EDAA45 Programmering, grundkurs

Läsvecka 12: Scala och Java

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2016

12 Scala och Java

- Veckans labb: `lthopoly-team`
- Jämförelse Scala och Java
- Array
- ArrayList
- Autoboxing
- Equals
- Fördjupning diverse
- Grumligt-lådan och Nyfiken-på-lådan

Veckans labb: lthopoly-team

Veckans labb: `lthopoly-team`

Förberedelse:

- Gör övning `scalajava`:
  - Övning 1: Översätt spelet Hangman från Java till Scala
  - Övning 2: Översätt Point från Scala till Java
  - Övning 3: Autoboxing
- Studera givna koden: workspace/w11_lthopoly_team

Grunduppgift:

- Implementera en förenklad variant av monopol i terminalen.

Extrauppgift:

- Implementera valfria utvidgningar t.ex. extra pengar vid ny runda

Jämförelse Scala och Java

Grundläggande likheter och skillnader

Några likheter:

- Kompilerar till bytekod som kör på JVM på många olika plattformar
- Statiskt typning: snabb maskinkod och kompilatorn hittar buggar vid kompilering

Liknande men viss skillnad:

Java

- **Objektorientering**, men inte ”äkta” (eng. *pure*) eftersom alla värden inte är objekt
- Primitivatyper är inte objekt; representeras effektivt, normalt **utan boxning**
- Visst stöd för **funktionsprogrammering**
- Typer måste alltid anges, ibland två gånger (variabeldeklaration + instansiering)

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**
- Typinfo ska finnas vid kompileringstid men kan ofta härledas av kompilatorn

Några saker som finns i Scala men inte i Java

- `case`-klasser
- Lokala funktioner
- Metoder som operatorer
- Infix operatornotation
- Defultargument
- 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
- Try, Success, Failure
- Samlingarna i Scalas standardbibliotek, speciellt de **oföränderliga** samlingarna `Vector`, `Map`, `Set`, `List`, etc.
- Enhetlig användning av samlingar inkl. Array
- Innehållslikhet med == för oföränderliga strukturer, inkl. `< <= > >=` på strängar
- Implicita värden och klasser
- Mer precis synlighetsreglering, `private[this]`, `private[mypackage]`
- Flexibilitet och namnändring vid `import`
- Flexibel filstruktur och filnamngivning
- Flexibel nästling av klasser, objekt, traits
- Typ-alias och abstrakta typer med `type`
- Implicita värden och klasser
- ...

Några saker som finns i Java men inte i Scala

- Variabledeklaration utan initialisering
- Förändringsbara paramterar
- C-liknande prefix och postfix inkrementering och dekrementering: `i++ ++i i-- --i`
- C-liknande **`for`**-sats
- Semikolon efter alla satser
- Parenteser efter alla metoder
- Specialsyntax för indexering av array `[]` ej som i andra samlingar
- Uppräknade typer med **`enum`**
- Hoppa ut ur loop med **`break`**
  docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html
- **`switch`** ”faller igenom” utan **`break`**
- Nästan alltid snabbare kompilering
- Mer omfattande IDE-stöd
- Kontrollerade undantag (eng. *checked exceptions*) och **`throws`**
- ...

Huvudprogram i Scala och Java

Scala

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

Java

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

Syntax för variabeldeklaration i Scala och Java

Exempel på variabeldeklarationer i

Scala

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

- i2 härledd typ; går ej i Java
- i3 typ 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;
int i4;
Point p = 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) {
  println(s(i))
}

//Loopa över index baklänges:

for (i <- s.length-1 to 0 by -1) {
  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-sats i Scala och Java

Scala

```scala
val s = "Abbasillen"

//Loopa över index framlänges:

for (i <- s.indices) {
  println(s(i))
}

//Loopa över index baklänges:

for (i <- s.indices.reverse) {
  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-each-sats i Scala och Java

Scala

```scala
val s = "Abbasillen"

//Loopa över alla tecken:

for (ch <- s) {
  println(ch)
}
```

Java

```java
String s = "Abbasillen";

//Loopa över alla tecken:

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

For-each-sats i Scala och Java

Scala

```scala
val s = "Abbasillen"

//Loopa över alla tecken:

for (ch <- s) {
  println(ch)
}
```

`s.foreach(println)`

Java

```java
String s = "Abbasillen";

//Loopa över alla tecken:

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

går ej i Java

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;
    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 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
}
```

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.

```java
public class JPerson {
    private String name;
    private int age;
    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 utantill så du snabbt får grejerna på plats!

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  //undantag?

  def isAdult: Boolean = age >= Person.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  //undantag?

  def isAdult: Boolean = age >= Person.AdultAge
}

object MutablePerson {
  val AdultAge = 18
}
```

```java
public class JMutablePerson {
    private String name;
    private int age = 0;
    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;
    }
}
```

Övning: Implementera dessa specifikationer

Specification **Vegitable**

```scala
/** Representerar en grönsak. */
class Vegitable(val name: String) {

  /** Returnerar nuvarande vikt i gram. */
  def weight: Int = ???

  /** Ändrar vikten till w gram.
   *  w ska vara positiv, blir annars 0 */
  def weight_=(w: Int): Unit = ???
}
```

class JVegitable

```java
/** Skapar en grönsak. */
JVegitable(String name);

/** Returnerar namnet. */
String getName();

/** Returnerar nuvarande vikt i gram. */
int getWeight();

/** Ändrar vikten till weight gram.
 *  w ska vara positiv, blir annars 0 */
void setWeight(int weight);
```

Övning: Implementera dessa specifikationer

Specification **Vegitable**

```scala
/** Representerar en grönsak. */
class Vegitable(val name: String) {

  /** Returnerar nuvarande vikt i gram. */
  def weight: Int = ???

  /** Ändrar vikten till w gram.
   *  w ska vara positiv, blir annars 0 */
  def weight_=(w: Int): Unit = ???
}
```

class JVegitable

```
/** Skapar en grönsak. */
JVegitable(String name);

/** Returnerar namnet. */
String getName();

/** Returnerar nuvarande vikt i gram. */
int getWeight();

/** Ändrar vikten till weight gram.
 *  w ska vara positiv, blir annars 0 */
void setWeight(int weight);
```

Fördjupning:
Kasta undantaget `IllegalArgumentException` vid försök till negativ vikt.
Läs om undantag i Java här: docs.oracle.com/javase/tutorial/essential/exceptions/

Oföränderlig datatyp i Scala och Java

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

Oföränderlig datatyp i Scala och 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 defaultargument så denna blir osmidig)
- `unapply`
  (men det finns ju inte mönstermatchning så denna struntar man nog i)

Array

Repetition: Primitiva 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 effektiv 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 vektor 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)

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;  //OBS! 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; // vektor 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     }
```

ArrayList

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 primitiv arrayer.
- Fungerar för alla typer av objekt.
- Utökar samlinges 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 autoboxing; mer om detta snart)
- Namnet `ArrayList` är inte helt lyckat, eftersom ordet ”lista” normalt används för länkade snarare än vektor-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.

Viktiga operationer på ArrayList (Urval)

class ArrayList

```
/** Skapar en ny lista */
ArrayList<E>();

/** 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.

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 loopvariabel: 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) {
    ???
}
```

Övning 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);
}
```

Autoboxing

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

- Men vad gör man om man vill ha element av primitiva typer, så som **`int`** och **`double`**? Detta går alltså **INTE** i Java:
  ~~`ArrayList<int> list = new ArrayList<int>();`~~

- 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*)

Detta kan bli ganska komplicerat och det finns fallgropar.

(Om du är nyfiken på alla intrikata detaljer, se java tutorial och javaspecifikationen.)

Wrapper-klassen `Integer`

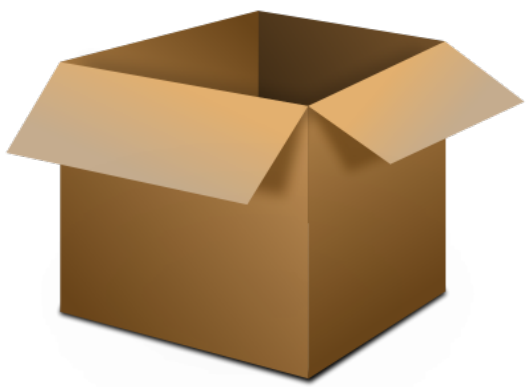
En skiss av klassen `Integer`
(ligger i paketet `java.lang` och importeras därmed implicit):

```
public class Integer {
    private int value;

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

    public Integer(int value) {
        this.value = 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

Med papper och penna:

- 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

```
1  import java.util.ArrayList;
2  import java.util.Scanner;
3
4  public class TestIntegerList {
5      public static void main(String[] args) {
6          ArrayList<Integer> list = new ArrayList<Integer>();
7          Scanner scan = new Scanner(System.in);
8          System.out.println("Skriv heltal med blank emellan. Avsluta med <CTRL+D>");
9          while (scan.hasNextInt()) {
10             int nbr = scan.nextInt();
11             Integer obj = new Integer(nbr);
12             list.add(obj);
13         }
14         System.out.println("Dina heltal i omvänd ordning:");
15         for (int i = list.size() - 1; i >= 0; i--) {
16             Integer obj = list.get(i);
17             int nbr = obj.intValue();
18             System.out.println(nbr);
19         }
20     }
21 }
```

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 skapare 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 autoboxing

```
1  import java.util.ArrayList;
2  import java.util.Scanner;
3
4  public class TestIntegerListAutoboxing {
5      public static void main(String[] args) {
6          ArrayList<Integer> list = new ArrayList<Integer>();
7          Scanner scan = new Scanner(System.in);
8          System.out.println("Skriv heltal med blank emellan. Avsluta med <CTRL+D>");
9          while (scan.hasNextInt()) {
10             int nbr = scan.nextInt();
11             list.add(nbr);   // motsvarar: list.add(new Integer(nbr));
12         }
13         System.out.println("Dina heltal i omvänd ordning:");
14         for (int i = list.size() - 1; i >= 0; i--) {
15             int nbr = list.get(i);  // motsvarar: int nbr = list.get(i).intValue();
16             System.out.println(nbr);
17         }
18     }
19 }
```

Koden finns här: scalajava/generics/TestIntegerListAutoboxing.java

Fallgropar vid autoboxing

- 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

Fallgrop med samlingar: metoden contains kräver implementation av equals

Antag att vi vill implementerar `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 implementerar `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 forts.kursen

Fördjupning: 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

Fördjupning diverse

Fördjupning: 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) 42 else 0
```

Java

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

Fördjupning: Typtest och typkonvertering

Scala

```
var x = "hej"

var isString = x.isInstanceOf[String]

var y = 42

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

Java

```
String x = "hej";

boolean isString = x instanceof String;

int y = 42;

double z = (double) y;
```

Fördjupning: Fånga undantag i Scala och Java

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

Scala

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

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

Java

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

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

Mer om undantag (eng. *exceptions*) i fortsättningskursen.

Fördjupning: scala.collection.JavaConverters

Med hjälp av `import scala.collection.JavaConverters._`
får man smidig **interoperabilitet** med Java och dess standardbibliotek,
speciellt metoderna **`asJava`** och **`asScala`**:

```
1  scala> import scala.collection.JavaConverters._
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: `http://docs.scala-lang.org/overviews/collections/conversions-between-java-and-scala-collections`

Fördjupning: 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`**.
  (Jämför Scalas **`with`** på **`trait`**s)
- 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>();`
- I Hangman-övningen:
  `Set<Character> found = new HashSet<Character>();`
- Mer om gränssnitt i fördjupningskursen.

Fördjupning: Skapa generisk Array

- 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`: hg.openjdk.java.net/.../ArrayList.java

Fördjupning: Skapa generisk Array

- 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`: hg.openjdk.java.net/.../ArrayList.java

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

```
scala> def fyrtiotvå[T](x: T): Array[T] = Array.fill(42)(x)
<console>:11: error: No ClassTag available for T

scala> import scala.reflect.ClassTag

scala> def fyrtiotvå[T: ClassTag](x: T): Array[T] = Array.fill(42)(x)
fyrtiotvå: [T](x: T)(implicit evidence$1: scala.reflect.ClassTag[T])Array[T]

scala> fyrtiotvå("hej")
res2: Array[String] = Array(hej, hej, hej, hej, hej, hej, hej, hej, hej, hej, hej, h

scala> fyrtiotvå(1)
res3: Array[Int] = Array(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

Grumligt-lådan och Nyfiken-på-lådan

Grumligt-lådan och Nyfiken-på-lådan

- Skriv lapp i **GRUMLIGT**-lådan om du har något **grundläggande begrepp** i kursen som du fortfarande tycker är **svårt att begripa**.
- Skriv lapp i **NYFIKEN-PÅ**-lådan om du vill veta mer om något ämne inom programmering och som går **bortom grunderna**.