Funkcionalno programiranje

Vežbe

01 Uvod

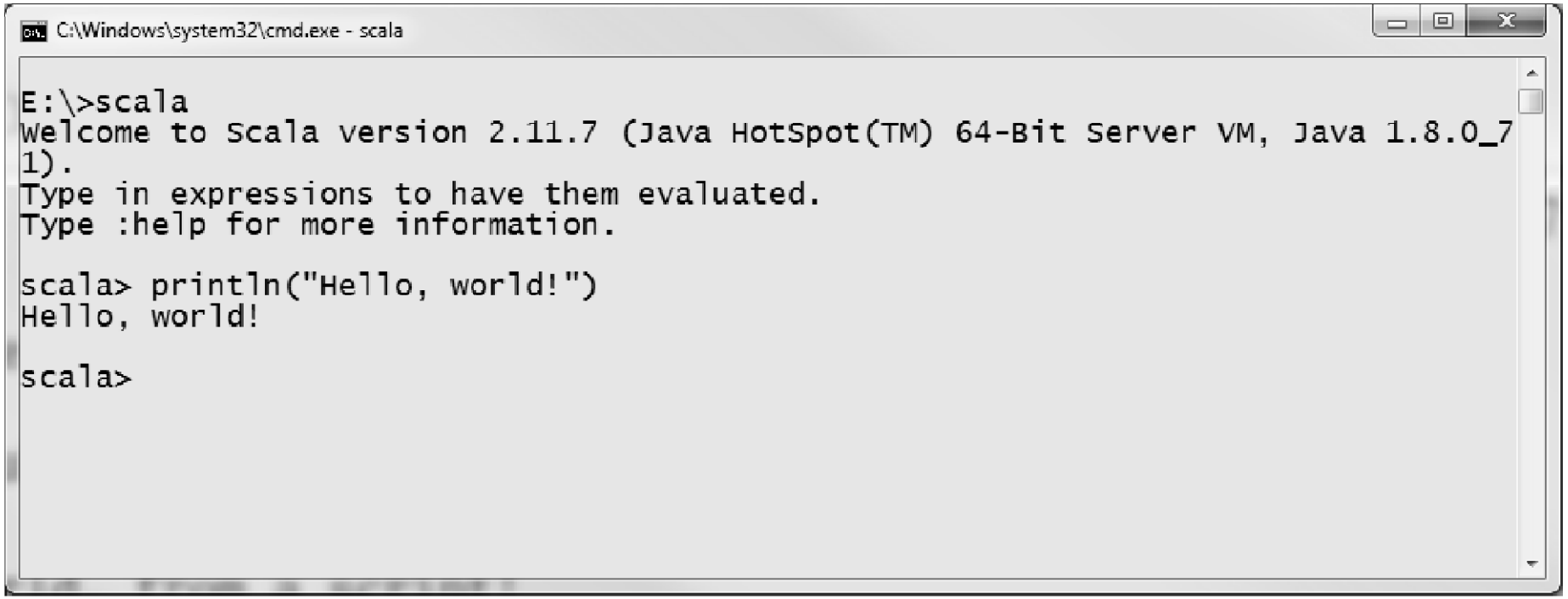
Uvod

- Par reci o raznim IDE, instalaciji + komandnoj liniji, skriptama, itd.

Zadatak 1.1 – "Hello, world"

Iz interpretera

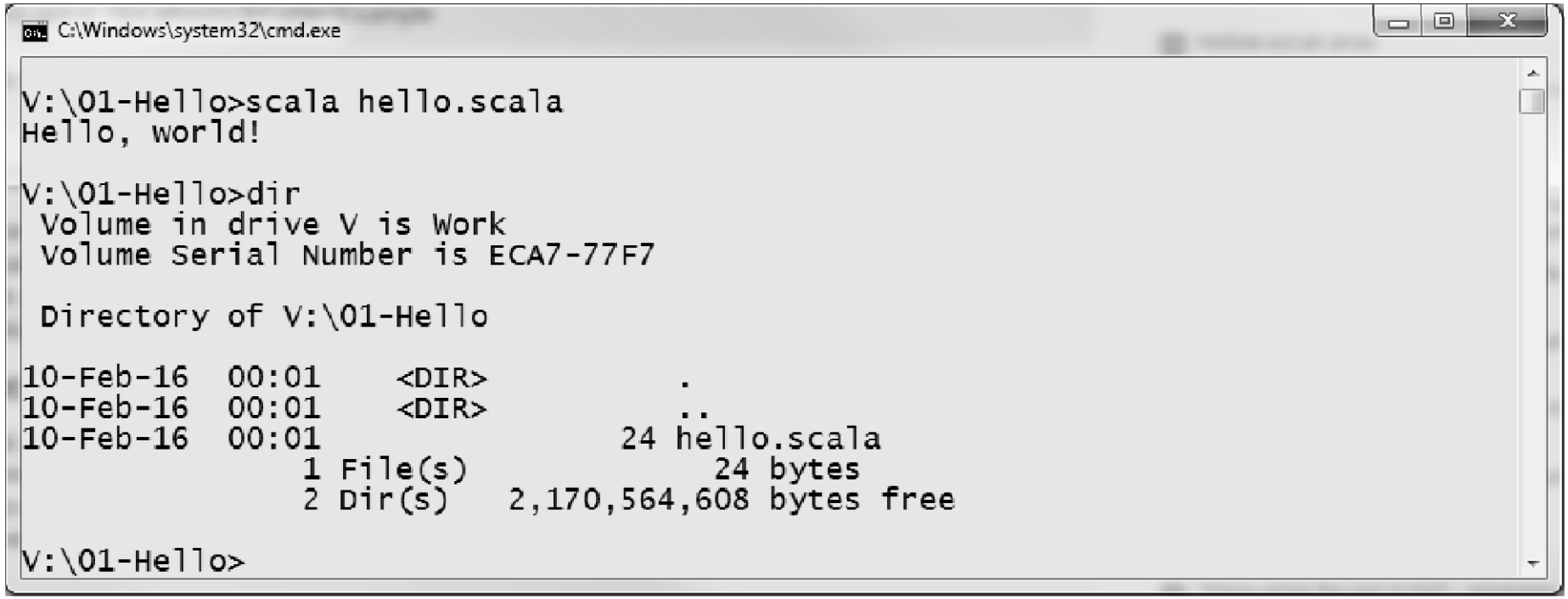
```
scala> println("Hello, world!")
Hello, world!
```



Zadatak 1.1 – "Hello, world"

U vidu skripte (hello.scala)

```
C:\> scala hello.scala
Hello, world!
```



Zadatak 1.2 - zbir

- Napisati skriptu koja iz komandne linije pročita dva cela broja i ispiše njihov zbir.

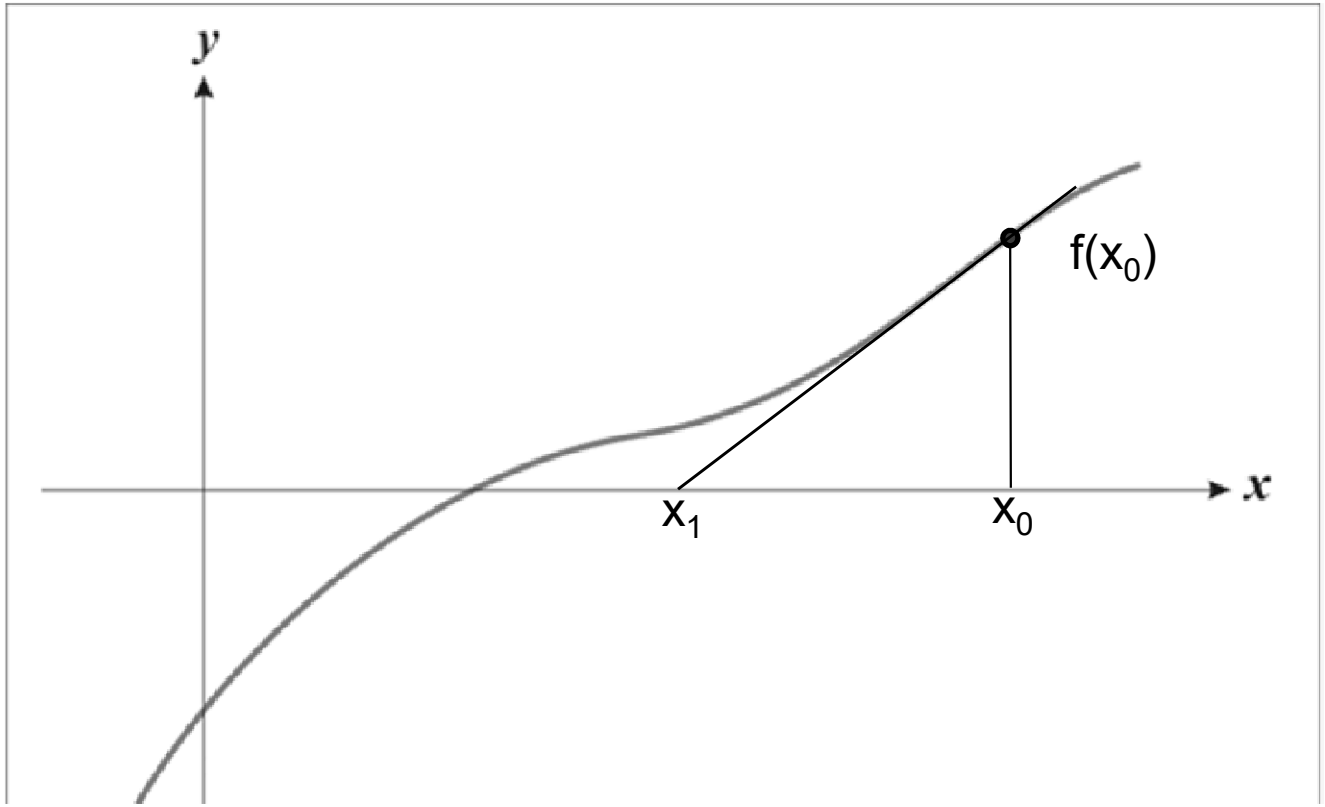
```
println(args(0) + " + " +args(1) + " = " +
        (
          Integer.parseInt(args(0))
            +
          Integer.parseInt(args(1))
        )
       )
```

Snimiti u fajl "suma.scala"

```
> scala suma.scala 5 8
> 5 + 8 = 13
```

Zadatak 1.3 – računanje kvadratnog korena

- Korišćenjem iterativne (rekurentne) formule prema Newton-Raphson-ovom metodu izračunati približno kvadratni koren datog broja



$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$$f(x) = x^2 - a$$

$$f'(x) = 2x$$

$$\Downarrow$$

$$x_{n+1} = x_n - \frac{x_n^{\,2} - a}{2x_n}$$

Zadatak 1.3 – računanje kvadratnog korena

- Generalna ideja
  - metod koristi sukcesivne aproksimacije
  - u i-tom koraku treba utvrditi da li je aproksimacija dovoljno dobra
  - ako nije, proceniti novu aproksimaciju i ponoviti

```
def sqrtIter(guess: Double, x: Double): Double =
  if (isGoodEnough(guess, x)) guess
  else sqrtIter(improve(guess, x), x)
```

- Primetiti
  - koristi se (specijalna) rekurzija – **mora da se navede tip rezultata**
  - rekurzija na dnu (*tail recursion*) – rekurzivan poziv je poslednji izraz
  - u tom slučaju, ponaša se kao **iteracija**, ne kao rekurzija

Zadatak 1.3 – računanje kvadratnog korena

```
object sqrtProgram {
  def sqrt(x: Double) = {
    def approx(guess: Double) = (guess + x/guess) / 2
    def goodApprox(guess: Double) =
      abs(sqr(guess)-x)/x < 1e-15
    def abs(x: Double) = if(x>=0) x else -x
    def sqr(x: Double) = x*x
    def sqrtIter(guess: Double): Double =
      if( goodApprox(guess) )   guess
      else                      sqrtIter( approx(guess) )

    sqrtIter(1)
  }                  //> sqrt: (x: Double)Double
   sqrt(15)          //> res0: Double = 3.872983346207417
}
```

Zadatak 1.4 – linearna i terminalna rekurzija

Funkcija sum koristi linearnu rekurziju. Napisati funkciju tako da koristi terminalnu rekurziju.

- Podsetnik sa predavanja:

```
def sum(f: Int => Double)(a: Int, b: Int): Double = {
  if ( a>b ) 0 else f(a) + sum(f)(a+1, b)
}
```

Zadatak 1.4 – linearna i terminalna rekurzija

Ideja: parcijalni rezultat treba prenositi u rekurzivne pozive

```
def sum(f: Int => Double)(a: Int, b: Int) = {
  def sumTR(a: Int, r: Double): Double =
    if ( a>b )      r
    else            sumTR(a+1, r+f(a))

   sumTR(a, 0)
}
```

Zadatak 1.5 – generalizacija pristupa

- a) Na osnovu funkcije sum iz zadatka 1.4, napisati funkciju `product` koja računa proizvod vrednosti u zadatom intervalu
- b) Na osnovu funkcije iz tačke (a) napisati funkciju koja računa n!
- c) Generalizovati pristup

```
def sum(f: Int => Double)(a: Int, b: Int) = {
  def sumTR(a: Int, r: Double): Double =
    if ( a>b )      r
    else            sumTR(a+1, r+f(a))

  sumTR(a, 0)
}
```

Zadatak 1.5 – generalizacija pristupa

a)

```
def product(f: Int => Double)(a: Int, b: Int) = {
  def productTR(a: Int, r: Double): Double =
    if ( a>b )      r
    else            productTR(a+1, r*f(a))

  productTR(a, 1)
}

product(x=>x)(2,5)    // Double = 120.0
```

b)

```
def fact(n: Int) = product(x=>x)(1,n)

fact(5)               // Double = 120.0
```

Zadatak 1.5 – generalizacija pristupa

c)

```
def genop(f: Int => Double, id: Int,
          g: (Double, Double)=> Double)(a: Int, b: Int) = {
  def tr(a: Int, r: Double): Double =
    if ( a>b )      r
    else            tr(a+1, g(r,f(a)))

  tr(a, id)
}

def fact2(n: Int)=genop(x=>x, 1, (x,y)=>x*y)(1,n) //(n: Int)Double
fact2(5)
```

Zadatak 1.6 – Rekurzija i kompozicija

- a) Napisati definiciju funkcije koja vraća funkciju koja vraca vrednost jedinog parametra tipa `Int`

- b) Napisati funkciju koja za date dve funkcije `f` i `g`, koje preslikavaju `Int` u `Int`, vraća funkciju koja vrši kompoziciju funkcija `f` i `g`: `y = g(f(x))`

- c) Napisati funkciju koja vraća funkciju koja predstavlja `n` puta komponovanu funkciju `f` nad parametrom `x`:
  `f(f(f ... f(x) ...))`

- d) Napisati funkciju iz tačke (c) upotrebom funkcija iz tačaka (a) i (b)

Zadatak 1.6 – Rekurzija i kompozicija

a)

```
def id = (x: Int) => x
```

b)

```
def compose(f: Int => Int, g: Int => Int): Int => Int =
   x => g(f(x))
```

c)

```
def repeated(f: Int => Int, n: Int): Int => Int =
   x => if (n == 0) x else repeated(f, n-1)(f(x))
```

```
repeated(f, 3)(10)
→  10           => repeated(f, 2)(f(10))
→  f(10)        => repeated(f, 1)(f(f(10)))
→  f(f(10))     => repeated(f, 0)(f(f(f(10))))
→  f(f(f(10))) => f(f(f(10)))
```

Zadatak 1.6 – Rekurzija i kompozicija

a)

```
def id = (x: Int) => x
```

b)

```
def compose(f: Int => Int, g: Int => Int): Int => Int =
   x => g(f(x))
```

c)

```
def repeated(f: Int => Int, n: Int): Int => Int =
   x => if (n == 0) x else repeated(f, n-1)(f(x))
```

d)

```
def repeated(f: Int => Int, n: Int): Int => Int =
   if(n == 0) id
   else compose(f, repeated(f, n-1))
```

Zadatak 1.7 – Nalaženje fiksne tačke funkcije

- Vraćanje na problem pronalaženja kvadratnog korena uz upotrebu funkcija višeg reda
- Broj x zove se fiksna tačka funkcije `f` ako važi `f(x) = x`
- Za određene funkcije, fiksna tačka se može odrediti uzastopnom primenom funkcije f, počevši od inicijalne procene: `x, f(x), f(f(x)), f(f(f(x))), ...`
- Napisati funkciju koja određuje fiksnu tačku zadate funkcije
- Zatim upotrebiti tu funkciju za definisanje funkcije sqrt(x)

Zadatak 1.7 – Nalaženje fiksne tačke funkcije

```
val tolerance = 0.0001
def isCloseEnough(x : Double, y : Double) =
   abs((x - y) / x) < tolerance
def fixedPoint(f: Double => Double)(firstGuess: Double) = {
   def iterate(guess : Double): Double = {
       val next = f(guess)
       if(isCloseEnough(guess, next)) next
       else iterate(next)
   }
   iterate(firstGuess)
}
```

Zadatak 1.7 – Nalaženje fiksne tačke funkcije

sqrt(x) = broj **y** takav da **y * y = x**
= broj **y** takav da **y = x / y**

Dakle, `sqrt(x)` se može napisati kao `(y => x / y)`

```
def sqrt(x: Double) = fixedPoint( y => x / y )(1.0)
```

Na žalost, funkcija ne konvergira (alternira 1.0, 2.0, 1.0, ...)

Rešenje, koje odgovara onom iz zadatka 1.3, je da se
procena računa kao srednja vrednost
dve uzastopne procene

```
def sqrt(x: Double) = fixedPoint( y => (y + x/y)/2 )(1.0)
```

Zadatak 1.7 – Nalaženje fiksne tačke funkcije

Pošto je tehnika "stabilizacije" fiksne tačke dovoljno generalna, zaslužuje da se apstrahuje posebnom funkcijom

```
def averageDamp(f: Double => Double)(x: Double) =
  (x + f(x)) / 2
```

Sada se može ponovo formulisati funkcija `sqrt(x)`

```
def sqrt(x: Double) =
  fixedPoint( averageDamp( y => x / y ) ) (1.0)
```

Pitanja:

- kako izračunati 3. koren?
- kako izračunati n-ti koren?

Funkcionalno programiranje

Vežbe

02 Klase

Zadatak 2.1

- Napisati hijerarhiju klasa za realizaciju ulančane liste sa parametrizovanim tipom elementa.

- S obzirom na favorizovanje rekurzije u funkcionalnim jezicima, lista se može predstaviti iz dva gradivna bloka:
  - Nil – prazna lista
  - Kons – ćelija koja sadrži **element** i referencu ka **ostatku liste**
    - Termin "cons" vodi poreklo iz jezika Lisp

- Funkcionalni jezici favorizuju **nepromenljive liste**

Zadatak 2.1

```
package w04

object list {

  trait List[T]  {
    def prazna: Boolean
    def glava: T
    def rep: List[T]
  }

  class Nil[T] extends List[T] {
    def prazna = true
    def glava = throw new NoSuchElementException("Nil.glava")
    def rep = throw new NoSuchElementException("Nil.rep")
  }

  class Elem[T](val glava : T, val rep : List[T]) extends List[T] {
    def prazna = false
  }
}
```

Zadatak 2.1

```
val testList = new Elem(1, new Elem(2, new Elem(3, new Nil)))
//> testList  : w04.list.Elem[Int] = w04.list$Elem@6aa8ceb6
```

Zadatak 2.2

- a) Napisati funkciju koja dohvata n-ti element liste iz zadatka 2.1
- b) Napisati funkciju koja odbacuje prvih n elemenata liste iz zadatka 2.1
- c) Napisati funkciju koja odbacuje elemente sa početka liste koji ispunjavaju zadat uslov

Zadatak 2.2

- a) Napisati funkciju koja dohvata n-ti element liste iz zadatka 2.1

```
@tailrec
def ntiElement[T](n: Int, lista: List[T]): T =
  if(lista.prazna) throw new IndexOutOfBoundsException
  else if(n == 0) lista.glava
  else ntiElement(n-1, xs.rep)
```

Zadatak 2.2

- b) Napisati funkciju koja odbacuje prvih n elemenata liste iz zadatka 2.1

```
@tailrec
def izbaciN[T](n: Int, lista: List[T]): List[T] =
  if(lista.prazna) throw new IndexOutOfBoundsException
  else if(n == 0) lista
  else izbaciN(n-1, xs.rep)
```

Zadatak 2.2

- c) Napisati funkciju koja odbacuje elemente sa početka liste koji ispunjavaju zadat uslov

```
@tailrec
def izbaciDok[T](lista: List[T], p:T => Boolean): List[T] =
  if(lista.prazna) throw new IndexOutOfBoundsException
  else if(! p(lista.glava)) lista
  else izbaciDok(xs.rep, p)
```

Zadatak 2.3

- Napisati potrebne klase za realizovanje logičkog tipa (Boolean) bez upotrebe kontrolne strukture if.

```
abstract class Boolean
{
	def ifThenElse(t : => Boolean, f : => Boolean) : Boolean

	def && (x : => Boolean) : Boolean = ifThenElse(x, False)
	def || (x : => Boolean) : Boolean = ifThenElse(True, x)

	def unary_! : Boolean = ifThenElse(False, True)

	def == (x : => Boolean) : Boolean = ifThenElse(x, !x)
	def != (x : => Boolean) : Boolean = ifThenElse(!x, x)

	def < (x : => Boolean) : Boolean = ifThenElse(False, x)
}
```

Zadatak 2.3

```
object True extends Boolean
{
	def ifThenElse(t : => Boolean, f : => Boolean) : Boolean = t
}

object False extends Boolean
{
	def ifThenElse(t : => Boolean, f : => Boolean) : Boolean = f
}
```

Zadatak 2.3

```
val a = True
//> a  : w04.Idealized.True.type = w04.Idealized$True$@f2a0b8e
val b = False
//> b  : w04.Idealized.False.type = w04.Idealized$False$@36d64342

val c = a || b
//> c  : w04.Idealized.Boolean = w04.Idealized$True$@f2a0b8e

val d = !c
//> d  : w04.Idealized.Boolean = w04.Idealized$False$@36d64342

val e = a < b
//> e  : w04.Idealized.Boolean = w04.Idealized$False$@36d64342
val f = a < a
//> f  : w04.Idealized.Boolean = w04.Idealized$False$@36d64342
val g = b < a
//> g  : w04.Idealized.Boolean = w04.Idealized$True$@f2a0b8e
}
```

Zadatak 2.4

- Napraviti klasu prema projektnom uzorku unikat (singleton)

- Analiza:
  - obezbediti da klasa ne bude apstraktna
  - ne može direktno da se instancira objekat date klase
  - treba da postoji metoda koja vrši dohvatanje jedinstvene instance

- Rešenje:
  - učiniti primarni konstruktor privatnim
  - napraviti prateći objekat sa metodom getInstance

Zadatak 2.4

```
class NarodnaBiblioteka private {
	override def toString : String = "Narodna biblioteka"
}

object NarodnaBiblioteka {
	val instance = new NarodnaBiblioteka
	def getInstance = instance
}

val tt = NarodnaBiblioteka.getInstance
//> tt  : w03.singleton.NarodnaBiblioteka = Narodna biblioteka
```

Zadatak 2.5 – proširivanje IntSet hijerarhije klasa

- Proširiti klase iz IntSet hijerarhije tako da podrže metode za određivanje unije i preseka skupova.

```
abstract class IntSet {
    def incl(x: Int): IntSet
    def contains(x: Int): Boolean
}

class Empty extends IntSet {
    def contains(x : Int) = false
    def incl(x : Int) =
            new NonEmpty(x, new Empty, new Empty)
}
```

Zadatak 2.5 – proširivanje IntSet hijerarhije klasa

```
class NonEmpty(elem: Int, left: IntSet, right: IntSet)
                                              extends IntSet {
    def this(elem: Int) = this(elem, new Empty, new Empty)
    def contains(x : Int) = {
      if (x < elem) left contains x
      else if (x > elem) right contains x
      else true
    }

    def incl(x : Int) = {
      if (x<elem) new NonEmpty(elem, left incl x, right)
      else if (x>elem) new NonEmpty(elem, left, right incl x)
      else this
    }
}
```

Zadatak 2.5 – proširivanje IntSet hijerarhije klasa

```
abstract class IntSet {
     def incl(x: Int): IntSet
     def contains(x: Int): Boolean
     def union(x: IntSet): IntSet
     def intersection(x: IntSet): IntSet
}

class Empty extends IntSet {
     def contains(x : Int) = false
     def incl(x : Int) = new NonEmpty(x, new Empty, new Empty)
     def union(x: IntSet) = x
     def intersection(x: IntSet) = new Empty
}
```

Zadatak 2.5 – proširivanje IntSet hijerarhije klasa

```
class NonEmpty(elem: Int, left: IntSet, right: IntSet)
                                              extends IntSet {
   def this(elem: Int) = this(elem, new Empty, new Empty)
   def contains(x : Int) = { ... }
   def incl(x : Int) = { ... }

   def union(x: IntSet) = { ??? }

   def intersection(x: IntSet) = { ??? }
 }
```

Zadatak 2.5 – proširivanje IntSet hijerarhije klasa

```
class NonEmpty(elem: Int, left: IntSet, right: IntSet)
                                              extends IntSet {
   def this(elem: Int) = this(elem, new Empty, new Empty)
   def contains(x : Int) = { ... }
   def incl(x : Int) = { ... }

   def union(x: IntSet) = ((left union right) union x) incl elem

   def intersection(x: IntSet) = { ??? }
 }
```

Zadatak 2.5 – proširivanje IntSet hijerarhije klasa

```
class NonEmpty(elem: Int, left: IntSet, right: IntSet)
                                              extends IntSet {
   def this(elem: Int) = this(elem, new Empty, new Empty)
   def contains(x : Int) = { ... }
   def incl(x : Int) = { ... }

   def union(x: IntSet) = ((left union right) union x) incl elem

   def intersection(x: IntSet) = {
      val y = if (x contains elem)
                     new NonEmpty(elem, new Empty, new Empty)
              else new Empty
    y union ((left intersection x) union (right intersection x))
   }

}
```

Zadatak 2.6

- Napisati klase i potrebne funkcionalnosti za formiranje statičkog Huffman-ovog koda od zadatog niza karaktera (string).

Zadatak 2.6 - Analiza

- Problem koji rešava Huffman-ov algoritam se najjednostavnije može modelirati binarnim stablom
- Napravićemo stablo po uzoru na listu
  - koristićemo rekurziju
  - potrebno je formirati
    - apstraktni tip za čvor stabla
    - konkretan tip za unutrašnji čvor stabla
    - konkretan tip za listove stabla
- Algoritam se takođe oslanja na prioritetni red
  - koristićemo gotovu klasu scala.collection.mutable.PriorityQueue

Zadatak 2.6

- Najpre treba prebrojati simbole u poruci
  - zbog nepromenljivosti objekata, formiraćemo listu simbola poruke
  - sortiraćemo listu po alfabetskom poretku
  - formiraćemo listu parova (simbol, broj)
  - umetnućemo elemente liste u prioritetni red
  - primenićemo Huffman-ov algoritam

Zadatak 2.6

```
object list
{
        def izStringa(s: String) : List[Char] =
        {
                var myList : List[Char] = new Nil[Char]
                for(c <- s)
                  myList = new Elem(c, myList)
                myList
        }

        def duzina[T](lista: List[T]) : Int =
        {
          @tailrec
          def duz[T](n : Int, lista : List[T]) : Int =
            if( lista.prazna ) n
            else duz(n+1, lista.rep)

          duz(0, lista)
        }
```

Zadatak 2.6

```
def mergeSort[T <% Double](lista : List[T]) : List[T] =   {

  def merge(levo: List[T], desno: List[T]) : List[T] =  {
      if( desno.prazna ) levo
      else if( levo.prazna ) desno
      else {
        if(levo.glava < desno.glava)
              new Elem(levo.glava, merge(levo.rep, desno))
        else new Elem(desno.glava, merge(levo, desno.rep))
      }
  }

  val n = duzina(lista)/2
  if( n < 1 ) lista
  else  {
    val (levo : List[T], desno : List[T]) = podeli(n, lista)
    merge( mergeSort(levo), mergeSort(desno))
  }
}
```

Zadatak 2.6

```
def prvihN[T](n: Int, list : List[T]) : List[T] =
{
  if( list.prazna ) throw new IndexOutOfBoundsException
  if( n == 0 ) new Nil
  else new Elem(list.glava, prvihN(n-1, list.rep))
}


def podeli[T](n: Int, list: List[T]) : (List[T], List[T]) =
{
  (prvihN(n, list), izostaviN(n, list) )
}
```

Zadatak 2.6

```
    class Par[T](val n : Int, val c : T) {
      override def toString = "(" + n + "," + c + ")"
    }
    implicit def parToDouble[T](p : Par[T]) : Double = p.n

    def napraviParove[T](lista : List[T]) : List[ Par[T] ] = {

      def izdvojIste(lista: List[T], n: Int, podatak: T) :
            List[ Par[T] ] = {
        if(lista.rep.prazna) new Elem( new Par(n, podatak), new Nil)
        else if(lista.rep.glava == podatak)
            izdvojIste(lista.rep, n+1, podatak)
        else  new Elem( new Par(n, podatak),
                    izdvojIste(lista.rep, 1, lista.rep.glava ))
      }

      if(lista.prazna) throw new IndexOutOfBoundsException
      izdvojIste(lista, 1, lista.glava)
    }
}
```

Zadatak 2.6

HuffmanTree.scala

```
trait HuffmanCvor
{
  def daLiJeKoren : Boolean
  def daLiJeList : Boolean
  def roditelj : HuffmanCvor
  def levi : HuffmanCvor
  def desni : HuffmanCvor
  def podat : list.Par[Char]
  def poseta(s: String) : Unit
}
```

Zadatak 2.6

```
class UnutrasnjiCvor(val levi: HuffmanCvor, val desni: HuffmanCvor)
        extends HuffmanCvor
{
  def daLiJeKoren = roditelj == null
  def daLiJeList = false
  var roditelj : HuffmanCvor = null
  val podat =
       new list.Par[Char](levi.podat.n + desni.podat.n, ' ')
  def poseta(s: String) =
  {
    levi.poseta(s + "0" )
    desni.poseta(s + "1" )
  }
}
```

Zadatak 2.6

```
class HuffmanList(val podat: list.Par[Char] ) extends HuffmanCvor
{
  val daLiJeKoren = false
  val daLiJeList = true
  def levi = throw new NoSuchElementException("List.levi")
  def desni = throw new NoSuchElementException("List.desni")
  var roditelj: UnutrasnjiCvor = null
  def poseta(s: String) =
  {
    println(podat.c + ":" + s)
  }
}
```

Zadatak 2.6


```
import scala.math.Ordering.Implicits._
import scala.collection.mutable.PriorityQueue

object Huffman
{
    def redosled(t: HuffmanCvor ) = -t.podat.n
    //> redosled: (t: w03.HuffmanCvor)Int

    val x = new PriorityQueue[ HuffmanCvor ]()(Ordering.by(redosled))
    //> x  : scala.collection.mutable.PriorityQueue[w03.HuffmanCvor] =
    //| PriorityQueue()
```

Zadatak 2.6

```
val dugacka = list.izStringa("OvojejednaveomadugackaporukA")
//> dugacka  : w03.list.List[Char] = w03.list$Elem@3b81a1bc

val dugackaSortirana = list.mergeSort(dugacka)
//> dugackaSortirana  : w03.list.List[Char] = w03.list$Elem@b97c004

var listaParova = list.napraviParove(dugackaSortirana)
//> listaParova  : w03.list.List[w03.list.Par[Char]] =
//| w03.list$Elem@4590c9c3

while( ! listaParova.prazna )
{
    val element = new HuffmanList( listaParova.glava )
    x.enqueue( element )
    listaParova = listaParova.rep
}

println(x.size)                                   //> 16
```

Zadatak 2.6

```
while(x.size > 1)
{
    val prvi = x.dequeue()
    val drugi = x.dequeue()

    val novi = new UnutrasnjiCvor(prvi, drugi)
    x.enqueue(novi)
}

val koren = x.dequeue()
//> koren  : w03.HuffmanCvor = w03.UnutrasnjiCvor@6d00a15d

println(koren.podat.n)                          //> 28
```

Zadatak 2.6

```
koren.poseta("")
```

```
//> c:0000
//| m:0001
//| e:001
//| o:010
//| A:01100
//| O:01101
//| n:01110
//| g:01111
//| d:1000
//| k:1001
//| a:101
//| u:1100
//| j:1101
//| v:1110
//| p:11110
//| r:11111
```


```
}
```

Funkcionalno programiranje

Vežbe

03 Kontrolne strukture

Zadatak 3.1

- (a) Napisati for ciklus koji ispiše sve elemente niza stringova

```
object loops {

 val marke = Array("BMW", "Mercedes", "Audi", "Porsche")
 //> marke  : Array[String] = Array(BMW, Mercedes, Audi, Porsche)

 for( marka <- marke )
       println(marka)
                                                    //> BMW
                                                     //| Mercedes
                                                     //| Audi
                                                     //| Porsche
}
```

```
def apply[T: ClassTag](xs: T*): Array[T] = {
   val array = new Array[T](xs.length)
   var i = 0
   for (x <- xs.iterator) { array(i) = x; i += 1 }
   array
}
```

Zadatak 3.1

- (b) Napisati for ciklus koji ispiše sve elemente niza stringova nakon pretvaranja svih slova u velika

```
object loops {

 val marke = Array("BMW", "Mercedes", "Audi", "Porsche")
 //> marke  : Array[String] = Array(BMW, Mercedes, Audi, Porsche)

 for( marka <- marke )  {
    val m = marka.toUpperCase     /* println( marka.toUpperCase ) */
    println(m)
 }
                                                    //> BMW
                                                     //| MERCEDES
                                                     //| AUDI
                                                     //| PORSCHE

}
```

Zadatak 3.2

- Napisati for ciklus koji formira nov niz elemenata tipa string u kojem su svi znaci velika slova

```
object loops {

 val marke = Array("BMW", "Mercedes", "Audi", "Porsche")
 //> marke  : Array[String] = Array(BMW, Mercedes, Audi, Porsche)

 val marke2 = for( marka <- marke )  yield marka.toUpperCase
//> marke2  : Array[String] = Array(BMW, MERCEDES, AUDI, PORSCHE)


 // yield vraća tip po kojem se iterira
 val marke3 = for(marka <- marke.toList) yield marka.toUpperCase
 //> marke3  : List[String] = List(BMW, MERCEDES, AUDI, PORSCHE)
}
```

Zadatak 3.3: string interpolation

```
val valuesAndNames = List( (1, "One"), (2, "Two") )
//> valuesAndNames  : List[(Int, String)] = List((1,One), (2,Two))

for(valueAndName <- valuesAndNames)
  println(s"${valueAndName._1} ... ${valueAndName._2}")
                                                  //> 1 ... One
                                                  //| 2 ... Two

val namesAndWeights = List ( ("John", 76.57), ("Andy", 88.863) )
//> namesAndWeights  : List[(String, Double)] = List((John,76.57),
//| (Andy,88.863))
var i = 0
for(nameAndWeight <- namesAndWeights) {
  i = i+1
  println(f"$i. ${nameAndWeight._1} weights ${nameAndWeight._2}%.1f")
}                                                 //> 1. John weights 76.6
                                                  //| 2. Andy weights 88.9
```

Zadatak 3.4 – custom string interpolation

http://docs.scala-lang.org/overviews/core/string-interpolation.html

- Napisati funkciju za namensku obradu znakovnih literala

```
implicit class Upper(val sc: StringContext) {
  def up(args: Any*): String = {
    val strings = sc.parts.iterator
    val expressions = args.iterator
    var buf = new StringBuffer(strings.next.toUpperCase)
    while(strings.hasNext) {
      buf append expressions.next
      buf append strings.next.toUpperCase
    }
    buf.toString
  }
}

val weight1 = 76.57
  val weight2 = 88.863
  println(up"average weight: ${ (weight1 + weight2)/2 }")
  //> AVERAGE WEIGHT: 82.7165
```

Zadatak 3.5

- Napisati funkciju za nalaženje zbira elemenata kvadratne matrice koji se nalaze u gornjem trouglu

```
type Matrix[T] = Array[ Array[T] ]

def sumTri[T <% Double](m : Matrix[T]) = {
  val length = m.length
  var value = 0 : Double
  for(row <- 0 until length; col <- row until length)
    value = value + m(row)(col)
  value
}

val matrix = Array.tabulate(3,3)( (x,y) => x+y )
//> matrix  : Array[Array[Int]] = Array(  Array(0, 1, 2),
//|                                      Array(1, 2, 3),
//|                                      Array(2, 3, 4))

sumTri(matrix)                  //> res0: Double = 12.0
```

Naznaka kako treba tumačiti literal

Zadatak 3.6

- Napisati funkciju koja vrši zadatu numeričku obradu nad zadatim elementima zadate matrice brojeva.

```
def processMatrix[@specialized(Int, Double, Long, Float) T ]
  (m : Matrix[T])
  (ident : Double)
  (ind : Iterable[(Int, Int)])
  (op : (Double, Double) => Double) : T =
  {
    var res = ident;
    for(i <- ind)
      res = op(res, matrix(i._1)(i._2));

    res.asInstanceOf[T]
  }
```

Generiše specijalizovanu metodu i metodu sa brisanjem tipa

Zadatak 3.6

```
val indices = List( (1, 2), (0, 2) )
//> indices  : List[(Int, Int)] = List((1,2), (0,2))

def maxEl(x:Double, y:Double) = if(x > y) x else y
//> maxEl: (x: Double, y: Double)Double

processMatrix(matrix)(0)(indices)(maxEl)
//> res1: Int = 3
```

Zadatak 3.6

```
val processMatrixSpec = processMatrix(matrix)(0)(_)
//> processMatrixSpec  :
//| Iterable[(Int, Int)] => (((Double, Double) => Double) => Int)
//| = <function1>

def elems[T](matrix : Matrix[T]) = {
   for(i <- 0 until matrix.length;
       j <- 0 until matrix.length if((i+j)%2==0) )
      yield (i, j)
}
//> elems: [T](matrix: w04.ranges.Matrix[T])(implicit evidence$2:
//| T => Double)scala.collection.immutable.IndexedSeq[(Int, Int)]

val myElems = elems(matrix)
//> myElems  : scala.collection.immutable.IndexedSeq[(Int, Int)]
//| = Vector((0,0), (0,2), (1,1), (2,0), (2,2))

val findAll = processMatrixSpec(myElems)(maxEl)
//> findAll  : Int = 4
```

Zadatak 3.7

- Za spisak knjiga, dat u formi liste naslova i autora, napisati iskaze u vidu *for-comprehension* za nalaženje
  - svih knjiga čiji je jedan od autora zadatog imena
  - svih knjiga čiji naslovi sadrže zadatu znakovni niz
  - svih autora koji su napisali najmanje dve knjige iz spiska

```
case class Book(title: String, authors: List[String])
val books: List[Book] = List(
    Book("Structure and Interpretation of Computer Programs",
       List("Abelson, Harold", "Sussman, Gerald J.")),
    Book("Principles of Compiler Design",
       List("Aho, Alfred", "Ullman, Jeffrey")),
    Book("Programming in Modula-2",
       List("Wirth, Niklaus")),
    Book("Introduction to Functional Programming",
       List("Bird, Richard")),
    Book("The Java Language Specification",
       List("Gosling, James", "Joy, Bill", "Steele, Guy",
            "Bracha, Gilad")))
```

Zadatak 3.7

- Za spisak knjiga, dat u formi liste naslova i autora, napisati iskaze u vidu *for-comprehension* za nalaženje
  - svih knjiga čiji je jedan od autora zadatog imena

```
for (b <- books; a <- b.authors if a startsWith "Ullman")
       yield b.title
//> res0: List[String] = List(Principles of Compiler Design)
```

Zadatak 3.7

- Za spisak knjiga, dat u formi liste naslova i autora, napisati iskaze u vidu *for-comprehension* za nalaženje
  - svih knjiga čiji naslovi sadrže zadati znakovni niz

```
for (b <- books if (b.title indexOf "Program") >= 0)
       yield b.title
//> res1: List[String] = List(Structure and Interpretation of
//| Computer Programs, Programming in Modula-2, Introduction to
//| Functional Programming)
```

Zadatak 3.7

- Za spisak knjiga, dat u formi liste naslova i autora, napisati iskaze u vidu *for-comprehension* za nalaženje
  - svih autora koji su napisali najmanje dve knjige sa spiska

```
for (   b1 <- books;
                b2 <- books if b1 != b2;
                a1 <- b1.authors;
                a2 <- b2.authors if a1 == a2)
        yield a1
//> res2: List[String] = List()
```

```
for {   b1 <- books
                b2 <- books if b1 != b2
                a1 <- b1.authors
                a2 <- b2.authors if a1 == a2
        }
        yield a1
//> res3: List[String] = List()
```

Mana rešenja:
dupliraće imena autora.

Za vežbu: popraviti.

Zadatak 3.8

- Prekidanje i nastavljanje ciklusa bez naredbi `break` i `continue`

```
import scala.util.control.Breaks._

object breakandcontinue {
  breakable {
    for (i <- 1 to 10) {
       println(i)
       if (i > 4) break  // break out of the for loop
    }
  }                                                    //> 1
                                                       //| 2
                                                       //| 3
                                                       //| 4
                                                       //| 5
  ...
}
```

Zadatak 3.8

Kako izgleda klasa Breaks?

```
package scala
package util.control

class Breaks {

  private val breakException = new BreakControl

  def breakable(op: => Unit) {
    try {
      op
    } catch {
      case ex: BreakControl =>
        if (ex ne breakException) throw ex
    }
  }

  def break(): Nothing = { throw breakException }
}
```

Zadatak 3.8

```
object breakandcontinue {
 ...
 var sumOdd = 0
 for (i <- 0 until 10) {
   breakable {
     if (i % 2 == 0)
     {
       break
     }
     else
     {
       sumOdd += i
     }
   }
  }
  println("Sum of odd numbers in range 0 to 10: " + sumOdd)
  //> Sum of odd numbers in range 0 to 10: 25
}
```

Zadatak 3.8

```
import util.control._

  val Inner = new Breaks
  val Outer = new Breaks
  Outer.breakable {
   for (i <- 1 to 5) {
     Inner.breakable {
       for (j <- 'a' to 'e')
       {
          if (i == 1 && j == 'c') Inner.break
          else println(s"i: $i, j: $j")

          if (i == 2 && j == 'b') Outer.break
       }
     }
  }
}
```

```
//> i: 1, j: a
//| i: 1, j: b
//| i: 2, j: a
//| i: 2, j: b
```

Zadatak 3.9

- Napisati funkciju koja sintaksno i semantički odgovara `while` ciklusu

```
sveDok(uslov) { telo }
```

U jeziku Scala ovo se može rešiti kerifikovanom funkcijom

- kada f-ja ima 1 parametar, on može biti u ( ) ili { }

```
@tailrec
def sveDok(testCondition: => Boolean)(codeBlock: => Unit) {
  if (testCondition) {
      codeBlock
      sveDok(testCondition)(codeBlock)
  }
}
```

Funkcionalno programiranje

Vežbe

04 Uparivanje obrazaca

Zadatak 4.1

- Napisati funkciju čija vrednost je deskriptivni opis (string) njenog argumenta upotrebom mehanizma uparivanja obrazaca.

```
package w05
object patternmatchingexamples {

  class Rational(x: Int, y: Int) {
    require(y != 0)
    private def gcd(a: Int, b: Int): Int = if(b==0) a else gcd(b, a%b)
    private val g = gcd(x.abs, y.abs)
    def numer = x / g
    def denom = y / g
    override def toString() = numer + "/" + denom
  }

  case class cRational(x : Int, y : Int) extends Rational(x, y)
  val cc = cRational(2,4)
  //> cc  : w05.patternmatchingexamples.cRational = 1/2
```

Zadatak 4.1

```
type F = (Int => Int)

def opisTipa(x: Any): String = x match {

  // Obrazac konstante
  case 0 => "nula"
  case true => "tacno"
  case "zdravo" => "tekst poruka 'zdravo'"
  case Nil => "Prazna lista"

  // Obrazac sekvence
  case List(0, _, _) => "Lista od 3 elementa koja pocinje nulom"
  case List(1, _*) =>
     "Lista proizvoljnog broja elemenata koja pocinje jedinicom"
  case List(d : Double, _*) =>
     "Lista proizvoljnog broja elemenata koja pocinje sa Double"
  case Vector(1, _*) =>
    "Vektor proizvoljnog broja elemenata koji pocinje jedinicom"
```

Zadatak 4.1

```
// Obrazac taplova
case (a, b) => s"Par sa elementima: $a i $b"
case (a, b, c) => s"Trojka sa elementima: $a, $b i $c"

// Obrasci konstruktora
case cRational(brojilac, 5) =>
    s"Racionalan broj, sa imeniocom 5 i brojiocem = $brojilac"

case cRational(brojilac, imenilac) =>
    s"Racionalan broj, $imenilac / $brojilac"
```

Zadatak 4.1

```
  // Obrasci tipa
  case s: String => s"String: $s"
  case i: Int => s"Ceo broj: $i"
  case f: Float => s"Realan broj: $f"
  case a: Array[Int] => s"Niz celih brojeva: ${a.mkString(",")}"
  case as: Array[String] => s"Niz stringova: ${as.mkString(",")}"
  case d: cRational => s"Racionalan broj: ${d}"
  case list: List[_] => s"Lista bilo kog tipa: $list"

  case m: Map[_, _] => m.toString

  case f : (Int => Int) => "Funkcija Int => Int"
  case f1 : F => "Funkcija Int => Int"

  // Obrazac koji uparuje sve
  case _ => "Nepoznato"

} //> opisTipa: (x: Any)String
```

non-variable type argument Int in type pattern Int ⇒ Int is unchecked since it is eliminated by erasure

Zadatak 4.1

```
  opisTipa( Array(1,2,3,4) )
  //> res0: String = Niz celih brojeva: 1,2,3,4
  opisTipa( Array("s1", "s2", "s3"))
  //> res1: String = Niz stringova: s1,s2,s3
  opisTipa( Array(1.0,2.0,3.0,4.0) )
  //> res2: String = Nepoznato

  opisTipa( List(1.0, 2, List('a','b')) )
  //> res3: String = Lista proizvoljnog broja elemenata koja pocinje
  //| jedinicom

 opisTipa( List(3.0, 2, List('a','b')) )
  //> res6: String = Lista proizvoljnog broja elemenata koja pocinje
  //| sa Double

  opisTipa( (x : Int ) => x)
  //> res4: String = Funkcija Int => Int
  opisTipa( (y : Float) => y*2 )
  //> res5: String = Funkcija Int => Int
}
```

Zadatak 4.1

```
case List(1, _*) =>
      "Lista proizvoljnog broja elemenata koja pocinje jedinicom"

case list: List[_] => s"Lista bilo kog tipa: $list"

case list2 : List(2, _*) =>
      s"Lista koja pocinje dvojkom: $list2"        // greška!
```

Da bi se ovo realizovalo, potrebno je kombinovati
obrazac vezivanja promenljive za obrazac sekvence

```
case list2 @ List(2, _*) =>
      s"Lista koja pocinje dvojkom: $list2"        // ispravno
```

Zadatak 4.2

- (a) Napisati funkciju za nalaženje najvećeg zajedničkog delioca dva cela broja primenom uparivanja obrazaca.

```
object gcd {
  def gcd(first: Int, second: Int): Int = (first, second) match {
    case (0, b) => b
    case (a, 0) => a
    case (a, b) if a > b => gcd(a - b, b)
    case (a, b) => gcd(a, b - a)
  } //> gcd: (first: Int, second: Int)Int

  gcd(3, 5)                                       //> res0: Int = 1
}
```

Zadatak 4.2

- (b) Proširiti funkcionalnost klase String tako da može da konvertuje znakovni niz u logičku vrednost:
  - `"", " ", "0", "zero"` u `false`
  - bilo koja druge vrednost u `true`

```
  implicit class StringImprovements(val s: String) {

    def asBoolean = s match {
      case "0" | "zero" | "" | " " => false
      case _ => true
    }
}

"zero".asBoolean                          //> res0: Boolean = false
"0".asBoolean                             //> res1: Boolean = false

"test".asBoolean                          //> res2: Boolean = true
if( "test".asBoolean ) true else false    //> res3: Boolean = true
```

Zadatak 4.2

- Komentar: funkcionalnost klase String je proširena, ali neće biti implicitno korišćena.

- Za implicitnu upotrebu treba napisati implicitne funkcije:

```
implicit def strToBool(s: String) : Boolean = s match {
  case "0" | "zero" | "" | " " => false
  case _ => true
}

if( "test" ) true else false  //> res3: Boolean = true
```

Zadatak 4.2

- (c) Proširiti funkcionalnost klase `String` dodavanjem bezbedne konverzije iz tipa `String` u tip `Int`, bez bacanja izuzetaka.

```
  implicit class StringImprovements(val s: String) {

    def toInteger : Option[Int] = {
      try {
        Some(Integer.parseInt(s))
      } catch {
        case e: NumberFormatException => None
      }
    }
}

"test".toInteger match {
    case Some(x) => println(x)
    case None => println("Not a number")
}                                          //> Not a number
```

Zadatak 4.3

- Novčanik sadrži određen broj kovanica u apoenima 1..5 dinara i novčanica u apoenima 10..5000 dinara. Sadržaj novčanika je predstavljen listom parova (apoen, količina). Odrediti:
  - da li je novčanik prazan?
  - da li ima novčanice?
  - koliko novčanica apoena 200 dinara sadrži?

Zadatak 4.3

```
val novcanik = List( (10, 0), (20, 1), (50, 2), (100, 5), (200, 3),
                     (500, 2), (1000, 1), (2000, 0), (5000, 0) )

novcanik match {

  case l : List[_] if l.length < 1 => "Prazan"

  case l : List[(Int,Int)] // ili List[(_,_)]
     if l.forall(x => x._2 == 0) => "Nema novcanica od 10 do 5000"


  case l : List[(Int,Int)]
     if l.exists(x => (x._1 == 200 && x._2 > 0)) =>
       s"Sadrzi ${l.find(x => x._1 == 200).get._2} novcanica po 200"

}
//> res6: String = Sadrzi 3 novcanica po 200
```

Zadatak 4.4

- Napisati potrebne klase za predstavljanje i računanje vrednosti numeričkih izraza i obezbediti njihov ispis.
  - Obezbediti operacije +, - i *
  - Pri ispisu koristiti zagrade samo kada su neophodne

Zadatak 4.4
Rešenje 1

```
object expression {

  trait Expression
  case class Number(n: Int) extends Expression
  case class Variable(s : String, var n : Int) extends Expression
  case class Add(e1: Expression, e2: Expression) extends Expression
  case class Sub(e1: Expression, e2: Expression) extends Expression
  case class Prod(e1: Expression, e2: Expression) extends Expression


  val c = Number(2)
                //> c  : w05.expression.Number = Number(2)
  println(c)
                //> Number(2)
```

Zadatak 4.4
Rešenje 1

```
def eval(e: Expression): Int = e match {
  case Number(n) => n
  case Variable(_, n) => n
  case Add(e1, e2) => eval(e1)+eval(e2)
  case Sub(e1, e2) => eval(e1)-eval(e2)
  case Prod(e1, e2) => eval(e1)*eval(e2)
  case _ => 0
}

eval(Sub(Number(3), Number(2)))           //> res0: Int = 1
eval(Sub(Number(4), Number(2)))           //> res1: Int = 2
eval(Prod(Number(2), Variable("x", 3)))   //> res2: Int = 6

val a = Variable("a", 5)
                  //> a  : w05.expression.Variable = Variable(a,5)
eval(a)                                   //> res3: Int = 5
a.n = 6
eval(a)                                   //> res4: Int = 6
```

Zadatak 4.4
Rešenje 1

```
def show(e: Expression): String = e match {
  case Number(n) => ""+n
  case Variable(s, _) => s

  case Prod(Add(e1, e2), e3) =>
         "(" + show(e1) + "+" + show(e2) + ")" + "*" + show(e3)

  case Prod(Sub(e1, e2), e3) =>
         "(" + show(e1) + "-" + show(e2) + ")" + "*" + show(e3)

  case Add(e1, e2) => show(e1) + "+" + show(e2)
  case Sub(e1, e2) => show(e1) + "-" + show(e2)
  case Prod(e1, e2) => show(e1) + "*" + show(e2)
  case _ => "???"
}
```

Zadatak 4.4
Rešenje 1

```
show(Number(3))                                  //> res5: String = 3

show(Add(Number(2), Number(3)))                  //> res6: String = 2+3

show(Add(Number(3), Sub(Number(4), Number(2))))
                                                 //> res7: String = 3+4-2

show(Prod(Number(2), Variable("x", 3)))          //> res8: String = 2*x

show(Add(Prod(Number(2), Number(3)), Variable("x", 4)))
                                                 //> res9: String = 2*3+x

show(Prod(Add(Number(2), Number(3)), Variable("x", 4)))
                                                 //> res10: String = (2+3)*4

show(Prod(Variable("y", 5), Sub(Variable("x", 3), Number(2))))
                                                 //> res11: String = y*x-2
```

Zadatak 4.4
Rešenje 1

- Analiza rešenja
- Posebno moraju da se razmatraju slučajevi
  - x * (y+z)
  - (y+z) * x
- Posebno se razmatra
  - zbir od proizvoda zbirovaa
  - razlika od proizvoda razlika
- Mnogo slučajeva koje je potrebno pokriti
- Nepraktično!

Zadatak 4.4
Rešenje 2

```
object expression2 {

  trait Expr {

    def +(that: Expr): Expr = (this, that) match {

      case (Num(0), e) => e

      case (Num(n), Num(m)) => Num(n+m)

      case (Var(x, a), Var(y, b)) if x==y => Num(2) * Var(x, a)

      case (Mul(e1, Var(x, a)), Mul(e2, Var(y, b))) if x==y =>
              (e1 + e2) * Var(x, a)

      case _ => Add(this, that)
    }
```

Zadatak 4.4
Rešenje 2

```
def -(that: Expr): Expr = (this, that) match {
  case (Num(0), e) => Mul(Num(-1), e)
  case (e, Num(0)) => e
  case (Num(n), Num(m)) => Num(n-m)
  case (Var(x, _), Var(y, _)) if x==y => Num(0)

  case (Mul(e1, Var(x, a)), Mul(e2, Var(y, b))) if x==y =>
            (e1 - e2) * Var(x, a)

  case _ => Sub(this, that)
}

def *(that: Expr): Expr = (this, that) match {
  case (Num(0), e) => Num(0)
  case (Num(1), e) => e
  case (Num(n), Num(m)) => Num(n*m)
  case (Var(x, a), Num(n)) => Num(n) * Var(x, a)
  case _ => Mul(this, that)
}
```

Zadatak 4.4
Rešenje 2

```
  def unary_! : Int = this match {
    case Num(x) => x
    case Var(_, x) => x
    case Add(a, b) => !a + !b
    case Sub(a, b) => !a - !b
    case Mul(a, b) => !a * !b
  }
}

case class Num(x: Int) extends Expr {
       override def toString = x.toString
}

case class Var(name: String, x : Int) extends Expr {
       override def toString = name
}
```

Zadatak 4.4
Rešenje 2

```
case class Add(e1: Expr, e2: Expr) extends Expr {
  override def toString = e1.toString + " + " + e2.toString
}

case class Sub(e1: Expr, e2: Expr) extends Expr {
  override def toString = e1.toString + " - " + e2.toString
}

case class Mul(e1: Expr, e2: Expr) extends Expr {
  override def toString = {
    def factorToString(e: Expr) = e match {
      case Add(_, _) => "(" + e.toString + ")"
      case Sub(_, _) => "(" + e.toString + ")"
      case _ => e.toString
    }

    factorToString(e1) + " * " + factorToString(e2)
  }
}
```

Zadatak 4.4
Rešenje 2

```
  !Num(3)                                          //> res0: Int = 3
  !(Num(2) + Num(3))                               //> res1: Int = 5
  !(Num(3) + Num(4) - Num(2))                      //> res2: Int = 5
  !(Num(2) * Var("x", 3))                          //> res3: Int = 6

  val r4= (Num(2) * Num(3) + Var("x", 4))
  println(r4)                                      //> 6 + x
  !r4                                              //> res4: Int = 10

  val r5 = (Num(2) * (Num(3) + Var("x", 4)))
  println(r5)                                      //> 2 * (3 + x)
  !r5                                              //> res5: Int = 14

  val r6 = (Var("y", 5) * (Var("x", 3) - Num(2)))

  println(r6)                                      //> y * (x - 2)
  !r6                                              //> res6: Int = 5
}
```

Zadatak 4.5

- Napisati potrebne klase za predstavljanje i manipulaciju stablima binarnog pretraživanja sa proizvoljnim tipom sadržaja čvorova uz upotrebu tehnike uparivanja obrazaca.

Zadatak 4.5
Rešenje 1 – samo za tip Int

```
object tree {

  abstract class IntTree

  case class Empty() extends IntTree

  case class Node(elem: Int, left: IntTree, right: IntTree)
    extends IntTree

  object Node {
    // Redundantno, kompajler prijavljuje gresku
    // def apply(elem: Int, left: IntTree, right: IntTree) =
    //     new Node(elem, left, right)
    def apply(elem: Int) = new Node(elem, Empty(), Empty())
  }
```

Zadatak 4.5
Rešenje 1

```
def contains(t: IntTree, v: Int): Boolean = t match {
  case n : Empty => false
  case Node(el, _, _) if(el == v) => true
  case Node(el, left, right) =>
    if( v < el ) contains(left, v) else contains(right, v)
}

def add(t: IntTree, v: Int) : IntTree = t match {
  case n : Empty => Node(v, Empty(), Empty())
  case Node(el, _, _) if( el == v ) => t
  case Node(el, left, right) =>
    if( v < el ) Node(el, add(left, v), right)
    else Node(el, left, add(right, v))
}
```

Zadatak 4.5
Rešenje 1

```
  val root  = Node(10, Node(3), Empty())
  //> root  : w05.tree.Node = Node(10,Node(3,Empty(),Empty()),Empty())

  contains(root, 5)
  //> res0: Boolean = false

  contains(root, 10)
  //> res1: Boolean = true

  contains(root, 3)
  //> res2: Boolean = true

  add(root, 5)
  //> res3: w05.tree.IntTree =
  //| Node(10,Node(3,Empty(),Node(5,Empty(),Empty())),Empty())
}
```

Zadatak 4.5
Rešenje 2

```
object tree2 {

  abstract class Tree[T]

  case class Empty[T]() extends Tree[T]

  case class Node[T](elem: T, left: Tree[T], right: Tree[T])
      extends Tree[T]

  object Node {
    def apply[T](elem: T) = new Node(elem, Empty(), Empty())
  }
```

Zadatak 4.5
Rešenje 2

```
def contains[T](t: Tree[T], v: T)
                (implicit ord: Ordering[T]): Boolean = t match {
  case n : Empty[T] => false
  case Node(el, _, _) if(el == v) => true
  case Node(el, left, right) =>
    if( ord.gt(el, v) ) contains(left, v)
    else contains(right, v)
}

def add[T](t: Tree[T], v: T)
          (implicit ord: Ordering[T]) : Tree[T] = t match {
  case n : Empty[T] => Node(v, Empty(), Empty())
  case Node(el, _, _) if( el == v ) => t
  case Node(el, left, right) =>
    if( ord.gt(el, v) ) Node(el, add(left, v), right)
    else Node(el, left, add(right, v))
}
```

Zadatak 4.5
Rešenje 2

```
  val root  = Node(10, Node(3), Empty())
  contains(root, 5)                          //> res0: Boolean = false
  contains(root, 10)                         //> res1: Boolean = true
  contains(root, 3)                          //> res2: Boolean = true

  add(root, 5)
  //> res3: w05.tree2.Tree[Int] =
  //| Node(10,Node(3,Empty(),Node(5,Empty(),Empty())) ,Empty())

  var root2 = Node("ABC")
  root2 = add(root2, "ADE")
  root2 = add(root2, "XYZ")
  root2 = add(root2, "ADA")
  //> root2  : w05.tree2.Tree[String] =
  //| Node(ABC,Empty(),Node(ADE,Node(ADA,Empty(
  //| ),Empty()),Node(XYZ,Empty(),Empty())))
}
```

Zadatak 4.5

Rešenje 3 – kombinovanje sa OO dekompozicijom

```
object tree3 {

  abstract class Tree[T] {
    def contains(v: T)
                (implicit ord: Ordering[T]): Boolean = this match {
      case n: Empty[T] => false
      case Node(el, _, _) if(el == v) => true
      case Node(el, left, right) =>
        if( ord.gt(el, v) ) left contains el
        else right contains el
    }

    private[tree3] def addInternal(v: T)(ord: Ordering[T]): Tree[T]

    def add(v: T)(implicit ord: Ordering[T]): Tree[T] = {
      if( this contains v ) this
      else addInternal(v)(ord)
    }
```

Zadatak 4.5

Rešenje 3 – kombinovanje sa OO dekompozicijom

```
  override def toString = this match {
    case n : Empty[T] => "."
    case Node(el, l, r) =>
      "(" + (l toString) + "_" + el + "_" + (r toString) + ")"
  }
}

case class Empty[T]() extends Tree[T] {
    override def addInternal(v: T)
                              (ord: Ordering[T]): Tree[T] = Node(v)
}

case class Node[T](elem: T, left: Tree[T], right: Tree[T])
     extends Tree[T] {
  override def addInternal(v: T)(ord: Ordering[T]): Tree[T] = {
    if( ord.gt(elem, v) )
      Node(elem, left.addInternal(v)(ord), right)
    else Node(elem, left, right.addInternal(v)(ord))
  }
}
```

Zadatak 4.5

Rešenje 3 – kombinovanje sa OO dekompozicijom

```
    object Node {
      def apply[T](elem: T) = new Node(elem, Empty(), Empty())
    }

    val root = Empty().add(10).add(5).add(15)
    //> root  : w05.tree3.Tree[Int] = ((._5_.)_10_(._15_.))

    root contains 3                          //> res0: Boolean = false

    val root2 = Node("ABC").add("ADE").add("XYZ").add("ADA")
    //> root2  : w05.tree3.Tree[String] =
    //| (._ABC_((._ADA_.)_ADE_(._XYZ_.)))
}
```

Funkcionalno programiranje

Vežbe

05 Crte

Zadatak 1

- Postoje dva standarda za električne komponente: standard po kojem električne komponente koriste mrežni napon od 110 V i standard po kojem koriste 220 V.
- Šuko utikači i utičnice su definisane za mrežni napon od 220 V.
- Usisivač A koristi mrežni napon od 220 V i šuko utikač.
- Obezbediti da usisivač B, koji koristi mrežni napon od 110 V ne može da poseduje šuko utikač.

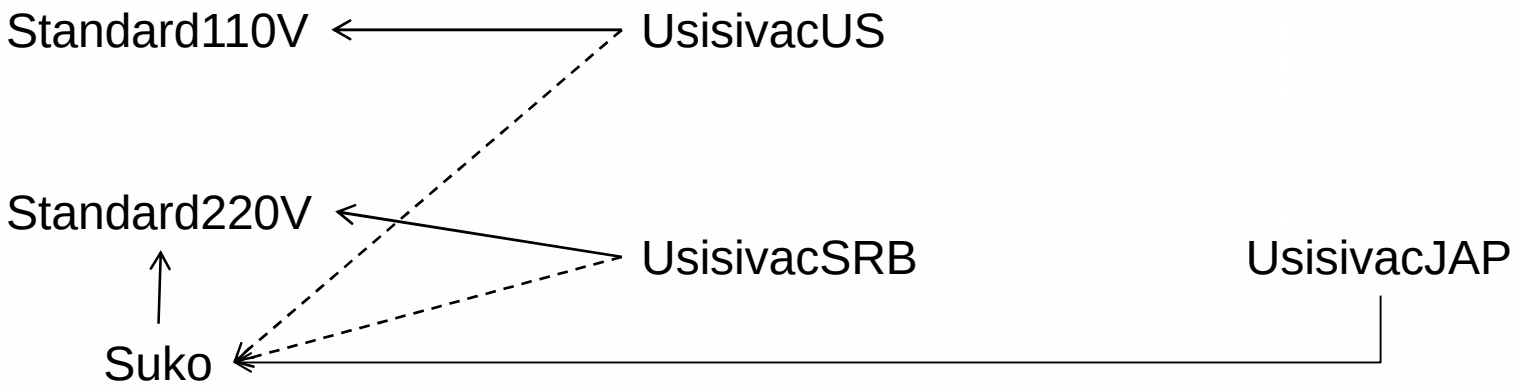
Zadatak 1

```
object elkomponente {
  class Standard110V
  class Standard220V
  trait Suko extends Standard220V

  class UsisivacSRB extends Standard220V with Suko

  class UsisivacUS extends Standard110V with Suko // greška

  class UsisivacJAP extends Suko // u redu???
}
```



Zadatak 1

- Problem – ne postoji ograničenje da uređaj mora da poštuje Standard 220 V.
- Implicitno ga poštuje, ali to može biti logička greška
  - Prevodilac ne detektuje
- Rešenje: crta mora da propiše nadklasu u koju će biti umetnuta

```
object elkomponente2 {
  class Standard110V
  class Standard220V
  trait Suko  {
    this: Standard220V =>
  }
  class UsisivacSRB extends Standard220V with Suko
  class UsisivacUS extends Standard110V with Suko // greska
  class UsisivacJAP extends Suko // greska
}
```

Zadatak 2

- Osobina pokretljivosti obezbeđuje pomeranje objekta u 2D ravni. Ova osobina zahteva da objekat može biti postavljen na proizvoljnu poziciju u 2D ravni.
- Napisati potrebne crte i klase tako da List i Kamen mogu imati osobinu pokretljivosti, ali Planina ne.

Zadatak 2

```
object movable {
  trait Position {
    protected var x: Int
    protected var y: Int
    def setPos(x: Int, y: Int) = {
      this.x = x
      this.y = y
    }
  }
  trait Movable extends Position {
    def move(x: Int, y: Int) = setPos(x,y)
  }
  class Leaf(protected override var x: Int,
             protected override var y: Int) extends Movable
  class Rock(protected override var x: Int,
             protected override var y: Int)  extends Movable
  class Mountain(protected override var x: Int,
                 protected override var y: Int)  extends Movable
  new Mountain(2,3)
  //> res0: w06.movable.Mountain = w06.movable$Mountain@f2a0b8e
}
```

Zadatak 2

```
object movable2 {

  trait Position {
    protected var x: Int
    protected var y: Int
    def setPos(x: Int, y: Int) = {
      this.x = x
      this.y = y
    }
  }

  trait Movable  {

    this: { def setPos(x: Int, y: Int): Unit } =>

    def move(x: Int, y: Int) = setPos(x,y)
  }
```

Zadatak 2

```
  class Leaf(protected override var x: Int,
             protected override var y: Int)
                                      extends Position with Movable

  class Rock(protected override var x: Int,
             protected override var y: Int)  extends Movable {
    def setPos(x: Int, y: Int) = { /*...*/ }
  }

  // greška
  class Mountain(protected override var x: Int,
                 protected override var y: Int) extends Movable
}
```

Funkcionalno programiranje

Vežbe

06 Liste

Zadatak 6.1

- Napisati implementaciju metode `:::` koja vrši konkatenaciju dve generičke liste.

- Analiza:
  - svi operatori koji se završavaju sa `:` grupišu se s desna ulevo
  - ```
    List(1, 2) ::: List(3, 4)
    = List(3, 4).:::(List(1, 2))
    = List(1, 2, 3, 4)
    ```
  - Dakle: parametar funkcije je zapravo prefiks rezultujuće liste

  - S obzirom na to da je lista kovarijantna, treba obezbediti da se u listu mogu nadovezivati liste nadtipova
    ```

Zadatak 6.1
Rešenje 1 – ručno napravljena lista

```
object lists {
  trait List[T] {
    def prazna: Boolean
    def glava: T
    def rep: List[T]
    def concat(l2 : List[T]) : List[T] = this match {
      case Nil() => l2
      case Elem(h, t) => Elem(h, t concat l2)
    }
    def :::(l2: List[T]) : List[T] = {
      l2.concat(this)
    }
  }

  case class Nil[T]() extends List[T] {
    def prazna = true
    def glava = throw new NoSuchElementException("Nil.glava")
    def rep = throw new NoSuchElementException("Nil.rep")
  }
```

Zadatak 6.1
Rešenje 1 – ručno napravljena lista

```
  case class Elem[T](val glava : T, val rep : List[T])
       extends List[T]  {
    def prazna = false
  }

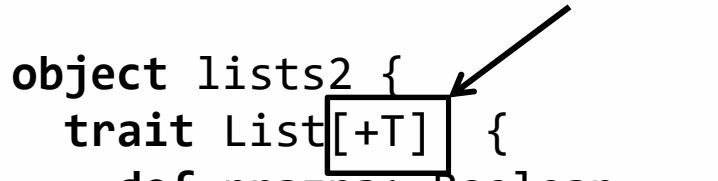
  val l1 = Elem(1, Nil()) //> l1:w06.lists.Elem[Int] = Elem(1,Nil())
  val l2 = Elem(2, Nil()) //> l2:w06.lists.Elem[Int] = Elem(2,Nil())

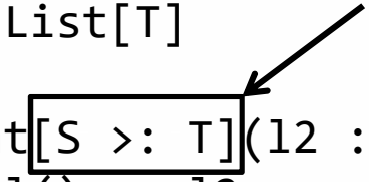
  l1.concat(l2)  //> res0:w06.lists.List[Int] = Elem(1,Elem(2,Nil()))

  l1 ::: l2      //> res1:w06.lists.List[Int] = Elem(1,Elem(2,Nil()))

  val l3 = Elem('a', Nil())
                 //> l3 : w06.lists.Elem[Char] = Elem(a,Nil())
  l1 ::: l3      // ????
}
```

Zadatak 6.1
Rešenje 2 – podrška za liste srodnih tipova





```
object lists2 {
  trait List[+T]  {
    def prazna: Boolean
    def glava: T
    def rep: List[T]

    def concat[S >: T](l2 : List[S]) : List[S] = this match {
      case Nil() => l2
      case Elem(h, t) => Elem(h, t concat l2)
    }
    def :::[S >: T](l2: List[S]) : List[S] =  {  l2 concat this  }
  }

  case class Nil[T]() extends List[T]  {
    def prazna = true
    def glava = throw new NoSuchElementException("Nil.glava")
    def rep = throw new NoSuchElementException("Nil.rep")
  }
```

Zadatak 6.1
Rešenje 2 – podrška za liste srodnih tipova

```
  case class Elem[T](val glava : T, val rep : List[T])
  extends List[T]  {    def prazna = false   }

  val l1 = Elem( 1, Nil() )
             //> l1:w06.lists2.Elem[Int] = Elem(1,Nil())
  val l2 = Elem( 'a', Nil() )
             //> l2:w06.lists2.Elem[Char] = Elem(a,Nil())

  l1 ::: l2  //> res1:w06.lists2.List[AnyVal] = Elem(1,Elem(a,Nil()))

  l2 ::: l1  //> res2:w06.lists2.List[AnyVal] = Elem(a,Elem(1,Nil()))

  val l3 = Elem( Nil(), Nil() )
  //> l3:w06.lists2.Elem[w06.lists2.Nil[Nothing]] = Elem(Nil(),Nil())

  l1 ::: l3
  //> res3: w06.lists2.List[Any] = Elem(1,Elem(Nil(),Nil()))
}
```

Zadatak 6.1
Rešenje 3 – `scala.List[T]`

```
sealed abstract class List[+A] extends AbstractSeq[A]
                              with LinearSeq[A]
                              with Product
                              with GenericTraversableTemplate[A, List]
                              with LinearSeqOptimized[A, List[A]]
                              with Serializable {
  ...
  def isEmpty: Boolean
  def head: A
  def tail: List[A]

  def ::[B >: A] (x: B): List[B] =
    new scala.collection.immutable.::(x, this)

  def :::[B >: A](prefix: List[B]): List[B] =
    if (isEmpty) prefix
    else if (prefix.isEmpty) this
    else (new ListBuffer[B] ++= prefix).prependToList(this)
  ...
```

Zadatak 6.2

- Napisati implementaciju funkcije `take(List[T], n)`, čiji rezultat je nova lista formirana od prvih n elemenata generičke liste.Ako je n veće od dužine liste, rezultat je sama lista.

Zadatak 6.2

```
def take[T](l : List[T], n : Int) : List[T] =
{
  def takea[T](l : List[T], n : Int) : List[T] =
    if( n > 0 )  l.head :: takea(l.tail, n-1)
    else Nil

  if( n > l.length ) l
  else takea(l, n)
}

val lista = 1 :: 2 :: 3 :: 5 :: 4 :: 8 :: 6 :: Nil

take(lista, 4)              //> res1: List[Int] = List(1, 2, 3, 5)
```

Zadatak 6.3

- Napisati implementaciju funkcije `drop(List[T], n)`, čiji rezultat je nova lista formirana izostavljajući prvih n elemenata generičke liste. Ako je n veće od dužine liste, rezultat je prazna lista.

Zadatak 6.3

```
def drop[T](l : List[T], n : Int) : List[T] =
{
  def dropa[T](l : List[T], n : Int) : List[T] =
      if( n > 0 ) dropa(l.tail, n-1 )
      else l

  if( n > l.length ) List()
  else  dropa(l, n)
}

val lista = 1 :: 2 :: 3 :: 5 :: 4 :: 8 :: 6 :: Nil

drop(lista, 3)            //> res0: List[Int] = List(5, 4, 8, 6)
```

Zadatak 6.4

- Napisati implementaciju funkcije `splitAt(List[T], n)`, čiji rezultat je par lista. Prvi element para je lista formirana izostavljajući prvih n elemenata ulazne liste. Drugi element para je lista koja se sastoji od preostalih elemenata ulazne liste.

Zadatak 6.4

```
def splitAtEasy[T](l : List[T], n : Int) : (List[T], List[T]) =
{

  (take(l, n), drop(l, n))

}

val lista = 1 :: 2 :: 3 :: 5 :: 4 :: 8 :: 6 :: Nil

splitAtEasy(lista, 3)
//> res2: (List[Int], List[Int]) = (List(1, 2, 3),List(5, 4, 8, 6))
```

Zadatak 6.4

```
def splitAt[T](l : List[T], n : Int) : (List[T], List[T]) =
{
  var a = l;

  def takefront(l : List[T], n : Int) : List[T] =
    if( n > 0 ) l.head :: takefront(l.tail, n-1)
    else {
      a = l
      Nil
    }

  if( n > l.length ) (l, Nil)
  else (takefront(l,n), a)
}

val lista = 1 :: 2 :: 3 :: 5 :: 4 :: 8 :: 6 :: Nil

splitAt(lista, 3)
//> res3: (List[Int], List[Int]) = (List(1, 2, 3),List(5, 4, 8, 6))
```

Zadatak 6.4

Implementacija u `scala.collection.immutable`

```
override def splitAt(n: Int): (List[A], List[A]) = {
  val b = new ListBuffer[A]
  var i = 0
  var these = this
  while (!these.isEmpty && i < n) {
    i += 1
    b += these.head
    these = these.tail
  }
  (b.toList, these)
}
```

Zadatak 6.5

- Implementirati metodu apply() koja vraća n-ti element generičke liste.

```
def apply[T](l : List[T], n : Int) : T =
  drop(l, n).head
//> apply: [T](l: List[T], n: Int)T


// Implementacija iz crte LinearSeqOptimized
def apply(n: Int): A = {
  val rest = drop(n)

  if (n < 0 || rest.isEmpty)
    throw new IndexOutOfBoundsException("" + n)

  rest.head
}
```

Zadatak 6.6

- Napisati funkciju za sortiranje generičke liste primenom tehnike uparivanja obrazaca.

Zadatak 6.6
Rešenje 1

```
def msort[T](cmp : (T,T) => Boolean)(l : List[T]) : List[T] =  {
  val n = l.length / 2
  if( n == 0 ) l
  else  {
    def merge(l1 : List[T], l2 : List[T]) : List[T] = l1 match  {
      case List() => l2
      case h :: t => l2 match {
            case List() => l1
            case h2 :: t2 => if( cmp(h, h2) ) h :: merge(t, l2)
                       else h2 :: merge(l1, t2)
          }
    }
    merge(msort(cmp)(l take n), msort(cmp)(l drop n))
  }
}

msort((x:Int,y:Int) => x < y) (List(3, 5, 1, 2, 7, 4))
//> res7: List[Int] = List(1, 2, 3, 4, 5, 7)
```

Zadatak 6.6
Rešenje 2

```
import math.Ordering
object MergeSort {
  def msort[T](list: List[T])(implicit ord: Ordering[T]): List[T] = {
    val mid = list.length / 2
    if (mid == 0) list
    else {
      def merge(xs: List[T], ys: List[T]): List[T] = (xs, ys) match {
        case (Nil, ys) => ys
        case (xs, Nil) => xs
        case (x :: xs1, y :: ys1) =>
          if (ord.lt(x, y)) x :: merge(xs1, ys)
          else y :: merge(xs, ys1)
      }
      val (first, second) = list.splitAt(mid)
      merge(msort(first), msort(second))
    }
  }
  println(msort(List(2, 1, 5, 2, 76, 7, 2)))
  //> List(1, 2, 2, 2, 5, 7, 76)
}
```

Zadatak 6.7

- Napisati funkciju `map` koja od zadate generičke liste formira novu listu primenom funkcije preslikavanja na njene elemente

```
def map[S, D](f : S => D)(l : List[S]) =
{
  for(e <- l)
    yield f(e)
}

map( (x : Int) => x + 1 )(List(1, 2, 3))
//> res8: List[Int] = List(2, 3, 4)
```

Zadatak 6.7

Implementacija u `scala.collection.immutable`

```
final override def map[B, That](f: A => B)
      (implicit bf: CanBuildFrom[List[A], B, That]): That = {
  if (bf eq List.ReusableCBF) {
    if (this eq Nil) Nil.asInstanceOf[That] else {
      val h = new ::[B](f(head), Nil)
      var t: ::[B] = h
      var rest = tail
      while (rest ne Nil) {
        val nx = new ::(f(rest.head), Nil)
        t.tl = nx
        t = nx
        rest = rest.tail
      }
      h.asInstanceOf[That]
    }
  }
  else super.map(f)
}
```

Zadatak 6.8

- Napisati funkciju koja od generičke liste proizvoljnih elemenata pravi njenu linearizaciju.

Primer:

```
List( List(List(4,5),List(5,6)), List(List(7,8)), 9)
```

pretvara u:

```
List(4, 5, 5, 6, 7, 8, 9)
```

Zadatak 6.8

- Generička klasa List poseduje metodu `flatten` koja obavlja sličan posao.

Primer:

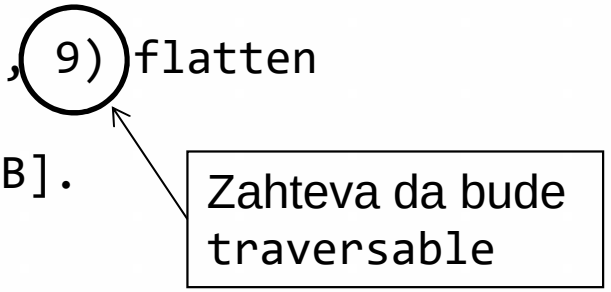
```
List(List(4,5), List(5,6), List(7,8)) flatten
//> res6: List[Int] = List(4, 5, 5, 6, 7, 8)
```

Ali:

```
List(List(List(4,5),List(5,6)), List(List(7,8))) flatten
//> res7: List[List[Int]] = List(List(4, 5), List(5, 6), List(7, 8))
```

Greška:

```
List(List(List(4,5),List(5,6)), List(List(7,8)), 9) flatten
//> No implicit view available from
//| Any => scala.collection.GenTraversableOnce[B].
```



Zadatak 6.8

```
def flatten(l : List[Any]) : List[Any] = l match {
  case Nil => Nil
  case (h : List[_]) :: t => flatten(h) ::: flatten(t)
  case h :: t => h :: flatten(t)
}
```

```
val test2 = List(List(List(4,5),List(5,6)), List(List(7,8)), 9)
//> test2  : List[Any] = List(List(List(4, 5), List(5, 6)),
//| List(List(7, 8)), 9)

flatten(test2)
//> res5: List[Any] = List(4, 5, 5, 6, 7, 8, 9)
```

Zadatak 6.9

- Napisati funkciju reverse, linearne vremenske složenosti, koja obrće redosled elemenata.

```
def reverse[A](l : List[A]) : List[A] = l
match {
  case List() => List()
  case List(x) => List(x)
  case h :: t => concat(reverse(t), List(h))
}

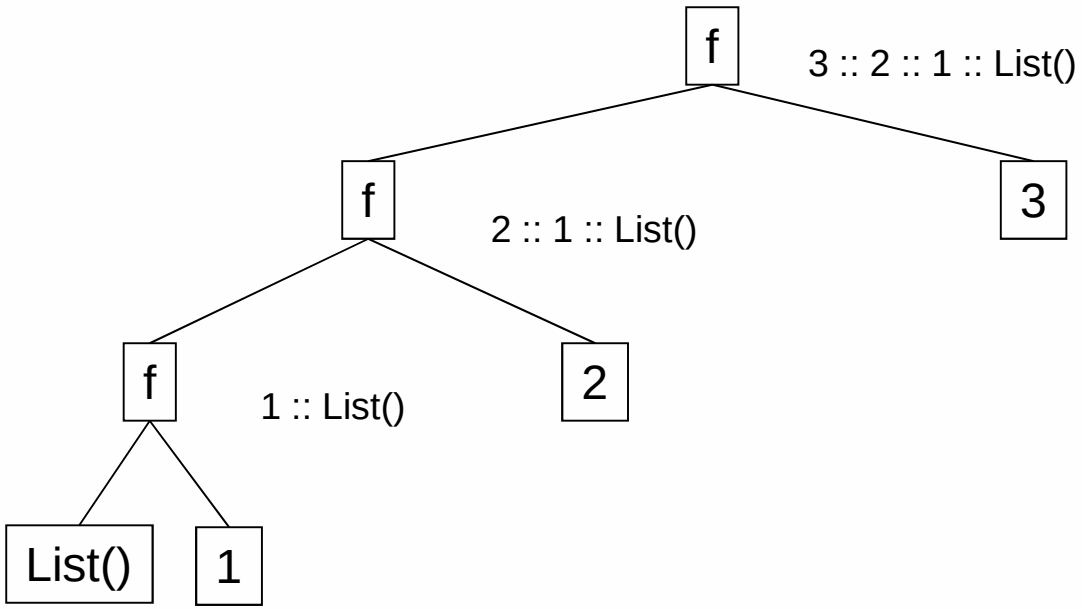
def concat[A](l1 : List[A], l2 :
List[A]) : List[A] = l1 match {
  case List() => l2
  case h :: t => h :: concat(t, l2)
}
```

Kvadratna složenost:
-listu obrće element po element
-konkatenacija ima linearnu složenost

Zadatak 6.9

```
def reverse[T](l : List[T]) : List[T] =
{
  (List[T]() /: l)( (left, right) => right :: left )
}

reverse( List(1, 2, 3) )
//> res13: List[Int] = List(3, 2, 1)
```



Funkcionalno programiranje

Vežbe

061 Kolekcije

Zadatak 1

- Ispisati elemente sekvencijalne strukture zajedno sa njihovim rednim brojevima.

```
val fruits = Array("apple", "banana", "orange")
//> fruits  : Array[String] = Array(apple, banana, orange)

for (i <- 0 until fruits.size) println(s"element $i is ${fruits(i)}")
                                                  //> element 0 is apple
                                                  //| element 1 is banana
                                                  //| element 2 is orange

for ((elem, count) <- fruits.zipWithIndex)
 println(s"element $count is $elem")              //> element 0 is apple
                                                  //| element 1 is banana
                                                  //| element 2 is orange
// zipWithIndex:
// Array( (apple, 0), (banana,1), (orange,2) )
```

Zadatak 1

```
for ((elem, count) <- fruits.view.zipWithIndex)
  println(s"element $count is $elem")
                                          //> element 0 is apple
                                          //| element 1 is banana
                                          //| element 2 is orange
```

```
for ((elem,count) <- fruits.zip(Stream from 1))
println(s"element $count is $elem")
                                          //> element 1 is apple
                                          //| element 2 is banana
                                          //| element 3 is orange
```

Zadatak 2

- Realizovati for/yield nad kolekcijom primenom metode map

Metoda map primenjuje zadatu funkciju nad svim elementima kolekcije i pravi novu kolekciju.

```
val gradovi = List("Beograd", "Cacak", "Pozega", "Novi Sad", "Nis")
//> gradovi  :
//  List[String] = List(Beograd, Cacak, Pozega, Novi Sad, Nis)

for(g <- gradovi if g.length() < 6) yield g.toUpperCase()
//> res0: List[String] = List(CACAK, NIS)


gradovi.map(x => if(x.length()<6) x.toUpperCase() )
//> res1: List[Any] = List((), CACAK, (), (), NIS)
```

Zadatak 2

- U opštem slučaju, samo `map` nije dovoljno
  - `map` mora da se primeni nad svakim elementom i da rezultujuće preslikavanje
  - ako postoji uslov koji nije ispunjen, prazna else grana tumači se kao `Unit`.

```
gradovi.map(x => if(x.length()<6) x.toUpperCase() )
//> res1: List[Any] = List((), CACAK, (), (), NIS)

gradovi.map(x => if(x.length()<6) x.toUpperCase() ).filter( _ != () )
//> res2: List[Any] = List(CACAK, NIS)

gradovi.filter( _.length()<6 ).map(_.toUpperCase())
//> res3: List[String] = List(CACAK, NIS)


gradovi.view.filter(_.length()<6).map(_.toUpperCase()).force
//> res4: Seq[String] = List(CACAK, NIS)
```

Zadatak 3

- Data je mapa koja preslikava nazive regija neke države u nazive vrsta stabala koje u njima rastu.
  Naći jedinstvene nazive vrsta stabala koje rastu u obuhvaćenim regijama.

```
/*
var r1 = List("Omorika", "Breza", "Javor", "Hrast")

var r2 = List("Jasen", "Bukva", "Kesten", "Bor")

var r3 = List("Lipa", "Omorika", "Topola", "Jablan", "Bukva")
*/

var vrste = Map("R1" -> r1, "R2" -> r2, "R3" -> r3)
```

Zadatak 3

- Zadata je kolekcija lista
- Potrebno je:
  - napraviti jedinstvenu listu (kolekciju)
  - izbaciti duplikate

```
vrste.flatten
```

No implicit view available from (String, List[String])
⇒ scala.collection.GenTraversableOnce[B].

```
vrste.flatten( (x) => x._2.iterator )
```

```
//> res8: scala.collection.immutable.Iterable[String] =
//  List(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten, Bor,
//       Lipa, Omorika, Topola, Jablan, Bukva)
```

Zadatak 3

```
vrste.flatten( (x) => x._2.iterator ).toList.distinct
//> res8: List[String] =
//  List(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten
//       Bor, Lipa, Topola, Jablan)

vrste.values.flatten.toList.distinct
//> res9: List[String] =
// List(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten
//      Bor, Lipa, Topola, Jablan)

vrste.values.flatten.toStream.distinct.force
//> res10: scala.collection.immutable.Stream[String] =
//  Stream(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten, Bor,
//         Lipa, Topola, Jablan)


vrste.values.toStream.flatten.distinct.force
//> res11: scala.collection.immutable.Stream[String] =
//  Stream(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten, Bor,
//         Lipa, Topola, Jablan)
```

Zadatak 3

```
vrste.values.flatten.toStream.distinct.force
//> res10: scala.collection.immutable.Stream[String] =
//  Stream(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten, Bor,
//         Lipa, Topola, Jablan)


vrste.values.toStream.flatten.distinct.force
//> res11: scala.collection.immutable.Stream[String] =
//  Stream(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten, Bor,
//         Lipa, Topola, Jablan)


vrste.values.view.flatten.toList.distinct
//> res12: List[String] =
//  List(Omorika, Breza, Javor, Hrast, Jasen, Bukva, Kesten,
//       Bor, Lipa, Topola, Jablan)
```

Zadatak 4

- Data je lista reči koja sadrži cele brojeve zapisane kao tekst i u decimalnoj cifarskoj reprezentaciji. Naći sumu brojeva zapisanih u decimalnoj cifarskoj reprezentaciji.

```
val brojevi = List("1", "2", "tri", "4", "sto sedamdeset pet")
//> brojevi  : List[String] = List(1, 2, tri, 4, sto sedamdeset pet)

def strToInt(s: String) : Option[Int] = {
  try { Some(Integer.parseInt(s.trim)) }
  catch { case e: NumberFormatException => None }
}
//> strToInt: (s: String)Option[Int]
```

Zadatak 4

```
brojevi.map(strToInt)
//> res0: List[Option[Int]] =
//        List(Some(1), Some(2), None, Some(4), None)

for(e <- brojevi.map(strToInt) if e != None ) yield e.get
//> res1: List[Int] = List(1, 2, 4)

brojevi.map(strToInt).flatten.sum          //> res2: Int = 7

brojevi.flatMap(strToInt).sum              //> res3: Int = 7
```

Zadatak 5

- Implementirati algoritam sortiranja metodom brojanja upotrebom funkcija višeg reda nad kolekcijama.

```
def countingSort(l : List[Int]) : List[Int] = {
  var maxVal = l.reduceLeft( _ max _ )
  var C = Array.fill(maxVal){0}

  for(e <- l) C(e-1) += 1
  C = C.scanLeft(0)(_ + _)

  var B = Array.ofDim[Int](l.length)

  for( e <- l.reverseIterator) {
    B(C(e)-1) = e
    C(e) -= 1
  }

  B.toList
}              //> countingSort: (l: List[Int])List[Int]
```

Zadatak 5

```
val niz = List(5, 12, 2, 8, 4, 16, 32, 12, 44, 5, 8, 6, 7, 5)
//> niz  : List[Int] =
//         List(5, 12, 2, 8, 4, 16, 32, 12, 44, 5, 8, 6, 7, 5)

countingSort(niz)
//| res0: List[Int] =
//        List(2, 4, 5, 5, 5, 6, 7, 8, 8, 12, 12, 16, 32, 44)
```

Funkcionalno programiranje

Vežbe

07 Akteri

Upotreba AKKA biblioteke

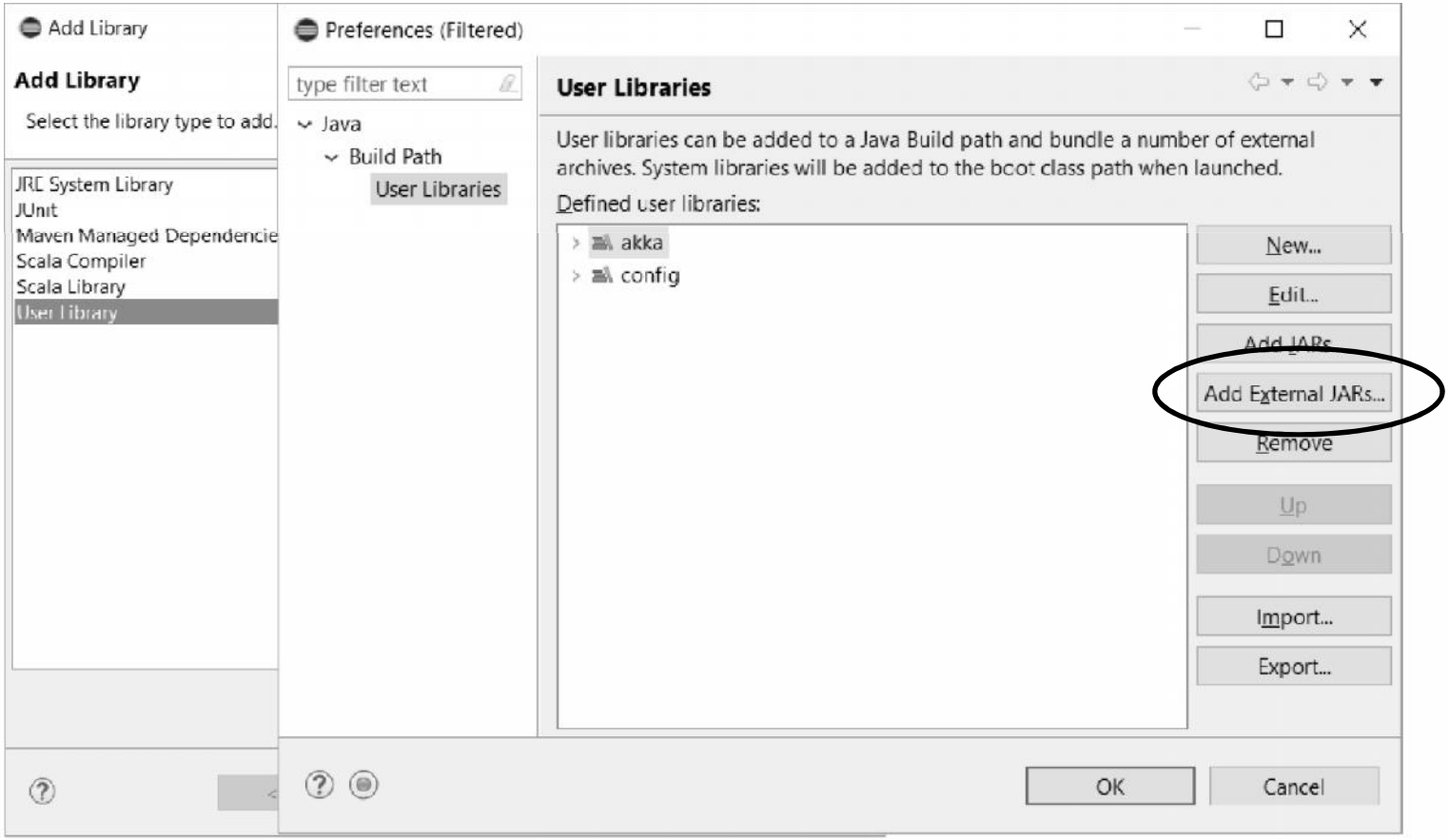
- http://akka.io
- http://akka.io/downloads/
- Aktuelna verzija 2.5.1

- Jedan od načina: upotreba SBT
  - Scala Build Tool
  - Alat za prevođenje Scala programa
  - Automatizacija prevođenja složenih projekata

- Alternativno: dostaviti .jar fajlove
  - preuzeti arhivu akka distribucije sa zvaničnog sajta
  - u konfiguraciji IDE postaviti odgovarajuće JAR fajlove kao biblioteke

Konfigurisanje

- Eclipse
  - Build path -> Add Library



Zadatak 1

- Napistati program poput višenitnog "ping-pong" programa u jeziku Java.

```
import akka.actor._

case object PingMessage
case object PongMessage
case object StartMessage
case object StopMessage
```

Zadatak 1

```
class Ping(pong: ActorRef) extends Actor {
  var count = 0
  def incrementAndPrint { count += 1; println("ping") }
  def receive = {
    case StartMessage =>
      incrementAndPrint
      pong ! PingMessage
    case PongMessage =>
      incrementAndPrint
      if (count > 99) {
        sender ! StopMessage
        println("ping zaustavljen")
        context.stop(self)
      } else {
        sender ! PingMessage
      }
    case _ => println("Primljena nepoznata poruka.")
  }
}
```

Zadatak 1

```
class Pong extends Actor {
  def receive = {
    case PingMessage =>
      println(" pong")
      sender ! PongMessage
    case StopMessage =>
      println("pong zaustavljen")
      context.stop(self)
    case _ => println("Primljena nepoznata poruka.")
  }
}
object PingPongTest extends App {
  val system = ActorSystem("PingPongSystem")
  val pong = system.actorOf(Props[Pong], name = "pong")
  val ping = system.actorOf(Props(new Ping(pong)), name = "ping")
  // start the action
  ping ! StartMessage
  //system.shutdown
}
```

Zadatak 2

- Napisati program na programskom jeziku Scala koji provodi aktera kroz njegov životni ciklus.

```
import akka.actor._

class Kenny extends Actor {
  println("entered the Kenny constructor")

  override def preStart { println("kenny: preStart") }

  override def postStop { println("kenny: postStop") }

  override def preRestart(reason: Throwable, message: Option[Any]) {
    println("kenny: preRestart")
    println(s" MESSAGE: ${message.getOrElse("")}")
    println(s" REASON: ${reason.getMessage}")
    super.preRestart(reason, message)
  }
```

Zadatak 2

```
  override def postRestart(reason: Throwable) {
    println("kenny: postRestart")
    println(s" REASON: ${reason.getMessage}")
    super.postRestart(reason)
  }

  def receive = {
    case ForceRestart => throw new Exception("Boom!")
    case _ => println("Kenny received a message")
  }
}
```

Zadatak 2

```
case object ForceRestart

object LifecycleDemo extends App {
  val system = ActorSystem("LifecycleDemo")
  val kenny = system.actorOf(Props[Kenny], name = "Kenny")

  println("===> sending kenny a simple String message")
  kenny ! "hello"
  Thread.sleep(1000)

  println("===> make kenny restart")
  kenny ! ForceRestart
  Thread.sleep(1000)

  println("===> killing kenny")
  system.stop(kenny)

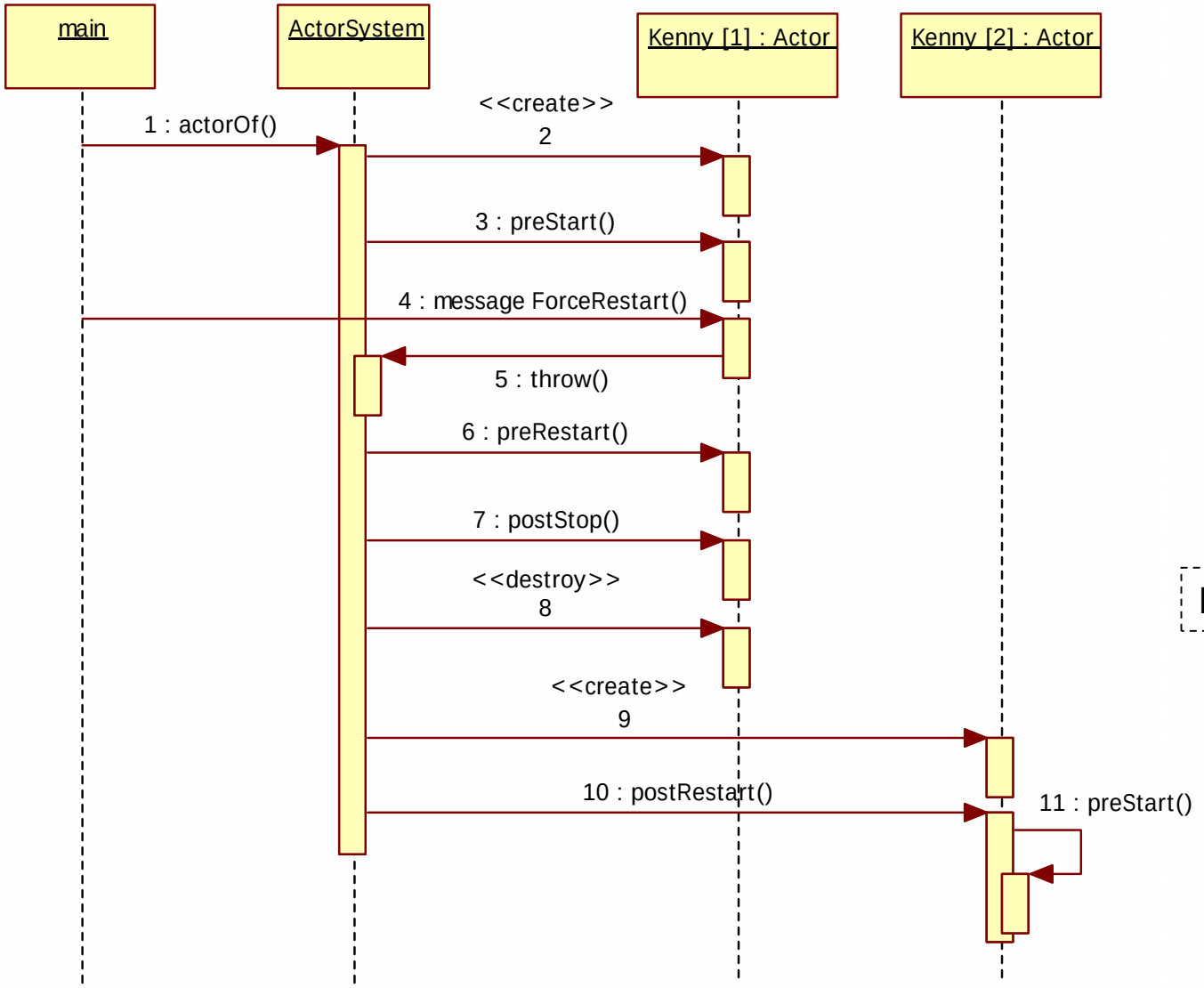
  println("shutting down system")
  system.shutdown
}
```

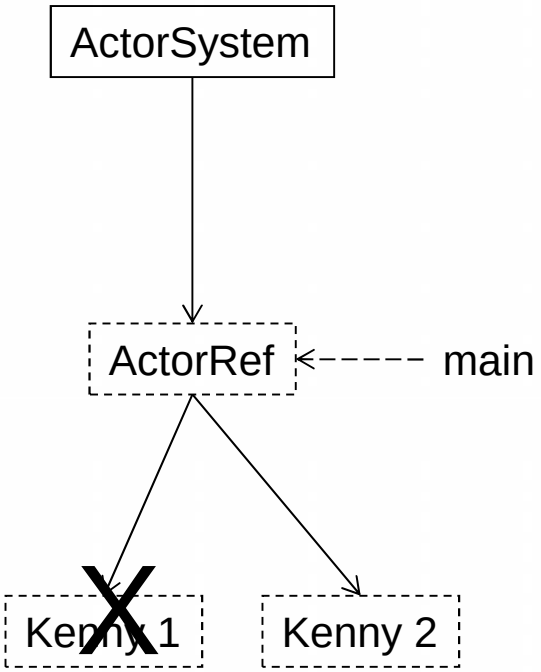
Zadatak 2

```
entered the Kenny 1 constructor
Kenny 1: preStart
===> sending kenny a simple String message
Kenny 1 received a message
==> make kenny restart
Kenny 1 received ForceRestart message
Kenny 1: preRestart
 MESSAGE: ForceRestart
 REASON: Boom!
Kenny 1: postStop
entered the Kenny 2 constructor
Kenny 2: postRestart
 REASON: Boom!
Kenny 2: preStart
[ERROR] [05/17/2017 21:38:54.420] [LifecycleDemo-akka.actor.default-dispatcher-4]
    [akka://LifecycleDemo/user/Kenny] Boom!
java.lang.Exception: Boom!
    ...

===> killing kenny
===> shutting down system
Kenny 2: postStop
```

Zadatak 2


main
ActorSystem
Kenny [1] : Actor
Kenny [2] : Actor
1 : actorOf()
<<create>>
2
3 : preStart()
4 : message ForceRestart()
5 : throw()
6 : preRestart()
7 : postStop()
<<destroy>>
8
<<create>>
9
10 : postRestart()
11 : preStart()


ActorSystem
ActorRef
main
Kenny 1
Kenny 2

Zadatak 3

- Napisati Scala akter koji na poruku `CreateChild(ime)` napravi novi akter kao svog potomka. Akteri-potomci, prilikom zaustavljanja, ispišu svoje ime i putanju.

Zadatak 3

```
import akka.actor._
case class CreateChild (name: String)
case class Name (name: String)

class Child extends Actor {
  var name = "No name"
  override def postStop {
    println(s"Aktor ($name) zaustavljen: ${self.path}")
  }

  def receive = {
    case Name(name) => this.name = name
    case _ => println(s"Potomak $name je primio poruku")
  }
}
```

Zadatak 3

```
class Parent extends Actor {
  def receive = {
    case CreateChild(name) =>
      val child = context.actorOf(Props[Child], name = s"$name")
      child ! Name(name)
    case _ => println("Roditelj je primio poruku.")
  }
}
```

Zadatak 3

```
object ParentTest extends App {
  val actorSystem = ActorSystem("ParentTest")
  val parent = actorSystem.actorOf(Props[Parent], name = "Parent")

  parent ! CreateChild("T-800")
  parent ! CreateChild("T-1000")

  Thread.sleep(500)

  println("Saljem otrovnu pililu (PoisonPill) ka T-800...")
  val t800 = actorSystem.actorSelection("/user/Parent/T-800")

  t800 ! PoisonPill

  println("T-800 neutralizovan")
  Thread.sleep(1000)
  actorSystem.shutdown
}
```

Zadatak 4

- Napisati Scala akter kome se ponašanje prosleđuje kao poruka.

```
import akka.actor._

case class Become(r : Actor.Receive)
case class Message(s : String)

class M extends Actor {

  def receive = {
    case Become(x) => context.become(x, false)
    case Message(s) => println(s)
    case _ => println("nepoznato")
  }

}
```

Zadatak 4

```
object Mutator extends App {
  val system = ActorSystem("system")
  val actor = system.actorOf(Props[M], name = "M")

  actor ! Message("Poruka")

  def laz : Actor.Receive =
  {
    case Message(s) => println("Nisam dobio poruku: " + s)
    case _ => println("Nisam nista primio")
  }

  actor ! Become( laz )
  actor ! Message("Zdravo")

  Thread.sleep(1000)
  system.shutdown()
}
```

Zadatak 5

- Demonstrirati obradu podataka pomoću budućih vrednosti
  - kada se izvršenje blokira čekajući buduću vrednost
  - kada završetak izračunavanja buduće vrednosti izazove događaj

Zadatak 5

```
// sa blokiranjem
import scala.concurrent.{Await, Future}
import scala.concurrent.duration._
import scala.concurrent.ExecutionContext.Implicits.global

object AwaitForFuture extends App {

  // Napravi buduću vrednost
  val f = Future {
    Thread.sleep(500)
    1 + 1
  }
  // Blokiraj izvršenje niti dok buduća vrednost ne završi
  // Baci izuzetak ako vreme istekne
  val result = Await.result(f, 1 second)

  println(result)
  Thread.sleep(1000)
}
```

Zadatak 5

```
// bez blokiranja
import scala.concurrent.{Future}
import scala.concurrent.ExecutionContext.Implicits.global
import scala.util.{Failure, Success}
import scala.util.Random
object FutureCallback extends App {
  println("počinjem ...")
  val f = Future { Thread.sleep(Random.nextInt(500)); 42 }
  println("Pre registrovanja povratnog poziva")
  f.onComplete {
    case Success(value) => println(s"Uspesno: $value")
    case Failure(e) => e.printStackTrace
  }
  println("A ..."); Thread.sleep(100)
  println("B ..."); Thread.sleep(100)
  println("C ..."); Thread.sleep(100)
  println("D ..."); Thread.sleep(100)
  println("E ..."); Thread.sleep(100)
  println("F ..."); Thread.sleep(100)
  Thread.sleep(2000)
}
```

```
počinjem  ...
Pre registrovanja povratnog poziva
A ...
B ...
C ...
Uspesno: 42
D ...
E ...
F ...
```

Zadatak 5

```
// bez blokiranja, resenje sa dva dogadjaja
import ...
object FutureTwoCallbacks extends App {
  val f = Future {
    Thread.sleep(Random.nextInt(500))
    if (Random.nextInt(500) > 250)
       throw new Exception("Izuzetak!") else 42
  }
  f onSuccess { case result => println(s"Uspesno: $result") }
  f onFailure { case t => println(s"Izuzetak: ${t.getMessage}") }
  // do the rest of your work
  println("A ..."); Thread.sleep(100)
  println("B ..."); Thread.sleep(100)
  println("C ..."); Thread.sleep(100)
  println("D ..."); Thread.sleep(100)
  println("E ..."); Thread.sleep(100)
  println("F ..."); Thread.sleep(100)
}
```

Zadatak 6

- Prikazati slučaj komunikacije dva aktera kod kojih inicijator poruke (pitanja) čeka na odgovor

```
import akka.actor._
import akka.pattern.ask
import akka.util.Timeout
import scala.concurrent.{Await, ExecutionContext, Future}
import scala.concurrent.duration._

case object AskNameMessage

class TestActor extends Actor {
  def receive = {
    case AskNameMessage => sender ! "Kenny"
    case _ => println("??")
  }
}
```

Zadatak 6

```
object AskTest extends App {

  // Napravi aktora
  val system = ActorSystem("AskTestSystem")
  val myActor = system.actorOf(Props[TestActor], name = "myActor")

  // Pitaj i čekaj odgovor
  implicit val timeout = Timeout(5 seconds)
  val future = myActor ? AskNameMessage
  val result = Await.result(future, timeout.duration)
                    .asInstanceOf[String]
  println(result)

  // Pitaj i odmah konvertuj u string
  val future2: Future[String] =
                       ask(myActor, AskNameMessage).mapTo[String]
  val result2 = Await.result(future2, 1 second)
  println(result2)

  system.shutdown
}
```