Funkcionalno programiranje

Interoperabilnost jezika Scala i Java

Prevođenje u Java bajt kod

- Svi Java tipovi imaju ekvivalentan tip u jeziku Scala
- Većina Scala koda se direktno preslikava u odgovarajući Java konstrukt
  - klase
  - definicija i pozivanje metoda
  - izuzeci

- Neki Scala konstrukti ne mogu da se direktno preslikaju, jer ne postoji (direktan) ekvivalent
  - crte
  - generički tipovi
- Kada ne postoji preslikavanje, kodiranje se vrši kombinovanjem struktura koje Java ima

Vrednosni tipovi

- Vrednosni tip poput `Int` se iz jezika Scala prevodi u jezik Java na dva načina
  - kad je moguće, pretvara se u primitivan tip `int` (performanse)
  - u suprotnom, koristi omotačku klasu `java.lang.Integer`

- Primer: `List[Any]`
  - može da sadrži samo objekte tipa `Int`, ali to nije garantovano
  - zato kompajler usvaja restriktivnu pretpostavku i koristi `Integer`

- Slično i za ostale vrednosne tipove

Unikatni objekti

- Prevodi se u kombinaciju
  statičkih i nestatičkih polja i metoda
- Za svaki unikatni objekat pravi se Java klasa
  čijem imenu se dodaje znak `$`
- Ova klasa ima sve metode i atribute unikatnog objekta
- Ima statički atribut `MODULE$`
  koje čuva instancu te klase koja predstavlja
  unikatni objekat

Unikatni objekti

```
object App {
  def main(args: Array[String]) = {
    println("Hello, world!")
  }
}
```

```
public final class App$ extends java.lang.Object
                        implements scala.ScalaObject {

  public static final App$ MODULE$;
  public static {};
  public App$();
  public void main(java.lang.String[]);
  public int $tag();
}
```

Unikatni objekti

- U slučaju da ne postoji prateća klasa (već samo unikatni objekat), kompajler će napraviti Java klasu sa statičkim metodama za pozivanje odgovarajućih metoda unikatnog objekta.

```
public final class App$ extends java.lang.Object
                        implements scala.ScalaObject {

  public static final App$ MODULE$;
  public static {};
  public App$();
  public void main(java.lang.String[]);
  public int $tag();
}
```

```
public final class App extends java.lang.Object{
  public static final int $tag();
  public static final void main(java.lang.String[]);
}
```

Crte

- Java ne podržava crte
- Kompajliranje crte stvara Java interfejs istog imena
- Ako crta ima samo apstraktne metode
  onda se direktno preslikava u Java intefejs
- U suprotnom, kompajler generiše kod
  potreban za simuliranje crta

Primer

Callback.scala

```
package w08
trait Callback {
  def print = println("Test!")
}
```

Primer

AWTTest.java

```
package w08;
import java.awt.*;
import java.awt.event.*;
public class AWTTest extends Frame {
  private boolean unisti = false;
  private Callback callback;

  private class TestAkcija implements ActionListener  {
    public void actionPerformed (ActionEvent d) {
      if( callback != null )   callback.print();
         }
  }
 private Panel panelTest()  {
    Panel p = new Panel();
    Button b = new Button("Test");
    b.addActionListener( new TestAkcija() );
    p.add( b );
    return p;
  }
```

Primer

AWTTest.java

```
  public void setCallback(Callback c) { callback = c; }

  public AWTTest (int x, int y) {
    super ("Test");
    setBounds (x, y, 305, 242);
    setResizable (false);
    add( panelTest(), BorderLayout.CENTER);
    addWindowListener (new WindowAdapter () {
      public void windowClosing (WindowEvent d)
        { if (unisti) dispose (); else setVisible (false); }
    });
  }

  public AWTTest (boolean unisti)  {
    this (250, 50);
    this.unisti = unisti;
  }
}
```

Primer

JavaScalaTest.scala

```
package w08

object JavaScalaTest {

  class SimpleCallback extends Callback

  def main(args: Array[String])
  {
      var test = new AWTTest(true)
      test.setVisible (true)
      test.setCallback( new SimpleCallback )
  }
}
```

Anotacije

- `@deprecated` – dodaje se na klase ili metode generišu se odgovarajuće Java anotiacije
- `@volatile` – dodaje Java modifikator `volatile` na generisan kod
- `@serializable` – klasi sa ovom anotacijom se dodaje interfejs Serializable
- `@transient` – atributu se dodaje modifikator `transient`
- `@throws` – metoda se proglašava da baca izuzetke
  - Primer:

```
@throws(classOf[IOException])
def read() = in.read()
```

Nespecificiran tip (džoker znak)

- Kako podržati sledeći Java kod?
  - `Iterator<?>`
  - `Iterator<? extends Component>`

- Koristi se simbol _
  (slično kao kod anonimnih ili parcijalno primenjenih f-ja)
- `Iteartor[_]`
- `Iterator[_ <: Component]`

Funkcijski literali i lambda izrazi

- Java 8 je uvela *lambda izraze*
  - suštinski: koncizan način pravljenja instanci anonimnih klasa

```
// Java 8
JButton button = new JButton();
button.addActionListener(event -> System.out.println("klik!"));
```

- Lambda izraz se može koristiti gde god se očekuje SAM (single abstract method) tip
- ActionListener je takav tip: jedino ima metodu `actionPerformed`

- Funkcijski literali u jeziku Scala se mogu koristiti na isti način
  - Razlika u verzijama jezika Scala

Funkcijski literali i lambda izrazi

```
val button = new JButton

// Pre Scala 2.12
implicit def function2ActionListener(f: ActionEvent => Unit) =
   new ActionListener {
     def actionPerformed(event: ActionEvent) = f(event)
   }

button.addActionListener( (_ : ActionEvent) => println("klik!") )



// Scala 2.12 +
button.addActionListener( _ => println("klik!") )
```