Vecka 12. Fördjupning, Projekt

Programmering, grundkurs (pgk)

Björn Regnell

Datavetenskap, LTH, Lunds universitet
https://lunduniversity.github.io/pgk

EDAA45, Lp1-2, HT2025

Kompilerad den 22 augusti 2025

12 Fördjupning, Projekt

- Om projektuppgiften
- Avslutning: munta och tenta
- Jämförelse Scala och Java
- Array i Java
- Autoboxning i Java
- Equals i Java
- Nästa vecka: repetition

Kom på föreläsning om systemutveckling!

- Tors. **12/12 kl 15-17 i E:1406** ger Björn Regnell en föreläsning i C:arnas kurs om digitalisering EITA65:
  - **Kravhantering** för mjukvara: hur bestämma vad koda?
  - Hur bäst dra nytta av **öppen källkod**?
- Alla **D-are, W-are** som går pgk är också **hjärtligt välkomna!!!**

Om projektuppgiften

Om din avslutande projektuppgift

Läs noga kompendium Del 1, kapitel -1 avsnitt ”Projektuppgift”!

Några viktiga punkter:

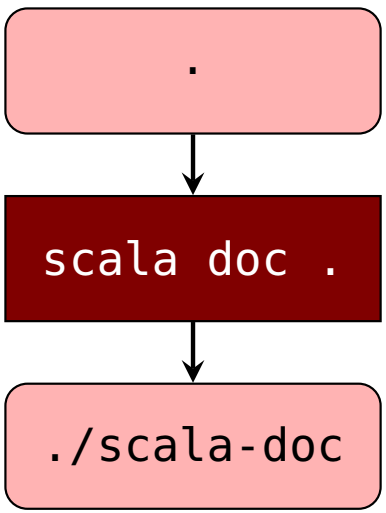
- Mål: skapa ett stort program med många samverkande klasser/moduler.
- Du väljer själv projektuppgift. Börja redan idag att planera ditt arbete!
- Projektet går över **två veckor**. Schemalagda labbar i december är **obligatoriska** för de som har kvar obligatoriska moment.
- I kompendium Del 2, kapitel 13, finns flera förslag att välja bland, men du kan också definiera ett eget projekt som passar just dig.
- Inför redovisningen ska du skapa automatiskt genererad dokumentation utifrån relevanta dokumentationskommentarer för minst hälften av dina publika metoder, enligt instruktioner i Appendix E.
- Redovisning sker i datorsal på schemalagd tid:
  - Förklara hur din kod fungerar.
  - Beskriv framväxten av ditt program.
  - Gå igenom den genererade dokumentationen av din kod.

Projektuppgifter

- `bank`
  - känd domän: skapa bank med transaktionshistorik
  - oföränderlig data tillsammans med tillståndsförändring
- `music`
  - skapa ett enkelt kompositionsverktyg som spelar musik
  - sätta sig in i en domänmodell
- `photo`
  - en enkel variant av photoshop
  - inblick i enkel matrismatematik
- egendefinierat projekt
  - Lagom svårt: ej för enkel uppgift, men ta dig inte vatten över huvudet!
  - Diskutera med en handledare och dokumentera egna uppgiften och dess omfattning och relation till lärandemålen i kursen så att andra handledare och kursansvarig kan förstå varför projektet passar i kursen.
  - Du måste få OK från en handledare innan du startar egendefinierat projekt.

Skapa dokumentation

Scala CLI kan ta dokumentationskommentarerna i källkoden och skapa en webbsajt med dokumentation.



Aktuell katalog med `.scala`-filer

`index.html` och en massa andra filer som ger en hel webbsajt!

Öppna `scala-doc/index.html` i en webbläsare.

Dokumentationskommentarer

För att kod ska bli begriplig för människor är det bra att dokumentera vad den gör. Det finns **tre olika sorters kommentarer**:

```
// Enradskommentarer börjar med dubbla snedstreck
//       men de gäller bara till radslut

/* Flerradskommentarer börjar med
   snedstreck-asterisk
   och slutar med asterisk-snedstreck.  */

/** Dokumentationskommentarer placeras före
 *   t.ex. en funktion och berättar vad den gör
 *   och vad eventuella parametrar används till.
 *   Börjar med snedstreck-asterisk-asterisk.
 *   Varje ny kommentarsrad börjar med asterisk.
 *   Avslutas med asterisk-stjärna.
 */
```

Kommentarer påverkar inte hur maskinen exekverar koden, men hjälper människor att använda koden och verktyg att visa hjälp. Se Appendix E.

Avslutning: munta och tenta

Avslutning och uppsamling

- Gör en **detaljerad** plan dag-för-dag för din kursavslutning.
- Föreläsningar denna vecka: om projekt, om munta, jämföra Scala–Java.
- Föreläsningarna nästa vecka består av repetition och tentaträning.
- Föreläsningarna anpassas efter era önskemål (se #önska i Discord).
- Inga föreläsningar sista läsveckan. **Sista pgk-föreläsningen** är tisdagen den 9:e december i E:A kl 13-15.
- Schematider i **sista läsveckan** är till för **muntligt prov, projektredovisning, redovisning av restlabbar**.
- Det är ok att redovisa projekt och göra munta i förtid i mån av plats. Men PLUGGA NOGA inför muntan!!! (Se nästa slajd.)
- Alla labbar klara innan du muntar (OK munta parallellt med projektet)
- För att få göra den valfria tentan krävs att **alla** obligatoriska moment är **godkända**.
- **Godkänd** grundkurs är **krav** för att få påbörja efterföljande `pfk`.
- **Kolla din status i Canvas** på sida med inklippt ögonblicksbild ur vårt system SAM där du ser vad du har **klarat** och vad du ev. har kvar.

Obligatoriskt muntligt prov

- Syftet med det muntliga provet är att säkerställa att alla som påbörjar efterföljande kurs har tillräckliga **grundkunskaper**, med fokus på **konceptuell förståelse**.
- Det muntliga provet tar mellan 10 och 30 min beroende på hur många frågor handledare behöver ställa för att kunna kan avgöra om du har tillräcklig förståelse.
- Enda tillåtna hjälpmedel: snabbref, och på anmodan av handledare även REPL. Medtag papper och penna!
- **Alla labbar måste vara godkända** innan du får göra provet.
- Det är ok att göra det muntliga provet även innan projektet är klart, i mån av tid om det inte är redovisningskö.
- Om du är godkänd på alla obligatoriska moment får du minst 3:a i kursen och du har då klarat förkunskapskraven för efterföljande kurser.
- Du ber handledare att få göra muntligt prov på samma sätt som du tidigare bett om att få redovisa labbar. Munta sker på plats i datorsal.
- Träna på provet här:
  `https://fileadmin.cs.lth.se/pgk/muntabot`

Gör den valfria tentan!

- **Alla** som kan & vill uppmuntras göra den valfria tentan!
- **Alla** obligatoriska moment måste vara klara innan du får tentera.
- Den valfria tentan ger högre betyg om du klarar gränserna för 4:a el. 5:a.
- Tentan liknar de extentor som finns på kurshemsidan.
- Tentan är på plats och skrivs med papper och penna.
- Enda hjälpmedel: snabbreferensen.
- Snabbreferens beställs i Canvas och hämtas hos `birger.swahn@cs.lth.se`
- Tentan ges en gång om året i januari. Det är tillåtet att ”plussa” ett senare år om du inte är nöjd med ditt betyg.
- **OBS! Obligatorisk anmälan** i LTH:s ordinarie tentasystem. `https://www.student.lth.se/mina-studier/tentamen/`

Jämförelse Scala och Java

Övning `scalajava` och labb `javatext`

Valfria uppgifter som ger dig en **flygande start** i efterföljande kurs:

- Övning `scalajava`:
  - Översätta från Java till Scala och från Scala till Java
  - Undersöka autoboxning (eng. *autoboxing*)
  - Använda `import scala.jdk.CollectionConverters.*`
- Laboration `javatext`:
  - Gör ett textspel för terminalen huvudsakligen i Java men vissa delar i Scala, enligt krav, tips och inspiration i labb-instruktionerna.
- OBS! Scala CLI och sbt kan blanda `.scala` och `.java` i samma projekt.

```
> scala run .
```

Ovan kommando kommer också kompilera `.java`-filer.
Observera Javas filnamnsregler: klass == filnamn, paket == katalog
Bra plats att lägga `.java`-filer: `src/main/java/`
Bra plats att lägga `.scala`-filer: `src/main/scala/`

”Hello world!” i Java.

Ett minimalt huvudprogram i Java:

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

Testa Java i `jshell`

- Java har en motsvarighet till Scalas REPL: kommandot `jshell`

```
> jshell
|  Welcome to JShell -- Version 11.0.11
|  For an introduction type: /help intro

jshell> /help intro
|
|                                  intro
|                                  =====
|
|  The jshell tool allows you to execute Java code, getting immediate results.
|  You can enter a Java definition (variable, method, class, etc), like:  int x = 8
|  or a Java expression, like:  x + x
|  or a Java statement or import.
|  These little chunks of Java code are called 'snippets'.
|
|  There are also the jshell tool commands that allow you to understand and
|  control what you are doing, like:  /list
|
|  For a list of commands: /help

jshell>
```

Grundläggande likheter och skillnader Java–Scala

Några likheter:

- Kompilerar till bytekod som kör på JVM på många olika plattformar.
- Statisk typning: ger snabb maskinkod, kompilatorn kan ge stöd vid förändring av kod (s.k. refactoring) och hittar många buggar redan vid kompilering.

Liknande men viss skillnad:

Java

- **Objektorientering**, men inte ”äkta” (eng. *pure*) eftersom inte alla värden är objekt
- Primitiva typer är inte objekt; representeras effektivt, normalt **utan boxning**
- Visst stöd för **funktionsprogrammering**

Scala

- **Äkta objektorienterat** eftersom alla värden är objekt, även funktioner
- `AnyVal`-instanser är äkta objekt men representeras ändå effektivt, normalt **utan boxning**
- Omfattande stöd för **funktionsprogrammering**

Huvudprogram i Scala och Java

Scala 2

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

Java

```java
public class JMain {
  public static void main(String[] args){
    System.out.println("Hello!");
  }
}
```

Huvudprogram i Scala och Java

Scala 2

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

Java

```
public class JMain {
  public static void main(String[] args){
    System.out.println("Hello!");
  }
}
```

Scala 3

```
@main
def sumFirst(n: Int, xs: Int*): Unit = println(xs.take(n).sum)
```

Loopa genom argumenten i ett Java-huvudprogram

```
> code HelloJavaArgs.java
```

```
public class HelloJavaArgs {
    public static void main(String[] args) {
    int i = 0;
    while (i < args.length) {
      System.out.println(args[i]);
      i = i + 1;
    }
  }
}
```

Kompilera och kör:

```
1 > javac HelloJavaArgs.java
2 > java HelloJavaArgs hej gurka tomat
3 hej
4 gurka
5 tomat
```

HIGHSCORE implementerad i Java

```
import java.util.Scanner;

public class HighScore {
    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        System.out.println("Hur många poäng fick du?");
        int points =  scan.nextInt();
        System.out.println("Vad var highscore före senaste spelet?");
        int highscore = scan.nextInt();
        if (points > highscore) {
            System.out.println("GRATTIS!");
        } else {
            System.out.println("Försök igen!");
        }
    }
}
```

Några saker som finns i Scala men inte i Java

- **`case`**-klasser
- Lokala funktioner
- Metoder som operatorer
- Infix operatornotation
- Defaultargument
- Namngivna argument
- Engångsinitialisering: **`val`**
- Fördröjd initialisering: **`lazy val`**
- Enhetlig access för **`def`**, **`val`**, **`var`**
- Egna setters med **`def`** `namn_=`
- Namnanrop, fördröjd evaluering
- Matchning, mönster och garder
- Klassparametrar, primärkonstruktor
- Singelobjekt: **`object`**
- Kompanjonsobjekt
- Inmixning: **`trait`**
- **`for`**-**`yield`**-uttryck
- Block är uttryck; slipper **`return`**
- Tomma värdet () av typen `Unit`
- `Option`, `Some`, `None` (Java har `Optional` som ger en del, men inte allt...)
- `Try`, `Success`, `Failure`
- Samlingarna i Scalas standardbibliotek, speciellt de **oföränderliga** samlingarna `Vector`, `Map`, `Set`, `List`, etc.
- Innehållslikhet med == för oföränderliga strukturer, inkl. `<` `<=` `>` `>=` på strängar
- **Enhetlig** användning av samlingar **inkl. Array** (förutom innehållslikhet för Array)
- Kontextuella abstraktioner **`given`** **`using`**
- Mer precis synlighet **`private`**`[mypack]`
- Namnändring vid **`import`**
- Flexibel filstruktur och filnamngivning
- Flexibel nästling av klasser, objekt, traits
- Typ-alias och abstrakta typer med **`type`**
- Extensionsmetoder **`extension`**
- ...

Några saker som finns i Java men inte i Scala

+ Snabbare kompilering
+ Mognare verktygsstöd
- Variabeldeklaration utan initialisering
- Förändringsbara parametrar
- C-liknande prefix- och postfix-inkrementering och -dekrementering:
  `i++ ++i i-- --i`
- C-liknande **`for`**-sats
- Semikolon krävs efter alla satser
- **`return`** krävs i alla metoder som har returvärde
- Nyckelordet **`void`**
- Parenteser efter alla metoder
- Specialsyntax för indexering av array `[ ]` ej som i andra samlingar
- Hoppa ut ur loop med **`break`**

  docs.oracle.com/javase/tutorial/
- **`switch`** ”faller igenom” om du glömmer skriva **`break`**
- Kontrollerade undantag (eng. *checked exceptions*) och **`throws`**
- ...

Begränsningar med funktionsprogrammering i Java

Av alla dessa funktionsprogrammeringskoncept i Scala...

- **överlagring**
- **anonyma funktioner**
- **mönstermatchning**
- funktioner etc. på toppnivå
- utelämna tom parameterlista (enhetlig access)
- defaultargument
- namngivna argument
- lokala funktioner
- funktioner som äkta värden
- klammerparentes vid ensam parameter
- multipla parameterlistor
- egendefinierade kontrollstrukturer
- namnanrop (fördröjd evaluering)
- stegade funktioner (”Curry-funktioner”)
- fångad variablelrymd i funktionsobjekt (”closure”)
- ad hoc polymorfism (”typklasser”)
- kontextuella abstraktioner

...kan man i Java endast göra: **överlagring** (”overloading”), **anonyma funktioner** (”lambda”), **mönstermatchning** (de senare två har starka begränsningar)

Läs mer om Java här:
`https://en.wikipedia.org/wiki/Java_version_history`
`https://en.wikipedia.org/wiki/Anonymous_function#Java_Limitations`

Grundtyper i Scala och primitiva typer Java

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

Javas switch-sats

De flesta C-liknande språk (men inte Scala) har en **`switch`**-sats som man kan använda istället för (vissa) nästlade if-else-satser:

```
import java.util.Scanner;

public class Switch {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv grönsak:");
        String g = scan.next();
        switch (g) {
        case "gurka":
            System.out.println("gott!");
            break;
        case "tomat":
            System.out.println("gott!");
            break;
        case "broccoli":
            System.out.println("ganska gott...");
            break;
        default:
            System.out.println("mindre gott...");
            break;
        }
    }
}
```

`switch` från Java 21 har begränsad mönstermatchning `https://docs.oracle.com/en/java/javase/21/language/pattern-matching.html`

Javas switch-sats utan break

Saknad **`break`**-sats ”faller igenom” till efterföljande gren:

```
import java.util.Scanner;

public class SwitchNoBreak {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv grönsak:");
        String g = scan.next();
        switch (g) {
        case "gurka":
        case "tomat":
            System.out.println("gott!");
            break;
        case "broccoli":
            System.out.println("ganska gott...");
            break;
        default:
            System.out.println(" mindre gott...");
            break;
        }
    }
}
```

En glömd **`break`** kan ge svårhittad bugg...

Javas switch-sats med glömd break

```
import java.util.Scanner;

public class SwitchForgotBreak {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv grönsak:");
        String g = scan.next();
        switch (g) {
        case "gurka":
            System.out.println("gott!");
        case "tomat":
            System.out.println("mycket gott!");
            break;
        case "broccoli":
            System.out.println("ganska gott...");
            break;
        default:
            System.out.println("mindre gott...");
            break;
        }
    }
}
```

Javas switch-sats med glömd break

```java
import java.util.Scanner;

public class SwitchForgotBreak {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv grönsak:");
        String g = scan.next();
        switch (g) {
        case "gurka":
            System.out.println("gott!");
        case "tomat":
            System.out.println("mycket gott!");
            break;
        case "broccoli":
            System.out.println("ganska gott...");
            break;
        default:
            System.out.println("mindre gott...");
            break;
        }
    }
}
```

```
> java SwitchForgotBreak
Skriv grönsak:
gurka
gott!
mycket gott!
```

Syntax för variabeldeklaration i Scala och Java

Exempel på variabeldeklarationer i

Scala

```
var i1: Int = 0
var i2 = 0
var i3 = (i2: Int) + 0
var p1: Point = new Point(0, 0)
var p2 = new Point(0, 0)
var (x, y) = (0, 0)
val a = 0
final val Constant = 42
```

- i2 härledd typ; går ej i Java 8,9 men finns med **var** i Java 10
- i3 typ varhelst i uttryck; går ej i Java
- (x, y) mönster i init; går ej i Java
- **val** ger ”engångsinit”; ingen exakt motsvarighet i Java men **final** kan ofta användas i stället

Java

```
int i1 = 0;
var i2 = 0; // från Java 10
int i4;
Point p1 = new Point(0, 0);
var p2 = new Point(0, 0);
final int CONSTANT = 42;
```

- i4 ej explicit init; går ej i Scala

For-sats i Scala och Java

Scala

```scala
val s = "Abbasillen"

// Loopa över index framlänges:

for i <- 0 until s.length do
  println(s(i))


// Loopa över index baklänges:

for i <- s.length-1 to 0 by -1 do
  println(s(i))
```

Java

```java
String s = "Abbasillen";

// Loopa över index framlänges:

for (int i = 0; i < s.length(); i++) {
    System.out.println(s.charAt(i));
}

// Loopa över index baklänges:

for (int i = s.length()-1; i >= 0; i--) {
    System.out.println(s.charAt(i));
}
```

I Scala är `s.indices` att föredra!

For-sats i Scala med indices

Scala

```scala
val s = "Abbasillen"

// Loopa över index framlänges:

for i <- s.indices do
  println(s(i))


// Loopa över index baklänges:

for i <- s.indices.reverse do
  println(s(i))
```

Java

```java
String s = "Abbasillen";

// Loopa över index framlänges:

for (int i = 0; i < s.length(); i++) {
    System.out.println(s.charAt(i));
}

// Loopa över index baklänges:

for (int i = s.length()-1; i >= 0; i--) {
    System.out.println(s.charAt(i));
}
```

For-satser och arrayer i Java

En for-sats i Java har följande struktur:

```
for (initialisering; slutvillkor; inkrementering) {
    sats1;
    sats2;
    ...
}
```

En primitiv heltals-array deklareras så här i Java:

```
int[] xs = new int[42];  // rymmer 42 st heltal, init 0:or
int[] ys = {10, 42, -1}; // initera med 3 st heltal
```

Exempel på for-sats: fyll en array med 1:or

```
for (int i = 0; i < xs.length; i++){
  xs[i] = 1;    // indexera med [i]
}
```

Implementation av SEQ-COPY i Java med `for`-sats

```
1  public class SeqCopyForJava {
2
3      public static int[] arrayCopy(int[] xs){
4          int[] result = new int[xs.length];
5          for (int i = 0; i < xs.length; i++){
6              result[i] = xs[i];
7          }
8          return result;
9      }
10
11     public static String test(){
12         int[] xs = {1, 2, 3, 4, 42};
13         int[] ys = arrayCopy(xs);
14         for (int i = 0; i < xs.length; i++){
15             if (xs[i] != ys[i]) {
16                 return "FAILED!";
17             }
18         }
19         return "OK!";
20     }
21
22     public static void main(String[] args) {
23         System.out.println(test());
24     }
25 }
```

Lite syntax och semantik för Java:

- En Java-klass med enbart statiska medlemmar motsvarar ett singelobjekt i Scala.
- Typen kommer **före** namnet.
- Man **måste** skriva **`return`**.
- Man **måste** ha semikolon efter varje sats.
- Metodnamn **måste** följas av parenteser; om inga parametrar finns används `()`
- En array i Java är inget vanligt objekt, men har ett ”attribut” `length` som ger antal element.
- **Övning**: skriv om med Javas **`while`**-sats i stället.

Element för element med speciell for-each-sats i Java

Scala

```scala
val s = "Abbasillen"

// Loopa över alla tecken:

for ch <- s do println(ch)
```

Java

```java
String s = "Abbasillen";

// Loopa över alla tecken:

for (char ch: s.toCharArray()) {
  System.out.println(ch);
}
```

Element för element med speciell for-each-sats i Java

Scala

```scala
val s = "Abbasillen"

// Loopa över alla tecken:

for ch <- s do println(ch)
```

Java

```java
String s = "Abbasillen";

// Loopa över alla tecken:

for (char ch: s.toCharArray()) {
  System.out.println(ch);
}
```

`s.foreach(println)` går ej i Java men från Java 8 finns metoden `chars` som ger en `IntStream` och då kan man:

```java
str.chars().forEachOrdered(i -> System.out.println((char) i));
```

Typisk utformning av Java-klass

Typisk ”anatomi” hos en Java-klass:

```
public class Klassnamn {
    attribut, normalt privata
    konstruktorer, normalt publika
    metoder: publika getters, och vid förändringsbara objekt även setters
    metoder: privata abstraktioner för internt bruk
    metoder: publika abstraktioner tänkta att användas av klientkoden
}
```

Statiska medlemmar i Java

- Man kan **inte** deklarera explicita singelobjekt i Java och det finns inget nyckelord **`object`**.
- I stället kan man deklarera **statiska medlemmar** i en klass med Java-nyckelordet **`static`**.
- Exempel på hur vi ska göra detta inuti en klassen `JPerson`:

```
public static final int ADULT_AGE = 18;
```

- Effekten blir den samma som ett singelobjekt i Scala:
  - Alla statiska medlemmar i en Java-klass allokeras automatiskt och hamnar i en egen singulär ”klassinstans” som existerar oberoende av de dynamiska instanserna.
  - De statiska medlemmarna accessas med punktnotation genom klassnamnet, **utan `new`**:

```
System.out.println(JPerson.ADULT_AGE);
```

Exempel: oföränderlig klass i Scala och Java

```
class Person(val name: String, val age: Int):
  def isAdult = age >= Person.AdultAge

object Person:
  val AdultAge = 18
```

Exempel: oföränderlig klass i Scala och Java

```scala
class Person(val name: String, val age: Int):
  def isAdult = age >= Person.AdultAge

object Person:
  val AdultAge = 18
```

```java
public class JPerson {
    private String name;
    private int age;
    public static final int ADULT_AGE = 18;

    public JPerson(String name, int age) {
      this.name = name;
      this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public boolean isAdult() {
        return age >= ADULT_AGE;
    }
}
```

Lär dig detta mönster för en typisk Java-klass utantill så du snabbt får grejerna på plats!

Exempel: oföränderlig klass i Scala och Java

```scala
class Person(val name: String, val age: Int):
  def isAdult = age >= Person.AdultAge

object Person:
  val AdultAge = 18
```

```java
public class JPerson {
    private String name;
    private int age;
    public static final int ADULT_AGE = 18;

    public JPerson(String name, int age) {
      this.name = name;
      this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public boolean isAdult() {
        return age >= ADULT_AGE;
    }
}
```

Övning:
Gör `Person` + `JPerson` **förändringsbara** så att namnet och åldern går att uppdatera och följande krav uppfylls:

- namnet ska ges vid konstruktion,
- åldern ska initieras till 0 vid konstr.,
- åldern ska aldrig kunna bli negativ.

Lär dig detta mönster för en typisk Java-klass utantill så du snabbt får grejerna på plats!

Exempel: Scala-klassen Complex

```
class Complex(val re: Double, val im: Double):
  def r  = math.hypot(re, im)
  def fi = math.atan2(im, re)
  def +(other: Complex) = new Complex(re + other.re, im + other.im)
  override def toString = s"$re + $im${Complex.imSymbol}"

object Complex:
  var imSymbol = 'i'
```

Scala: `https://github.com/lunduniversity/introprog/blob/master/compendium/examples/complex7.scala`

Java: `https://github.com/lunduniversity/introprog/blob/master/compendium/examples/JComplex.scala`

Exempel: Motsvarande Java-klassen JComplex

```java
public class JComplex { // man kan ej deklarera klassparametrar i Java
    private double re;  // initialiseras i konstruktorn nedan
    private double im;  // initialiseras i konstruktorn nedan
    public static char imSymbol = 'i'; // publikt förändringsbart attribut (ovanligt i Java)

    public JComplex(double real, double imag){ // konstruktor, körs vid new
        re = real;
        im = imag;
    }

// en "getter" som ger attributvärdet, hindrar förändring av re
    public double getRe(){
        return re;
    }

// ej bruklig formattering i Java, så metoder blir minst 3 rader
    public double getIm(){ return im; }

    public double getR(){
        return Math.hypot(re, im);       // Math med stort M i Java
    }

    public double getFi(){
        return Math.atan2(re, im);
    }

// Javametodnamn får ej ha operatortecken t.ex. +, därav namnet add
    public JComplex add(JComplex other){
        return new JComplex(re + other.getRe(), im + other.getIm());
    }

    @Override public String toString(){
        return re + " + " + im + imSymbol;
    }
}
```

Exempel: Använda JComplex i Scala-kod

```
$ javac JComplex.java
$ scala
Welcome to Scala 2.12.9 (OpenJDK 64-Bit Server VM, Java 1.8.0_222).
Type in expressions for evaluation. Or try :help.

scala> val jc1 = JComplex(3, 4)
jc1: JComplex = 3.0 + 4.0i

scala> val polarForm = (jc1.getR, jc1.getFi)
polarForm: (Double, Double) = (5.0,0.6435011087932844)

scala> val jc2 = JComplex(1, 2)
jc2: JComplex = 1.0 + 2.0i

scala> jc1 add jc2
res0: JComplex = 4.0 + 6.0i
```

Exempel: Använda JComplex i Java-kod

```
public class JComplexTest {
    public static void main(String[] args){
        JComplex jc1 = new JComplex(3,4);
        String polar = "(" + jc1.getR() + ", " + jc1.getFi() + ")";
        System.out.println("Polär form: " + polar);
        JComplex jc2 = new JComplex(1,2);
        System.out.println(jc1.add(jc2));
    }
}
```

- I Java måste man skriva **new**.
- I Java måste man skriva **tomma parentes-par** efter metodnamnet vid **anrop av parameterlösa metoder**.
- **Tupler finns inte** i Java, så det går inte på ett enkelt sätt att skapa par av värden som i Scala; ovan görs polär form till en sträng för utskrift.
- **Operatornotation för metoder finns inte** i Java, så man måste i Java använda punktnotation och skriva: `jc1.add(jc2)`

Exempel: Förändringsbar klass i Scala och Java

```
class MutablePerson(var name: String):
  private var _age = 0

  def age: Int = _age

  def age_=(a: Int): Unit =
    if (a >= 0) _age = a else _age = 0
      // eller hellre kasta undantag?

  def isAdult: Boolean =
    age >= MutablePerson.AdultAge

object MutablePerson:
  val AdultAge = 18
```

Exempel: Förändringsbar klass i Scala och Java

```scala
class MutablePerson(var name: String):
  private var _age = 0

  def age: Int = _age

  def age_=(a: Int): Unit =
    if (a >= 0) _age = a else _age = 0
      // eller hellre kasta undantag?

  def isAdult: Boolean =
    age >= MutablePerson.AdultAge

object MutablePerson:
  val AdultAge = 18
```

```java
public class JMutablePerson {
    private String name;
    private int age = 0;
    public static final int ADULT_AGE = 18;

    public JMutablePerson(String name) {
      this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        if (age >= 0) {
          this.age = age;
        } else {
          this.age = 0;
        }
    }

    public boolean isAdult() {
        return age >= ADULT_AGE;
    }
}
```

Scalas ”case-klass-godis” finns inte i Java

En oföränderlig datatyp implementeras i **Scala** helst som en

Scalas ”case-klass-godis” finns inte i Java

En oföränderlig datatyp implementeras i **Scala** helst som en **`case`**-klass:

```
case class Person(name: String, age: Int):
  def isAdult = age >= Person.AdultAge

object Person:
  val AdultAge = 18
```

Scalas ”case-klass-godis” finns inte i Java

En oföränderlig datatyp implementeras i **Scala** helst som en `case`-klass:

```
case class Person(name: String, age: Int):
  def isAdult = age >= Person.AdultAge

object Person:
  val AdultAge = 18
```

En oföränderlig datatyp i **Java** med **motsvarande** funktionalitet kräver egen implementation av dessa metoder:

- en getter för varje attribut
- `equals`
- `hashCode` (förklaras i forts.kurs)
- `apply`
  (men man kallar nog den `create` el. likn.; namnet måste ju skrivas)
- `toString`
- `copy`
  (men det finns ju inte namngivna parametrar och default-argument så denna blir osmidig)
- `unapply`
  (men det finns ju inte mönstermatchning så denna struntar man nog i)

Array i Java

Repetition: Den primitiva typen Array i JVM

- Primitiva arrayer (`Array` i Scala, `[]` i Java) har **fördelar**:[1]
  - Det är den snabbaste indexerbara datastrukturen i JVM: att läsa och uppdatera ett element på en viss plats är mycket effektivt om man vet platsens index.
  - Fungerar lika bra med både primitiva värden och objektreferenser
- ... men också **nackdelar**:
  - Man måste bestämma sig för antalet element som ska allokeras när man gör `new`.
  - Man kan ta i lite extra när man allokerar om man behöver plats för fler senare, men då måste man hålla reda på hur många platser man använder och veta var nästa lediga plats finns.
  - Det är krångligt att stoppa in (eng. *insert*) och ta bort (eng. *delete*) element.
  - Vill man ha fler platser måste man allokera en helt ny, större array och kopiera över alla befintliga element.

[1] stackoverflow.com/questions/2843928/benefits-of-arrays

Syntax för Array i Scala och Java

Scala

```scala
var xs = Array(42, 43, 44)




val n = xs.length

var strings = new Array[String](42)
// eller      Array.ofDim[String](42)
// eller      Array.fill(42)(null: String)

strings(0) = "first"

strings(1) = "second"
```

Java

```java
int[] xs = new int[]{42, 43, 44};

// samma som ovan, men kortare:
int[] xs2 = {42, 43, 44};

int n = xs.length;  // EJ length()

String[] strings = new String[42];



strings[0] = "first";

strings[1] = "second";
```

Exempel: Polygon med primitiv array i Java

```
1  public class Polygon {
2      private Point[] vertices; // array med hörnpunkter
3      private int n;            // antalet hörnpunkter
4
5      /** Skapar en polygon */
6      public Polygon() {
7          vertices = new Point[1];
8          n = 0;
9      }
10
11     ...
```

Polygon med primitiv array i Java: stoppa in sist och vid behov skapa mer plats

Implementera:

```
private void extend()                    // dubbla storleken
public void addVertex(int x, int y)  // lägg till hörnpunkt
```

Polygon med primitiv array i Java: stoppa in sist och vid behov skapa mer plats

Implementera:

```
private void extend()                  // dubbla storleken
public void addVertex(int x, int y)    // lägg till hörnpunkt
```

```
1   private void extend(){
2       Point[] oldVertices = vertices;
3       vertices = new Point[2 * vertices.length]; // skapa dubbel plats
4       for (int i = 0; i < oldVertices.length; i++) {  // kopiera
5           vertices[i] = oldVertices[i];
6       }
7   }
8
9   public void addVertex(int x, int y) {
10      if (n == vertices.length) extend();
11      vertices[n] = new Point(x, y);
12      n++;
13  }
```

Polygon med primitiv array i Java: stoppa in mitt i på angiven plats

Implementera:

```
/** Sätt in hörnpunkt på plats pos */
public void insertVertex(int pos, int x, int y)
```

Polygon med primitiv array i Java: stoppa in mitt i på angiven plats

Implementera:

```
/** Sätt in hörnpunkt på plats pos */
public void insertVertex(int pos, int x, int y)
```

```
1    public void insertVertex(int pos, int x, int y) {
2        if (n == vertices.length) extend();   // utöka vid behov
3        for (int i = n; i > pos; i--) {       // flytta element bakifrån
4            vertices[i] = vertices[i - 1];
5        }
6        vertices[pos] = new Point(x, y);
7        n++;
8    }
```

Scanna filer och strängar med `java.util.Scanner`

- I Scala kan man läsa från fil så här (se quickref sid 3 längst ner):

```
val names = scala.io.Source.fromFile("src/names.txt").getLines.toVector
```

- Klassen `java.util.Scanner` kan också läsa från fil (se Java Snabbref sid 4):

```
def readFromFile(fileName: String): Vector[String] = {
  val file = new java.io.File(fileName)
  val scan = new java.util.Scanner(file)
  val buffer = scala.collection.mutable.ArrayBuffer.empty[String]
  while (scan.hasNext) {
    buffer += scan.next
  }
  scan.close
  buffer.toVector
}
```

- Med `new java.util.Scanner(System.in)` kan man även scanna tangentbordet.
- Med `new java.util.Scanner("hej 42")` kan man även scanna en sträng.
- Scanna `Int` och `Double` med metoderna `nextInt` och `nextDouble`.
  Se doc: docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

Exempel: Scanner

```
scala> val scan = new java.util.Scanner("hej 42 42.0   42 slut")

scala> scan.hasNext
res0: Boolean = true

scala> scan.hasNextInt
res1: Boolean = false

scala> scan.next
res2: String = hej

scala> scan.hasNextInt
res3: Boolean = true

scala> scan.nextInt
res4: Int = 42

scala> while (scan.hasNext) println(scan.next)
42.0
42
slut
```

Använda Java-samlingar i Scala med `CollectionConverters`

Med hjälp av **`import`** `scala.jdk.CollectionConverters.*`
får du smidig **interoperabilitet** med Java och dess standardbibliotek,
speciellt metoderna **`asJava`** och **`asScala`**:

```
1  scala> import scala.jdk.CollectionConverters.*
2
3  scala> Vector(1,2,3).asJava
4  res0: java.util.List[Int] = [1, 2, 3]
5
6  scala> val xs = new java.util.ArrayList[String]()
7  xs: java.util.ArrayList[String] = []
8
9  scala> xs.add("hej")
10 res1: Boolean = true
11
12 scala> xs.asScala
13 res2: scala.collection.mutable.Buffer[String] = Buffer(hej)
```

Läs mer här: `https://docs.scala-lang.org/overviews/collections-2.13/conversions-between-java-and-scala-collections.html`

Generiska samlingar i Java

- Från och med version 5 av Java (2004) så introducerades **generics** vilket möjliggör skapandet av klasser som kan erbjuda generell behandling av olika typer av objekt.
- Generiska klasser i Java känns igen med syntaxen `Klassnamn<Typ>`, till exempel `ArrayList<Point>`
- Fördjupning: docs.oracle.com/javase/tutorial/extra/generics/intro.html, mer om detta i fördjupningskursen.

Om ArrayList i Java

`java.util.ArrayList` liknar `scala.collection.mutable.ArrayBuffer` som båda har dessa fördelar:

- Lagrar sina element internt i snabbindexerade primitiva arrayer.
- Fungerar för alla typer av objekt.
- Utökar samlingens storlek av sig själv vid behov.

Det finns dock vissa nackdelar med `ArrayList` i Java (som inte gäller för `ArrayBuffer` i Scala):

- Fungerar **inte** rakt av med primitiva typer `int`, `double`, `char`, ... (men det finns sätt komma runt detta, tack vare s.k. wrapper-klasser och autoboxning; mer om detta snart)
- Namnet `ArrayList` är inte helt lyckat, eftersom ordet ”lista” normalt används för länkade snarare än array-liknande strukturer.

Polygon med ArrayList i Java

Klassen `Polygon`, nu med ett attribut av typen `ArrayList<Point>`:

```
public class Polygon {
    private ArrayList<Point> vertices; // lista med hörnpunkter

    /** Skapar en polygon */
    public Polygon() {
        vertices = new ArrayList<Point>();
    }

    ...
```

Det behövs inget attribut n eftersom vi inte själva behöver hålla reda på antalet allokerade platser: allokering, insättning, och utökning av antalet platser sköts helt automatiskt av `ArrayList`-klassen vid behov.

Några viktiga operationer på ArrayList<E>

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/ArrayList.html

```
/** Tar reda på elementet på plats pos */
E get(int pos);

/** Lägger in objektet obj sist */
void add(E obj);

/** Lägger in obj på plats pos; efterföljande flyttas */
void add(int pos, E obj);

/** Tar bort elementet på plats pos och returnerar det */
E remove(int pos);

/** Tar reda på antalet element i listan */
int size();
```

Lär dig vad som finns om ArrayList i snabbreferensen för Java
Överkurs för den nyfikne: kolla implementation av ArrayList här:

https://hg.openjdk.org/jdk8/jdk8/jdk/file/tip/src/share/classes/java/util/ArrayList.java#l106

Övning ArrayList: new och add

Skriv Java-kod som skapar en lista med element av typen `Point` och lägger in tre punkter i listan med koordinaterna: (50, 50), (50,10) och (30, 40).

Övning ArrayList: new och add

Skriv Java-kod som skapar en lista med element av typen `Point` och lägger in tre punkter i listan med koordinaterna: (50, 50), (50,10) och (30, 40).

Lösning:

```
ArrayList<Point> vertices = new ArrayList<Point>();
vertices.add(new Point(50, 50));
vertices.add(new Point(50, 10));
vertices.add(new Point(30, 40));
```

For-each-sats i Java:

- Antag att vi vill gå igenom alla element i en lista.

```
ArrayList<String> words = new ArrayList<String>();
```

- Det finns två olika typer av **`for`**-satser i Java som kan göra detta:
  - Vanlig **`for`**-sats:

```
for (int i = 0; i < words.size(); i++) {
    System.out.println(i + ": " + words.get(i));
}
```

  - Så kallad **for-each-sats** med denna syntax:
    `for` `(Elementtyp element: samling) { ... }`

    Exempel:

```
for (String s: words) {
    System.out.println(s);
}
```

    Men vi får ingen indexvariabel då...

Polygon med ArrayList: metoderna blir enklare

```
    public void addVertex(int x, int y) {
        vertices.add(new Point(x, y));
    }

    public void move(int dx, int dy) {
        for (Point p: vertices){
            p.move(dx, dy);
        }
    }

    public void insertVertex(int pos, int x, int y) {
        vertices.add(pos, new Point(x, y));
    }

    public void removeVertex(int pos) {
        vertices.remove(pos);
    }
```

Se hela lösningen här:
compendium/examples/scalajava/list/Polygon.java

Polygon med ArrayList: iterera över alla hörnpunkter i draw med indexering

```
public void draw(SimpleWindow w) {
    if (vertices.size() == 0) {
        return;
    }
    Point start = vertices.get(0);
    w.moveTo(start.getX(), start.getY());
    for (int i = 1; i < vertices.size(); i++) {
        w.lineTo(vertices.get(i).getX(),
                 vertices.get(i).getY());
    }
    w.lineTo(start.getX(), start.getY());
}
```

Övning: Skriv om med for-each-sats.

Polygon med ArrayList: iterera över alla hörnpunkter i draw med foreach-sats

```
    public void draw(SimpleWindow w) {
        if (vertices.size() == 0) {
            return;
        }
        Point start = vertices.get(0);
        w.moveTo(start.getX(), start.getY());
        for (Point p: vertices){
            w.lineTo(p.getX(), p.getY());
        }
        w.lineTo(start.getX(), start.getY());
    }
```

Se hela lösningen här:
compendium/examples/scalajava/list/Polygon.java

Övning ArrayList: implementera metoden hasVertex

Skriv kod som implementerar denna metod i klassen `Polygon`:

```
/** Undersöker om polygonen har någon hörnpunkt med koordinaterna x, y. */
public boolean hasVertex(int x, int y) {
    ???
}
```

Lösning ArrayList: implementera metoden hasVertex

```
public boolean hasVertex(int x, int y) {
    for (Point p: vertices) {
        if (p.getX() == x && p.getY() == y) {
            return true;
        }
    }
    return false;
}
```

For-each-sats med array

For-each-sats fungerar även med primitiv array:

```
String[] stringArray = {"hej", "på", "dej"};
for (String s: stringArray) {
    System.out.println(s);
}
```

Autoboxning i Java

Generiska klasser (t.ex. ArrayList) med primitiva typer

Detta går tyvärr **INTE** i Java:

~~`ArrayList<int> list = new ArrayList<int>();`~~

Generiska klasser (t.ex. ArrayList) med primitiva typer

Detta går tyvärr **INTE** i Java:

~~`ArrayList<int> list = new ArrayList<int>();`~~

- Hur gör man om man vill ha heltalselement (eller andra primitiva värden) i en generisk samling?
- Javas lösning på problemet består av två delar:
  - Klasser som packar in primitiva typer, (eng. *wrapper classes*)
  - Speciella regler för implicita konverteringar, s.k. ”auto-boxing” (eng. *Boxing / Unboxing conversions*)

Ofta fungerar det fint, men det finns fallgropar.

(Om du är nyfiken på alla intrikata detaljer, se Java tutorial och Javaspecifikationen.)

Wrapper-klassen `Integer`

En skiss av klassen `Integer`
(ligger i paketet `java.lang` och importeras därmed implicit):

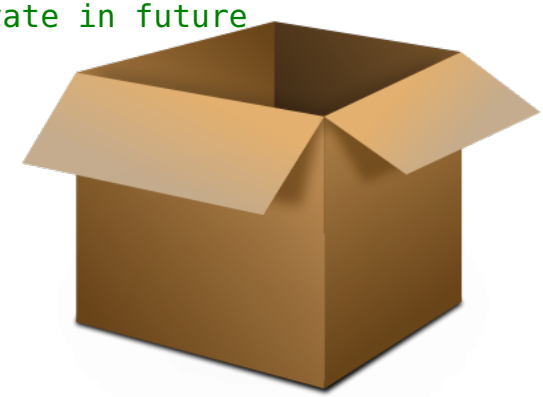
```
public class Integer {
    private int value;

    public static final MIN_VALUE = -2147483648;
    public static final MAX_VALUE = 2147483647;

    public Integer(int value) { // deprecated; will be private in future
        this.value = value;
    }

    public static valueOf(int value) {
      return new Integer(value)
    }

    public int intValue() {
        return value;
    }
    ...
}
```



Javadoc för klasen `Integer` finns här:

`http://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html`

Wrapper-klasser i `java.lang`

Primitiv typ	**Inpackad typ**
boolean	Boolean
byte	Byte
short	Short
char	Character
int	Integer
long	Long
float	Float
double	Double

Övning: primitiva versus inpackade typer

- Deklarera en variabel med namnet `gurka` av den primitiva heltalstypen och initiera den till värdet 42.
- Deklarera en referensvariabel med namnet `tomat` av den inpackade (”wrappade”) heltalstypen och initiera den till värdet 43.
- Rita hur det ser ut i minnet.

Exempel: Lista med heltal utan autoboxning

```
import java.util.ArrayList;
import java.util.Scanner;

public class TestIntegerList {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<Integer>();
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv heltal med blank emellan. Avsluta med <CTRL+D>");
        while (scan.hasNextInt()) {
            int nbr = scan.nextInt();
            Integer obj = Integer.valueOf(nbr);
            list.add(obj);
        }
        System.out.println("Dina heltal i omvänd ordning:");
        for (int i = list.size() - 1; i >= 0; i--) {
            Integer obj = list.get(i);
            int nbr = obj.intValue();
            System.out.println(nbr);
        }
    }
}
```

Koden finns här: compendium/examples/scalajava/TestIntegerList.java

Specialregler för wrapper-klasser

- Om ett `int`-värde förekommer där det behövs ett `Integer`-objekt, så lägger kompilatorn **automatiskt** ut kod som skapar ett `Integer`-objekt som packar in värdet.
- Om ett `Integer`-objekt förekommer där det behövs ett `int`-värde, lägger kompilatorn **automatiskt** ut kod som anropar metoden `intValue()`.

Samma gäller mellan alla primitiva typer och dess wrapper-klasser:

```
boolean   ⇔   Boolean
   byte   ⇔   Byte
  short   ⇔   Short
   char   ⇔   Character
    int   ⇔   Integer
   long   ⇔   Long
  float   ⇔   Float
 double   ⇔   Double
```

Exempel: Lista med heltal och autoboxning

```
import java.util.ArrayList;
import java.util.Scanner;

public class TestIntegerListAutoboxing {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<Integer>();
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv heltal med blank emellan. Avsluta med <CTRL+D>");
        while (scan.hasNextInt()) {
            int nbr = scan.nextInt();
            list.add(nbr);   // motsvarar: list.add(Integer.valueOf(nbr));
        }
        System.out.println("Dina heltal i omvänd ordning:");
        for (int i = list.size() - 1; i >= 0; i--) {
            int nbr = list.get(i);  // motsvarar: int nbr = list.get(i).intValue();
            System.out.println(nbr);
        }
    }
}
```

Koden finns här: scalajava/generics/TestIntegerListAutoboxing.java

Fallgropar vid autoboxning

- Jämförelser med == och `!=`
  compendium/examples/scalajava/generics/TestPitfall1.java
- Kompilatorn hittar inte förväxlad parameterordning, t.ex. `add(pos, item)` i fel ordning: ~~`add(item, pos)`~~
  compendium/examples/scalajava/generics/TestPitfall2.java

Equals i Java

Referenslikhet eller innehållslikhet i Scala och Java

Det finns två **principiellt olika** sorters **likhet**:

- **Referenslikhet** (eng. *reference equality*): två referenser anses lika om de refererar till **samma instans** i minnet.
- **Innehållslikhet**, ä.k. strukturlikhet (eng. *structural equality*): två referenser anses lika om de refererar till objekt med **samma innehåll**.

Referenslikhet eller innehållslikhet i Scala och Java

Det finns två **principiellt olika** sorters **likhet**:

- **Referenslikhet** (eng. *reference equality*): två referenser anses lika om de refererar till **samma instans** i minnet.
- **Innehållslikhet**, ä.k. strukturlikhet (eng. *structural equality*): två referenser anses lika om de refererar till objekt med **samma innehåll**.
- I Scala finns flera metoder som testar likhet:
  - metoden `eq` testar **referenslikhet** och `r1.eq(r2)` ger **`true`** om `r1` och `r2` refererar till **samma** instans.
  - metoden `ne` testar referens**o**likhet och `r1.ne(r2)` ger **`true`** om `r1` och `r2` refererar till **olika** instanser.
  - metoden == som anropar metoden `equals` som default testar referenslikhet men som **kan överskuggas** om man **själv vill bestämma** om det ska vara referenslikhet eller strukturlikhet.

Referenslikhet eller innehållslikhet i Scala och Java

Det finns två **principiellt olika** sorters **likhet**:

- **Referenslikhet** (eng. *reference equality*): två referenser anses lika om de refererar till **samma instans** i minnet.
- **Innehållslikhet**, ä.k. strukturlikhet (eng. *structural equality*): två referenser anses lika om de refererar till objekt med **samma innehåll**.
- I Scala finns flera metoder som testar likhet:
  - metoden `eq` testar **referenslikhet** och `r1.eq(r2)` ger **`true`** om `r1` och `r2` refererar till **samma** instans.
  - metoden `ne` testar referens**o**likhet och `r1.ne(r2)` ger **`true`** om `r1` och `r2` refererar till **olika** instanser.
  - metoden == som anropar metoden `equals` som default testar referenslikhet men som **kan överskuggas** om man **själv vill bestämma** om det ska vara referenslikhet eller strukturlikhet.
- Scalas **standardbibliotek** och **grundtyperna** `Int`, `String` etc. testar **innehållslikhet** genom metoden ==

Referenslikhet eller innehållslikhet i Scala och Java

Det finns två **principiellt olika** sorters **likhet**:

- **Referenslikhet** (eng. *reference equality*): två referenser anses lika om de refererar till **samma instans** i minnet.
- **Innehållslikhet**, ä.k. strukturlikhet (eng. *structural equality*): två referenser anses lika om de refererar till objekt med **samma innehåll**.
- I Scala finns flera metoder som testar likhet:
  - metoden `eq` testar **referenslikhet** och `r1.eq(r2)` ger **`true`** om `r1` och `r2` refererar till **samma** instans.
  - metoden `ne` testar referens**o**likhet och `r1.ne(r2)` ger **`true`** om `r1` och `r2` refererar till **olika** instanser.
  - metoden == som anropar metoden `equals` som default testar referenslikhet men som **kan överskuggas** om man **själv vill bestämma** om det ska vara referenslikhet eller strukturlikhet.
- Scalas **standardbibliotek** och **grundtyperna** `Int`, `String` etc. testar **innehållslikhet** genom metoden ==
- I **Java** är det **annorlunda**: symbolen == är ingen metod i Java utan **specialsyntax** som vid instansjämförelse alltid testar **referenslikhet**, medan metoden `equals` kan överskuggas med valfri likhetstest.

Fallgrop med samlingar: metoden contains kräver implementation av equals

Antag att vi vill implementera `hasVertex()` i klassen `Polygon` genom att använda metoden `contains` på en lista. Hur gör vi då?

Fallgrop med samlingar: metoden contains kräver implementation av equals

Antag att vi vill implementera `hasVertex()` i klassen `Polygon` genom att använda metoden `contains` på en lista. Hur gör vi då?

```
public boolean hasVertex(int x, int y) {
    return vertices.contains(new Point(x, y)); // FUNKAR INTE om ...
    // ... inte Point har en equals som kollar innehållslikhet
}
```

Vi behöver implementera metoden `equals(Object obj)` i klassen `Point` som kollar innehållslikhet och ersätter den `equals` som finns i `Object` som kollar referenslikhet, eftersom metoden `contains` i klassen `ArrayList` anropar `equals` när den letar igenom listan efter lika objekt.
Se exempel här: compendium/examples/scalajava/generics/TestPitfall3.java

Det krävs ofta även att man även ersätter `hashCode`, mer om det i fortsättningskursen.

Fullständigt recept för `equals`

För den nyfikne inför fortsättningskursen efter jul:

Läs om fallgropar för att implementera equals i **Java** här:
www.artima.com/lejava/articles/equality.html

Läs receptet för att implementera equals i **Scala** här:
www.artima.com/pins1ed/object-equality.html#28.4

Villkorsuttryck i Java

Det går att använda villkorsuttryck i Java, men med syntax från språket C:

Scala

```
var r = math.random()
var answer = if r > 0.5 then 42 else 0
```

Java

```
double r = Math.random();
int answer = (r > 0.5) ? 42 : 0;
```

Typtest och typkonvertering

Scala

```scala
var x = "hej"

var isString = x.isInstanceOf[String]

var y = 42

var z = y.asInstanceOf[Double]
```

Java

```java
String x = "hej";

boolean isString = x instanceof String;

int y = 42;

double z = (double) y;
```

Typtest och typkonvertering

Scala

```scala
var x = "hej"

var isString = x.isInstanceOf[String]

var y = 42

var z = y.asInstanceOf[Double]
```

Java

```java
String x = "hej";

boolean isString = x instanceof String;

int y = 42;

double z = (double) y;
```

Detta görs ju i Scala bäst med **`match`** och typmönster!

Regler för överskuggning i Java

```
http://docs.oracle.com/javase/tutorial/java/
IandI/override.html
```

Fånga undantag i Scala och Java

Typisk skillnad mellan Scala och Java:
konstruktioner som är **uttryck** i Scala är ofta **satser** i Java.

Scala

```
val a = try 2 / 0 catch
  case e: ArithmeticException => 0

val b = try 4 / 2 catch
  case e: ArithmeticException => 0
```

Java

```
int a;
try {
    a = 2 / 0;
} catch (ArithmeticException e) {
    a = 0;
}

int b;
try {
    b = 4 / 2;
} catch (ArithmeticException e) {
    b = 0;
}
```

Gränssnittet `List` i Java

- I Java finns inte **`trait`** och inmixning.
- I stället finns **`interface`** som liknar **`trait`** men är mer begränsad vad gäller vilka medlemmar som får finnas.
- Man kan bara göra **`extends`** på exakt en annan klass, men man kan i Java göra **`implements`** på flera **`interface`**.
- Exempel:

```
public class ArrayList<E> extends AbstractList<E>
    implements List<E>, RandomAccess, Cloneable, java.io.Serializable
```

- Att implementera ett gränssnitt innebär att uppfylla ett kontrakt som utlovar att vissa speciella metoder finns tillgängliga.
- Gränssninttet `List` uppfylls av en av dess implementationer `ArrayList` på liknande sätt i Scala där gränssnittet `Seq` uppfylls av `Vector` etc.
  `List<String> xs = new ArrayList<String>();`
- Liknande exempel från övningen Hangman:
  `Set<Character> found = new HashSet<Character>();`
- En Scala-trait med enbart abstrakta medlemmar kompileras till ett Java-interface i JVM bytekode.
- Mer om gränssnitt i Java i fördjupningskursen.

Det går inte att skapa generisk Array i Java

- I Java kan man **inte** skapa en primitiv array av godtycklig typ enligt generisk typparameter: ~~`T[] xs = new T[42]`~~
- Man måste istället skapa en array av den mest generella referenstypen:
  `Object[] xs = new Object[42]`
  och sedan typtesta och typkonvertera under körtid; se t.ex. implementationen av `ArrayList` på rad 119:
  http://developer.classpath.org/doc/java/util/ArrayList-source.html

Det går inte att skapa generisk Array i Java

- I Java kan man **inte** skapa en primitiv array av godtycklig typ enligt generisk typparameter: ~~`T[] xs = new T[42]`~~
- Man måste istället skapa en array av den mest generella referenstypen:
  `Object[] xs = new Object[42]`
  och sedan typtesta och typkonvertera under körtid; se t.ex. implementationen av `ArrayList` på rad 119:
  http://developer.classpath.org/doc/java/util/ArrayList-source.html

- Detta går faktiskt att göra i Scala med hjälp av `reflect.ClassTag`

Det går inte att skapa generisk Array i Java

- I Java kan man **inte** skapa en primitiv array av godtycklig typ enligt generisk typparameter: ~~`T[] xs = new T[42]`~~
- Man måste istället skapa en array av den mest generella referenstypen:
  `Object[] xs = new Object[42]`
  och sedan typtesta och typkonvertera under körtid; se t.ex. implementationen av `ArrayList` på rad 119:
  http://developer.classpath.org/doc/java/util/ArrayList-source.html

- Detta går faktiskt att göra i Scala med hjälp av `reflect.ClassTag` så här:

```
scala> def fyll[T](n: Int, x: T): Array[T] = Array.fill(n)(x)
-- Error:
1 |def fyll[T](n: Int, x: T): Array[T] = Array.fill(n)(x)
  |                                                   ^
  |  No ClassTag available for T

scala> def fyll[T: reflect.ClassTag](n: Int, x: T): Array[T] = Array.fill(n)(x)

scala> fyll(42, "hej")
res2: Array[String] = Array(hej, hej, hej, hej, hej, hej, hej, hej, hej, hej, hej, h
```

Jämföra strängar i Java

- I Java kan man **inte** jämföra strängar med operatorerna <, <=, >, och >=
- Dessutom ger operatorerna == och != *inte* innehålls(o)likhet utan **referens(o)likhet** :(
- För innehållslikhet måste du använda metoderna `equals` och `compareTo`.

Jämföra strängar i Java

- I Java kan man **inte** jämföra strängar med operatorerna <, <=, >, och >=
- Dessutom ger operatorerna == och != *inte* innehålls(o)likhet utan **referens(o)likhet** :(
- För innehållslikhet måste du använda metoderna `equals` och `compareTo`.
- `s1.compareTo(s2)` ger ett heltal som är:
  - `0` om s1 och s2 har samma innehåll
  - **negativt** om s1 < s2 i lexikografisk mening, alltså s1 ska sorteras **före**
  - **positivt** om s1 > s2 i lexikografisk mening, alltså s1 ska sorteras **efter**

Jämföra strängar i Java

- I Java kan man **inte** jämföra strängar med operatorerna <, <=, >, och >=
- Dessutom ger operatorerna == och != *inte* innehålls(o)likhet utan **referens(o)likhet** :(
- För innehållslikhet måste du använda metoderna `equals` och `compareTo`.
- `s1.compareTo(s2)` ger ett heltal som är:
  - `0` om s1 och s2 har samma innehåll
  - **negativt** om s1 < s2 i lexikografisk mening, alltså s1 ska sorteras **före**
  - **positivt** om s1 > s2 i lexikografisk mening, alltså s1 ska sorteras **efter**
- Undersök följande:

```
1  scala> new String("hej") eq new String("hej") // motsvarar == i Java
2  scala> "hej".equals("hej")                    // samma som == i Scala
3  scala> "hej".compareTo("hej")
4  scala> "hej".compareTo("HEJ")         // alla stora är 'före' alla små
5  scala> "HEJ".compareTo("hej")
```

- Ibland ger `"hej" == "hej"` förvånande nog värdet **`true`** i Java. Detta beror på en minnesoptimering som kallas stränginternalisering (eng. *string interning*) som ofta sker, men inte alltid. Det är därför en svårhittad bug!

docs.oracle.com/javase/8/docs/api/java/lang/String.html#compareTo-java.lang.String-

Jämföra strängar i Java: exempel

Vad skriver detta Java-program ut?

```
public class StringEqTest {
    public static void main(String[] args){
        boolean eqTest1 =
          (new String("hej")) == (new String("hej"));
        boolean eqTest2 =
          (new String("hej")).equals(new String("hej"));
        int eqTest3 =
          (new String("hej")).compareTo(new String("hej"));
        System.out.println(eqTest1);
        System.out.println(eqTest2);
        System.out.println(eqTest3);
    }
}
```

Jämföra strängar i Java: exempel

Vad skriver detta Java-program ut?

```
public class StringEqTest {
    public static void main(String[] args){
        boolean eqTest1 =
          (new String("hej")) == (new String("hej"));
        boolean eqTest2 =
          (new String("hej")).equals(new String("hej"));
        int eqTest3 =
          (new String("hej")).compareTo(new String("hej"));
        System.out.println(eqTest1);
        System.out.println(eqTest2);
        System.out.println(eqTest3);
    }
}
```

```
1 $ javac StringEqTest.java
2 $ java StringEqTest
3 false
4 true
5 0
```

Nästa vecka: repetition

Vad är grumligt? Vad är du nyfiken på?

- Skriv önskemål till de sista föreläsningarna på Discord i kanalen `#önska`
- Skriv ämnen/koncept ur kursen som du önskar att jag förklarar mer om – sådant som fortfarande känns **grumligt**.
- Skriv ämnen om programmering och systemutveckling som du är **nyfiken** på så tar jag upp dem i mån av tid.
- Jag kommer också ge tips inför valfria tentan.
- OBS! Inga föreläsningar w14.