

Programmering, grundkurs

Kompendium

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

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

Editor: Björn Regnell

Contributors in alphabetical order: Anders Buhl, Anton Andersson, Björn Regnell, Casper Schreiter, Cecilia Lindskog, Erik Bjäreholt, Erik Grampp, Fredrik Danebjer, Gustav Cedersjö, Jonas Danebjer, Måns Magnusson, Maj Stenmark, Oskar Berg, Patrik Persson, Per Holm, Sandra Nilsson, Valthor Halldorsson,

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 and projects.

Copyright © 2015-2016.

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

Framstegsprotokoll

Genomförda övningar

Till varje laboration hör en övning med uppgifter som utgör förberedelse inför labben. Du behöver minst behärska 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 ingenjörstudenter 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
II Moduler	17
1 Introduktion	19
1.1 Vad är programmering?	20
1.2 Vad är en kompilator?	20
1.3 Vad består ett program av?	21
1.4 Exempel på programmeringsspråk	21
1.5 Varför Scala + Java som förstaspråk?	22
1.6 Hello world	22
1.7 Utvecklingscykeln	23
1.8 Utvecklingsverktyg	23
1.9 Literaler	23
1.10 Uttryck	23
1.11 Identifierare	24
1.12 Speciella identifierare	24
1.13 Övning: expressions	25
1.13.1 Grunduppgifter	25
1.13.2 Extrauppgifter: öva mer på grunderna	32
1.13.3 Fördjupningsuppgifter: avancerad nivå	33

1.14	Laboration: kojo	35
1.14.1	Obligatoriska uppgifter	35
1.14.2	Frivilliga extrauppgifter	41
2	Kodstrukturer	45
2.1	Övning: programs	46
2.1.1	Grunduppgifter	46
2.1.2	Extrauppgifter: öva mer på grunderna	55
2.1.3	Fördjupningsuppgifter: avancerad nivå	55
3	Funktioner, Objekt	57
3.1	Övning: functions	58
3.1.1	Grunduppgifter	58
3.1.2	Extrauppgifter: öva mer på grunderna	66
3.1.3	Fördjupningsuppgifter: avancerad nivå	67
3.2	Laboration: bugs	69
3.2.1	Obligatoriska uppgifter	69
3.2.2	Frivilliga extrauppgifter	69
4	Datastrukturer	71
4.0.3	Att göra denna vecka	72
4.1	Denna vecka: Fatta datastrukturer	72
4.1.1	Olika sorters datastrukturer	72
4.2	Olika sätt att skapa datastrukturer	72
4.2.1	Tupler	72
4.3	Vad är en tupel?	72
4.4	Övning: data	73
4.4.1	Grunduppgifter	73
4.4.2	Extrauppgifter: öva mer på grunderna	86
4.4.3	Fördjupningsuppgifter: avancerad nivå	87
4.5	Laboration: pirates	91
4.5.1	Förberedelseuppgifter	91
4.5.2	Obligatoriska uppgifter	94
4.5.3	Frivilliga extrauppgifter	97
5	Sekvensalgoritmer	99
5.1	Vad är en sekvensalgoritm?	100
5.2	Några indexerbara samlingar	100
5.3	Algoritm: SEQ-COPY	100
5.4	Övning: sequences	101
5.4.1	Grunduppgifter	101
5.4.2	Extrauppgifter: öva mer på grunderna	111
5.4.3	Fördjupningsuppgifter: avancerad nivå	113
5.5	Laboration: cards	116
5.5.1	Bakgrund	116
5.5.2	Obligatoriska uppgifter	116
5.5.3	Frivilliga extrauppgifter	117

6	Klasser	119
6.1	Vad är en klass?	120
6.2	Designexempel: Klassen Complex	120
6.3	Specifikationer av klasser i Scala	120
6.4	Specifikationer av klasser och objekt	121
6.5	Specifikationer av Java-klasser	122
6.6	Övning: classes	123
6.6.1	Grunduppgifter	123
6.6.2	Extrauppgifter: öva mer på grunderna	129
6.6.3	Fördjupningsuppgifter: avancerad nivå	129
6.7	Laboration: turtlegraphics	130
6.7.1	Bakgrund	130
6.7.2	Obligatoriska uppgifter	131
6.7.3	Frivilliga extrauppgifter	134
7	Arv	137
7.1	TODO: Begrepp att förklara	138
7.2	Medlemmar och arv	138
7.3	Regler för override i Scala.	138
7.4	När använda en trait och när använda en klass som supertyp?	139
7.5	Designexempel: Klassen ???	139
7.6	Övning: traits	140
7.6.1	Grunduppgifter	140
7.6.2	Extrauppgifter: öva mer på grunderna	150
7.6.3	Fördjupningsuppgifter: avancerad nivå	150
7.7	Laboration: turtle-ace-team	153
7.7.1	Obligatoriska uppgifter	153
7.7.2	Frivilliga extrauppgifter	155
8	Mönster, Undantag	157
8.1	TODO: Begrepp att förklara	158
8.2	Javas switch-sats	158
8.3	Javas switch-sats	158
8.4	TODO: Begrepp att förklara	159
8.5	Undantag	159
8.6	Övning: matching	160
8.6.1	Grunduppgifter	160
8.6.2	Extrauppgifter: öva mer på grunderna	170
8.6.3	Fördjupningsuppgifter: avancerad nivå	170
8.7	Laboration: chords-team	171
8.7.1	Bakgrund	171
8.7.2	Obligatoriska uppgifter	171
8.7.3	Extrauppgifter	174

9	Matriser, Typparametrar	177
9.1	Övning: matrices	178
9.1.1	Grunduppgifter	178
9.1.2	Extrauppgifter: öva mer på grunderna	179
9.1.3	Fördjupningsuppgifter: avancerad nivå	179
9.2	Laboration: maze	180
9.2.1	Bakgrund	180
9.2.2	Obligatoriska uppgifter	180
9.2.3	Frivilliga extrauppgifter	182
10	Sökning, Sortering	185
10.1	Övning: sorting	186
10.1.1	Grunduppgifter	186
10.1.2	Extrauppgifter: öva mer på grunderna	186
10.1.3	Fördjupningsuppgifter: avancerad nivå	186
10.2	Laboration: surveydata-team	187
10.2.1	Obligatoriska uppgifter	187
10.2.2	Frivilliga extrauppgifter	187
11	Scala och Java	189
11.1	Övning: scalajava	190
11.1.1	Grunduppgifter	190
11.1.2	Extrauppgifter: öva mer på grunderna	190
11.1.3	Fördjupningsuppgifter: avancerad nivå	190
11.2	Laboration: lthopoly-team	191
11.2.1	Bakgrund	191
11.2.2	Kodstruktur	192
11.2.3	Obligatoriska uppgifter	196
11.2.4	Frivilliga extrauppgifter	198
12	Trådar	199
12.1	Övning: threads	200
12.1.1	Grunduppgifter	200
12.1.2	Extrauppgifter: öva mer på grunderna	203
12.1.3	Fördjupningsuppgifter: avancerad nivå	203
12.2	Laboration: life	205
12.2.1	Bakgrund	205
12.2.2	Reglerna	205
12.2.3	Obligatoriska uppgifter	206
12.2.4	Frivilliga extrauppgifter	206
13	Design	209
13.1	Projektuppgift: bank	210
13.1.1	Fokus	210
13.1.2	Bakgrund	210
13.1.3	Krav	210
13.1.4	Design	211

13.1.5	Obligatoriska uppgifter	213
13.1.6	Frivilliga extrauppgifter	214
13.1.7	Exempel på körning av programmet	214
13.2	Projektuppgift: tictactoe	220
13.2.1	Bakgrund	220
13.2.2	Regler	220
13.2.3	Teori	220
13.2.4	Obligatoriska uppgifter	221
13.2.5	Frivilliga extrauppgifter	223
13.3	Projektuppgift: imageprocessing	224
13.3.1	Bakgrund	224
13.3.2	Uppgiften	224
13.3.3	Frivilliga extrauppgifter	228
14	Tentaträning	231
III	Appendix	233
A	Virtuell maskin	235
A.1	Vad är en virtuell maskin?	235
A.2	Installera kursens vm	235
A.3	Vad innehåller kursens vm?	236
B	Terminalfönster och kommandoskal	237
B.1	Vad är ett terminalfönster?	237
B.2	Några viktiga terminalkommando	237
C	Editera	239
C.1	Vad är en editor?	239
C.2	Välj editor	239
D	Kompilera och exekvera	241
D.1	Vad är en kompilator?	241
D.2	Java JDK	241
D.2.1	Installera Java JDK	241
D.3	Scala	241
D.3.1	Installera Scala-kompilatorn	241
D.4	Read-Evaluate-Print-Loop (REPL)	241
D.4.1	Scala REPL	241
E	Dokumentation	243
E.1	Vad gör ett dokumentationsverktyg?	243
E.2	scaladoc	243
E.3	javadoc	243

F	Integrerad utvecklingsmiljö	245
F.1	Vad är en IDE?	245
F.2	Kojo	245
F.2.1	Installera Kojo	245
F.2.2	Använda Kojo	245
F.3	Eclipse och ScalaIDE	245
F.3.1	Installera Eclipse och ScalaIDE	245
F.3.2	Använda Eclipse och ScalaIDE	245
G	Byggverktyg	247
G.1	Vad gör ett byggverktyg?	247
G.2	Byggverktyget sbt	247
G.2.1	Installera sbt	247
G.2.2	Använda sbt	247
H	Versionshantering och kodlagring	249
H.1	Vad är versionshantering?	249
H.2	Versionshanteringsverktyget git	249
H.2.1	Installera git	249
H.2.2	Använda git	249
H.3	Vad är nyttan med en kodlagringsplats?	249
H.4	Kodlagringsplatsen GitHub	249
H.4.1	Installera klienten för GitHub	249
H.4.2	Använda GitHub	249
H.5	Kodlagringsplatsen Atlassian BitBucket	249
H.5.1	Installera SourceTree	249
H.5.2	Använda SourceTree	249
I	Nyckelord	251
I.1	Vad är ett nyckelord ord?	251
I.2	Nyckelord i Scala	251
I.3	Nyckelord i Java	251
J	Hur bidra till kursmaterialet?	253
J.1	Bidrag är varmt välkomna!	253
J.2	Instruktioner	253
J.2.1	Vad behövs för att kunna bidra?	253
J.2.2	Svenska eller engelska?	253
J.3	Exempel	254
K	Lösningförslag till övningar	257
K.1	expressions	258
K.1.1	Grunduppgifter	258
K.1.2	Extrauppgifter: öva mer på grunderna	262
K.1.3	Fördjupningsuppgifter: avancerad nivå	262
K.2	programs	263
K.3	functions	264

K.4 data	265
K.5 sequences	266
K.6 classes	267
K.7 traits	268
K.8 matching	269
K.9 matrices	270
K.10 sorting	271
K.11 scalajava	272
K.12 threads	273
L Ordlista	275

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.

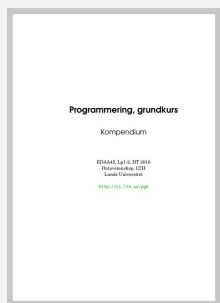
Vad lär du dig?

- 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!**

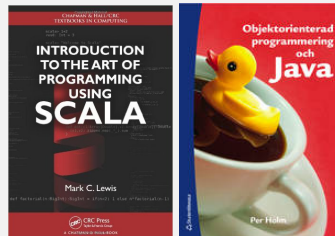
Kurslitteratur



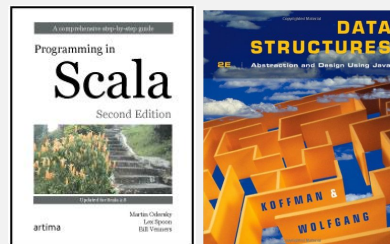
- **Kompendium** med föreläsningsanteckningar, övningar & laborationer
- Säljs på KFS
<http://www.kfsab.se/>

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

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"/> identifierare | <input type="checkbox"/> false |
| <input type="checkbox"/> variabel | <input type="checkbox"/> MinValue |
| <input type="checkbox"/> typ | <input type="checkbox"/> MaxValue |
| <input type="checkbox"/> tilldelning | <input type="checkbox"/> aritmetik |
| <input type="checkbox"/> namn | <input type="checkbox"/> slumpstal |
| <input type="checkbox"/> val | <input type="checkbox"/> math.random |
| <input type="checkbox"/> var | <input type="checkbox"/> logiska uttryck |
| <input type="checkbox"/> def | <input type="checkbox"/> de Morgans lagar |
| <input type="checkbox"/> inbyggda typer | <input type="checkbox"/> while-sats |
| <input type="checkbox"/> Int | <input type="checkbox"/> for-sats |
| <input type="checkbox"/> Long | |

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
 - Kompatibilitet: 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; ...

```
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 Literaler

Literaler representerar ett fixt värde i koden. Literaler används för att skapa data i ett program.

1.10 Uttryck

Räknar ut något nytt baserat på existerande delar.

1.11 Identifierare

Namn på saker.

1.12 Speciella identifierare

Backticks för att komma runt krockar med nyckelord. 'var'

1.13 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 deklarerera och använda variabler och tilldelning, samt kunna rita bilder av minnessituationen då variablers värden förändras.
- ☐ Förstå skillnaden mellan olika numeriska typer, kunna omvandla mellan dessa och vara medveten om noggrannhetsproblem som kan uppstå.
- ☐ Förstå booleska uttryck och värdena **true** och **false**, samt kunna förenkla booleska uttryck.
- ☐ Förstå skillnaden mellan heltalsdivision och flyttalsdivision, samt användning av rest vid heltalsdivision.
- ☐ Förstå precedensregler och användning av parenteser i uttryck.
- ☐ Kunna använda **if**-satser och **if**-uttryck.
- ☐ Kunna använda **for**-satser och **while**-satser.
- ☐ Kunna använda `math.random` för att generera slumptal i olika intervall.

Förberedelser

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

1.13.1 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 = "Robinsson"
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 'Robinsson' har 9 bokstäver.
```

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

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

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

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

Vad har följande uttryck för värden och typ?

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

Uppgift 20. for-sats.

- 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>

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

```
scala> math.random
```

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

```
scala> for (i <- 1 to 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 (slumpthal > 0.2) { println(slumpthal); 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**

1.13.2 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`.)

1.13.3 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. Backticks. `val`konstig val` = 42`

Uppgift 28. `Integer.toBinaryString`, `Integer.toHexString`

Uppgift 29. Typannoteringar.

Uppgift 30. `0x2a`

Uppgift 31. `i += 1; i *= 1; i /= 2`

Uppgift 32. `BigInt`, `BigDecimal`

Uppgift 33. Vad händer här?

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

Uppgift 34. 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 35. 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 36. För typerna `Byte`, `Short`, `Char`, `Int`, `Long`, `Float`, `Double`: Undersök hur många bitar som behövs för att representera varje typs omfång?

Tips: Några användbara uttryck:

```
Integer.toBinaryString(Int.MaxValue + 1).size
```

```
Integer.toBinaryString((math.pow(2,16) - 1).toInt).size
```

`1 + math.log(Long.MaxValue)/math.log(2)` Se även språkspecifikationen för Scala, kapitlet om heltalsliterals:

<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 37. 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
```


1.14 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 1.13.
- ☐ 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.

1.14.1 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. Vilka autoformateringar gör programmet lättare att läsa? Vilka manuella formateringar tycker du bör göras för att öka läsbarheten? Diskutera läsbarheten med en handledare.

Uppgift 5. Alternativ.

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

```
sudda; sakta(5000)

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

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

1.14.2 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 < 1e8.toLong) { 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 centralprocessors 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         isInstanceOf
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?

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

```
{
  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:

- ☐ definera funktion
- ☐ anropa funktion
- ☐ parameter
- ☐ returtyp
- ☐ värdeandrop
- ☐ namnanrop
- ☐ default-argument
- ☐ namngivna argument
- ☐ applicera funktion på alla element i en samling
- ☐ procedur
- ☐ värdeanrop vs namnanrop
- ☐ uppdelad parameterlista
- ☐ skapa egen kontrollstruktur
- ☐ objekt
- ☐ modul
- ☐ punktnotation
- ☐ tillstånd
- ☐ metod
- ☐ medlem
- ☐ funktionsvärde
- ☐ funktionstyp
- ☐ äkta funktion
- ☐ stegad funktion
- ☐ apply
- ☐ lazy val
- ☐ lokala funktioner
- ☐ anonyma funktioner
- ☐ lambda
- ☐ aktiveringspost
- ☐ rekursion
- ☐ basfall
- ☐ anropsstacken
- ☐ objektheapen
- ☐ algoritm: GCD (största gemensamma delare)
- ☐ 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)
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 `Slumpal` 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 funktionsapplieringssyntax.

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:

- | | |
|--|---|
| <input type="checkbox"/> attribut (fält) | <input type="checkbox"/> List |
| <input type="checkbox"/> medlem | <input type="checkbox"/> Vector |
| <input type="checkbox"/> metod | <input type="checkbox"/> Set |
| <input type="checkbox"/> tupel | <input type="checkbox"/> Map |
| <input type="checkbox"/> klass | <input type="checkbox"/> typparameter |
| <input type="checkbox"/> Any | <input type="checkbox"/> generisk samling som parameter |
| <input type="checkbox"/> instanceof | <input type="checkbox"/> översikt samlingsmetoder |
| <input type="checkbox"/> toString | <input type="checkbox"/> översikt strängmetoder |
| <input type="checkbox"/> case-klass | <input type="checkbox"/> läsa/skriva textfiler |
| <input type="checkbox"/> räkna med bråk och klassen Frac | <input type="checkbox"/> Source.fromFile |
| <input type="checkbox"/> föränderlighet vs oföränderlighet | <input type="checkbox"/> 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. *Typpparameter.* Vissa funktioner är generella för många typer och tar en så kallad **typpparameter** 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

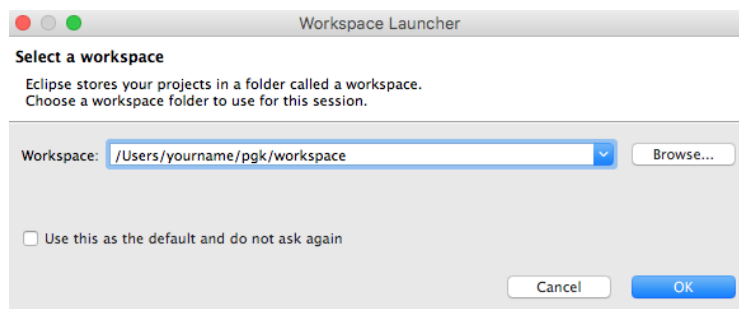
- ☐ Kunna använda utvecklingsmiljön Eclipse och dess Scala IDE.
- ☐ Kunna använda case classer för att spara data.
- ☐ Kunna spara data till fil.
- ☐ Kunna läsa in data från fil med `scala.io`.
- ☐ Kunna skapa och använda klasser för att behandla data.
- ☐ Kunna använda samlingstyperna vektor och map samt Option.
- ☐ Förstå och kunna använda Option, Some och None.
- ☐ Förstå skillnaden mellan kompileringsfel och exekveringsfel.
- ☐ Kunna avlusa (debugga) program med hjälp av Eclipse.

Förberedelser

- ☐ Gör övning 4.
- ☐ Läs om Eclipse i Appendix.
- ☐ Läs igenom laborationen och gör förberedelseuppgiften.

4.5.1 Förberedelseuppgifter

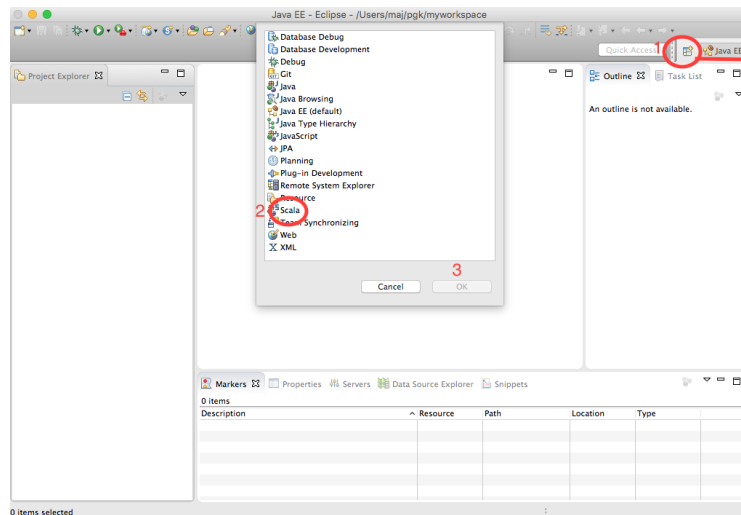
Uppgift 1. Ladda hem zip-filen med kursens workspace från git-repot (XX) och packa upp det på valfritt ställe på din dator. Starta programmet Eclipse och följ instruktionerna i figurer för att skapa ett program och exekvera det.



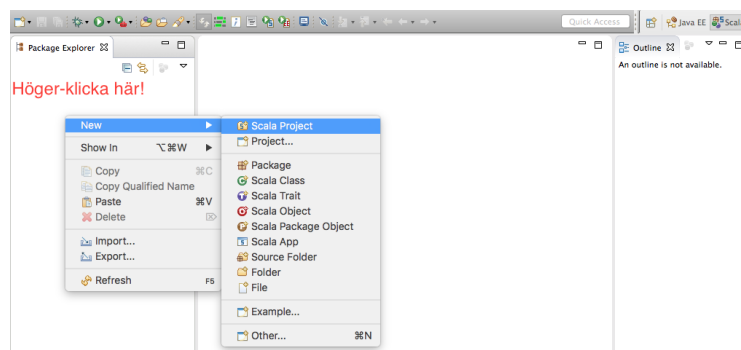
1. Bläddra fram till kurs-workspacet.



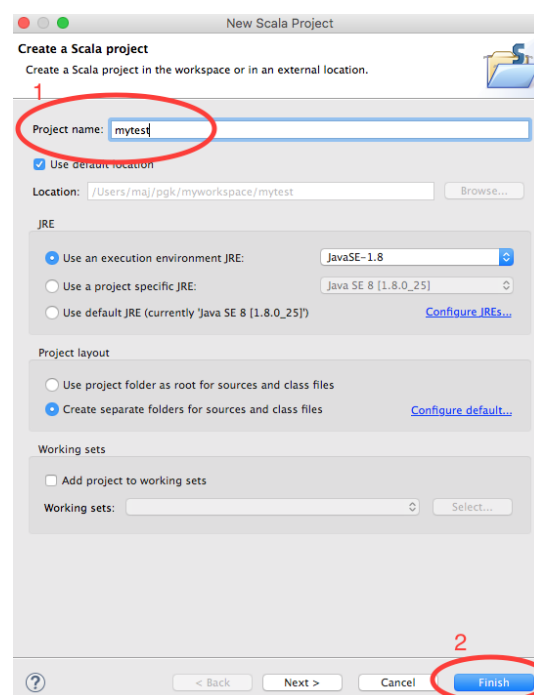
2. Första gången Eclipse öppnas har den en välkomstskärm, klicka på **Workbench**.



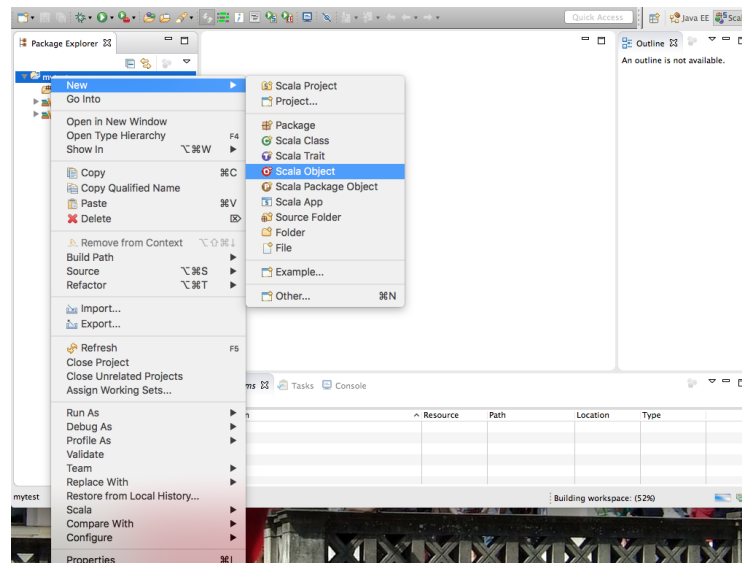
3. Kontrollera att du har Scala igång uppe i högra hörnet, annars klickar du på knapp nummer 1 och väljer Scala bland dina plugin (2) och **OK**.



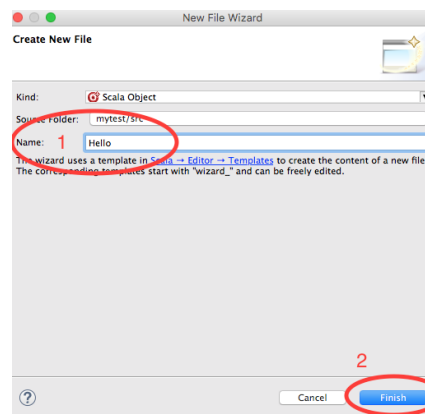
4. Skapa ett nytt projekt genom att höger-klicka i **Package Explorer**.



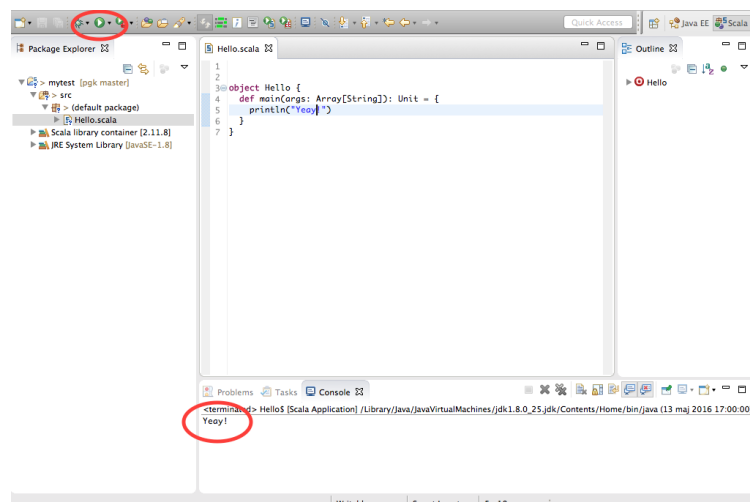
5. Döp ditt projekt till ett beskrivande namn.



6. Genom att höger-klicka på projektet kan man lägga till filer, t ex skapa ett paket eller en klass. Börja med att skapa ett objekt ...



7 ... som du döper till nåt beskrivande!



8. Lägg till kod som skriver ut en hälsning. Exekvera sedan genom att trycka på den gröna pilen och se om utskriften kommer ut i Eclipse-konsollen.

Prova att stava fel till `println`, då dyker det upp ett rött kryss till vänster på samma kodrad som berättar vad du gjort för fel. Kompilatorn kör i bakgrunden och innan koden kan exekveras måste alla kompileringsfel åtgärdas. Men betyder detta att programmet alltid kommer bete sig korrekt under körningen?

4.5.2 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 2. *Save your crew.*

a) Kung George är villig att benåda fem personer ur din besättning! Skapa en lista (nåja, vektor) där personerna sparas med förnamn, efternamn och befattning genom att läsa in dem från konsolen i Eclipse. Inläsning kan göras med `scala.io` med kodraden:

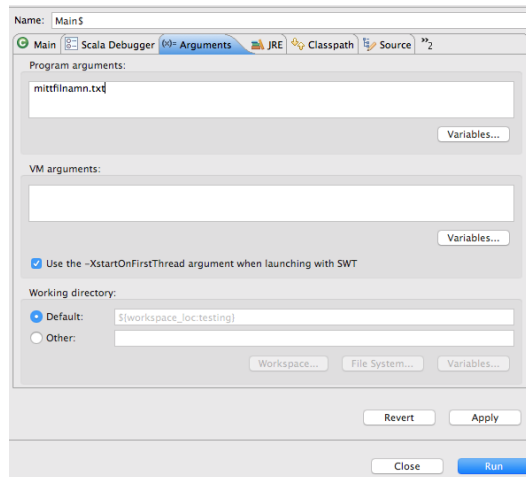
```
val first = scala.io.StdIn.readLine("Förnamn: ").
```

Namnen och befattningen kan sparas som en `case class` `CrewMember`! Skriv ett program som läser in namn och befattning på dina besättningsmedlemmar och spara dem i en vektor.

b) Vi vill skriva besättningen till en fil så att den faktiskt sparas. Till din hjälp får du följande kodsnuitt som tar en sträng `s` och sparar den till en fil `fileName`:

```
def write(s: String, fileName: String): Unit = {  
  import java.nio.file.{Paths, Files, StandardOpenOption}  
  Files.write(Paths.get(fileName), s.getBytes("UTF-8"),  
    StandardOpenOption.CREATE,  
    StandardOpenOption.TRUNCATE_EXISTING) // will append  
}
```

Istället för att hårdkoda filnamnet går det att ange som argument genom att högerklicka på klassfilen i **Project explorer**, välja **Run As -> Run Configurations** och under fliken **Arguments** skriva in argument (separerade med mellanslag) och sen välja **Run** eller **Apply**.



Lägg till kodrader i ditt program som sparar besättningen till filnamnet i det första argumentet om ett sådant har angivits, annars till filen `crew.txt`.

c) Det går att överskugga `toString()` i `CrewMember` och på så sätt ändra utskriften genom att lägga till följande kodrad inne i klassen:

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

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

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

Uppgift 3. Avlusa din besättning.


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

En fil `fileName` kan läsas rad för rad till en vektor med följande kodexempel:

```
def readLines(fileName: String): Vector[String] = {  
    scala.io.Source.fromFile(fileName).getLines.toVector  
}
```

Skapa en funktion `readCrewMember(s:String)` som tar en rad från filen och returnerar en `CrewMember`.

b) Skriv ett testprogram som läser in din besättning från filen och skriver ut dem i konsolen. Stämmer det med filinnehållet?

c) Det går att följa exekveringen av programmet stegvis genom att köra det i *debug mode* som startas med 


Då öppnas en debuggvy i Eclipse. Variablerna visas uppe i högra hörnet. Där går att klicka och se vilka värden varje variabel har för tillfället.

Genom att lägga till *break points* (klicka på sidan av koden för att lägga till och ta bort dem) kan programexekveringen pausas just innan raden exekveras

```

7   for(s <- data){
8     Line breakpoint:readTest [line: 8] - main
9
10

```

så att programmeraren kan kontrollera variablernas värden och sen köra vidare med .

d) Den konkurrerande kaptenen Charles Vane betalar dig för att sabotera listan genom att lägga till nonsensrader i filen. Gör det. Vad händer då när du exekverar ditt testprogram?

e) Ändra din funktion `readCrewMember(s:String)` så att en korrekt formaterad rad returnerar en `CrewMember` medan felaktiga rader ger `None`. Det går att göra genom att använda en behållartyp som heter `Option` som fungerar som en samling som kan innehålla `None` eller något `Some(CrewMember(first, last, post))`. Returtypen kan då deklarerars som

```
def readCrewMember(s:String): Option[CrewMember] = ...
```

Resultatet från ett `option`-objekt o hämtas med t ex `o.get` eller genom att skriva ut felmeddelanden om det är `None`, o.`getOrElse("I'm no one")`. Testa ditt nya program och se om det blir som förväntat genom lämpliga break points och utskrifter.

Uppgift 4. Lögner, förbannade lögner och statistik.

För att du inte ska bli överlistad av dina sluga, lögnaktiga skeppskamrater behöver du kunna gissa hur en pirat tänker. Därför förkovrar du dig i Robert Louis Stevensons *Skattkammarön*³ som finns i filen `skattkammarön.txt` i workspacet. Genom att för varje ord spara det mest frekventa nästkommande ordet går det att förutse vad som kommer sägas⁴.

a) Det går att läsa in ord från en fil med `scala.io.Source.fromFile` genom att först läsa in alla rader, ta bort allt som *inte* är svenska bokstäver och sen göra en split på *white space*:

```
def readWords(fileName: String): Vector[String] = {
  scala.io.Source.fromFile(fileName).getLines.
  map(_.replaceAll("[^a-zA-ZåäöÅÄÖ\\s]", " ")).
  flatMap(_.split("\\s+")).filter(!_.isEmpty).toVector
}
```

Skapa objektet `PirateSpeech` och lägg till kodrader som läser in alla ord i filen `skattkammarön.txt`. Skapa sen en klass `FrequencyCounter` som för varje ord kan räkna nästkommande ord (sådana ordpar kallas bigram). Varje ord får en egen `FrequencyCounter` som i sin tur innehåller en samling, t ex en `Map` som listar alla efterföljande ord och deras antal så att vi kan få deras frekvens.

b) Lägg till en funktion `add(next:String)` i `FrequencyCounter` som ökar frekvensen av det ordet `next` eller lägger till det med räknaren satt till 1 om

³Vars copyright har gått ut så du behöver inte piratkopiera den.

⁴Detta används till exempel i Swiftkey på smarttelefoner.

det inte finns.

- c) Skriv ett program som läser in alla ord och räknar ut deras frekvens. Tips: håll reda på orden och deras bigramfrekvensräknare med en `[Map[String, FrequencyCounter]]`.
- d) Lägg till en funktion `getBestGuess()` i `FrequencyCounter` som returnerar det mest frekventa efterföljande ordet.
- e) Det tar lång tid att läsa hela boken varje gång programmet körs, det gäller ju att vara kvicktänkt och vi är ju bara intresserade av den mest sannolika gissningen! Spara ordpar `word, bestGuess` med den bästa gissningen till varje ord på en fil och läs bara in paren i nästa uppgift.
- f) Skriv ett main-program som läser in ett ord från användaren och gissar vad de ska säga sen. Efterföljs "James" oftast av "Hawkins" och eller av "Flint"?

4.5.3 Frivilliga extrauppgifter

Uppgift 5. Läs in större mängder text, implementera ett tangentbord till Android och bli rik.

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 slumpvalsfrö.
- ☐ 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)
```

- Använd metoderna `filter` och `size` på `xs` för att filtrera ut alla 6:or och räkna hur många de är.
- Använd metoderna `filter` och `size` på `xs` för att filtrera ut alla jämna kast och räkna hur många de är.
- 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 pseudoslumtalssekvenser med java.util.Random.* Klassen `java.util.Random` ger möjlighet att generera en sekvens av tal som verkar slumpmässiga. Genom att välja ett visst s.k. **frö** (eng. *seed*) kan man få samma sekvens av pseudoslumptal varje gång.

-  a) Sök upp och studera dokumentationen för `java.util.Random`. Hur skapar man en ny instans av klassen `Random`? Vad gör operationen `nextInt` på `Random`-objekt.
b) 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. *Algoritm: 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. *Algoritm: SEQ-INSERT.* Man kan implementera algoritmen SEQ-INSERT på plats i en `Array[Int]` så att alla elementen efter pos flyttas fram ett steg och att sista elementet "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

- ☐ Kunna använda vektorer.
- ☐ Kunna använda SHUFFLE-algoritmen för kortblandning.
- ☐ Kunna räkna frekvenser.

Förberedelser

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

5.5.1 Bakgrund

Denna labb handlar om kortblandning. Att blanda kort så att varje möjlig permutation är lika sannolik är icke-trivialt; en osystematiskt blandning leder till en skev fördelning. Förutsatt att det finns en bra slumpgenerator, går det att blanda en kortlek genom att lägga alla kort i en hög och sedan ta ett slumpvist kort från högen och lägga det överst i leken, tills alla kort flyttats från högen till leken. Enligt denna princip 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    $r \leftarrow$  slumptal 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.

- a) Implementera metoden `shuffle` i klassen `CardDeck`. Följ algoritmen noga, och använd `cards.length` för att få fram längden på kortleken.
- b) Kör `TestingDeck` för att testa att blandningen är jämnt fördelad. `TestingDeck` blandar en kortlek med tre kort och räknar hur ofta olika permutationer dyker upp. Du bör få en utskrift med sex (3!) ungefär lika långa staplar.

Uppgift 2. Fyll i de ofärdiga delarna av klassen `CardDeck`.

- a) Skriv kod för att skapa en array innehållande en 52-korts standardlek. Använd konstanterna i `Cards`. Importera antingen `UnicodeSuits` eller `LetterSuits` om det första inte fungerar. Tänk på att en **for/yield**-sats inte nödvändigtvis ger en Array, men att alla samlingar kan omvandlas till en sådan med `toArray`.

b) Kör CardDeck och kontrollera så att alla kort ser ut att finnas med.

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.

a) Implementera funktionen `test` i `PokerProbability`. Använd de färdiga funktionerna `Hand.drawFrom` och `testHand` för att dra och klassificera en hand från en kortlek. Lagra frekvenserna i en muterbar Map (`collection.mutable.Map` finns redan importerad).

b) Kör `PokerProbability`. Du bör få ungefär

Royal flush	0.000154%
Straight flush	0.00139%
Flush	0.197%
Straight	0.39%
High card	99.41%

5.5.3 Frivilliga extrauppgifter

Uppgift 4. Implementera metoden `tally` i klassen `Hand` så att simuleringen även kan registrera kortkombinationerna fyrtal, kåk, triss, tvåpar och par. Kör sedan `PokerProbability` igen.

Kapitel 6

Klasser

Koncept du ska lära dig denna vecka:

- ☐ objektorientering
- ☐ klass
- ☐ Point
- ☐ Square
- ☐ Complex
- ☐ new
- ☐ null
- ☐ this
- ☐ inkapsling
- ☐ accessregler
- ☐ private
- ☐ private[this]
- ☐ kompanjonsobjekt
- ☐ getters och setters
- ☐ klassparameter
- ☐ primär konstruktor
- ☐ objektfabriksmetod
- ☐ överlagring av metoder
- ☐ referenslikhet vs strukturelikhet
- ☐ eq vs ==

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 Designexempel: Klassen Complex

TODO:

- Bygg upp **case class** Complex(re: Double, im: Double) steg för steg inspirerat av Pins3ed kap 6 i likhet med hur de gör med Rational
- Illustrera följande begrepp: this (behövs i max(that)), method overloading behövs för att plussa med både Complex och Double
- Till fördjupningsövning: dekorera Double med metoderna im och re samt (Double, Double) med metoden ir (för irrational) med implicit klass
- Till extrauppgift: implementera klassen Polar(r, fi) med polära koordinater https://sv.wikipedia.org/wiki/Pol%C3%A4ra_koordinater

6.3 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.

6.4 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.

6.5 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;
```


6.6 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.

6.6.1 Grunduppgifter

Uppgift 1. *Instansiering med **new** och värdet **null**.* Man skapar instanser av klasser med **new**. Då anropas konstruktorn och plats reserveras i datorns minne för objektet. Variabler av referenstyp som inte refererar till något objekt har värdet **null**.

a) Vad händer nedan? Vilka rader ger felmeddelande och i så fall hur lyder felmeddelandet?

```
1 scala> class Gurka(val vikt: Int)
2 scala> var g: Gurka = null
3 scala> g.vikt
4 scala> g = new Gurka(42)
5 scala> g.vikt
6 scala> g = null
7 scala> g.vikt
```



b) Rita minnessituationen efter raderna 2, 4, 6.

Uppgift 2. *Klasser och instanser.*

a) Vad händer nedan?

```
1 scala> :pa
2 class Arm(val ärTillVänster: Boolean)
3 class Ben(val ärTillVänster: Boolean)
```

```

4 class Huvud(val harHår: Boolean)
5 class RymdVarelse {
6     var arm1 = new Arm(true)
7     var arm2 = new Arm(false)
8     var ben1 = new Ben(true)
9     var ben2 = new Ben(false)
10    var huvud1 = new Huvud(false)
11    var huvud2 = new Huvud(true)
12    def ärSkallig = !huvud1.harHår && !huvud2.harHår
13 }
14 scala> val alien = new RymdVarelse
15 scala> alien.ärSkallig
16 scala> val predator = new RymdVarelse
17 scala> predator.ärSkallig
18 scala> predator.huvud2 = alien.huvud1
19 scala> predator.ärSkallig

```

b) Rita minnessituationen efter rad 18.

c) Vad händer så småningom med det ursprungliga huvud2-objektet i predator efter tilldelningen på rad 18? Går det att referera till detta objekt på något sätt?

Uppgift 3. *Synlighet i primärkonstruktörer.* Undersök nedan vad nyckelorden **val** och **private** får för konsekvenser. Förklara vad som händer. Vilka rader ger vilka felmeddelanden?

```

1 scala> class Gurka1(vikt: Int)
2 scala> new Gurka1(42).vikt
3 scala> class Gurka2(val vikt: Int)
4 scala> new Gurka2(42).vikt
5 scala> class Gurka3(private val vikt: Int)
6 scala> new Gurka3(42).vikt
7 scala> class Gurka4(private val vikt: Int, kompis: Gurka4){
8     def kompisVikt = kompis.vikt
9 }
10 scala> val ingenGurka: Gurka4 = null
11 scala> new Gurka4(42, ingenGurka).kompisVikt
12 scala> new Gurka4(42, new Gurka4(84, null)).kompisVikt
13 scala> class Gurka5(private[this] val vikt: Int, kompis: Gurka5){
14     def kompisVikt = kompis.vikt
15 }
16 scala> class Gurka6 private (vikt: Int)
17 scala> new Gurka6(42)
18 scala> :pa
19 class Gurka7 private (var vikt: Int)
20 object Gurka7 {
21     def apply(vikt: Int) = {
22         require(vikt >= 0, s"negativ vikt: $vikt")
23         new Gurka7(vikt)
24     }
25 }
26 scala> new Gurka7(-42)
27 scala> Gurka7(-42)
28 scala> val g = Gurka7(42)

```

```

29 scala> g.vikt
30 scala> g.vikt = -1
31 scala> g.vikt

```


Uppgift 4. Egendefinierad setter kombinerat med privat 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 strukturellikhet.* Metoden == på case-klasser ger **strukturellikhet** (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
```

- Förklara vad som händer ovan.
- 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 heltal-sattributen `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
```

- Vad ger `Point.origin eq Point.origin` för resultat om `origin` istället implementeras som **def** `origin: Point = Point(0, 0)`



- 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: Int, dy: 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?

6.6.2 Extrauppgifter: öva mer på grunderna

Uppgift 11. `class` Frac `private` (numerator: BigInt, denominator: BigInt)
TODO: Ungefär som Rational i pinsled med GCD

Uppgift 12. *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 = ???  
}
```

6.6.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 13.

6.7 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

- ☐ Klasser.
- ☐ Muterbarhet.
- ☐ Abstraktion

6.7.1 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

```
/** Creates a window and makes it visible. */
public SimpleWindow(int width, int height, String title);

/** Returns the width of the window. */
public int getWidth();

/** Returns the height of the window. */
public int getHeight();

/** Clears the window. */
public void clear();

/** Closes the window.*/
public void close();

/** Opens the window. */
public void open();

/** Moves the pen to a new position. */
public void moveTo(int x, int y) ;
```



```

/** Moves the pen to a new position while drawing a line. */
public void lineTo(int x, int y);

/** Writes a string at the current position. */
public void writeText(String txt);

/** Draws a bitmap image at the current position.*/
public void drawImage(Image image);

/** Returns the pen's x coordinate. */
public int getX();

/** Returns the pen's y coordinate. */
public int getY();

/** Sets the line width. */
public void setLineWidth(int width);

/** Sets the line color. */
public void setLineColor(Color col);

/**Returns the current line width. */
public int getLineWidth();

/** Returns the current line color. */
public Color getLineColor();

/** Waits for a mouse click. */
public void waitForMouseClicked();

/** Returns the mouse x coordinate at the last mouse click. */
public int getMouseX();

/** Returns the mouse y coordinate at the last mouse click. */
public int getMouseY();

/**Adds a sprite to the window. */
public void addSprite(Sprite sprite);

/** Wait for a specified time. */
public static void delay(int ms);

```

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

6.7.2 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.

Specification Point

```

/** Represents a single Point in the x,y plane. */

```

```

case class Point(val x: _, val y: _) {

  ** Returns a new Point which has been moved some number of pixels */
  def translate(dx: _, dy: _) = ???
}

```

- Hur borde specifikationen för Point se ut? Vilka typer bör attributen ha?
- Implementera klassen Point.

Uppgift 2. Skapa en klass Turtle enligt följande specifikation:

Specification Turtle

```

/**
  * 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) {

  /** Gets the Turtle's current pixel position on the x axis */
  def x: Int = ???

  /** Gets the Turtle's current pixel position on the y axis
      (measured from the top left) */
  def y: Int = ???

  /**
    * Moves the turtle to a new position without drawing a line.
    */
  def jumpTo(newPosition: Point) : 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? Vilken/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?
- Använd din Turtle för att rita en cirkel. För att göra detta kan du t.ex. låta din Turtle gå ett kort steg och svänga någon grad tills den har gjort ett fullt varv.
- Skapa två stycken Turtles i samma fönsterobjekt som rör sig alternerande. Fungerar allt som tänkt?

Tips:

- SimpleWindow har sitt origo i övre vänstra hörnet, till skillnad från det nedre vänstra hörnet som är vanligt inom matematik.
- Om din cirkel inte blev som förväntad, fundera på din implementation av Point. Var förekommer avrundningar?

Uppgift 3. Implementera klassen Rectangle.

Specification Rectangle

```

/** Immutable class representing a rectangle.
 * @param position a Point representing the upper left corner of the
 *                 rectangle (before rotation)
 * @param width    the width of the rectangle
 * @param height   the height of the rectangle
 * @param angle    the angle of the rectangle (rotated around
 *                 the upper left corner)
 */
class Rectangle(position: Point, width: Double,
                height : Double, angle : Double) {
  /** Draws the rectangle using a turtle */
  def draw(turtle: Turtle): Unit = ???

  /** Returns a new Rectangle that is rotated to the left */
  def rotateLeft(degrees: Double): Rectangle = ???

  /** Returns a new Rectangle that is rotated to the right */
  def rotateRight(degrees: Double): Rectangle = ???

  /** Returns a new Rectangle that has been scaled by a size factor */
  def scale(factor: Double): Rectangle = ???

  /** Returns a new Rectangle that has been moved some number of pixels */
  def translate(dx: Double, dy: Double): Rectangle = ???
}

```

6.7.3 Frivilliga extrauppgifter

Uppgift 4. Skapa en klass `RectangleSequence`. I denna klass skall `draw`-metoden rita ut ett antal rektanglar där varje rektangel har förflyttat sig, roterats och skalats jämfört med föregående rektangel i sekvensen. Se bilder nedan.

Specification `RectangleSequence`

```

/** Represents a sequence of Rectangles that have been translated,
 *     rotated, and scaled down.
 *
 * @param rectangle    the rectangle to use as a base for the shape
 * @param count        the number of rectangles to draw
 * @param angle        the number of degrees to rotate the image
 * @param step         the number of pixels to move each rectangle
 *                     (in the direction of angle)
 * @param rotationStep the number of degrees to shift each rectangle
 *                     with each roll
 */
case class RectangleSequence(rectangle: Rectangle,
                             count: Int,
                             angle: Double,
                             step: Double,
                             rotationStep: Double) {

  /** Draws the image using a given Turtle */
}

```

```
def draw(turtle: Turtle): Unit = ???

/** Returns a new RectangleSequence that has been scaled with a size factor */
def scale(factor: Double): RectangleSequence = ???

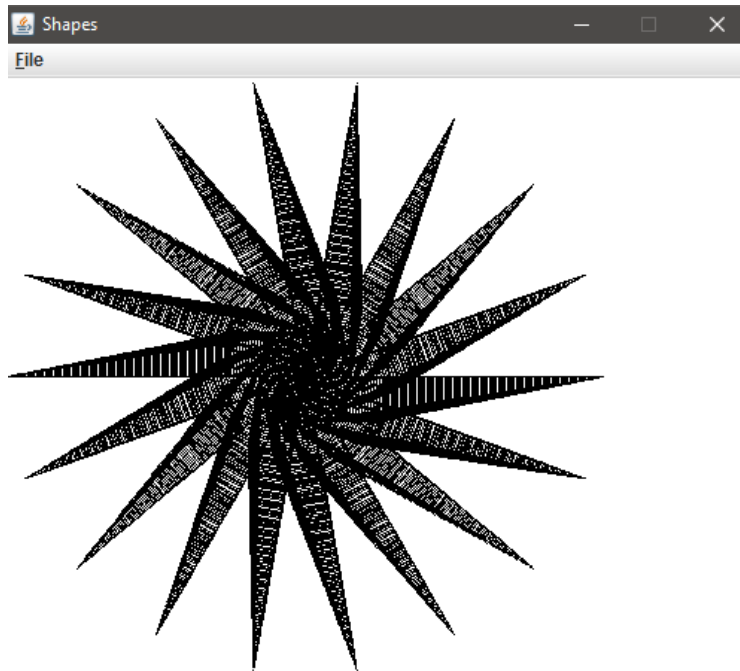
/** Returns a new RectangleSequence that has been rotated to the left */
def rotateLeft(degrees: Double): RectangleSequence = ???

/** Returns a new RectangleSequence that has been rotated to the right */
def rotateRight(degrees: Double): RectangleSequence = ???

/** Returns a new RectangleSequence that has moved some number of pixels */
def translate(dx: Double, dy: Double): RectangleSequence = ???
}
```



Resultatet av
`RectangleSequence(Rectangle(Point(200, 200), 50, 30, 0), 5, 0, 70, -30).`



Resultatet av en `RectangleSequence` roll med stort count och utan rotation kört i följande loop (givet en Turtle `t`):

```
for (i <- 0 to 360 by 20) roll.rotateLeft(i).draw(t)
```

Kapitel 7

Arv

Koncept du ska lära dig denna vecka:

- | | |
|---|--|
| <input type="checkbox"/> arv | Ref Object AnyVal Null Nothing |
| <input type="checkbox"/> polymorfism | <input type="checkbox"/> referenstyper vs värdetyper |
| <input type="checkbox"/> trait | <input type="checkbox"/> klasshierarkin i scala.collection |
| <input type="checkbox"/> extends | <input type="checkbox"/> Shape som bastyp till Point och Rectangle |
| <input type="checkbox"/> asInstanceOf | <input type="checkbox"/> accessregler vid arv |
| <input type="checkbox"/> with | <input type="checkbox"/> protected |
| <input type="checkbox"/> inmixning | <input type="checkbox"/> final |
| <input type="checkbox"/> supertyp | <input type="checkbox"/> klass vs trait |
| <input type="checkbox"/> subtyp | <input type="checkbox"/> abstract class |
| <input type="checkbox"/> bastyp | <input type="checkbox"/> case-object |
| <input type="checkbox"/> override | <input type="checkbox"/> typer med uppräknade värden |
| <input type="checkbox"/> klasshierarkin i Scala: Any Any- | |

7.1 TODO: Begrepp att förklara

Tänk igenom ordningen:

- OO, arv, supertyp, subtyp, bastyp, polymorfism, ...

7.2 Medlemmar och arv

Vid arv kan man ersätta en medlem (eng. *override a member*), så att en medlem i en subtyp får sin egen speciella skepnad.

Olika sorters medlemmar i **Scala**:

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

Olika sorters medlemmar i **Java**:

- variabel
- metod

Variabler kan vara instansvariabler eller klassvariabler (nyckelord **static**)

När man konstruerar ett objektorienterat språk gäller det att man definierar **sunda regler för arv**, som bestämmer hur en medlem kan ersättas.

7.3 Regler för override i Scala.

En medlem M1 i en supertyp får ersättas av en medlem M2 i en subtyp, givet reglerna:

1. M1 och M2 ska ha samma namn och typerna ska matcha.
2. **def** får bytas ut mot: **def**, **val**, **var**, **lazy val**
3. **val** får bytas ut mot: **val**, och om M1 är abstrakt mot en **lazy val**.
4. **var** får bara bytas ut mot en **var**.
5. **lazy val** får bara bytas ut mot en **lazy val**.
6. Om en medlem i en supertyp är abstrakt *behöver* man inte använda nyckelordet **override** i subtypen. (Men det är bra att göra det ändå så att kompilatorn kollar att man verkligen byter ut något.)
7. Om en medlem i en supertyp är konkret *måste* man använda nyckelordet **override** i subtypen, annars ges kompileringsfel.
8. M1 får inte vara **final**.
9. M1 får inte vara **private** eller **private[this]**, men kan vara **private[X]** om M2 också är **private[X]**, eller **private[Y]** om X innehåller Y.
10. Om M1 är **protected** måste även M2 vara det.

7.4 När använda en trait och när använda en klass som supertyp?

TODO

Använd en trait som supertyp om:

1. Du är osäker på vilket som är bäst.
(Du kan alltid ändra till en klass senare.)
2. etc
3. etc

Använd en klass om:

1. Du vill ge supertypen en parameter vid konstruktion.
2. etc
3. etc

7.5 Designexempel: Klassen ???

TODO:

-

7.6 Övning: traits

Mål

- ☐ Förstå följande begrepp:
 - ☐ bastyp,
 - ☐ supertyp,
 - ☐ subtyp,
 - ☐ abstrakt typ,
 - ☐ polymorfism.
- ☐ Kunna deklarerar och använda en arvshierarki i flera nivåer med nyckelordet **extends**.
- ☐ Kunna deklarerar och använda inmixning med flera traits och nyckelordet **with**.
- ☐ Kunna deklarerar och känna till nyttan med finala klasser och finala attribut och nyckelordet **final**.
- ☐ Känna till synlighetsregler vid arv och nyttan med privata och skyddade attribut.
- ☐ Kunna deklarerar och använda skyddade attrinbute med nyckelordet **protected**.
- ☐ Känna till hur typtester och typkonvertering vid arv kan göras med metoderna `isInstanceOf` och `asInstanceOf` och känna till att detta görs bättre med **match** (som kommer i kapitel 8).
- ☐ Känna till begreppet anonym klass.
- ☐ Kunna deklarerar och använda överskuggade metoder med nyckelordet **override**.
- ☐ Känna till reglerna som gäller vid överskuggning av olika sorters medlemmar.
- ☐ Kunna deklarerar och använda hierarkier av klasser där konstruktorparametrar överförs till superklasser.
- ☐ Kunna deklarerar och använda uppräknade värden med case-objekt och gemensam bastyp.

Förberedelser

- ☐ Studera teorin i kapitel 7.

7.6.1 Grunduppgifter

Uppgift 1. *Gemensam bastyp.* Man vill ofta lägga in objekt av olika typ i samma samling.

```
1 class Gurka(val vikt: Int)
2 class Tomat(val vikt: Int)
3 val gurkor = Vector(new Gurka(100), new Gurka(200))
4 val grönsaker = Vector(new Gurka(300), new Tomat(42))
```

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

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

```
1 gurkor.map(_.vikt).sum
2 grönsaker.map(_.vikt).sum
```

c) Vi kan göra så att vi kan komma åt vikten på alla grönsaker genom att ge gurkor och tomater en gemensam bas typ som de olika konkreta grönsakstyperna utvidgar med nyckelordet **extends**. Man säger att subtyperna Gurka och Tomat **ärver** egenskaperna hos supertypen Grönsak.

Attributet vikt i traiten Grönsak nedan initialiseras inte förrän konstruktorerna anropas när vi gör **new** på någon av klasserna Gurka eller Tomat.

```
1 trait Grönsak { val vikt: Int }
2 class Gurka(val vikt: Int) extends Grönsak
3 class Tomat(val vikt: Int) extends Grönsak
4 val gurkor = Vector(new Gurka(100), new Gurka(200))
5 val grönsaker = Vector(new Gurka(300), new Tomat(42))
```

d) Vad blir det nu för typ på 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. Den sägs vara abstrakt eftersom den saknar definition – medlemmen har bara ett namn och en typ men inget värde. Du kan instansiera den abstrakta traiten Grönsak om du fyller i det som "fattas", nämligen ett värde på vikt. Man kan fylla på det som fattas i genom att "hänga på" ett block efter typens namn vid instansiering. Man får då vad som kallas en **anonym** klass, i detta fall en ganska konstig grönsak som inte är någon speciell sorts grönsak med som ändå har en vikt.

Vad får anonymGrönsak nedan för typ och strängrepresentation?

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

Uppgift 2. *Polymorfism i samband med arv.* Polymorfism betyder "många skepnader". I samband med arv innebär det att flera subtyper, till exempel Ko och Gris, kan hanteras gemensamt som om de vore instanser av samma supertyp, så som Djur. Subklasser kan implementera en metod med samma namn på

olika sätt. Vilken metod som exekveras bestäms vid körtid beroende på vilken subtyp som instansieras. På så sätt kan djur komma i många skepnader.

a) Implementera funktionen `skapaDjur` nedan så att den returnerar antingen en ny `Ko` eller en ny `Gris` med lika sannolikhet.

```
1 scala> trait Djur { def väsnas: Unit }
2 scala> class Ko extends Djur { def väsnas = println("Muuuuuuu") }
3 scala> class Gris extends Djur { def väsnas = println("Nöffnöff") }
4 scala> def skapaDjur: Djur = ???
5 scala> val bondgård = Vector.fill(42)(skapaDjur)
6 scala> bondgård.foreach(_.väsnas)
```

b) Lägg till ett djur av typen `Häst` som väsnas på lämpligt sätt och modifiera `skapaDjur` så att det skapas kor, grisar och hästar med lika sannolikhet.

Uppgift 3. *Bastypen `Shape` och subtyperna `Rectangle` och `Circle`.* Du ska nu skapa ett litet bibliotek för geometriska former med oföränderliga objekt implementerade med hjälp av case-klasser. De geometriska formerna har en gemensam bastyp kallad `Shape`. Skriv nedan kod i en editor och klistra sedan in den i REPL med kommandot `:paste`.

```
case class Point(x: Double, y: Double) {
  def move(dx: Double, dy: Double): Point = Point(x + dx, y + dy)
}

trait Shape {
  def pos: Point
  def move(dx: Double, dy: Double): Shape
}

case class Rectangle(
  pos: Point,
  dx: Double,
  dy: Double
) extends Shape {
  override def move(dx: Double, dy: Double): Rectangle =
    Rectangle(pos.move(dx, dy), this.dx, this.dy)
}

case class Circle(pos: Point, radius: Double) extends Shape {
  override def move(dx: Double, dy: Double): Circle =
    Circle(pos.move(dx, dy), radius)
}
```

a) Instansiera några cirklar och rektanglar och gör några relativa förflyttningar av dina instanser genom att anropa `move`.

b) Lägg till metoden `moveTo` i `Point`, `Shape`, `Rectangle` och `Circle` som gör en absoult förflyttning till koordinaterna `x` och `y`. Klistra in i REPL och testa på

några instanser av `Rectangle` och `Circle`.

c) Lägg till metoden `distanceTo(that: Point): Double` i case-klassen `Point` som räknar ut avståndet till en annan punkt. Klistra in i REPL och testa på några instanser av `Rectangle` och `Circle`.

d) Läg till en konkret metod `distanceTo(that: Shape): Double` i traiten `Shape` som räknar ut avståndet till positionen för en annan `Shape` med hjälp av `math.hypot`. Klistra in i REPL och testa på några instanser av `Rectangle` och `Circle`.

Uppgift 4. Inmixning. Man kan utvidga en klass med multipla traits med nyckelordet **with**. På så sätt kan man fördela medlemmar i olika traits och återanvända gemensamma koddelar genom så kallad **inmixning**, så som nedan exempel visar.

En alternativ fågeltaxonomi, speciellt populär i Skåne, delar in alla fåglar i två specifika kategorier: Kråga respektive Ånka. Krågor kan flyga men inte simma, medan Ånkor kan simma och oftast även flyga. Fågel i generell, kollektiv bemärkelse kallas på gammal skånska för Fyle.¹ Skriv in nedan kod i en editor och spara den för kommande uppgifter. Klistra in koden i REPL med kommandot `:paste`.

```
trait Fyle {
  val läte: String
  def väsnas: Unit = print(läte * 2)
  val ärSimkunnig: Boolean
  val ärFlygkunnig: Boolean
}

trait KanSimma      { val ärSimkunnig = true }
trait KanInteSimma { val ärSimkunnig = false }
trait KanFlyga      { val ärFlygkunnig = true }
trait KanKanskeFlyga { val ärFlygkunnig = math.random < 0.8 }

class Kråga extends Fyle with KanFlyga with KanInteSimma {
  val läte = "krax"
}

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

a) En flitig ornitolog hittar 42 fåglar i en skog där alla fågelsorter 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`, `asInstanceOf`, `ärFlygkunnig` och `size`.

¹www.klangfix.se/ordlista.htm

```

1 scala> val fyle =
2     Vector.fill(42)(if (math.random > 0.5) new Kråga else new Ånka)
3 scala> fyle.foreach(_.väsnas)
4 scala> val antalFlygånkor: Int = ???

```

```

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

```

b) Om alla de fåglar som ornitologen hittade skulle väsnas exakt en gång var, hur många krax och hur många kvack skulle då höras? Använd metoderna `filter` och `size`, samt predikatet `ärSimkunnig` för att beräkna antalet krax respektive kvack.

```

1 scala> val antalKrax: Int = ???
2 scala> val antalKvack: Int = ???

```

```

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

```

Uppgift 5. Finala klasser. Om man vill förhindra att man kan göra `extend` på en klass kan man göra den `final` genom att placera nyckelordet **final** före nyckelordet **class**.

a) Eftersom klassificeringen av fåglar i uppgiften ovan i antingen Ånkor eller Krågor är fullständig och det inte finns några subtyper till varken Ånkor eller Krågor är det lämpligt att göra dessa finala. Ändra detta i din kod.

b) Testa att ändå försöka göra en subklass `Simkråga` **extends** `Kråga`. Vad ger kompilatorn för felmeddelande om man försöker utvidga en final klass?

Uppgift 6. Accessregler vid arv och nyckelordet `protected`. Om en medlem i en supertyp är privat så kan man inte komma åt den i en subklass. Ibland vill man att subklassen ska kunna komma åt en medlem även om den ska vara otillgänglig i annan kod.

```

1 trait Super {
2     private val minHemlis = 42
3     protected val vårHemlis = 42
4 }
5 class Sub extends Super {
6     def avslöja = minHemlis
7     def krypisk = vårHemlis * math.Pi
8 }

```

a) Vad blir felmeddelandet när klassen `Sub` försöker accessa `minHemlis`?

b) Deklarera `Sub` på nytt, men nu utan den förbjudna metoden `avslöja`. Vad blir felmeddelandet om du försöker komma åt `vårHemlis` via en instans av klassen `Sub`? Prova till exempel med `new Sub.vårHemlis`

c) Fungerar det att anropa metoden `kryptisk` på instanser av klassen `Sub`?

Uppgift 7. *Användning av `protected`.* Den flitige ornitologen från uppgift 4 ska ringmärka alla 42 fåglar hen hittat i skogen. När hen ändå håller på bestämmer hen att i även försöka ta reda på hur mycket oväsen som skapas av respektive fågelsort. För detta ändamål apterar den flitige ornitologen en linuxdator på allt infångat fyle. Du ska hjälpa ornitologen att skriva programmet.

a) Inför en `protected var` `räknaLäte` i traiten `Fyle` och skriv kod på lämpliga ställen för att räkna hur många läten som respektive fågelinstans yttrar.

b) Inför en metod `antalLäten` som returnerar antalet krax respektive kvack som en viss fågel yttrat sedan dess skapelse.



c) Varför inte använda `private` i stället för `protected`?



d) Varför är det bra att göra räknar-variabeln oåtkomlig från "utsidan"?

Uppgift 8. *Typtester med `isInstanceOf` och typkonvertering med `asInstanceOf`.* Gör nedan deklARATIONER.

```
1 scala> trait A; trait B extends A; class C extends B; class D extends B
2 scala> val (c, d) = (new C, new D)
3 scala> val a: A = c
4 scala> val b: B = d
```

a) Rita en bild över vilka typer som ärver vilka.

b) Vilket resultat ger dessa typtester? Varför?

```
1 scala> c.isInstanceOf[C]
2 scala> c.isInstanceOf[D]
3 scala> d.isInstanceOf[B]
4 scala> c.isInstanceOf[A]
5 scala> b.isInstanceOf[A]
6 scala> b.isInstanceOf[D]
7 scala> a.isInstanceOf[B]
8 scala> c.isInstanceOf[AnyRef]
9 scala> c.isInstanceOf[Any]
10 scala> c.isInstanceOf[AnyVal]
11 scala> c.isInstanceOf[Object]
12 scala> 42.isInstanceOf[Object]
13 scala> 42.isInstanceOf[Any]
```

c) Vilka av dessa typkonverteringar ger felmeddelande? Vilket och varför?

```
1 scala> c.asInstanceOf[B]
2 scala> c.asInstanceOf[A]
3 scala> d.asInstanceOf[C]
4 scala> a.asInstanceOf[B]
5 scala> a.asInstanceOf[C]
6 scala> a.asInstanceOf[D]
7 scala> a.asInstanceOf[E]
8 scala> b.asInstanceOf[A]
```

Uppgift 9. Regler för *override*, *private* och *final*.

a) Undersök överskuggning av abstrakta, konkreta, privata och finala medlemmar genom att skriva in raderna nedan en i taget i REPL. Vilka rader ger felmeddelande? Varför? Vid felmeddelande: notera hur felmeddelandet lyder och ändra deklarationen av den felande medlemmen så att koden blir kompilerbar (eller om det är enda rimliga lösningen: ta bort den felaktiga medlemmen), innan du provar efterkommande rad.

```

1 trait Super1 { def a: Int; def b = 42; private def c = "hemlis" }
2 class Sub2 extends Super1 { def a = 43; def b = 43; def c = 43 }
3 class Sub3 extends Super1 { def a = 43; override def b = 43 }
4 class Sub4 extends Super1 { def a = 43; override def c = "43" }
5 trait Super5 { final def a: Int; final def b = 42 }
6 class Sub6 extends Super5 { override def a = 43; def b = 43 }
7 class Sub7 extends Super5 { def a = 43; override def b = 43 }
8 class Sub8 extends Super5 { def a = 43; override def c = "43" }
9 trait Super9 { val a: Int; val b = 42; lazy val c: String = "lazy" }
10 class Sub10 extends Super9 { override def a = 43; override val b = 43 }
11 class Sub11 extends Super9 { val a = 43; override lazy val b = 43 }
12 class Sub12 extends Super9 { val a = 43; override var b = 43 }
13 class Sub13 extends Super9 { val a = 43; override lazy val c = "still lazy" }
14 class SubSub extends Sub13 { override val a = 44 }
15 trait Super14 { var a: Int; var b = 42; var c: String }
16 class Sub15 extends Super14 { def a = 43; override var b = 43; val c = "?" }

```

b) Skapa instanser av klasserna Sub3, Sub13 och SubSub från ovan deluppgift och undersök alla medlemmarnas värden för respektive instans. Förklara varför de har dessa värden.

c) Läs igenom reglerna i kapitel ?? om vad som gäller vid arv och överskuggning av medlemmar. Gör några egna undersökningar i REPL som försöker bryta mot någon regel som inte testades i deluppgift a.

Uppgift 10. *Supertyp med parameter.* En trait kan inte ha någon parameter. Vill man ha en parameter till supertypen måste man använda en klass istället, enligt nedan exempel.

Utbildningsdepartementet vill i sitt system hålla koll på vissa personer och skapar därför en klasshierarki enligt nedan. Skriv in koden i en editor och klipp sedan in den i REPL.

```

class Person(namn: String)


class Akademiker(
  namn: String,
  universitet: String) extends Person(namn)

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

```



```
class Forskare(
  namn: String,
  universitet: String,
  titel: String) extends Akademiker(namn, universitet)
```

- a) Deklara 4 olika **val**-värden med lämpliga namn som refererar till olika instanser av alla olika klasser ovan och ge attributen valfria initialvärden genom olika parametrar till konstruktörerna.
- b) Skriv två satser: en som först stoppar in instanserna i en Vector och en som sedan loopar igenom vektorn och skriv ut alla instansers toString och namn.
- c) Utbildningsdepartementet bestämmer sig för att bara hålla koll på studenter och forskare och man vill då att det inte ska gå att instansiera objekt av typerna Person och Forskare. Det kan åstadkommas genom att placera nyckelordet **abstract** före **class**. Uppdatera koden i enlighet med detta. Vilket blir felmeddelande om man försöker instansiera en **abstract class**?
- d) Utbildningsdepartementets utvecklare vill slippa implementera toString och slippa skriva **new** vid instansiering och gör därför om typerna Student och Forskare till case-klasser. Skapa instanser av case-klasserna Student och Forskare och skriv ut deras toString. Hur ser utskriften ut?
- e) Eftersom utbildningsdepartementet numera deklarerar Personer och Akademiker som abstrakta, väljer man att göra om dessa typer till traits istället för klasser. Man inför också en case-klass IckeAkademiker som man tänker använda i olika statistiska medborgarundersökningar. Dessutom förser man alla personer med ett personnummer representerat som en Int. Hur ser utvecklarnas kod ut? Skriv ett testprogram som skapar några instanser och skriver ut deras attribut.
-  f) I vilka sammanhang lämpligt eller nödvändigt att använda en **trait** respektive en **class**.

Uppgift 11. *Uppräknade värden.* Ett sätt att hålla reda på uppräknade värden, t.ex. färgen på olika kort i en kortlek, är att använda olika heltal som får representera de olika värdena, till exempel så här:²

```
object Färg {
  val Spader = 1
  val Hjärter = 2
  val Ruter = 3
  val Klöver = 4
}

case class Kort(färg: Int, valör: Int)
```

²Om namnkonventioner för konstanter i Scala: läs under rubriken "Constants, Values, Variable and Methods" här docs.scala-lang.org/style/naming-conventions.html

Man kan hålla reda på färgen med en variabel av typen `Int` och tilldela den en viss färg med ovan konstanter. Och när man skapar ett kort behöver man inte komma ihåg vilket numret är.

```
1 scala> val f = Färg.Spader
2 scala> import Färg._
3 scala> Kort(Ruter, 7)
```

En annan fördelen med detta är att man lätt kan loopa från 1 till 4 för att gå igenom alla färger.

```
1 scala> val kortlek = for (f <- 1 to 4; v <- 1 to 13) yield Kort(f, v)
```

Nackdelen är att kompilatorn vid kompileringstid inte kollar om variablerna av misstag råkar ges något värde utanför det giltiga intervallet, t.ex. 42. Detta får vi själv hålla koll på vid körtid, till exempel med funktionen `require` eller `if`-satser, etc.

Istället kan man använda case-objekt enligt nedan deluppgifter och få hjälp av kompilatorn att hitta eventuella fel vid kompileringstid. Ett case-objekt är som ett vanligt singleton-objekt men det får automatiskt en `toString` samma som namnet och kan användas i matchningar etc. (mer om `match` i kapitel 8).

a) Deklarera följande uppräknade värden som singleton objekt med gemensam bastyp i en editor och klistra in i REPL med kommandot `:paste`. Skapa därefter några exemplinstanser av klassen. `Kort`.

```
trait Färg
case object Spader extends Färg
case object Hjärter extends Färg
case object Ruter extends Färg
case object Klöver extends Färg

case class Kort(färg: Färg, valör: Int)
```

b) Om man vill kunna iterera över alla värden är det bra om de finns i en samling med alla värden. Vi kan lägga en sådan i ett kompanjonsobjekt till bastypen. Uppdatera koden enligt nedan och klistra in på nytt i REPL med kommandot `:paste`. Skriv ut alla färgvärden med en `for`-sats.

```
trait Färg
object Färg {
  val values = Vector(Spader, Hjärter, Ruter, Klöver)
}
case object Spader extends Färg
case object Hjärter extends Färg
case object Ruter extends Färg
case object Klöver extends Färg

case class Kort(färg: Färg, valör: Int)
```

Skapa en kortlek med 52 olika kort och blanda den sedan med `Random.shuffle` enligt nedan. Använd en dubbel **for**-sats och **yield**.

```
1 scala> val kortlek: Vector[Kort] = ???
2 scala> val blandad = scala.util.Random.shuffle(kortlek)
```

```
// kod till facit
for (f <- Färg.values; v <- 1 to 14) yield Kort(f,v)
```

c) Skriv en funktion **def** `blandadKortlek: Vector[Kort] = ???` som ger en ny blandad kortlek varje gång den anropas med metoden i föregående uppgift.

```
// kod till facit
def blandadKortlek: Vector[Kort] = {
  val kortlek =
    for (f <- Färg.values; v <- 1 to 14) yield Kort(f,v)
  scala.util.Random.shuffle(kortlek)
}
```

d) Om man även vill ha en heltalsrepresentation med en metod `toInt` för alla värden, kan man ge basstypen en parameter och i stället för en trait (som inte kan ha några parametrar) använda en abstrakt klass.

```
abstract class Färg(val toInt: Int)
object Färg {
  val values = Vector(Spader, Hjärter, Ruter, Klöver)
}
case object Spader extends Färg(0)
case object Hjärter extends Färg(1)
case object Ruter extends Färg(2)
case object Klöver extends Färg(3)

case class Kort(färg: Färg, valör: Int)

def blandadKortlek: Vector[Kort] = ???
```

Skapa en funktion `färgPoäng` som räknar ut summan av heltalsrepresentationen av alla färger hos en vektor med kort, och använd den sedan för att räkna ut `färgPoäng` för de första fem korten.

```
1 scala> def färgPoäng(xs: Vector[Kort]): Int = ???
2 scala> färgPoäng(blandadKortlek.take(5))
```

```
// kod till facit
def färgPoäng(xs: Vector[Kort]): Int = xs.map(_.färg.toInt).sum
```

7.6.2 Extrauppgifter: öva mer på grunderna

Uppgift 12. Polynom ??? Hur kommer undantag in här?

```
//kod till facit TODO: INTE ALLS FÄRDIGT

sealed trait Term {
  def const: BigDecimal
  protected def silentConst: String =
    if (const == BigDecimal(1)) "" else const.toString
  def *(that: Term): Term
  def ^(e: Int): Term
}

case class Const(const: BigDecimal) extends Term {
  override def toString = const.toString
  def *(that: Term): Term = that match {
    case Const(c)      => Const(c * const)
    case Var(c, n, e) => Var(c * const, n, e)
    case Prod(c, vs)  => ???
  }
}

case class Var(const: BigDecimal, name: Char, exp: BigInt) extends Term {
  private def silentExp: String =
    if (exp == BigInt(1)) "" else "^"+exp.toString
  def ^(e: Int) = Var(const.pow(e), name, e * exp)
  def *(c: BigDecimal) = Var(const * c, name, exp)
  def /(c: BigDecimal) = Var(const / c, name, exp)
  def constUnit: Var = Var(1.0, name, exp)
  def *(that: Var): Term = {
    Prod(const * that.const, Vector(this.constUnit, that.constUnit))
  }
  override def toString = s"$silentConst$name$silentExp"
}

object Var{
  def apply(name: Char) = new Var(1, name, 1)
  def apply(const: Double, name: Char) = new Var(const, name, 1)
}

case class Prod(const: BigDecimal, vars: Vector[Var]) extends Term {
  override def toString = silentConst + vars.mkString
}

case class Poly(xs: Vector[Term]) { // etc etc
}
```

7.6.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 13. TODO: Gör en fördjupningsuppgift inspirerat av denna <http://stackoverflow.com/questions/16173477/usages-of-null-nothing-unit-in-scala>

Uppgift 14. TODO: Gör en fördjupningsuppgift om extraherare (eng. *extractors*) och implementera unapply.

Uppgift 15. Pynomdivision ???

Uppgift 16. Numbers TODO: Fundera på om detta är lönt eller blir exemplet för krångligt ??? Vad har detta med **match** och undantag att göra ??? Hitta på ngt bättre...

```
// INTE FÄRDIGT
package numbers

trait Number {
  def reduce: Number = this // reduce if possible to a simpler number
  def isZero: Boolean
  def isOne: Boolean
  def +(that: Number): Number = ??? // should be abstract
}

object Number {
  def Zero = Nat(0)
  def One = Nat(1)
  def Im(im: BigDecimal) = Complex(Real(0), Real(im))
  def Im(im: Real) = Complex(Real(0), im)
  def j = Complex(Real(0), Real(1))
  def Re(re: BigDecimal) = Complex(Real(re), Real(0))
  def Re(re: Real) = Complex(re, Real(0))
  implicit class IntDecorator(i: Int){ def j = Im(i) }
  implicit class DoubleDecorator(d: Double){ def j = Im(d) }
}

trait AbstractComplex extends Number {
  def re: AbstractReal
  def im: AbstractReal
  def abs = Real(math.hypot(re.decimal.toDouble, im.decimal.toDouble))
  def fi = Real(math.atan2(im.decimal.toDouble, re.decimal.toDouble))
  override def isZero: Boolean = re.decimal == 0 && im.decimal == 0
  override def isOne: Boolean = abs.decimal == 1.0
  override def reduce: AbstractComplex = if (im.decimal == 0) re.reduce else this
}

case class Complex(re: Real, im: Real) extends AbstractComplex
case object Complex {
  def apply(re: BigDecimal, im: BigDecimal) = new Complex(Real(re), Real(im))
}

case class Polar(override val abs: Real, override val fi: Real) extends AbstractComplex {
  override def re = Real(abs.decimal.toDouble * math.cos(fi.decimal.toDouble))
  override def im = Real(abs.decimal.toDouble * math.sin(fi.decimal.toDouble))
}
case object Polar {
  def apply(abs: BigDecimal, fi: BigDecimal) = new Polar(Real(abs), Real(fi))
}

trait AbstractReal extends AbstractComplex {
  def decimal: BigDecimal
  override def isZero = decimal == 0
  override def isOne = decimal == 1
  override def re = this
}
```

```

    override def im = Number.Zero
    override def reduce: AbstractReal =
      if (decimal == 0) Number.Zero else if (decimal == 1) Number.One else this
  }

case class Real(decimal: BigDecimal) extends AbstractReal

trait AbstractRational extends AbstractReal {
  def numerator: AbstractInteger
  def denominator: AbstractInteger
  override def decimal = BigDecimal(numerator.integ)
  override def isOne = numerator.integ == denominator.integ
  override def reduce: AbstractRational =
    if (denominator.isOne) numerator.reduce else this // should use gcd
}

case class Frac(numerator: Integ, denominator: Integ) extends AbstractRational {
  require(denominator.integ != 0, "denominator must be non-zero")
}

case object Frac {
  def apply(n: BigInt, d: BigInt) = new Frac(Integ(n), Integ(d))
}

trait AbstractInteger extends AbstractRational {
  def integ: BigInt
  override def numerator = this
  override def denominator = Number.One
  override def isZero = integ == 0
  override def isOne = integ == 1
  override def decimal: BigDecimal = BigDecimal(integ)
  override def reduce: AbstractInteger =
    if (isZero) Number.Zero else if (isOne) Number.One else this
}

case class Integ(integ: BigInt) extends AbstractInteger

trait AbstractNatural extends AbstractInteger

case class Nat(integ: BigInt) extends AbstractNatural {
  require(integ >= 0, "natural numnbers must be non-negative")
}

```

7.7 Laboration: turtlerace-team

Mål

- ☐ Kunna arv
- ☐ Kunna traits

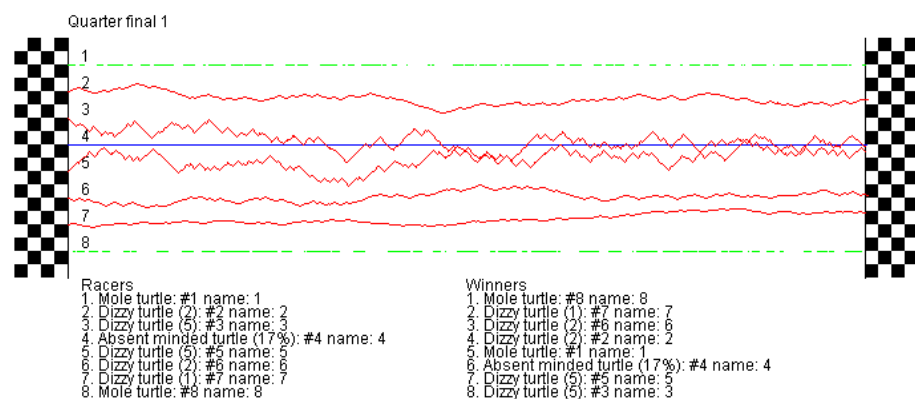
Förberedelser

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

7.7.1 Obligatoriska uppgifter

Uppgift 1. RaceWindow

- a) Skapa RaceWindow som ärver från SimpleWindow med fix storlek och förbestämmd titel. Lämplig storlek är 800x400. Det ska finnas två attribut startX och endX som är förbestämda och motsvarar start och mål.
- b) Skapa tre metoder startX, endX och startY. startX och endX ska retunera värdet på motsvarande attribut. startY ska ta in ett heltal nbr som är ett startnummer och retunera y-värdet för det startnumret.
- c) Skapa en metod draw som ritar upp ett race i fönstret och skriver ut startnummer. Definiera gärna en klass som ritar upp ett start-/målfält. Bilden är enbart ett exempel på vad man kan göra. Rita något kul! Titeln och alla tävlande turtles kommer att skrivas ut senare i uppgiften.



- d) Skapa en metod printTitle som tar in en sträng och skriver ut den som titel på racet.
- e) Skapa en metod printRacers som tar in en lista med RaceTurtle, ett x-värde och en titel på listan och skriver ut listan med början på angivet x-värde.

Se exemplet med listan *Racers* och *Winners*.

Uppgift 2. RaceTurtle.

- a) Högerklicka på projektet `w07_turtlerace_team` och välj Build Path → Configure Build Path. Välj fliken Projects och tryck Add... Markera projektet `w06_turtlegraphics` och tryck OK.
- b) Implementera klassen `RaceTurtle` som ska ärva från `Turtle`. `Turtle` och `Point` behöver importeras från `turtlegraphics`. Startpositionen för en `RaceTurtle` hämtas från `RaceWindow`.

Uppgift 3. TurtleRace

- a) Implementera metoden `race` som ska börja med att skriva ut tävlande och titel i `RaceWindow`. Skapa en tom `ArrayBuffer` med namnet `winners` (för detta behöver man importera `scala.collection.mutable.ArrayBuffer`). Låt varje `RaceTurtle` ta ett steg tills en passerar mållinjen. Lägg över alla som passerat mållinjen i `winners` och kör detta tills alla `RaceTurtle` har hamnat i `winners`. Använd `SimpleWindow.delay(5)` mellan varje steg för att se animeringen. Skriv ut vinnarna i `RaceWindow` och vänta på musklick. Retunera listan med vinnare.

Uppgift 4. ColorTurtle.

- a) Skapa en ny klass `ColorTurtle` som ärver från `RaceTurtle` och tar in en extra parameter `color` av typen `java.awt.Color`.
- b) Metoden `raceStep` är det enda som skiljer klasserna. Börja med att spara undan nuvarande färg i `RaceWindow` och byt till den angivna färgen `color`. Anropa `raceStep` i superklassen och ändra sedan tillbaka färgen i `RaceWindow`.

Uppgift 5. Dizziness, AbsentMindness och Mole

- a) Implementera tre **trait** som ärver från `RaceTurtle` och som **override** `raceStep` och `toString`. `toString` ska utöver `toString` från `RaceTurtle` även bestå av vilken typ av `RaceTurtle` det är och graden av yrsel/tankspriddhet den har.
 - **Dizziness**. Slumpa ett heltal `dizziness` mellan 1 och 5. För varje steg ska en riktningsförändring slumpas fram som blir större desto större `dizziness` är. Slumpa även om den avviker åt höger eller vänster och använd `turnRight` och `turnLeft`.
 - **AbsentMindness**. Slumpa ett heltal `absent` mellan 0 och 99 som anger i procent hur tankspridd `RaceTurtle` är. För varje steg ska det vara `absent%` chans att ett steg inte tas.
 - **Mole**. Med 50% sannolikhet ska denna typ `RaceTurtle` gräva ner sig i marken. För varje steg ska det vara 50% chans att pennan är uppe och 50% chans att pennan är nere. Använd metoderna `penUp` och `penDown`.

Uppgift 6. TurtleTournament

- a) Börja med att skapa en hjälpmetod `randTurtle` som tar in ett `RaceWindow`, ett nummer och ett namn som parameter. Slumpa med lika stor sannolikhet mellan att skapa en `ColorTurtle` med en av de tre olika egenskaperna och låt de olika egenskaperna ha olika färger.
- b) Skapa ett `RaceWindow`. Slumpa fram 32 sköldpaddor och låt dem utföra fyra `TurtleRace`. Ta vara på vinnarna och låt de fyra bästa från varje lopp köra två lopp till. De fyra bästa från båda dessa loppen går vidare till finalen. **Tänk på att rensa `RaceWindow` efter varje lopp och rita ut det på nytt innan varje lopp.**

7.7.2 Frivilliga extrauppgifter**Uppgift 7.** En labbuppgiftsbeskrivning.

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

Kapitel 8

Mönster, Undantag

Koncept du ska lära dig denna vecka:

- ☐ mönstermatchning
- ☐ match
- ☐ Option
- ☐ try
- ☐ catch
- ☐ Try
- ☐ unapply
- ☐ sealed

- ☐ flatten
- ☐ flatMap
- ☐ partiella funktioner
- ☐ collect
- ☐ implementera equals utan arv för Complex
- ☐ implementera equals med arv för Shape

8.1 TODO: Begrepp att förklara

Tänk igenom ordningen:

- java switch, scala match ...

8.2 Javas switch-sats

A switch in Java works with the byte, short, char, and int primitive data types. It also works with enumerated types (discussed in Enum Types), the String class, and a few special classes that wrap certain primitive types: Character, Byte, Short, and Integer

8.3 Javas switch-sats

```
public class Switch {
    public static void main(String[] args) {
        String favorite = "selleri";
        if (args.length > 0) {
            favorite = args[0];
        }
        System.out.println("Din favoritgrönsak: " + favorite);
        char firstChar = Character.toLowerCase(favorite.charAt(0));
        System.out.print("Jag tycker ");
        switch (firstChar) {
            case 'g':
                System.out.println("gurka är gott!");
                break;
            case 't':
                System.out.println("tomat är gott!");
                break;
            case 'b':
                System.out.println("brocolli är gott!");
                break;
            default:
                System.out.println(favorite + " är äckligt!");
                break;
        }
    }
}
```

8.4 TODO: Begrepp att förklara

Tänk igenom ordningen:

- java switch, scala match ...

8.5 Undantag

```
try  
catch
```

8.6 Övning: matching

Mål



Förberedelser



8.6.1 Grunduppgifter

Uppgift 1. Hur funkar en **switch**-sats i Java (och flera andra språk)? Det händer ofta att man vill testa om ett värde är ett av många olika alternativ. Då kan man använda en sekvens av många **if-else**, ett för varje alternativ. Men det finns ett annat sätt i Java och många andra språk: man kan använda **switch** som kollar flera alternativ i en och samma sats, se t.ex. en.wikipedia.org/wiki/Switch_statement.

a) Skriv in nedan kod i en kodeditor. Spara med namnet `Switch.java` och kompilera filen med kommandot `javac Switch.java`. Kör den med `java Switch` och ange din favoritgrönsak som argumentet till programmet. Vad händer? Förklara hur **switch**-satsen fungerar.

```
1 public class Switch {
2     public static void main(String[] args) {
3         String favorite = "selleri";
4         if (args.length > 0) {
5             favorite = args[0];
6         }
7         System.out.println("Din favoritgrönsak: " + favorite);
8         char firstChar = Character.toLowerCase(favorite.charAt(0));
9         System.out.print("Jag tycker ");
10        switch (firstChar) {
11            case 'g':
12                System.out.println("gurka är gott!");
13                break;
14            case 't':
15                System.out.println("tomat är gott!");
16                break;
17            case 'b':
18                System.out.println("brocolli är gott!");
19                break;
20            default:
21                System.out.println(favorite + " är äckligt!");
22                break;
23        }
24    }
25 }
```

b) Vad händer om du tar bort **break**-satsen på rad 16?

Uppgift 2. Matcha på konstanta värden. I Scala finns ingen **switch**-sats. I stället har Scala ett **match**-uttryck som är mer kraftfullt. Dock saknar Scala nyckelordet **break** och Scalas **match**-uttryck kan inte "falla igenom" som skedde i uppgift 1b.

a) Skriv nedan program med en keditor och spara i filen `Match.scala`. Kompilera med `scalac Match.scala`. Kör med `scala Match` och ge som argument din favoritgrönsak. Vad händer? Förklara hur ett **match**-uttryck fungerar.

```

1 object Match {
2   def main(args: Array[String]): Unit = {
3     val favorite = if (args.length > 0) args(0) else "seleri"
4     println("Din favoritgrönsak: " + favorite)
5     val firstChar = favorite.toLowerCase.charAt(0)
6     val meThink = firstChar match {
7       case 'g' => "gurka är gott!"
8       case 't' => "tomat är gott!"
9       case 'b' => "brocolli är gott!"
10      case _   => favorite + " är äckligt!"
11    }
12    println("Jag tycker " + meThink)
13  }
14 }

```

b) Vad blir det för felmeddelande om du tar bort case-grenen för defaultvärdet och indata väljs så att inga case-grenar matchar? Är det ett körtidsfel eller ett kompileringsfel?



c) Beskriv några skillnader i syntax och semantik mellan Javas flervalssats **switch** och Scalas flervalsuttryck **match**.

Uppgift 3. Gard i case-grenar. Med hjälp en gard (eng. *guard*) i en case-gren kan man begränsa med ett villkor om grenen ska väljas.

Utgå från koden i uppgift 2a och byt ut case-grenen för 'g'-matchning till nedan variant med en gard med nyckelordet **if** (notera att det inte behövs parenteser runt villkoret):

```
case 'g' if math.random > 0.5 => "gurka är gott ibland..."
```

Kompilera om och kör programmet upprepade gånger med olika indata tills alla grenar i **match**-uttrycket har exekverats. Förklara vad som händer.

Uppgift 4. Mönstermatcha på attributen i case-klasser. Scalas **match**-uttryck är extra kraftfulla om de används tillsammans med **case**-klasser: då kan attribut extraheras automatiskt och bindas till lokala variabler direkt i case-grenen som nedan exempel visar (notera att `v` och `rutten` inte behöver deklarerats explicit). Detta kallas för **mönstermatchning**.

a) Vad skrivs ut nedan? Varför? Prova att byta namn på `v` och `rutten`.

```

1 scala> case class Gurka(vikt: Int, ärRutten: Boolean)
2 scala> val g = Gurka(100, true)
3 scala> g match { case Gurka(v,rutten) => println("G" + v + rutten) }

```

b) Skriv sedan nedan i REPL och tryck TAB två gånger efter punkten. Vad har unapply-metoden för resultattyp?

```
1 scala> Gurka.unapply // Tryck TAB två gånger
```

Bakgrund för kännedom: Case-klasser får av kompilatorn automatiskt ett kompanjonsobjekt (eng. *companion object*), i detta fallet **object** Gurka. Det objektet får av kompilatorn automatiskt en unapply-metod. Det är unapply som anropas ”under huven” när case-klassernas attribut extraheras vid mönstermatchning, men detta sker alltså automatiskt och man behöver inte explicit nyttja unapply om man inte själv vill implementera s.k. extraherare (eng. *extractors*); om du är nyfiken på detta, se fördjupningsuppgift 17.

c) Anropa unapply-metoden enligt nedan. Vad blir resultatet?

```
1 scala> Gurka.unapply(g)
```

Vi ska i senare uppgifter undersöka hur typerna Option och Some fungerar och hur man kan ha nytta av dessa i andra sammanhang.

d) Spara programmet nedan i filen vegomatch.scala och kompilera med scalac vegomatch.scala och kör med scala vegomatch.Main 1000 i terminalen. Förklara hur predikatet ärÄtvärd fungerar.

```
1 package vegomatch
2
3 trait Grönsak {
4   def vikt: Int
5   def ärRutten: Boolean
6 }
7
8 case class Gurka(vikt: Int, ärRutten: Boolean) extends Grönsak
9 case class Tomat(vikt: Int, ärRutten: Boolean) extends Grönsak
10
11 object Main {
12   def slumpvikt: Int = (math.random*500 + 100).toInt
13   def slumprutten: Boolean = math.random > 0.8
14   def slumpgurka: Gurka = Gurka(slumpvikt, slumprutten)
15   def slumptomat: Tomat = Tomat(slumpvikt, slumprutten)
16   def slumpgrönsak: Grönsak =
17     if (math.random > 0.2) slumpgurka else slumptomat
18
19   def ärÄtvärd(g: Grönsak): Boolean = g match {
20     case Gurka(v, rutten) if v > 100 && !rutten => true
21     case Tomat(v, rutten) if v > 50 && !rutten => true
22     case _ => false
23   }
24
25   def main(args: Array[String]): Unit = {
26     val skörd = Vector.fill(args(0).toInt)(slumpgrönsak)
27     val ätvärda = skörd.filter(ärÄtvärd)
28     println("Antal skördade grönsaker: " + skörd.size)
29     println("Antal ätvärda grönsaker: " + ätvärda.size)
30   }
31 }
```


Uppgift 5. Man kan åstadkomma urskiljningen av de ätbara grönsakerna i uppgift 4 med polymorfism i stället för **match**.

a) Gör en ny variant av ditt program enligt nedan riktlinjer och spara den modifierade koden i filen `vegopoly.scala` och kompilera och kör.

- Ta bort predikatet `ärÄtvärd` i objektet `Main` och inför i stället en abstrakt metod **def** `ärÄtbar`: `Boolean` i traiten `Grönsak`.
- Inför konkreta **val**-medlemmar i respektive grönsak som definierar ätbarheten.
- Ändra i huvudprogrammet i enlighet med ovan ändringar så att `ärÄtvärd` anropas som en metod på de skördade grönsaksobjekten när de ätvärda ska filtreras ut.

b) Lägg till en ny grönsak **case class** `Brocolli` och definiera dess ätbarhet. Ändra i slump-funktionerna så att `brocolli` blir ännu ovanligare än `gurka`.



c) Jämför lösningen med **match** i uppgift 4 och lösningen ovan med polymorfism. Vilka är för- och nackdelarna med respektive lösning. Diskutera två olika situationer på ett hypotetiskt företag som utvecklar mjukvara för jordbrukssektorn: 1) att uppsättningen grönsaker inte ändras särskilt ofta medan definitionerna av ätbarhet ändras väldigt ofta och 2) att uppsättningen grönsaker ändras väldigt ofta men att ätbarhetsdefinitionerna inte ändras särskilt ofta.

Uppgift 6. Matcha på case-objekt och nyttan med **sealed**. Skapa nedan kod i en editor, och klistra in i REPL med kommandot `:pa`. Notera nyckelordet **sealed**.

```
sealed trait Färg
object Färg {
  val values = Vector(Spader, Hjärter, Ruter, Klöver)
}
case object Spader extends Färg
case object Hjärter extends Färg
case object Ruter extends Färg
case object Klöver extends Färg
```

a) Skapa en funktion **def** `parafärg(f: Färg): Färg` i en editor, som med hjälp av ett `match`-uttryck returnerar parallellfärgen till en färg. Parallellfärgen till `Hjärter` är `Ruter` och vice versa, medan parallellfärgen till `Klöver` är `Spader` och vice versa. Klistra in funktionen i REPL.

```
1 scala> parafärg(Spader)
2 scala> val xs = Vector.fill(5)(Färg.values((math.random * 4).toInt))
3 scala> xs.map(parafärg)
```

```
// kod till facit
def parafärg(f: Färg): Färg = f match {
  case Spader => Klöver
```


```

case Hjärter => Ruter
case Ruter   => Hjärter
case Klöver  => Spader
}

```

b) Vi ska nu undersöka vad som händer om man glömmer en av case-grenarna i matchningen i parafärg? ”Glöm” alltså avsiktligt en av case-grenarna och klistra in den nya parafärg med den ofullständiga matchningen. Hur lyder varningen? Kommer varningen vid körtid eller vid kompilering?

c) Anropa parafärg med den ”glömda” färgen. Hur lyder felmeddelandet? Är det ett kompileringsfel eller ett körtidsfel?

d) Förklara vad nyckelordet **sealed** innebär och vilken nytta man kan ha av att **försegla** en supertyp. 

Uppgift 7. Betydelsen av små och stora begynnelsebokstaver vid matchning.

För att åstadkomma att namn kan bindas till variabler vid matchning utan att de behöver deklarerats i förväg (som vi såg i uppgift 4a) så har identifierare med liten begynnelsebokstav fått speciell betydelse: den tolkas av kompilatorn som att du vill att en variabel binds till ett värde vid matchningen. En identifierare med stor begynnelsebokstav tolkas däremot som ett konstant värde (t.ex. ett case-objekt eller ett case-klass-mönster).

a) *En case-gren som fångar allt.* En case-gren med en identifierare med liten begynnelsebokstav som saknar gard kommer att matcha allt. Prova nedan i REPL, men försök lista ut i förväg vad som kommer att hända. Vad händer?

```

1 scala> val x = "urka"
2 scala> x match {
3     case str if str.startsWith("g") => println("kanske gurka")
4     case vadsomhelst => println("ej gurka: " + vadsomhelst)
5 }
6 scala> val g = "gurka"
7 scala> g match {
8     case str if str.startsWith("g") => println("kanske gurka")
9     case vadsomhelst => println("ej gurka: " + vadsomhelst)
10 }

```

b) *Fallgrop med små begynnelsebokstäver.* Innan du provar nedan i REPL, försök gissa vad som kommer att hända. Vad händer? Hur lyder varningarna och vad innebär de?

```

1 scala> val any: Any = "gurka"
2 scala> case object Gurka
3 scala> case object tomat
4 scala> g match {
5     case Gurka => println("gurka")
6     case tomat => println("tomat")
7     case _ => println("allt annat")
8 }

```

c) Använd *backticks* för att tvinga fram *match* på konstant värde. Det finns en utväg om man inte vill att kompilatorn ska skapa en ny lokal variabel: använd specialtecknet *backtick*, som skrivs ``` och kräver speciella tangentbordsstryck.¹ Gör om föregående uppgift men omgärda nu identifieraren *tomat* i *tomat-case*-grenen med *backticks*, så här: `case `tomat` => ...`

Uppgift 8. Använda *Option* och *matcha* på värden som kanske saknas. Man behöver ofta skriva kod för att hantera värden som eventuellt saknas, t.ex. saknade telefonnummer i en persondatabas. Denna situation är så pass vanlig att många språk har speciellt stöd för saknade värden.

I Java² används värdet `null` för att indikera att en referens saknar värde. Man får då komma ihåg att testa om värdet saknas varje gång sådana värden ska behandlas, t.ex. med `if (ref != null) { ... } else { ... }`. Ett annat vanligt trick är att låta `-1` indikera saknade positiva heltal, till exempel saknade index, som får behandlas med `if (i != -1) { ... } else { ... }`.

I Scala finns en speciell typ *Option* som möjliggör smidig och typsäker hantering av saknade värden. Om ett kanske saknat värde packas in i en *Option* (eng. *wrapped in an Option*), finns det i en speciell slags samling som bara kan innehålla *inget* eller *något* värde, och alltså har antingen storleken `0` eller `1`.

a) Förklara vad som händer nedan.

```
1 scala> var kanske: Option[Int] = None
2 scala> kanske.size
3 scala> kanske = Some(42)
4 scala> kanske.size
5 scala> kanske.isEmpty
6 scala> kanske.isDefined
7 scala> def ökaOmFinns(opt: Option[Int]): Option[Int] = opt match {
8     case Some(i) => Some(i + 1)
9     case None    => None
10 }
11 scala> kanske.map(öka)
```

b) Mönstermatchingen ovan är minst lika knölig som en *if*-sats, men tack vare att en *Option* är en slags (liten) samling finns det smidigare sätt. Förklara vad som händer nedan.

```
1 val meningen = Some(42)
2 val ejMeningen = Option.empty[Int]
3 meningen.map(_ + 1)
4 ejMeningen.map(_ + 1)
5 ejMeningen.map(_ + 1).orElse(Some("saknas")).foreach(println)
6 meningen.map(_ + 1).orElse(Some("saknas")).foreach(println)
```

c) *Samlingsmetoder som ger en Option*. Förklara för varje rad nedan vad som händer. En av raderna ger ett felmeddelande; vilken rad och vilket felmeddelande?


¹Fråga någon om du inte hittar hur man gör *backtick* ``` på ditt tangentbord.

²Scala har också `null` men det behövs bara vid samverkan med Java-kod.


```

1 val xs = (42 to 84 by 5).toVector
2 val e = Vector.empty[Int]
3 xs.headOption
4 xs.headOption.get
5 xs.headOption.getOrElse(0)
6 xs.headOption.orElse(Some(0))
7 e.headOption
8 e.headOption.get
9 e.headOption.getOrElse(0)
10 e.headOption.orElse(Some(0))
11 Vector(xs, e, e, e)
12 Vector(xs, e, e, e).map(_._lastOption)
13 Vector(xs, e, e, e).map(_._lastOption).flatten
14 xs.lift(0)
15 xs.lift(1000)
16 e.lift(1000).getOrElse(0)
17 xs.find(_ > 50)
18 xs.find(_ < 42)
19 e.find(_ > 42).foreach(_ => println("HITTAT!"))

```

d) Vilka är fördelarna med Option jämfört med **null** eller -1 om man i sin kod glömmer hantera saknade värden? 

TODO: Till facit: något i stil med: kompileringsfel vs körtidsfel + dokumentation av returvärden i funktioner direkt i kod istf i kommentarer. Vad menas med "typsäker"?

e) Studera dokumentationen för Option här och se om du känner igen några av metoderna som också finns på samlingen Vector: 

www.scala-lang.org/api/current/index.html#scala.Option

Förklara hur metoden contains på en Option fungerar med hjälp av dokumentationens exempel.


Uppgift 9. Kasta undantag. Om man vill signalera att ett fel eller en onormal situation uppstått så kan man **kasta** (eng. *throw*) ett **undantag** (eng. *exception*). Då avbryts programmet direkt med ett felmeddelande, om man inte väljer att **fånga** (eng. *catch*) undantaget.


a) Vad händer nedan?

```

1 scala> throw new Exception("PANG!")
2 scala> java.lang. // Tryck TAB efter punkten
3 scala> throw new IllegalArgumentException("fel fel fel")
4 scala> val carola = try {
5     throw new Exception("stormvind!")
6     42
7 } catch { case e: Throwable => println("Fångad av en " + e); -1 }

```

b) Nämn ett par undantag som finns i paketet `java.lang` som du kan gissa vad de innebär och i vilka situationer de kastas. 

c) Vilken typ har variabeln `carola` ovan? Vad hade typen blivit om `catch`-grenen hade returnerat en sträng i stället? 

Uppgift 10. Fånga undantag i Java med en **try-catch**-sats. Det finns som vi såg i förra uppgiften inbyggt stöd i JVM för att hantera när program avbryts på oväntade sätt, t.ex. på grund av division med noll eller ej förväntade indata från användaren. Skriv in nedan Java-program i en editor och spara i en fil med namnet TryCatch.java och kompilera med javac TryCatch.java i terminalen.

```

1 // TryCatch.java
2
3 public class TryCatch {
4     public static void main(String[] args) {
5         int input;
6         int output;
7         if (args[0].equals("safe")) {
8             try {
9                 input = Integer.parseInt(args[1]);
10                System.out.println("Skyddad division!");
11                output = 42 / input;
12            } catch (Exception e) {
13                System.out.println("Undantag fångat: " + e);
14                System.out.println("Dividerar ändå med säker default!");
15                input = 1;
16                output = 42 / input;
17            }
18        } else {
19            input = Integer.parseInt(args[0]);
20            System.out.println("Oskyddad division!");
21            output = 42 / input;
22        }
23        System.out.println("42 / " + input + " == " + output);
24    }
25 }

```

a) Förklara vad som händer när du kör programmet med olika indata:

```

1 $ java TryCatch 42
2 $ java TryCatch 0
3 $ java TryCatch safe 42
4 $ java TryCatch safe 0
5 $ java TryCatch

```

b) Vad händer om du "glömmer bort" raden 15 och därmed missar att initialisera input? Hur lyder felmeddelandet? Är det ett körtidsfel eller kompileringsfel?



c) Beskriv några skillnader och likheter i syntax och semantik mellan **try-catch** i Java respektive Scala.

Uppgift 11. Fånga undantag i Scala med `scala.util.Try`. I paketet `scala.util` finns typen `Try` med stort T som är som en slags samling som kan innehålla antingen ett "lyckat" eller "misslyckat" värde. Om beräkningen av värdet lyckades och inga undantag kastas blir värdet wrappat i en `Success`, annars blir undantaget wrappat i en `Failure`. Man kan extrahera värdet,

respektive undantaget, med mönstermatchning, men det är oftast smidigare att använda samlingsmetoderna `map` och `foreach`, i likhet med hur `Option` används. Det finns även en smidig metod `recover` på objekt av typen `Try` där man kan skicka med kod som körs om det uppstår en undantagssituation.

a) Förklara vad som händer nedan.

```

1 scala> def pang = throw new Exception("PANG!")
2 scala> import scala.util.{Try, Success, Failure}
3 scala> Try{pang}
4 scala> Try{pang}.recover{case e: Throwable => s"desermerad bomb: $e"}
5 scala> Try{"tyst"}.recover{case e: Throwable => s"desermerad bomb: $e"}
6 scala> def kanskePang = if (math.random > 0.5) "tyst" else pang
7 scala> def kanskeOk = Try{ kanskePang}
8 scala> val xs = Vector.fill(100)(kanskeOk)
9 scala> xs(13) match {
10     case Success(x) => ":"
11     case Failure(e) => ":( " + e
12 }
13 scala> x(13).isSuccess
14 scala> x(13).isFailure
15 scala> xs.count(_.isFailure)
16 scala> xs.find(_.isFailure)
17 scala> val badOpt = xs.find(_.isFailure)
18 scala> val goodOpt = xs.find(_.isSuccess)
19 scala> badOpt
20 scala> badOpt.get
21 scala> badOpt.get.get
22 scala> badOpt.map(_.getOrElse("bomben desermerad!")).get
23 scala> goodOpt.map(_.getOrElse("bomben desermerad!")).get
24 scala> xs.map(_.getOrElse("bomben desermerad!")).foreach(println)
25 scala> xs.map(_.toOption)
26 scala> xs.map(_.toOption).flatten
27 scala> xs.map(_.toOption).flatten.size

```

b) Vad har funktionen `pang` för returtyp?

c) Varför får funktionen `kanskePang` den härledda returtypen `String`?



Uppgift 12. *TODO flatten och flatMap med Option och Try* Ska detta vara fördjupning???

Uppgift 13. *TODO partiella funktioner och metoderna collect och collectFirst på samlingar* Ska detta vara fördjupning???

Uppgift 14. *Metoden equals.* Om man överskuggar den befintliga metoden `equals` så kommer metoden `==` att fungera annorlunda. Man kan då själv åstadkomma innehållslighet i stället för referenslighet. Vi börjar att studera den befintliga `equals` med referenslighet.

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

```

1 scala> class Gurka(val vikt: Int, ärÄtbar: Boolean)
2 scala> val g1 = new Gurka(42, true)

```

```

3 scala> val g2 = g1
4 scala> val g3 = new Gurka(42, true)
5 scala> g1 == g2
6 scala> g1 == g3
7 scala> g1.equals // tryck TAB två gånger

```



b) Rita minnessituationen efter rad 4.

c) *Överskugga metoderna equals och hashCode.*

Bakgrund för kännedom: Det visar sig förvånande komplicerat att implementera innehållslighet med metoden equals så att den ger bra resultat under alla speciella omständigheter. Till exempel måste man även överskugga en metod vid namn hashCode om man överskuggar equals, eftersom dessa båda används gemensamt av effektivitetsskäl för att skapa den interna lagringen av objekten i vissa samlingar. Om man missar det kan objekt bli "osynliga" i hashCode-baserade samlingar – men mer om detta i senare kurser. Om objekten ingår i en öppen arvshierarki blir det också mer komplicerat; det är enklare om man har att göra med finala klasser. Dessutom krävs speciella hänsyn om klassen har en typparameter.

Definera klassen nedan i REPL med överskuggade equals och hashCode; den ärver inte något och är final.

```

// fungerar fint om klassen är final och inte ärver något
final class Gurka(val vikt: Int, ärÄtbar: Boolean) {
  override def equals(other: Any): Boolean = other match {
    case that: Gurka => this.vikt == that.vikt
    case _ => false
  }
  override def hashCode: Int = (vikt, ärÄtbar).## //förklaras sen
}

```

d) Vad händer nedan?

```

1 scala> val g1 = new Gurka(42, true)
2 scala> val g2 = g1
3 scala> val g3 = new Gurka(42, true)
4 scala> g1 == g2
5 scala> g1 == g3

```



e) Hur märker man ovan att den överskuggade equals medför att == nu ger innehållslighet? Jämför minnessituationen efter rad 3 med uppgift a.

f) *Klassen Complex och metoden equals.* Implementera klassen Complex som representerar ett komplext tal med realdel och imaginärdel. I stället för att, som man ofta gör i Scala, använda en case-klass och en equals-metod som automatiskt ger innehållslighet, ska du träna på att implementera en egen equals.

```

class Complex(re: Double, im: Double) extends Number {
  def abs: Double = math.hypot(re, im)
  override def toString = s"Complex($re, $im)"
  override def equals(other: Any): Boolean = ???
}

```

```
    override def hashCode: Int = ???  
  }  
  case object Complex {  
    def apply(re: Double, im: Double): Complex = new Complex(re, im)  
  }
```

Uppgift 15. Implementera en egen, typsäker innehållstest med metoden `==`.
TODO ??? Ska detta vara med ???

8.6.2 Extrauppgifter: öva mer på grunderna

Uppgift 16. Gör motsvarande program i Scala som visas i uppgift 10, men utnyttja att Scalas `try-catch` är ett uttryck. Kompilera och kör och testa så att de ur användarens synvinkel fungerar precis på samma sätt. Notera de viktigaste skillnaderna mellan de båda programmen.

8.6.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 17. Speciella matchningar. `@` och `_*`

Uppgift 18. Skapa din egen extraktor med metoden `unapply`.

Uppgift 19. Överskugga `equals` vid arv. Ska detta vara fördjupning???

```
trait Number  
class Complex(re: Double, im: Double) extends Number {  
  override def equals(other: Any): Boolean = ???  
}  
class Rational(numerator: Int, denominator: Int) extends Number {  
  override def equals(other: Any): Boolean = ???  
}
```


8.7 Laboration: chords - team

Mål

- ☐ Kunna använda mönstermatchning
- ☐ Känna till exceptions
- ☐ Förstå hur Try fungerar

Förberedelser

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

8.7.1 Bakgrund

Inom musik utgår man från en skala med 12 olika toner (C, C#, D, D#, E, F, F#, G, G#, A, A#, B). Nästa ton efter B är C och tillhör nästa oktav. Ett ackord är uppbyggt av ett antal olika toner som spelas tillsammans. Laborationen kommer att utgå från två olika instrument: gitarr och ukulele. Skillnaden för dessa två instrument är antalet strängar och vilken tonart de är stämde i. På båda instrumenten har en grepbräda med ett antal olika band. Ett ackord spelas genom att man med ett finger trycker ner på strängen på band i . Om strängen spelas kommer tonen att vara ett halvt tonsteg högre än om man håller ner strängen på plats $i - 1$.

Laborationen kommer bestå av ett textbaserat användargränssnitt där man kommer ha möjlighet bl.a. att lägga till nya ackord, rita upp ackord och spela ackord. En ton anges på följande format "E2", vilket innebär andra oktaven tonen E. Rekommenderad stämning för gitarr resp. ukulele är: E2, A2, D3, G3, B3, E4 resp. G4, C4, E4, A4. Denna stämning kommer fungera för de fördefinierad ackorden i filen chords.txt.

Om ni är fler än fyra i gruppen behöver ni även göra extrauppgiften, men om ni däremot är enbart tre behöver ni inte implementera play.

8.7.2 Obligatoriska uppgifter

Uppgift 1. Notes. Objektet ska kunna omvandla en tons namn (t.ex. "E2") till en heltalsrepresentation och tvärtom.

- a) Implementera först metoden `fromNbrToNote` med hjälp av %-operatoren
- b) Implementera metoden `unapply` som ska ta in en sträng (t.ex. "E2") och tonens nummer. "E2" kommer översättas till 16 och "C1" till 0 och tänk på att hantera specialfallet då till exempel "C#1" ska ge värdet 1. Använd attributet `toNumber` för att översätta en ton utan oktav ("C#", "E") till ett nummer. Titta gärna på vad metoden `zipWithIndex.toMap` gör och använd den i `toNumber`.

c) Använd `TestNotes` som ligger i paketet `test` för att se till så att `Notes` är rätt implemeterat. Starta `TestNotes` och rätta till eventuella fel.

Uppgift 2. `Chord`. Representation av de två olika ackorden. Lägga märken till hur stämning (eng. *tuning*) och ett grepp (eng. *grip*) representeras i `Chord`. `-1` i ett grep betyder att strängen inte ska spelas.

a) Implementera `toString` så att den matchar utskriften i filen *chords.txt*. I objektet `Chord` ska `toString` vara på formen `D:-1 -1 0 2 3 2`

b) Implementera metoden `isIncludedBy`. Använd `.split(";")` för att dela upp strängen `filter` i de olika filtersträngarna. Kolla för varje filtersträng om den inkluderas i `toString`. Exempel på filtersträng: `git:G;B;uku`. **Tänk på att metoden `forall` kollar om ett villkor gäller för alla möjligheter. I det här fallet räcker det att det gäller för ett av fallen**

c) Implementera `isGit` och `isUku` som jämför början av strängen `s` och returnerar **true** om strängen matchar ett gitarrackord resp. ukuleleackord.

d) Implementera klassen `Guitar`. `toString` ska vara på samma format som i filen *chords.txt*.

e) Skapa en ny klass `Ukulele` som har liknande beteende som `Guitar`.

f) Implementera `instToChord` med hjälp av `matching` och låt basfallet (om ackordet är varken gitarrackord eller ukuleleackord) retunera `None`.

Uppgift 3. `database`. Kommer hålla reda på alla ackord i programmet. Testa varje metod innan nästa blir implemeterad. Det blir då lättare att upptäcka fel i koden.

a) Börja med att implemetera `add` och `allChords`.

b) Implementera `find` och `updateFilter`. `find` ska inte retunera enbart den första förekomsten av söksträngen utan alla. Metoden `find` i `Vector` retunerar enbart första förekomsten.

c) Implementera `filteredChords` och `sort` med hjälp av metoden `isIncludedBy` i `Chord`. Tom filtersträng innebär att inget filter appliceras. Vid `sort` kan man ta hjälp av metoden `sortBy` i `Vector`.

d) Implementera `delete`. **Tänk på att `delete` ska radera ackordet på plats i `filteredChords` och inte i `codeallChords`**

Uppgift 4. `textui`

a) Provkör programmet och prova några olika kommandon (t.ex. `add`, `del`, `help`). Använd metoderna i `database` för att implementera kommandona.

b) Titta på objektet `Help`. Använd liknande matchning för att ta hand om alla fall för de olika kommandona.

- `Add`. Argumenten måste först bli en sträng med hjälp av `mkString(" ")` för att sedan delas upp vid `' '`. Användaren kan skriva in felaktiga ackord, vilket måste hanteras. Metoden `fromString` i `Chord` retunerar ett `Option[Chord]`. Titta närmare på metoden `flatMap`.

- Lst. Använd matching för att ta hand om fallet när användaren anger argument till kommandot, vilket inte ska vara med. Använd metoden i database. Listan ska vara numrerad från 1.
- Del. Använd matching för att ta hand om felfallen inget argument och fler än ett argument. Ett tredje felfall är om användaren anger något annat än en siffra som argument. Använd Try och matching för att ta hand om felet.
- Filter. Använd metoderna i database. De filtrerade ackorden behöver inte skrivas ut efter filtrering, utan användaren behöver använda kommandot list för att lista de filtrerade ackorden.
- Find. Använd matching för att ta hand om felfallen inget argument eller fler än ett argument.
- Load. Använd matching för att ta hand om felfallen inget argument eller fler än ett argument. Använd metoden load i objektet io för att läsa in från den angivna filen. Metoden kommer likna metoden Add.
- Save. Använd matching för att ta hand om felfallen inget argument eller fler än ett argument. Använd metoden save i objektet io för att skriva till den angivna filen. Om filen inte finns kommer den skapas.
- Sort. Använd matching för att ta hand om fallet när användaren anger argument till kommandot, vilket inte behövs. Använd metoden i database.
- Quit. Använd matching för att ta hand om fallet när användaren anger argument till kommandot, vilket inte behövs. Skapa en hjälpmetod quitPrompt som returnerar en Boolean med värdet **true** om användaren anger 'y', False om användaren anger 'n' och som körs igen vi felaktigt svar (allt annat än 'y' och 'n'). Även 'Y' och 'N' ska vara acceptabelt svar.
- Play. Använd matching för att bryta ut de olika argumenten och för att ta hand om felfallet inga argument. Första argumentet är tempo i millisekunder och resten är siffror som motsvarar platsen för ackordet i den filtrerade listan. Varje sträng i argumentlistan ska omvandlas till Int och det är viktigt att ta hand om fallet då användaren anger något annat än en siffra. Använd Try och matching för att ta hand om felet. Använd ChordPlayer för att spela upp ett ackord. **Kom ihåg att lägga till kommandot i listan med kommandon i doCommand**
- Draw. Använd matching för att ta hand om felfallen inget argument eller fler än ett argument. Argumentet motsvarar ackordets plats i den filtrerade listan. Använd Try och matching för att ta hand om felet att användaren anger något annat än en siffra. Använd ChordDraw för att rita upp ackord. **Kom ihåg att lägga till kommandot i listan med kommandon i doCommand**

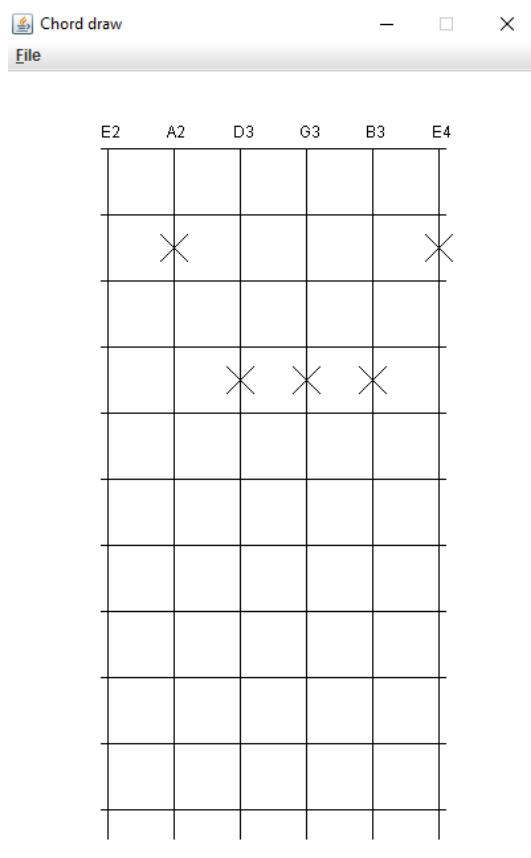
Uppgift 5. ChordPlayer. Ska spela upp enstaka ackord chord med en viss längd time i millisekunder.

- Titta vilka parametrar metoden `play` i `SimpleNotePlayer` behöver. Heltalet `note` ska vara heltalsrepresentationen av en ton.
- För att omvandla ett ackord till toner behöver först stämningen, `tuning`, omvandlas till sin heltalsrepresentation och sen ska greppet, `grip`, adderas till stämningen för varje sträng. Varje strängs ton ska sedan spelas upp under den angivna tiden. Koden för att spela ackordet ska placeras ovanför den existerande koden. **Tänk på att strängar med greppet -1 inte ska spelas**
- Skapa ett nytt kommando i `textui` som heter `play`. Se instruktioner för `textui`.

8.7.3 Extrauppgifter

Uppgift 6. ChordDraw

- Rita upp en greppbräda liknande bilden nedan (kryssen läggs till i kommande uppgifter). Antalet strängar ska variera beroende på instrument.



- Skapa en hjälpmetod `cross` som tar in två heltal `x` och `y`. Metoden ska rita upp ett kryss som är 20x20 pixlar och har sitt centrum i en angiven koordinaten.

- c) Rita ut ett kryss där en sträng trycks ner. **Tänk på att -1 och 0 anger att en sträng inte trycks ner.**
- d) Implementera metoden `play` som börjar med att vänta på ett event från `SimpleWindow`, sedan kollar om eventet är av typen `SimpleWindow.MOUSE_EVENT`. Sedan ska man kolla om användaren trycket på någon sträng (ett intervall på -10 till +10 i förhållande till strängens x-koordinat kan anses vara på strängen). Om användaren tryckt på en sträng ska denna spelas med hjälp av `SimpleNotePlayer`. Metoden `play` ska köras tills användaren kryssar ner fönstret, vilket motsvarar `SimpleWindow.CLOSE_EVENT`.
- e) Skapa ett nytt kommando i `textui` som heter *draw*. Se instruktioner för `textui`.

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

- ☐ Kunna skapa och iterera över matriser med nästlade for-loopar.
- ☐ Kunna använda sig av och förstå arv.
- ☐ Förstå och träna på olika villkor med if-satser.
- ☐ Känna till algoritmer för att lösa problem så som att ta sig igenom en labyrint eller slumpmässigt skapa en labyrint

Förberedelser

- ☐ Läs om matriser.
- ☐ Läs om arv.
- ☐ Läs om olika algoritmer för att ta sig igenom en labyrint:
()
- ☐ Läs om olika sätt att skapa en slumpmässig labyrint:
().
- ☐ Läs om olika algoritmer för att ta sig igenom en labyrint

9.2.1 Bakgrund

I denna laboration kommer du att få skapa labyrinter och sedan implementera algoritmer för att ta dig igenom dessa. En labyrint är ett rum som har en ingång och en utgång. Ingången är i de här fallen alltid längst ner i labyrinten, och utgången högst upp. Alla väggar är också parallella med antingen x-axeln eller y-axeln. Ett sätt att beskriva en sådan labyrint i kod är med hjälp av en matris. Varje element i matrisen motsvarar då en "ruta" i labyrinten. Värdet TRUE representerar att det finns en vägg och FALSE representerar att det är fri väg att gå där.

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

Den frivilliga uppgiften

I den frivilliga uppgiften ska en slumpmässig labyrint genereras. Det finns flera olika algoritmer för att göra detta men den vi kommer använda här är en slumpmässig variant av Prims algoritm. Du kan läsa mer under https://en.wikipedia.org/wiki/Maze_generation_algorithm (De representerar en labyrint på ett annat sätt än vi gör i den här uppgiften vilket innebär att det inte kommer se precis likadant ut).

9.2.2 Obligatoriska uppgifter

Uppgift 1. I denna uppgift ska du implementera en metod som kan 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 '.' representerar en gång. Utifrån detta skapar Maze en Boolean-matris "mapMatrix" som representerar labyrinthen, där TRUE står för vägg och FALSE står för gång. Fråga om något är oklart!
- b) Implementera metoden DRAW i Maze som ritar upp labyrinthen i SimpleWindow. För att göra detta behöver du gå igenom matrisen mapMatrix i Maze och undersöka elementen på varje plats. Om ett element är TRUE så betyder det att det här ska finnas (det vill säga ritas upp) en vägg i labyrinthen, om FALSE att det här ska finnas en gång. Ta hjälp av metoden brickInTheWall som finns tillgänglig i Maze-klassen.

Uppgift 2. En labbuppgiftsbeskrivning.

- a) Skapa en ny klass AMazeIngRace genom att välja File -> New -> Scala Class. I denna klass ska du skriva en main-metod där du skapar ett objekt av klassen Maze genom att läsa in från fil (börja exempelvis med filen maze1.txt). Du måste även skapa ett objekt av SimpleWindow för att skicka med i draw-metoden. Anropa sedan metoden draw på Maze-objektet och kolla att labyrinthen ritas upp som den ska. Gör samma sak för resterande av filerna mazeX.txt, alla ska kunna ritas upp korrekt. Testa att rita en egen labyrinth genom att skapa en textfil och lägg i samma mapp som övriga maze-filer. Kontrollera så att även denna ritas upp som den ska, och fixa annars till metoden draw så att den fungerar som tänkt.

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

- a) Skapa en ny klass MazeTurtle som ärver från klassen ColourTurtle. MazeTurtle ska ta ett extra argument, nämligen ett av typen Maze som är den labyrinth som sköldpaddan ska gå i.
- b) Definiera i MazeTurtle en ny metod åalk". Implementera sedan denna metod. I metoden ska sköldpaddan med hjälp av tekniken "hålla vänster hand i väggen" ta sig genom labyrinthen, från början till slut. Varje steg motsvarar att flytta sig från en ruta till en annan i Boolean-matrisen i Maze. Sköldpaddan kommer alltså ta sig fram genom att undersöka för varje steg om den borde svänga vänster, det vill säga om den inte har någon vägg till vänster om sig längre, om den borde gå rakt fram eller om den borde svänga höger.
- c) Lägg till kod i AMazeIngRace som skapar en sköldpadda och sedan låter denna gå igenom labyrinthen med metoden walk. Testa att din MazeTurtle fungerar som den ska! Sköldpaddan ska klara att ta sig igenom alla labyrinth i filerna maze1.txt - maze3.txt samt din egna labyrinth.

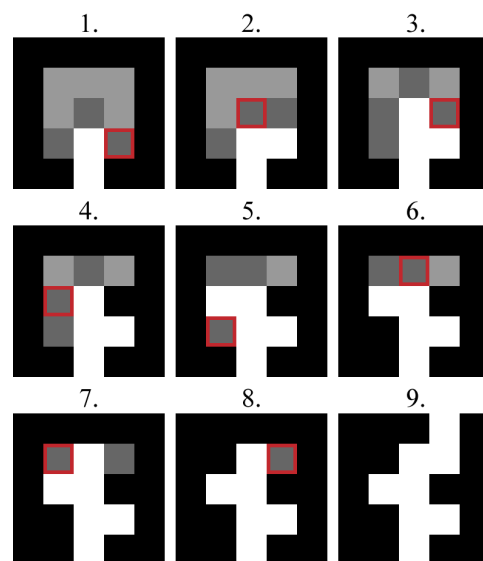
9.2.3 Frivilliga extrauppgifter

Prims algoritm är ett sätt att hitta hur man kopplar ihop alla punkterna i en graf med så få kopplingar som möjligt. Detta låter nog som rena grekiskan men tänk dig att punkterna är varje ruta i labyrinten och en hopkoppling är där du kan gå vilket innebär två öppna rutor intill varandra. Eftersom algoritmen vill ha så få kopplingar som möjligt får inga "rundor" uppstå utan det kommer endast finnas en väg igenom labyrinten

I praktiken funkar algoritmen genom att den väljer en slumpmässig ruta intill en öppen ruta och kollar om det är någon mer öppen ruta intill. Om så är fallet låter den rutan vara en vägg annars öppnar den rutan. Den upprepar detta tills alla rutor antingen öppnats eller låtit vara en vägg.

Eftersom det endast ska finnas en in- och utgång måste detta hanteras på lite annats sätt.

1. Börja med att välja en av rutorna i botten och öppna den och rutan över.
2. Applicera ovanstående algoritm på alla rutor utom de rutor som är runt (Så att vi får en vägg som går runt hela).
3. Efteråt letar vi efter en slumpmässig ruta på näst översta raden som är öppen och öppnar rutan ovanför.



Figur 9.1: Visualisering av algoritmen när den genererar en labyrint av storleken 5x5

Psuedokod

```
def random(rows, cols)
  labyrint = matris(rows, cols) fylld med true
```

```
buffer = lista(koordinater till rutor)

Välj en slumpmässig ruta i nedersta raden och öppna den

Öppna rutan ovanför
Lägg till rutorna runt till buffern

while(buffer har element kvar)
    element = välj slumpmässigt element i buffer
    if (element har max en öppen ruta intill sig)
        öppna rutan element innehåller
        lägg till rutorna runt till buffern
        ta bort element från buffer

Hitta en ruta som är öppen i näst översta raden
Öppna rutan ovanför
```

Uppgift 4. I den här uppgiften ska du implementera en algoritm för att skapa en slumpmässig labyrinth.

- Inspektera ovanstående pseudokod och försök förstå den. Fråga gärna om något är oklart! Läs också de färdigskrivna metoderna i metoden `random` i Maze och se om du kan förstå vad de gör.
- Implementera metoden `random` i Maze som skapar och returnerar en slumpmässigt utformad labyrinth med hjälp av pseudokoden ovan (eller på egen hand för den modiga/nyfikna!). Ta även hjälp av de färdigskrivna metoderna `addWallsToBuffer` och `checkWallsAround`.
- Skapa en ny labyrinth i `AMazeIngRace` genom att anropa din `random`-metod! Vad får du för resultat? Ett bra värde att använda på storleken när du anropar metoden är mellan 50 och 100. Dvs anropa metoden med till exempel `random(50, 50)`. Om du lyckas få fram en labyrinth och din `random`-metod är korrekt, testa att låta sköldpaddan gå igenom labyrinthen och se vad som händer!

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"/> algoritim: 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`
- ☐ översiktligt om relationen mellan trait och interface
- ☐ namnkonventioner för konstanter
- ☐ 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
- ☐ Statiskt
- ☐ Autoboxing
- ☐ Arv
- ☐ Läs igenom Bakgrunden, Kodstrukturen och alla kommentarer i kodskelettet.
- ☐ Exceptions

11.2.1 Bakgrund

I denna labb skall ni tillverka lthopoly, en variant av det välkända brädspelet 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. Beroende på vilken av de tre möjliga ruttyperna spelaren hamnar på sker olika saker:

- **MoveSpace:** Om en spelare hamnar på denna ruta slumpas ett kort från en vektor av MoveCards, varpå spelaren förflyttas antingen framåt eller bakåt det antal steg som kortet anger. Kortets deskriptiva text skrivs även ut i spelfönstret.
- **MoneySpace:** Om en spelare hamnar på denna ruta slumpas ett kort från en vektor av MoneyCards, varpå spelaren antingen förlorar pengar eller vinnar pengar enl. kortet. Kortets deskriptiva text skrivs även ut i spelfönstret.
- **HouseSpace:** Skulle en spelare hamnar här gäller olika saker beroende på rutans två tillstånd. Hamnar spelaren på en HouseSpace som ingen annan spelare har köpt upp skall spelaren få möjlighet att köpa denna om den har råd (inköpspriset är en hyra). Är det istället så att rutan ägs av en annan spelare den nuvarande spelaren betala hyra till ägaren (hyran är samma som inköpspriset).

11.2.2 Kodstruktur

Klassen `GameBoard` håller koll på spelets tillstånd. `GameBoard` använder sig några sorters hjälpobjekt för att hålla koll på detaljerna kring spelet. För varje spelare finns ett `Player`-objekt håller koll på information gällande den spelaren och `GameBoard` lagrar dessa in en `List<Player>`.

De ovannämnda tre ruttyperna implementeras som tre olika subklasser till `BoardSpace`. `GameBoard` lagrar dessa i en `ArrayList<BoardSpace>` som representerar hela spelplanen, där objektens index i listan motsvaras av rutornas position på spelplanen. Första rutan har index 0 och sista rutan på spelplanen har index `spaces.size() - 1`.

`GameBoard` har en metod `getPossibleActions` som returnerar en lista över alla möjliga spelarhandlingar för den nuvarande spelaren. Denna används av `main`-metoden för att be användaren välja nästa handling. Olika sorters spelarhandlingar representeras av statiska `int`-variabler i klassen `GameBoard`. Vid val av handling matar användaren in handlingens siffervärde i konsolen. Varje handling motsvaras då alltid av samma inmatningsvärde för användaren.

Den visuella representationen av spelet sker via konsolfönstret med hjälp av klassen `TextUI`. `TextUI` är en färdigskriven klass med metoder som gör det enkelt att skriva ut spelplanen och en logg av spelhistoriken i konsolfönstret. Alla utskrifter under spelets gång ska gå via `TextUI`.

Utskriften kan exempelvis se ut såhär:

```

1 =====
2
3 Oskar slog en 2:a!
4 Oskar drog ett kort: Jädrans! Studiebidraget har sänkts. Förlora 40 SEK
5 Oskar har avslutat sin runda.
6 Nästa spelare: Jonas
7 Jonas slog en 3:a!
8 Jonas drog ett kort: Det lönade sig att leva på nudlar! Inkassera 50 SEK
9 Jonas har avslutat sin runda.
10 Nästa spelare: Valthor
11 Valthor slog en 5:a!
12 Grattis, Valthor är nu den stolta ägaren av V-Huset
13 Valthor har avslutat sin runda.
14 Nästa spelare: Oskar
15 Oskar slog en 2:a!
16 Namn-----Position-----Pengar----
17 Oskar*           Moroten och piskan(40)      260
18 Jonas            ChansRuta                    350
19 Valthor          V-Huset [Valthor](45)        255
20 -----
21 Välj ett alternativ:
22
23 3. Köp ett hus
24 5. Avsluta din runda
25 8. Visa standardvyn
26 9. Visa spelplanen
27 0. Avsluta Lthopoly
28
29 =====

```

Färdigimplementerat

Specification TextUI

```

    /** Prints an ASCII plot of the total amount
    of money in the game as a function of the turn index*/
    def plotStatistics(x: Buffer[Int]): Unit

    /** Appends the String s to the end of the UI's event log */
    def addToLog(s: String): Unit

    /** Reprints the current state of the UI using the given
    GameBoard to print the status bar*/
    def updateConsole(board: GameBoard): Unit

    /** Asks the user to select an option from a list of options
    *
    * @param options an Array of tuples of the form (choice, description)
    *                where choice is the number the user should enter to select
    *                the choice represented by description,
    *                e.g. (0, "End Game") allows the user to input 0 to end
    *                the game.
    * @return        the selected choice
    */
    def promptForInput(options: Array[(Int, String)]): Int

    /** Prints the entire GameBoard */
    def printBoard(board: GameBoard): Unit

```

Implementeras Själv

class Player

```

    /** Creates a new player*/
    public Player(String name, int money, int pos);

    /** Returns the players money*/
    public int getMoney() ;

    /** Adjusts the players money*/
    public void adjustMoney(int money);

    /** Returns the players position*/
    public int getPosition();

    /** Returns a string representation of the player*/
    public String toString();

    /** Sets the players position*/
    public void setPosition(int pos) ;

```

class BoardSpace

```

/** Returns a array of int describing possible
 * game actions available while on this space*/
public abstract int[] getPossibleActions(GameBoard board);

/** Executes a game action available while on this space*/
public abstract void action(GameBoard board, int action);

/** Returns a string representation of this BoardSpace*/
public abstract String toString();

```

class GameBoard

```

/** Creates a new board ready to play */
public GameBoard(List<Player> players);

/**Returns an int array containing possible game actions.
 * A game action can be any of the static constants in
 * GameBoard*/
public int[] getPossibleActions() ;

/** Checks whether the game is over or not */
public boolean isGameOver();

/** Returns the player with the most money */
public Player getRichestPlayer();

/** Returns a list of all players */
public List<Player> getPlayers();

/** Returns a list of all BoardSpaces */
public List<BoardSpace> getBoardSpaces();

/** Performs an action for the current player */
public void doAction(int action);

/** Returns the currently active player */
public Player getCurrentPlayer();

/** Returns the boardspace corresponding to the position
 * of the current player. */
public BoardSpace getCurrentBoardSpace();

/** Moves the currently active player adjustments spaces forward.
 * Negative adjustment moves the player backwards*/
public void moveCurrentPlayer(int adjustment);

/** Returns an ArrayList<Integer> containing the total
 * sum of in the game after a round */
public ArrayList<Integer> getStatistics();

/** String Representation of the GameBoard */
public String toString() ;

```

class DocumentParser


```
/**Returns a ArrayList of Boardspaces loaded from a file*/
public static ArrayList<BoardSpace> getBoard();

/**Returns a array of MoneyCards loaded from file*/
public static MoneyCard[] getMoneyCards();

/**Returns a array of MoveCards loaded from file*/
public static MoveCard[] getMoveCards();
```

class MoneyCard

```
/**Creates a new MoneyCard*/
public MoneyCard( String description, int money);

/**Returns the cards money adjustment value*/
public int getMoney();

/**Returns the description of why the money is adjusted*/
public String getDescription();
```

class MoveCard

```
/**Creates a new MoveCard*/
public MoveCard( String description, int positionAdjustment) ;

/**Returns the position adjustment*/
public int getPositionAdjustment();

/**Returns the description of why the position is adjusted*/
public String getDescription();
```

Spelregler

- Varje spelare måste alltid börja sin runda med att slå en tärning innan den gör någon annan spelhandling, d.v.s. någon annan handling som påverkar spelets tillstånd (exempelvis kan man alltid visa spelplanen eller avsluta spelet).
- Om någon spelare har mindre än 0 SEK kvar skall spelet sluta.
- Om någon spelare hamnar på en husruta som ägs av en annan spelare måste denne betala ägaren husets hyra i SEK. Om ingen äger huset ges spelaren möjlighet att köpa det för ett belopp motsvarande en hyra (förutsatt att den har råd).
- Om en spelare hamnar på en MoveSpace eller ett MoneySpace får spelaren möjligheten att dra ett kort. För MoveCard innebär detta en förflyttning (bakåt eller framåt) medan för MoneyCard en minskning eller ökning av pengar.
- Spelplanen skall vara cyklisk, d.v.s. att rutan direkt efter sista rutan är den första rutan på spelplanen.

Nedan visas ett förenklat flödesdiagram för en spelrunda.

11.2.3 Obligatoriska uppgifter

Uppgift 1. All information om olika kort och om spelplanens upplägg finns i textfiler som måste läsas in med hjälp av metoderna i klassen `DocumentParser`. Textfilerna `moneycards.txt` och `movecards.txt` innehåller information om de olika korten som finns. Varje rad innehåller en förklaring för kortet följt av ett värde separerat med semikolon. Dessa sparas i en vektor och när en spelare hamnar på en `MoveSpace` eller `MoneySpace` dras ett slumpmässigt kort från vektorn. Vektorn måste alltså skickas med till motsvarande ruta när rut-objektet skapas. Filen `board.txt` innehåller spelplanen, men behandlas först i uppgift 3.

För att skapa ett `File`-objekt med informationen från filerna kan följande kod användas:

```
File f = new File(DocumentParser.class.getResource("/moneycards.txt").getFile());
```

Därefter kan ni använda er av ett scanner-objekt som får tillgång till Filen för att läsa in en rad i taget.

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

Ni kan nyttja metoden `String.split(String delimiter)` för att dela en sträng i en array av fält, där `delimiter` är den avgränsade strängen som används vid uppdelningen.

- Implementera klassen `MoveCard`.
- Implementera klassen `MoneyCard`.
- Implementera metoderna `getMoneyCards()` och `getMoveCards()` i `DocumentParser`.
- Implementera klassen `Player`.

Uppgift 2. I denna uppgift skall de tre olika subklasserna till `BoardSpace` implementeras.

- Implementera en klass för varje typ av spelruta. Tänk på att `MoveSpace` och `MoneySpace` behöver tillgång till respektive kortlekar. För att skriva metoden `action` kommer ni behöva nyttja att klassen `GameBoard` har statiska konstanterna som representerar de olika spelarvalen.

Obs! Än så länge kommer logiken inte fungera då inga metoder är implementerade i `BoardGame`, det går trots detta bra att anropa metoderna utan kompileringsfel (i väntan på att de implementeras).

Uppgift 3. Nu är det dags att implementera `getBoard()` i klassen `DocumentParser`.

I denna metod skall ni läsa in från filen board.txt och nyttja de metoder ni redan skrivit för att nu kunna skapa MoveSpaces och MoneySpaces. Radernas ordning i filen bestämmer deras ordning på spelplanen. Varje rad börjar antingen med orden "Move", "Money" eller "House". House-rader följs dessutom av husets hyra och dess namn separerat av semikolon. Baserat på radens första ord skall ett motsvarande objekt konstrueras och läggas till i en ArrayList<BoardSpace> som slutligen returneras.

a) Implementera getBoard().

Fundera på

- Behöver flera objekt skapas av varje ruttyp?

Uppgift 4. Implementera GameBoard enligt specifikationen.

Tips:

- Ni kan använda privata hjälpmetoder för att underlätta implementeringen.
- Metoden printStatistics i klassen TextUI tar en vektor av int-värden som inparameter, vilket är opassande då det underlättar att lagra pengahistoriken i en arrayList (eftersom dess storlek inte är bestämd). Det är därför lämpligt att skriva en metod som flyttar över samtliga Integer-objekt från ArrayList<Integers> till en vektor av primitiva int-värden. Detta fungerar trots att de har olika typer p.g.a. autoboxing.
- Tänk på att spelarna skall kunna gå runt spelplanen ett obegränsat antal gånger.
- Glöm inte att alla utskrifter skall gå via TextUI.
- Se flödesdiagrammet för att få en överblick för vilka actions som är tillåtna vid en given tidpunkt.

Fundera på

- Varför tar konstruktorn i GameBoard emot en List<Player> istället för en ArrayList<Player>?

Uppgift 5. Med spelplanen implementerad behövs en main-metod för att kunna starta spelet. I main-metoden skapas GameBoard samt alla Spelare. Spelet körs sedan som en loop där GameBoard tillfrågas vilka spelarhandlingar som är tillåtna för den nuvarande spelaren, erbjuder spelaren möjligheten att välja något av dessa alternativ, och matar sedan in spelarens val tillbaka till GameBoard som hanterar valet. GameBoard ska alltså hantera all spellogik internt. Spelloopen skall köras tills dess att spelet är över enligt GameBoard.isGameOver.

- a) Implementera `getAction` i Scala. Metoden ska anropa `TextUI.promptForInput` med en lämplig lista av tupler för att begära input från användarna. Metoden skall nyttja de statiska variablerna från `GameBoard` för att ge en lämplig utskrift.
- b) Implementera `main`-metoden i Scala.
- c) Efter att spelet har avslutats skall den totala mängden pengar i spelet plottas som en funktion av `rundindexet`. Själva uppritningen sker med hjälp av den färdigskrivna metoden `plotStatistics` i `TextUI` som kräver en `Buffer[Int]` innehållande varje rundas totalsumma.

Tips:

- Nyttja `scala.collection.JavaConverters` för att konvertera Javas datatyper till Scala.

11.2.4 Frivilliga extrauppgifter

Uppgift 6. Utöka spelet med ny spelmekanik.

- a) Implementera funktionalitet för att varje spelare ska få extra pengar då den passerar första spelrutan.
- b) Implementera funktionalitet för att varje spelare som hamnar på en ruta de äger sedan tidigare har möjlighet att öka hyran för rutan ifall någon annan spelare skulle hamna på den.
- c) Implementera funktionalitet för 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

- ☐ Känna till vad en tråd är och kunna förklara begreppet jämlöpande exekvering.
- ☐ Känna till vad metoderna `run` och `start` gör i klassen `Thread`.
- ☐ Kunna skapa och starta en tråd där metoden `run` är ersatt.
- ☐ Kunna skapa ett enkelt program som från två trådar tävlar om att uppdatera en variabel och förklara varför beteendet kan bli oförutsägbart.
- ☐ Kunna använda en `Future` för att köra igång flera parallella beräkningar.
- ☐ Kunna registrera en callback på en `Future` med metoden `onComplete`.
- ☐ Känna till att webbsidor beskrivs av HTML-kod och kunna skapa en minimal webbsida.
- ☐ Kunna ladda ner en webbsida med `scala.io.Source.fromURL`.

Förberedelser

- ☐ Studera teorin i kapitel 12.

12.1.1 Grunduppgifter

Uppgift 1. Trådar. Klassen `java.lang.Thread` används för att skapa **trådar** med jämlöpande exekvering (eng. *concurrent execution*). På så sätt kan man få olika koddelar att köra samtidigt.

Klassen `Thread` definierar metoden `run` som är tom. Vill man att tråden ska göra något vettigt får man ersätta metoden med det man vill ska göras. En tråd körs igång med metoden `start` och då anropas automatiskt `run`-metoden och tråden exekverar koden jämlöpande med övriga trådar. Om man anropar `run` direkt blir det *inte* jämlöpande exekvering. Med metoden `isAlive` kan man kolla om tråden "lever", d.v.s om den håller på att köra sin `run`-metod.

a) Skapa en tom tråd och kör igång den enligt nedan. Varför händer det inget när du anropar `start`.

```
1 val t1 = new Thread
2   t1.setName("tomma tråden")
3   t1.getName
4   t1.toString
5   t1.isAlive
6   t1.start
7   t1.isAlive
```

b) Skapa en tråd som gör något som tar lite tid och kör med `run` resp. `start`.

```
1 def zzz = { print("zzzzzz"); Thread.sleep(5000); println(" VAKEN!") }
2 zzz
3 val t2 = new Thread{ override def run = zzz }
4 t2.run
5 t2.run; println("Gomorrn!")
```

```

6 t2.start; println("Gomorra!")
7 t2.start

```

c) Vad händer om man anropar start mer än en gång på samma tråd?

d) Skapa två trådar med icke-tomma run-metoder och kör igång dem samtidigt. Vilken ordning skrivs hälsningarna ut efter rad 3 resp. rad 4 nedan? Förklara vad som händer.

```

1 val g = new Thread{ override def run = for (i <- 1 to 100) print("Gurka ") }
2 val t = new Thread{ override def run = for (i <- 1 to 100) print("Tomat ") }
3 g.run; t.run
4 g.start; t.start

```

e) Använd Thread.sleep enligt nedan. Är beteendet helt förutsägbart (deterministiskt)? Förklara vad som händer.

```

1 def ibland(block: => Unit) = new Thread {
2   override def run = while(true) { block; Thread.sleep(600) }
3 }.start
4 ibland(print("zzz ")); ibland(print("snark ")); ibland(println("hej!"))

```

Uppgift 2. Jämlöpande variabeluppdatering. Skriv klasserna Bank och Kund i en editor och klistra sedan in koden i REPL.

```

class Bank {
  private var saldo = 0;
  def visaSaldo: Unit = println(s"saldo: $saldo")
  def sättIn: Unit = { saldo += 1 }
  def taUt: Unit = { saldo -= 1 }
}

class Kund(bank: Bank) {
  def slösaSpara = {bank.taUt; Thread.sleep(1); bank.sättIn}
}

```

a) Använd funktionen ibland från föregående uppgift. Förklara vad som händer nedan.

```

1 val bank = new Bank
2 bank.visaSaldo
3 bank.sättIn
4 bank.visaSaldo
5 bank.taUt
6 bank.visaSaldo
7
8 val bamse = new Kund(bank)
9 val skutt = new Kund(bank)
10
11 bamse.sparaSlösa
12 skutt.sparaSlösa
13 bank.visaSaldo

```

```

14
15 def ofta(block: => Unit) = new Thread {
16   override def run = while(true) { block; Thread.sleep(1) }
17 }.start
18
19 ofta(bamse.slösaSpara); ofta(skutt.slösaSpara)
20
21 ibland(bank.visaSaldo)

```

Uppgift 3. *Jämlöpande exekvering med `scala.concurrent.Future`.* Att hålla reda på trådar kan bli ganska knepigt. Med hjälp av `scala.concurrent.Future` kan man sätta igång beräkningar i nya trådar och sedan köra vidare med annat. Detta sker ”under huven” med hjälp av ett avancerat ramverk för multitrådning i s.k. trådpooler som heter Akka¹ och möjliggör parallellprogrammering på en högre abstraktionsnivå.

I denna uppgift ska du ladda ner webbsidor parallellt med hjälp `Future` så att en nedladdning kan avslutas under tiden en annan dröjer.

a) Koden för en minimal webbsida ser ut som nedan. Du kan beskåda sidan här: <http://fileadmin.cs.lth.se/pgk/mini.html> eller skriva in nedan kod i en fil som heter något som slutar på `.html` och öppna filen i din webbläsare.

```

<!DOCTYPE html>
<html>
<body>
HELLO WORLD!
</body>
</html>

```

b) För att simulera slöa webbservrar kan man ladda ner en sida via sajten <http://deelay.me/>. Ladda ner ovan sida med 2 sekunders fördröjning:

<http://deelay.me/2000/http://fileadmin.cs.lth.se/pgk/mini.html>

c) Man kan ladda ner webbsidor med `scala.io.Source`. Vad händer nedan? Hur lång tid måste du i medeltal, bästa fall respektive värsta fall vänta innan du kan se första webbsidan i vektorn `laddningar` nedan?

```

1 scala> def ladda(url: String) = scala.io.Source.fromURL(url).getLines.toVector
2 scala> def slöladda(url: String) = {
3     val delay = (math.random * 1000 + 2000).toInt
4     val delaySite = s"http://deelay.me/$delay/"
5     ladda(delaySite+url)
6 }
7 scala> ladda("http://fileadmin.cs.lth.se/pgk/mini.html")
8 scala> def seg = slöladda("http://fileadmin.cs.lth.se/pgk/mini.html")
9 scala> val laddningar = Vector.fill(10)(seg)
10 scala> laddningar(0)

```

d) Innan vi kan köra igång en `Future` så måste vi importera den underliggande exekveringsmiljön som är redo att parallelisera ditt program i trådar utan

¹<http://akka.io/>

att du själv måste skapa dem. Detta görs med nedan import-satser. Förklara vad som händer nedan.

```
1 scala> import scala.concurrent._
2 scala> import ExecutionContext.Implicits.global
3 scala> val f = Future{ seg }
4 scala> f // kolla om den är klar annars prova igen senare
5 scala> f
```

e) Ladda indata utan att blockera (eng. *non-blocking input*). Förklara vad som händer nedan.

```
1 scala> val nonblock = Future{ Vector.fill(10)(seg) }
2 scala> nonblock // kolla igen senare om ej klar
3 scala> nonblock
```

f) Ladda indata separat i olika parallella trådar. Förklara vad som händer nedan.

```
1 scala> val para = Vector.fill(10)(Future{ seg })
2 scala> para
3 scala> para.map(_.isCompleted)
4 scala> para.map(_.isCompleted) // studera hur de blir färdiga en efter en
5 scala> para(0)
```

g) Registrera en s.k. *callback* med metoden `onComplete`. Förklara vad som händer nedan.

```
1 scala> val action = Vector.fill(10)(Future{ seg })
2 scala> action(0).onComplete(xs => println(s"ready:$xs"))
3 scala> // vänta tills laddning på plats 0 är klar
```

12.1.2 Extrauppgifter: öva mer på grunderna

Uppgift 4. TODO

12.1.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 5. Sök upp och studera dokumentationen för klassen `java.lang.Thread`.

a) Klassen `Thread` har funnits med länge och allteftersom JVM:en har utvecklats så har några metoder tillkommit och andra har blivit utfasade (eng. *deprecated*). Vilka metoder i klassen `Thread` är markerade *deprecated* och ska därmed fasas ut?

b) Läs mer om `Future` och jämlöpande exekvering i Scala här:

alvinalexander.com/scala/future-example-scala-cookbook-oncomplete-callback

c) Läs lite mer om jämlöpande exekvering och multitrådade program i Java här: www.tutorialspoint.com/java/java_multithreading.htm

När man skriver program med jämlöpande exekvering finns det många fallgror; det kan bli kapplöpning (eng. *race conditions*) om gemensamma resurser och dödläge (eng. *deadlock*) där inget händer får att trådar väntar på varandra. Mer om detta i senare kurser.

12.2 Laboration: life

Mål

- ☐ Kunna hur man kan använda matriser som en datastruktur.
- ☐ Kunna separera beteende från vy med hjälp av *Model-View* uppdelning.
- ☐ Känna till grundläggande cellulära automata.
- ☐ Känna till trådar, en grundläggande metod för att köra flera metoder *samtidigt*.

Förberedelser

- ☐ Läs igenom laborationen.

12.2.1 Bakgrund

Spelet Life (även kallat *Conway's Game of Life* efter skaparen och matematikern John Horton Conway) simulerar en koloni av encelliga organismer som lever, förökar sig och dör på ett bräde. Varje enskild cells överlevnad bestäms av några enkla regler som beror på dess omgivning, detta är en s.k. *cellulär automata*². Spelet går ut på att simulera flera generationer av en cellkoloni.

Spelet har inga medvetna spelare (ett så kallat 'zero-player game') och 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

²Detta är ett exempel på s.k. 'emergence' and 'self-organization'

12.2.3 Obligatoriska uppgifter

Uppgift 1. Skapa en modell 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:

- ☐ designexempel
- ☐ the expression problem
- ☐ utvecklingsprocessen
- ☐ krav-design-implementation-
- ☐ test
- ☐ översiktligt om trait som gräns-snitt

13.1 Projektuppgift: bank

13.1.1 Fokus

- ☐ Kunna implementera ett helt program efter given specifikation
- ☐ Kunna sätta samman olika delar från olika moduler
- ☐ Förstå hur Java-klasser kan användas i Scala
- ☐ Förstå och bedöma när immutable/mutable såväl som var/val bör användas i större sammanhang
- ☐ Kunna använda sig av kompanjons-objekt
- ☐ Kunna läsa och skriva till fil
- ☐ Kunna söka i olika datastrukturer på olika sätt

13.1.2 Bakgrund

I detta projekt ska du skriva ett program som håller reda på bankkonton och kunder i en bank. Programmet ska även hålla reda på bankens nuvarande tillstånd, såväl som föregående. Tillstånden ska vid varje tillståndsförändring skrivas till fil så att utifall banken skulle krascha finns alla transaktioner som genomförts sparade, banken kan således återställas.

Programmet ska vara helt textbaserat, man ska alltså interagera med programmet via konsollen där en meny skrivs ut och input görs via tangentbordet.

Du ska skriva hela programmet själv, men det ska dock följa de specifikationer som ges i uppgiften, såväl som de objektorienterade principer du lärt dig i kursen.

13.1.3 Krav

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

- Programmet ska ha följande menyval:
 - 1. Hitta konton för en viss kontoinnehavare.
 - 2. Söka efter kontoinnehavare på (del av) namn.
 - 3. Sätta in pengar på ett konto.
 - 4. Ta ut pengar på ett konto.
 - 5. Överföra pengar mellan två olika konton.
 - 6. Skapa ett nytt konto.
 - 7. Ta bort ett befintligt konto.
 - 8. Skriv ut bankens alla konton, sorterade i bokstavsordning efter innehavare.
 - 9. Återställa banken till ett tidigare tillstånd för ett givet datum. För simplicitet får alla transaktioner genomförda efter det datum banken återställts till permanent kasseras.

- 10. Avsluta.
- Programmet ska skapa ett nytt tillstånd med tidsstämpel och spara gamla tillstånd varje gång då:
 - Pengar sätts in eller tas ut från ett konto.
 - Pengar överförs mellan två konton.
 - Ett konto skapas.
 - Ett konto tas bort.
- Då bankens tillstånd förändras ska detta skrivas till fil.
- Då banken startas upp ska transaktionshistoriken läsas in så att banken laddar senaste sparade tillståndet.
- Inga utskrifter eller inläsningar får göras i klasserna Customer, BankAccount, Bank, State eller Transaction. Allt som berör användargränssnittet ska ske i BankApplication. Det är tillåtet att använda valfritt antal hjälpmetoder och hjälpklasser i klassen BankApplication.
- Alla metoder och attribut ska ha lämpliga åtkomsträttigheter.
- Valet av val/var och immutable/mutable måste vara lämpliga.
- Din indata måste ge samma resultat som i exemplen (som kommer komma i framtiden) i bilagan.
- Rimlig felhantering ska finnas, det är alltså önskvärt att programmet inte kraschar då man matar in felaktig input, utan istället säger till användaren att input är ogiltig.
- Programdesignen ska följa de specifikationer som är angivna nedan.

13.1.4 Design

Nedan följer specifikationerna för de olika klasserna bankapplikationen måste innehålla:

Specification BankAccount

```
/**
 * Creates a new bank account for the customer provided.
 * The account is given a unique account number and initially
 * got a balance of 0 kr.
 */
class BankAccount(val holder: Customer) = ???

/**
 * Deposits provided amount on this account.
 */
```

```

def deposit(amount: Int): Unit = ???

/**
 * Returns the balance of this account.
 */
def getBalance: Int = ???

/**
 * Withdraws provided amount from this account, if there
 * is enough money on the account. Returns true if the
 * transaction was successful, otherwise false.
 */
def withdraw(amount: Int): Boolean = ???

```

Specification Bank

```

/**
 * Creates a new bank with no accounts and no state.
 */
class Bank() = ???

    /**
     * Adds a new account in the bank.
     * The account number generated for the account is returned.
     */
    def addAccount(name: String, id: Int): Int = ???

    /**
     * Returns the customer with provided id. If no such customer
     * existed, return null.
     */
    def findHolder(id: Int): Customer = ???

    /**
     * Removes the bank account with provided account number,
     * returns true if successful, otherwise false is returned.
     */
    def removeAccount(accountNbr: Int): Boolean = ???

    /**
     * Returns a list with every bank account in the bank.
     */
    def getAllAccounts(): ArrayBuffer[BankAccount] = ???

    /**
     * Returns the account holding provided account number.
     * If no such account exists null is returned.
     */
    def findByNumber(accountNbr: Int): BankAccount = ???

    /**
     * Returns a list with every account belonging to the customer
     * with provided id.
     */
    def findAccountsForHolder(id: Int): ArrayBuffer[BankAccount] = ???

```

```

/**
 * Returns a list with all customers which names matches
 * with provided name pattern.
 */
def findByName(namePattern: String): ArrayBuffer[Customer] = ???

/**
 * Executes a transaction in the bank.
 */
def makeTransaction(transaction: Transaction): String = ???

/**
 * Resets the bank to the state corresponding to given date.
 * Returns true if successful, otherwise false.
 */
def returnToState(returnDate: Date): Boolean = ???

```

Specification Transaction

```

/**
 * Describes a transaction for a bank.
 */
abstract class Transaction = ???

```

Specification Customer

```

/**
 * Describes a customer of a bank with provided name and id.
 */
class Customer(val name: String, val id: Int) = ???

```

För att använda tidsstämplar ska klassen Date som finns bifogat i kursens workspace användas. Det är en enkel wrapper av Java.time.

13.1.5 Obligatoriska uppgifter

Uppgift 1. Implementera klassen Customer

Uppgift 2. Implementera klassen BankAccount

Uppgift 3. Implementera den transaktionsklass som skapar ett nytt konto.

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 klassen Bank

a) Implementera menyval 6 och 8. Testa noga.

- b) Implementera tillståndsfunktionaliteten. Varje ny transaktion ska ge upphov till ett nytt tillstånd och gamla tillstånd ska sparas som historik till det nya tillståndet.
- c) Implementera alla andra menyval, förutom menyval 9. Implementera även de klasser som förlänger Transaction utefter att de behövs för nya menyval. Testa de nya menyvalen noga efterhand som du implementerar dem, i synnerhet så att tillståndsförändringarna fungerar korrekt. Gör de utökningar du anser behövs.

Uppgift 6. Implementera menyval 9. Testa noga. Det är viktigt att denna funktionalitet fungerar bra innan man går vidare.

Uppgift 7. Implementera säkerhetskopiering av tillstånden.

- a) Implementera utskriften till fil då ett nytt tillstånd skapas, utskriften ska ske omedelbart. Banken ska ej behöva avslutas för att utskriften ska hamna på fil, om så vore fallet kan information fortfarande gå förlorad om banken kraschar.
- b) Implementera inläsningen från fil då banken startas.

13.1.6 Frivilliga extrauppgifter

Gör först klart inlämningsuppgiftens obligatoriska delar. Därefter kan du, om du vill, utöka ditt program enligt följande.

Uppgift 8. Skriv en eller flera av klasserna Customer, BankAccount och State i Java istället och använd istället för Scala versionen.

Uppgift 9. Implementera ett nytt menyalternativ som skriver ut all kontohistorik för en given person. I historiken ska dåvarande saldo vid varje transaktion synas.

13.1.7 Exempel på körning av programmet

Nedan visas möjliga exempel på körning av programmet. Data som matas in av användaren är markerad i fetstil. Ditt program måste inte se identiskt ut, men den övergripande strukturen såväl som resultat av körningen ska vara densamma. När exemplet börjar förutsätts det att banken inte har några konton.

1. Hitta ett konto för en given kund
2. Sök efter en kund utifrån (del av) angivet namn
3. Sätt in pengar
4. Ta ut pengar
5. Överför pengar mellan konton
6. Skapa nytt konto
7. Radera existerande konto
8. Skriv ut alla konton i banken

9. Återställ banken till ett tidigare datum

10. Avsluta

Val: **6**

Namn: **Adam Asson**

Id: **6707071234**

Nytt konto skapat med kontonummer: 1001

10:03:0 CET 5 / 14 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton
 6. Skapa nytt konto
 7. Radera existerande konto
 8. Skriv ut alla konton i banken
 9. Återställ banken till ett tidigare datum
 10. Avsluta

Val: **6**

Namn: **Berit Besson**

Id: **8505255678**

Nytt konto skapat med kontonummer: 1001

10:12:0 CET 5 / 14 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton
 6. Skapa nytt konto
 7. Radera existerande konto
 8. Skriv ut alla konton i banken
 9. Återställ banken till ett tidigare datum
 10. Avsluta

Val: **8**

Konto 1000 (Adam Asson, id 850127) 0 kr

Konto 1001 (Berit Besson, id 900318) 0 kr

10:13:0 CET 5 / 14 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton

6. Skapa nytt konto
7. Radera existerande konto
8. Skriv ut alla konton i banken
9. Återställ banken till ett tidigare datum
10. Avsluta

Val: **6**

Namn: **Berit Besson**

Id: **8505255678**

Nytt konto skapat med kontonummer: 1002

13:56:0 CET 5 / 14 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton
 6. Skapa nytt konto
 7. Radera existerande konto
 8. Skriv ut alla konton i banken
 9. Återställ banken till ett tidigare datum
 10. Avsluta

Val: **8**

Namn: **erit** Konto 1001 (Berit Besson, id 900318) 0 kr

Konto 1002 (Berit Besson, id 900318) 0 kr

14:01:0 CET 5 / 14 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton
 6. Skapa nytt konto
 7. Radera existerande konto
 8. Skriv ut alla konton i banken
 9. Återställ banken till ett tidigare datum
 10. Avsluta

Val: **3**

Kontonummer: **1000**

Summa: **5000**

Transaktionen lyckades.

14:36:0 CET 5 / 14 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn

3. Sätt in pengar
4. Ta ut pengar
5. Överför pengar mellan konton
6. Skapa nytt konto
7. Radera existerande konto
8. Skriv ut alla konton i banken
9. Återställ banken till ett tidigare datum
10. Avsluta

Val: **5**

Kontonummer att överföra ifrån: **1000**

Kontonummer att överföra till: **1001**

Summa: **1000**

Transaktionen lyckades.

14:37:0 CET 5 / 14 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton
 6. Skapa nytt konto
 7. Radera existerande konto
 8. Skriv ut alla konton i banken
 9. Återställ banken till ett tidigare datum
 10. Avsluta

Val: **8**

Konto 1000 (Adam Asson, id 850127) 4000 kr

Konto 1001 (Berit Besson, id 900318) 1000 kr

Konto 1002 (Berit Besson, id 900318) 0 kr

14:52:0 CET 5 / 14 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton
 6. Skapa nytt konto
 7. Radera existerande konto
 8. Skriv ut alla konton i banken
 9. Återställ banken till ett tidigare datum
 10. Avsluta

Val: **9**

Vilket datum vill du återställa banken till? År: **2016**

Månad: **5**

Datum (dag): **8**

Timme: **10**

Minut: **5**

Banken återställd.

15:00:0 CET 5 / 14 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton
 6. Skapa nytt konto
 7. Radera existerande konto
 8. Skriv ut alla konton i banken
 9. Återställ banken till ett tidigare datum
 10. Avsluta

Val: **8**

Konto 1000 (Adam Asson, id 850127) 0 kr

15:01:0 CET 5 / 14 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton
 6. Skapa nytt konto
 7. Radera existerande konto
 8. Skriv ut alla konton i banken
 9. Återställ banken till ett tidigare datum
 10. Avsluta

Val: **3**

Kontonummer: **1001**

Summa: **5000**

Transaktionen misslyckades. Inget sådant konto hittades.

15:06:0 CET 5 / 14 - 2016

-
1. Hitta ett konto för en given kund
 2. Sök efter en kund utifrån (del av) angivet namn
 3. Sätt in pengar
 4. Ta ut pengar
 5. Överför pengar mellan konton
 6. Skapa nytt konto
 7. Radera existerande konto
 8. Skriv ut alla konton i banken
 9. Återställ banken till ett tidigare datum

10. Avsluta

Val: **3**

Kontonummer: **1001**

Summa: **5000**

Transaktionen misslyckades. Otillräckligt saldo.

15:23:0 CET 5 / 14 - 2016

13.2 Projektuppgift: tictactoe

Mål

- ☐ Implementera ett helt program efter specifikation.
- ☐ Få en inblick i hur rekursion kan användas, utöver svans-rekursion.
- ☐ Bli introducerad till spelteori och hur man kan uttrycka optimal strategi för spelet tictactoe.
- ☐ Träna på att använda abstrakta klasser.
- ☐ Kunna byta mellan representationer av en spelplan.

13.2.1 Bakgrund

I detta projektet ska du implementera din egen version av spelet tic-tac-toe (eller som vi på svenska kallar det, tre i rad)! Du kommer börja med att implementera en version där du kan spela mot en kursare och sen gå vidare till att implementera en datorspelare som lägger sin pjäs slumpmässigt och till slut en som inte kan förlora!

13.2.2 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 spelpjäs eller om spelplanen är fylld.

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

13.2.3 Teori

Representationen är vald till en endimensionell vektor av typen `Int` av storlek 9 där element $[0,2]$ ¹ 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`. Vi har valt representationen där 0 representerar tom plats, 1 representerar x och -1 representerar o. Denna representation är dels smidig för vår framtida `OptimalP` och även för att avgöra

¹Med beteckningen $[x,y]$ menas alla heltal från x till y, dvs: x, x+1, x+2, ... , y-1, y. $[0,2] = 0,1,2$

om spelare x eller o har vunnit. Man kan exempelvis summera en rad och kolla om radens summa är 3, då har x vunnit eller -3, då har o vunnit.

13.2.4 Obligatoriska uppgifter

Uppgift 1. Implementera ett fungerande spel genom att utöka kodskeletten i klasserna Player, HumanP och Game.

- Implementera funktionen gameWon i klassen Player som testar huruvida spelaren who vunnit spelet.
- Implementera HumanPs move-funktion.
- Implementera en version av Game. Börja med att alltid spela ett spel och alltid rita spelplanen. main, draw och play behöver implementeras. All funktionalitet i main behöver ännu inte finnas.²

Uppgift 2. Randomized player

- Skapa en ny utökning av Player (kopiera HumanP, och byt namn till RandP) där move istället för att läsa från System.in väljer ett random giltigt drag.
- Ändra Game så att användaren tillåts stänga av ritfunktionen och i så fall tillåts välja antalet spel.
- Vad är sannolikheterna för att x vinner, o vinner och att det blir oavgjort om två RandP spelar mot varandra?

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

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

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

Uppgift 3. Optimal player

Betrakta den givna funktionen eval

```
/* returns 1 if there is a guaranteed strategy for who to win
 * returns 0 if there is a guaranteed strategy for who to draw
 * returns -1 if the opponent can force a win,
 * no matter what who does.
 * This is done by min,max-evaluation.
 * Find the move that gives the oppoent the worst possible
 * position and return -min, this is our max.
 */
def eval(game: Array[Int], depth: Int, who: Int): Int = {
```

²Notera att man behöver inverta spelplanen om den ska skickas till spelare två (alternativt låta spelaren hålla reda på om den är x eller o). Förslagsvis löses detta med en extra funktion invGame som skapar en ny array med omvända tecken till originalarrayen.

```

if(gameWon(game, -who)) return -1;
if(depth == 9) return 0;
var min = 1;
for(i <- 0 until 9) {
  if(game(i) == 0) {
    game(i) = who;
    val score = eval(game, depth+1, -who);
    if(score < min){
      min = score;
    }
    game(i) = 0;
  }
}
-min;
}

```

eval avgör om du är i en vinnande, förlorande eller oavgjord situation, givet att båda spelare spelar optimalt. Det som nog är svårast att förstå är varför vi returnerar -min på slutet. min sparar det sämsta värdet som vår motståndare kan få givet våra möjliga drag. Vi observerar att vi är i precis omvänd situation jämfört med vår motståndare. Om vår motståndare definitivt vinner förlorar vi definitivt, om det blir oavgjort för vår motståndare blir det också oavgjort för oss, och om vår motståndare definitivt förlorar, då vinner vi. Vi representerade ju vinst med 1, oavgjort med 0 och förlust med -1. Det är alltså härifrån minustecknet kommer ifrån. Vill man läsa mer om detta kan man kolla in wikipedias artikel <https://en.wikipedia.org/wiki/Minimax> om minmax-evaluering. Vi tar helt enkelt det draget som är sämst för vår motståndare.

- Implementera move-funktionen till OptimalP.
- Låt två OptimalP spela mot varandra några gånger (10-typer). Det skall alltid bli oavgjort.
- Testa att spela mot din OptimalP med en HumanP. Kan du spela lika? Kan du vinna?
- Vad händer om du sätter en RandP mot OptimalP? Blir det någonsin oavgjort, hur ofta? Blir det någon skillnad man byter vem som får spela först.

Uppgift 4. Säkerhet

I nuläget finns det förmodligen ett problem med din nuvarande implementation, och det är att du skickar iväg en mutable datastruktur till en Player som utifrån den mutable datan skall göra ett drag. Tänk om en elak programmerare bestämmer sig för att ändra på spelplanen i sin egna players move-metod. Då skulle man i princip kunna fuska. För att lösa detta kan man skicka en kopia till spelaren från Game.

13.2.5 Frivilliga extrauppgifter

Uppgift 5. Hashning.

Om du låter en OptimalP spela mot en RandP 1000 gånger lär det ta ganska lång tid. Det behöver det inte göra. När OptimalP bestämmer vilket drag den skall göra första gången går den ju igenom alla andra möjliga drag man kan komma till. Det visar sig att det inte finns så många unika spelbräden. Färre än $3^9 < 20000$. Sparar man ett värde till varje sådant spelbräde i en HashMap kan man bara fråga HashMapen vilket drag som är bäst givet ett spelbräde. Detta går väldigt snabbt, jämfört med ungefär $9! > 300000$ funktionsanrop för ett drag på ett tomt spelbräde. Således behöver vi bara gå igenom våra evauleringsmetoder en gång för att bygga hashmapen, sedan går det jättesnabbt för vår HashOptimalP att spela tictactoe.

Man får dock vara lite klurig, en `Array[Int]` går inte att använda som key i en hashMap, då den inte har en implementerad `hashCode`-funktion. Enklarest är att göra om vår array till en sträng, genom att lägga värdena i arrayen efter varandra i strängen. Vi kan göra en privat funktion `hash(Array[Int]):String` som konkatenerar värdena i arrayen. `hash([0,0,0,1,1,0,-1,-1,0])` skall alltså returnera `"000110-1-10"`.

- a) Skapa en ny subklass `HashOptimalP`.
- b) Skapa och implementera en privat metod `hash(Array[Int]):String`
- c) Implementera en konstruktör som skapar och genererar en `HashMap`.

Här kan du använda koden som används i evalfunktionen, men du måste komma ihåg att innan du returnerar måste ett key-value-par läggas in i din hashMap.

- d) Låt `move`-funktion göra en hashlookup med hjälp av `hash`-funktionen och din `HashMap`.
- e) Testa att låta två `HashOptimalP` spela mot varandra. Du märker nog att skapandet av en sådan spelare kommer ta lite tid, typ en halv sekund. Sedan skall det dock gå jättesnabbt när spelarna spelar. 100000 spel skall gå utan problem på någon sekund, vilket borde gå på tiotals minuter för den gamla `OptimalP`.

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. Filterklasserna ska ärva från en abstrakt ImageFilter-klass som är skriven i Java.

Följande beskriver ImageFilter-klassen.

```
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);

/**
 * Berättar huruvida ett filter behöver ett parmetervärde eller inte
 * @return true ifall parametervärde behövs, annars false
 */
public abstract boolean needsParameter();

/**
```

```

* 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);

```

Utöver filterklasserna ska du även implementera FilterChooser som hanterar val av filter och FilterList som kan applicera ett variabelt antal filter på en och samma bild. Du får tillgång till en Image-klass som representerar en bild samt ett ImageUI som hjälper att ladda in en JPEG bild.

Specification Image

```

class Image(val image: BufferedImage);

/** Returns a matrix of Color-objects that represents an image */
def getColorMatrix: Array[Array[Color]];

/** Updates the image in accordance with the given Color-matrix */
def updateImage(pixels: Array[Array[Color]]): Unit;

```

Specification FilterList

```

class FilterList = ???

/** Adds a filter to the FilterList */
def addFilter(filter: ImageFilter): Unit = ???

/** Applies all the filters on the given Image and draws it on SimpleWindow */
def applyFilters(image: Image, sw: SimpleWindow): Unit = ???

```

Specification FilterChooser

```

/** Creates a FilterChooser with all the available filters */
class FilterChooser(filters: Array[ImageFilter]) = ???

/** Shows which filters are available and lets the user choose filters
 * until an escape sequence has been given and returns a FilterList which
 * contain the chosen filters
 * Example:
 * Tryck på 1 för Blått-filter
 * Tryck på 2 för Kontrast-filter
 * Tryck på 3 för Gauss-filter

```

```

* Tryck på 4 för Sobel-filter
* Tryck 42 om du inte vill ha fler filter
*/
def chooseFilters(): FilterList = ???

```

Uppgift 1. Implementera ett blåfilter. Det vill säga skapa ett filter där varje pixelelement bara innehåller den blå komponenten. Testa filteret genom att skriva en main metod. Använd ImageUI för att välja en bild på följande sätt:

```
val im = new Image(ImageUI.getImage)
```

Använd SimpleWindow för att visa bilden.

Uppgift 2. 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 3. 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 är 105 så ska ett nytt Color objekt med värdena (105, 105, 105) skapas.

Uppgift 4. Skapa ett filter som krypterar bilden med xor-operatoren. Varje bildelement krypteras genom att använda xor-operatoren med ursprungsvärdena för rött, grönt och blått tillsammans med ett slumpmässigt heltalsvärde som genereras av Javas Random klass.

Uppgift 5. 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. Enklast är att flytta över dessa element oförändrade från inPixels till outPixels. Man kan också sätta dem till Color.WHITE, men då kommer den filtrerade bilden att se något mindre ut.

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

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

Uppgift 6. 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. Koefficienterna i matrisen `X_SOBEL` uttrycker derivering i x-led, i `Y_SOBEL` faltning i y-led. För att förklara varför koefficienterna ibland är 1 och ibland 2 måste man studera den bakomliggande teorin noggrant, men det gör vi inte här.

3. Om resultaten av faltningen i en punkt betecknas med `sx` och `sy` så får man en indikator på närvaron av en kontur med `Math.abs(sx) + Math.abs(sy)`. Absolutbelopp behöver man eftersom man har negativa koefficienter i faltningsmatriserna.
4. Sätt pixeln till svart om indikatorn är större än tröskelvärdet, till vit annars. Tröskelvärdet bestäms av `paramValue`. Skriv en klass `SobelFilter` som implementerar denna algoritm.

Uppgift 7. Implementera `FilterList` enligt specifikationerna ovan.

Uppgift 8. Implementera `FilterChooser` enligt specifikationerna ovan.

Uppgift 9. Knyt ihop allt i ett `ImageProcessing` object som ska innehålla en `main` metod där allt körs från. Utskrifterna ska se ut på följande sätt:
Välj en av följande bilder genom att mata in en siffra

- 0. boy.jpg
- 1. car.jpg
- 2. duck.jpg
- 3. facade.jpg
- 4. jay.jpg
- 5. moon.jpg
- 6. obidos.jpg
- 7. sgrada.jpg
- 8. shuttle.jpg

Ditt val: 1

Bild car.jpg laddad

Tryck på 0 för Vanligt-filter

Tryck på 1 för Blått-filter

Tryck på 2 för Krypterat-filter

Tryck på 3 för Inverterat-filter

Tryck på 4 för Grått-filter

Tryck på 5 för Kontrast-filter
Tryck på 6 för Gauss-filter
Tryck på 7 för Sobel-filter
Tryck 42 om du inte vill använda fler filter
Välj ett filter 1
Välj ett filter 42
Välja ny bild? (y/n) n

Tänk på att användaren kan mata in otillåtna värden. Detta ska hanteras på lämpligt sätt.

13.3.3 Frivilliga extrauppgifter

Uppgift 10. 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 $\text{histogram}[0] + \text{histogram}[1] + \dots + \text{histogram}[\text{lowCut}] = 0.015 * 100000$ (så nära det går att komma, det blir troligen inte exakt likhet). Beräkna highCut på liknande sätt. Sammanfattning av algoritmen:

1. Beräkna 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.

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

Hur bidra till kursmaterialet?

J.1 Bidrag är varmt välkomna!

Ett av huvudsyftena med att göra detta kursmaterial fritt och öppet är att möjliggöra bidrag från alla som är intresserade. Speciellt välkommet är bidrag från studenter som vill vara delaktiga i att utveckla undervisningen.

J.2 Instruktioner

J.2.1 Vad behövers för att kunna bidra?

Om du hittar ett problem, t.ex. ett enkelt stavfel, eller har något mer omfattande som borde förbättras, men ännu inte känner till eller har tillgång till de verktyg som beskriv nedan och som behövs för att göra bidrag, kontakta då någon som redan bidragit till materialet, så att någon annan kan implementera ditt förslag.

Innan du själv kan implementera ändringar direkt i materialet, behöver du känna till, och ha tillgång till, ett eller flera av följande verktyg (beroende på vad ändringen gäller):

- Latex: en.wikibooks.org/wiki/LaTeX
- Scala: en.wikipedia.org/wiki/Scala_%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

J.2.2 Svenska eller engelska?

Vi blandar engelska och svenska enligt följande principer:

- Publika diskussioner, t.ex. i issues och pull requests på GitHub, sker på engelska. I en framtid kan delar av materialet komma att översättas till

engelska och då är det bra om även icke-engelskspråkiga kan förstå vad som har hänt. Alla ändringshändelser sparas och man kan söka och gå tillbaka i historiken.

- Kompendiet finns för närvarande bara på svenska eftersom kursen initialt endast ges för svenskspråkiga studenter, men texten ska hjälpa läsaren att tillgodogöra sig motsvarande engelsk terminologi. Skriv därför mostvarande engelska begrepp (eng. *concept*) i parentes med hjälp av latex-kommandot `\Eng{concept}`.
- På övningar och föreläsningar är svenska variabelnamn ok. Svenska kan användas för att hjälpa den som håller på att lära sig att skilja på ord som vi själv hittar på och ord som finns i programmeringsspråket. Detta signalerar också att när man lär sig och experimenterar kan man hitta på 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.

J.3 Exempel

Som exempel på hur det går till i ett typiskt öppen-källkodsprojekt, beskrivs nedan vad som hände i ett verkligt fall: en dokumentationsuppdatering av Scala-dokumentationen efter att ett fel upptäckts. Detta 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 avlutade) PR här:
<https://github.com/scala/scala.github.com/pull/517>
11. När jag skapat en PR fick de som sköter repot ett automatiskt meddelande om denna nya PR och den efterföljande granskningsfasen inträdde. Den brukar sluta med att en eller flera andra personer kommenterar PR i webbgränssnittet med 'LGTM'. LGTM = "*Looks Good To Me*" och betyder ungefär "jag har kollat på detta nu och det verkar (vad jag kan bedöma) vara utmärkt och alltså redo för *merge*". Om det inte ser bra ut så förväntas granskaren föreslå vad som behöver förbättras i en saklig och trevlig ton.
12. När PR är granskad så kan en person, som har rättigheter att ändra, "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.

Appendix K

Lösningsförslag till övningar

K.1 expressions

K.1.1 Grunduppgifter

Uppgift 1.

- a) REPL skriver ut "hejsan REPL"
- b) man får fortsätta på nästa rad
- c) Värde: gurkatomat. Typ: String
- d) värde: gurkatomat upprepat 42 gånger. Typ:String

Uppgift 2.

Uppgift 3.

- a) Int
- b) Long
- c) char
- d) String
- e) Double
- f) Double
- g) Double
- h) Float
- i) Float
- j) Boolean
- k) Boolean

Uppgift 4. skriver ut hejsan42 gurka

klammer definierar funktionen och semikolon tillåter en att skriva flera satser på samma rad

Uppgift 5. a) uttryck är evaluering utav matematiska operationer. satser är kommandon

- b) println()
- c)

värdesaknas innehåller Unit

skriver ut Unit

skriver ut "()"

skriver ut "()"

skriver först ut hej med det innersta anropet och sen () med det yttre anropet

- d) Unit
- e) Unit

Uppgift 6.

- a) Int, 42
- b) Float, 42
- c) Double, 42
- d) Double, 42
- e) Float, 1.042E42
- f) Long, 42E6
- g) String, gurka
- h) Char, 'A'
- i) Int, 65
- j) Int, 48
- k) Int, 49
- l) Int, 57
- m) Char, 'q'
- n) Char, '*'

Uppgift 7. a) lint, 84

- b) Float, 21
- c) Float, 41.8
- d) Double, 44

Uppgift 8. a) Int, 46

- b) Int, 88
- c) Double, 13.3
- d) Int, 13

Uppgift 9. a) int, 21

- b) int, 10
- c) float, 10.5
- d) int, 0
- e) int, 1
- f) int, 3
- g) int, 0

Uppgift 10. a) 127, -128

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

Uppgift 11. a) java: PI scala: Pi

b) andvänder sig utav pythagoras sats

c)

Uppgift 12. a) den blir Int.MinValue

b) kastar exeption

c) 1.0000000000000001E8

d) avrundas till 1E8

e) 45.00000000000001

f) returnerar en double som är oändlig

g) Int.MaxValue

h) NaN

i) NaN

j)

Uppgift 13. a) true

b) false

c) false

d) false

e) true

f) true

g) true

h) false

i) true

j) false

k) false

l) true

m) true

n) false

o) true

p) true

q) true

r) false

s) true

t) false

u) true

v) true

Uppgift 14. a = 42 b = 43 c = Double 170 b = 0 a = 0 c = Double 171

Uppgift 15.

a)

x blir 42 x blir 43 skriv ut x x = 44 skriv ut x false constant värde y blir 42 fel skriv ut gurka och z blir 42 funktionen w blir det inom måsvingarna skriv ut z skriv ut z z blir 43 anropa w anropa w fel

b)

rad 8 och 16 y är konstant och kan ej modifieras kan ej modifiera en funktion

c)

var är en modifierbar variabel val är ett konstant värde def är en funktion

Uppgift 16.

a blir 42 b blir 84 a blir 52 b blir 74 a blir 104 b blir 37

Uppgift 17.

a) Namnet Kim Robinsson har 12 bokstäver.

b)

val fTot = f.size val eTot = e.size println(s"fharfTot bokstäver.") println(s"ehareTot bokstäver.")

Uppgift 18.

skriver ut sant eftersom den hoppar över andra kod blocket skriver ut falsk för den hoppar över det första kodblocket skriver ut sant eftersom den hoppar över andra kod blocket skriver ut falsk för den hoppar över det första kodblocket definerar en funktion som 50kastar kronan tre gånger

Uppgift 19.

a) String, inte gott b) string, gott c) string, likastora d) string, gurka

e) string, banan

Uppgift 20.

a)

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

b)

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

Uppgift 21.

a)

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

b)

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

Uppgift 22.

- a) 0 till 9 0, 2, 4, 6, 8, 10, 12
- b) `var k = 0 while(k <= 43) print(Ä" + k + ", ") k = k + 3`
- c) `foreach`

Uppgift 23.

- a) Double
- b) 0, less than 1.0
- c)
- d) man får olika slumpmässiga tal
- e) den skriver ut flera slumptal
- f) `for (i <- 1 to 100) println((math.random * 9).toInt)`
- g) `for (i <- 1 to 100) println((math.random * 5 + 1).toInt)`
- h) gurka skrivs ut olika antal gånger
- i) `while (math.random > 0.01) println(gurka")`
- j) sama sak som i dem förra fast man skriver ut slumptalet

Uppgift 24.

- a) `poäng > 1000`
- b) `poäng > 100`
- c) `poäng < highscore`
- d) `poäng < 0 || poäng > highscore`
- e) `poäng > 0 && poäng < highscore`
- f) `klar`
- g) `!klar`

K.1.2 Extrauppgifter: öva mer på grunderna**Uppgift 25.**

- a)
- b) Lösningstext.

K.1.3 Fördjupningsuppgifter: avancerad nivå**Uppgift 26.**

- a) 42
- b) Lösningstext.

K.2 programs

Uppgift 1.

- a) 42
- b) Lösningstext.

Uppgift 2.

Uppgift 3.

- a) 42
- b) Lösningstext.

Uppgift 4.

- a) 42
- b) Lösningstext.

K.3 functions

Uppgift 1.

- a) 42
- b) Lösningstext.

Uppgift 2.

Uppgift 3.

- a) 42
- b) Lösningstext.

Uppgift 4.

- a) 42
- b) Lösningstext.

K.4 data

Uppgift 1.

- a) 42
- b) Lösningstext.

Uppgift 2.

Uppgift 3.

- a) 42
- b) Lösningstext.

Uppgift 4.

- a) 42
- b) Lösningstext.

K.5 sequences

Uppgift 1.

- a) 42
- b) Lösningstext.

Uppgift 2.

Uppgift 3.

- a) 42
- b) Lösningstext.

Uppgift 4.

- a) 42
- b) Lösningstext.

K.6 classes

Uppgift 1.

- a) 42
- b) Lösningstext.

Uppgift 2.

Uppgift 3.

- a) 42
- b) Lösningstext.

Uppgift 4.

- a) 42
- b) Lösningstext.

K.7 traits

Uppgift 1.

- a) 42
- b) Lösningstext.

Uppgift 2.

Uppgift 3.

- a) 42
- b) Lösningstext.

Uppgift 4.

- a) 42
- b) Lösningstext.

K.8 matching

Uppgift 1.

- a) 42
- b) Lösningstext.

Uppgift 2.

Uppgift 3.

- a) 42
- b) Lösningstext.

Uppgift 4.

- a) 42
- b) Lösningstext.

K.9 matrices

Uppgift 1.

- a) 42
- b) Lösningstext.

Uppgift 2.

Uppgift 3.

- a) 42
- b) Lösningstext.

Uppgift 4.

- a) 42
- b) Lösningstext.

K.10 sorting

Uppgift 1.

- a) 42
- b) Lösningstext.

Uppgift 2.

Uppgift 3.

- a) 42
- b) Lösningstext.

Uppgift 4.

- a) 42
- b) Lösningstext.

K.11 scalajava

Uppgift 1.

- a) 42
- b) Lösningstext.

Uppgift 2.

Uppgift 3.

- a) 42
- b) Lösningstext.

Uppgift 4.

- a) 42
- b) Lösningstext.

K.12 threads

Uppgift 1.

- a)
- b)
- c)
- d)
- e)

Uppgift 2.

- a)

Uppgift 3.

- a)
- b)
- c)
- d)
- e)
- f)
- g)

Uppgift 4.

Uppgift 5.

- a)
- b)
- c)

Appendix L

Ordlista