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
- Grumligt- 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

Exempel: typisk 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: typisk 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: typisk 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;
    }
}
```

Övning:

Gör `Person` och `JPerson` förändringsbar:

- Namnet ska ges vid konstruktion.
- Åldern ska alltid börja på 0.
- Namn och ålder ska gå att ändra.
- Åldern ska aldrig kunna bli negativ.

Lär dig detta mönster utantill så du snabbt får grejerna på plats!

Exempel: typisk förändringsbar klass i Scala och Java

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

  def age: Int = _age

  def age_=(newAge: Int): Unit =
    if (newAge >= 0) _age = newAge
    else throw new Exception(s"Bad age: $newAge")

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

object MutablePerson {
  val AdultAge = 18
}
```

Exempel: typisk förändringsbar klass i Scala och Java

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

  def age: Int = _age

  def age_=(newAge: Int): Unit =
    if (newAge >= 0) _age = newAge
    else throw new Exception(s"Bad age: $newAge")

  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 {
          throw new Exception(s"Bad age: $newAge");
        }
    }

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

Övning: Implementera dessa specifikationer

Specification **Grönsak**

```
/** Representerar en grönsak
 *  med namn och vikt som från början är 0
 */
class Vegitable(val name: String) {

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

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

class JVegitable

```
/** Skapar en grönsak med namnet name och
*   vikten weigth som är 0 från början.
*/
JVegitable(String name);

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

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

/** Ändrar vikten till weight gram
*   som ska vara positiv annars undantag */
void setWeight(int weight);
```

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)

Syntax för variabeldeklaration i Scala och Java

Syntax för Array i Scala och Java

For-sats i Scala och Java

Huvudprogram i Scala och Java

Förändringsbar samling i Scala och Java

Autoboxing

Iterera över samling i Scala och Java

Fördjupning: Skapa generisk Array av viss typ

Grumligt- och Nyfiken-på-lådan

Grumligt- och Nyfiken-på-lådan