.zip>
OBS! Då filen är på nästan 4GB kan nedladdningen ta mycket lång tid.
3. Packa upp filen vbox.zip i biblioteket "VirtualBox VMs" som du fick i din hemkatalog när du installerade VirtualBox. Du får då 3 filer som heter något med "introprog-ubuntu-64bit".
4. Kolla med hjälp av denna sida:
<https://md5file.com/calculator>
så att filen "introprog-ubuntu-64bit.vdi" har denna sha256-checksumma:
— ska-stå-checksumma-här-sen —
5. Öppna VirtualBox och lägg till maskinen introprog-ubuntu-64bit genom menyn "add".
6. Starta maskinen.
7. Öppna ett terminalfönster och skriv `scala` och du är igång och kan göra första övningen!

A.3 Vad innehåller kursens vm?

Den virtuella maskinen kör Xubuntu 14.04 med fönstermiljön XFCE, vilket är samma miljö som E-husets linuxdatorer kör.

I den virtuella maskinen finns detta förinstallerat:

- Java JDK 8
- Scala 2.11.8
- Kojo 2.4.08
- Eclipse Mars.2 med ScalaIDE 4.3
- gedit med syntaxfärgning för Scala och Java
- git
- sbt
- Ammonite REPL

Appendix B

Terminalfönster och kommandoskal

B.1 Vad är ett terminalfönster?

I ett terminalfönster kan man skriva kommandon som kör program och hanterar filer på din dator. När man programmerar använder man ofta terminalkommando för att kompilera och exekvera sina program. Man kan använda terminalkommandon för att navigera och manipulera filerna på datorns disk.

Terminal i Linux

PowerShell i Microsoft Windows

Microsoft Windows är inte Unix-baserat, men i kommandotolken PowerShell finns alias definierat för en del vanliga unix-kommandon. Du startar Powershell t.ex. genom att trycka på Windows-knappen och skriva powershell.

Terminal i Apple OS X

Apple OS X är ett Unix-baserat operativsystem. Många kommandon som fungerar under Linux fungerar också under Apple OS X.

B.2 Några viktiga terminalkommando

Tipsa om ss64.com

Appendix C

Editera

C.1 Vad är en editor?

C.2 Välj editor

Appendix D

Kompilera och exekvera

D.1 Vad är en kompilator?

D.2 Java JDK

D.2.1 Installera Java JDK

D.3 Scala

D.3.1 Installera Scala-kompilatorn

D.4 Read-Evaluate-Print-Loop (REPL)

För många språk, t.ex. Scala och Python, finns det en interaktiv tolk som gör det möjligt att exekvera enstaka programrader och direkt se effekten. En sådan tolk kallas Read-Evaluate-Print-Loop eftersom den läser en rad i taget och översätter till maskinkod som körs direkt.

D.4.1 Scala REPL

Kommandon i REPL

`:paste`

Kortkommandon: Ctrl+K etc.

Appendix E

Dokumentation

E.1 Vad gör ett dokumentationsverktyg?

E.2 scaladoc

<http://docs.scala-lang.org/style/scaladoc.html>

E.3 javadoc

Appendix F

Integrerad utvecklingsmiljö

F.1 Vad är en IDE?

F.2 Kojo

F.2.1 Installera Kojo

www.kogics.net/kojo-download

F.2.2 Använda Kojo

Tabell F.1: Några av sköldpaddans funktioner. Se även lth.se/programmera

<i>Svenska</i>	<i>Engelska</i>	<i>Vad händer?</i>
fram	forward	Paddan går 25 steg frammåt.
fram(50)	forward(50)	Paddan går 50 steg frammåt.
höger	right	Paddan vrider sig 90 grader åt höger.
upprepa(10){???}	repeat(10){???}	Repetition av ??? 10 gånger.

Koden för den svenska paddans api finns här: bitbucket.org/lalit_pant/kojo/

F.3 Eclipse och ScalaIDE

F.3.1 Installera Eclipse och ScalaIDE

F.3.2 Använda Eclipse och ScalaIDE

Appendix G

Byggverktyg

G.1 Vad gör ett byggverktyg?

G.2 Byggverktyget sbt

G.2.1 Installera sbt

G.2.2 Använda sbt

Appendix H

Versionshantering och kodlagring

H.1 Vad är versionshantering?

H.2 Versionshanteringsverktyget git

H.2.1 Installera git

H.2.2 Använda git

H.3 Vad är nyttan med en kodlagringsplats?

H.4 Kodlagringsplatsen GitHub

H.4.1 Installera klienten för GitHub

H.4.2 Använda GitHub

H.5 Kodlagringsplatsen Atlassian BitBucket

H.5.1 Installera SourceTree

H.5.2 Använda SourceTree

Appendix I

Nyckelord

I.1 Vad är ett nyckelord ord?

Nyckelord är ord i ett programmeringsspråk som har speciell betydelse och reserverade för endast ett användningsområde. Nyckelord kallas även *reserverade ord*¹. Man kan till exempel inte använda nyckelordet **def** som namn på en variabel. Nyckelord ges ofta en speciell färg av de kodeditorer som erbjuder *syntaxstyrd färgning*.

I.2 Nyckelord i Scala

abstract	case	catch	class	def
do	else	extends	false	final
finally	for	forSome	if	implicit
import	lazy	macro	match	new
null	object	override	package	private
protected	return	sealed	super	this
throw	trait	try	true	type
val	var	while	with	yield
_	:	=	=>	<-
			<:	<%
			>:	#
				@

I.3 Nyckelord i Java

Here is a list of keywords in the Java programming language. You cannot use any of the following as identifiers in your programs. The keywords `const` and `goto` are reserved, even though they are not currently used. `true`, `false`, and `null` might seem like keywords, but they are actually literals; you cannot use them as identifiers in your programs.

abstract continue for new switch

¹Läs mer här: en.wikipedia.org/wiki/Reserved_word

```
assert *** default goto * package synchronized
boolean do if private this
break double implements protected throw
byte else import public throws
case enum **** instanceof return transient
catch extends int short try
char final interface static void
class finally long strictfp ** volatile
const * float native super while
*      not used
**      added in 1.2
***      added in 1.4
****      added in 5.0
```

Appendix J

Lösningsförslag till övningar

J.1 expressions

J.1.1 Grunduppgifter

Uppgift 1.

- a) 43
- b) Lösningstext.

J.1.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.1.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.2 programs

J.2.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

Uppgift 2.

J.2.2 Extrauppgifter: öva mer på grunderna

Uppgift 3.

- a) 42
- b) Lösningstext.

J.2.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 4.

- a) 42
- b) Lösningstext.

J.3 functions

J.3.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.3.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.3.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.4 data

J.4.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.4.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.4.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.5 sequences

J.5.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.5.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.5.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.6 classes

J.6.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.6.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.6.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.7 traits

J.7.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.7.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.7.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.8 matching

J.8.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.8.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.8.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.9 matrices

J.9.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.9.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.9.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.10 sorting

J.10.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.10.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.10.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.11 `scalajava`

J.11.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.11.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.11.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.12 threads

J.12.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.12.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.12.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

Appendix K

Ordlista