

# Funkcionalno programiranje

Vežbe  
02 Klase

## Zadatak 2.1

- Napisati hijerarhiju klasa za realizaciju ulanane 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 – elija 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 po etka 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 po etka 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 logi kog 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:
  - u `__init__` primarni konstruktor privatnim
  - napraviti `__new__` i 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 stati kog Huffman-ovog koda od zadanog niza karaktera (string).

## Zadatak 2.6 - Analiza

- Problem koji rešava Huffman-ov algoritam se najjednostavnije može modelirati binarnim stablom
- Napravi `emo` stablo po uzoru na listu
  - koristi `emo` rekurziju
  - potrebno je formirati
    - apstraktni tip za `vor` stabla
    - konkretan tip za unutrašnji `vor` stabla
    - konkretan tip za listove stabla
- Algoritam se tako `e` oslanja na prioritetni red
  - koristi `emo` gotovu klasu `scala.collection.mutable.PriorityQueue`

## Zadatak 2.6

- Najpre treba prebrojati simbole u poruci
  - zbog nepromenljivosti objekata, formira emo listu simbola poruke
  - sortira emo listu po alfabetskom poretku
  - formira emo listu parova (simbol, broj)
  - umetnu emo elemente liste u prioritetni red
  - primeni emo 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

```
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()
```

Huffman.sc

## 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
```

```
}
```



Huffman.sc