Zbirke (kolekcije) in generiki

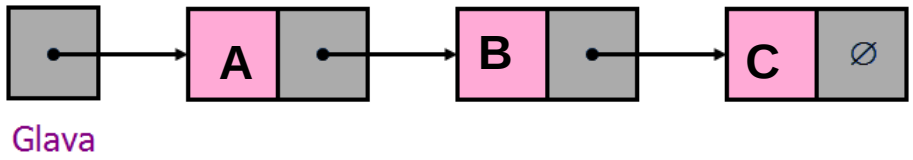
Podatkovne strukture v Javi

Kdaj in kako uporabiti katero podatkovno strukturo?

Podatkovne strukture

1. Podatkovna struktura je organizacija podatkov v pomnilniku računalnika.

2. Pod tem razumemo sezname, sklad, binarna drevesa, Razpršene (hash) tabele itd.



3. Algoritmi obdelujejo podatke v teh strukturah na različne načine, na primer z iskanjem in sortiranjem.

Prednosti in slabosti podatkovnih struktur

Tip podatkovne strukture	Prednosti	Slabosti
Array	Hitro vstavljanje, zelo hiter dostop, če poznamo indeks.	Počasno vstavljanje, počasno brisanje, fiksna dolžina.
Ordered array	Hitrejše iskanje kot pri neurejenem polju	Počasno vstavljanje in brisanje, fiksna dolžina
Stack	Zadnji noter, prvi ven	Počasen dostop do drugih elementov
Queue	Prvi noter, prvi ven (FIFO).	Počasen dostop do drugih elementov
Linked list	Hitro vstavljanje in brisanje	Počasno iskanje
Array List	Naključen dostop	Počasno vstavljanje in brisanje

Razred Arrays

- Razred Arrays
  - Nudi statične metode za rokovanje s polji
  - Nudi “visoko nivojske” metode
    - Metoda binarySearch za iskanje v urejenih poljih
    - Metoda equals za primerjavo polj
    - Metoda fill za vstavljanje vrednosti v polja
    - Metoda sort za urejanje polj

Primer uporabe Arrays

```
import java.util.*;

public class UsingArrays {
        private int intValues[] = { 1, 2, 3, 4, 5, 6 };
        private double doubleValues[] = { 8.4, 9.3, 0.2, 7.9, 3.4 };
        private int filledInt[], intValuesCopy[];

      // iniciacija polja
      public UsingArrays()  {
          filledInt = new int[ 10 ];
          intValuesCopy = new int[ intValues.length ];

          Arrays.fill( filledInt, 7 );   // napolnimo s sedmicami

          Arrays.sort( doubleValues );   // sort doubleValues ascending

        // kopiramo polje  intValues v polje intValuesCopy
        System.arraycopy( intValues, 0, intValuesCopy,
          0, intValues.length );
      }
```

Polje napolnimo s statično metodo fill razreda Arrays

Polje uredimo po naraščajočih vrednostih s statično metodo sort razreda Arrays

S statično metodo arraycopy razreda System kopiramo polje intValues v polje intValuesCopy

Primer uporabe Arrays (nadaljevanje 1)

```
// izpis vrednosti vsakega polja
public void printArrays() {
   System.out.print( "doubleValues: " );
   for ( int count = 0; count < doubleValues.length; count++ )
      System.out.print( doubleValues[ count ] + " " );

   System.out.print( "\nintValues: " );
   for ( int count = 0; count < intValues.length; count++ )
       System.out.print( intValues[ count ] + " " );

   System.out.print( "\nfilledInt: " );
   for ( int count = 0; count < filledInt.length; count++ )
      System.out.print( filledInt[ count ] + " " );

   System.out.print( "\nintValuesCopy: " );
   for ( int count = 0; count < intValuesCopy.length; count++ )
       System.out.print( intValuesCopy[ count ] + " " );

   System.out.println();
} // end method printArrays
```

Primer uporabe Arrays (nadaljevanje 2)

```
// iskanje vrednosti v polju intValues
public int searchForInt( int value )  {
   return Arrays.binarySearch( intValues, value );
}

// primerjava vsebin polj
public void printEquality()       {
    boolean b = Arrays.equals( intValues, intValuesCopy );

   System.out.println( "intValues " + ( b ? "==" : "!=" ) +
      " intValuesCopy" );

   b = Arrays.equals( intValues, filledInt );

   System.out.println( "intValues " + ( b ? "==" : "!=" ) +
      " filledInt" );
}
```

S statično metodo binarySearch iz razreda Arrays izvedemo binarno iskanje v polju

S statično metodo equals iz razreda Arrays ugotovimo, ali so vrednosti dveh polj ekvivalentne

Primer uporabe Arrays (nadaljevanje 3)

```
   public static void main( String args[] )      {
      UsingArrays usingArrays = new UsingArrays();  // razred UsingArrays smo definirali malo nazaj

      usingArrays.printArrays();
      usingArrays.printEquality();

      int location = usingArrays.searchForInt( 5 );
      System.out.println( ( location >= 0 ? “Najdena 5 v elementu " +
         location : "5 ni najdena" ) + " v intValues" );

      location = usingArrays.searchForInt( 8763 );
      System.out.println( ( location >= 0 ? “Najdeno 8763 v elementu " +
        location : "8763 ni najdeno" ) + " v  intValues" );
   }
} // end class UsingArrays
```

```
doubleValues: 0.2 3.4 7.9 8.4 9.3
intValues: 1 2 3 4 5 6
filledInt: 7 7 7 7 7 7 7 7 7 7
intValuesCopy: 1 2 3 4 5 6
intValues == intValuesCopy
intValues != filledInt
Najdena 5 v elementu 4 v intValues
8763 ni najdeno v intValues
```

Demo

Primer: uporaba asList

```
import java.util.*;
public class UsingAsList {
        private static final String values[] = { "red", "white", "blue" };
        private List list;
        // initialize List and set value at location 1
        public UsingAsList()  {
            list = Arrays.asList( values );   // get List
            list.set( 1, "green" );          // change a value
        }

        // output List and array
        public void printElements() {
          System.out.print( "List elements : " );

          for ( int count = 0; count < list.size(); count++ )
            System.out.print( list.get( count ) + " " );

          System.out.print( "\nArray elements: " );
```

Metoda asList razreda Arrays vrne pogled List *na vrednosti v polju*

Metoda set razreda List spremeni vsebino elementa 1 na "green"

Metoda size razreda List vrne število elementov v List

Metoda get razreda List vrne posamezni element v List

Primer: uporaba asList (nadaljevanje)

```
    for ( int count = 0; count < values.length; count++ )
        System.out.print( values[ count ] + " " );
    System.out.println();
  }

  public static void main( String args[] ) {
    new UsingAsList().printElements();
  }

} // end class UsingAsList
```

Izpis

```
List elements : red green blue
Array elements: red green blue
```

Demo

Sklad

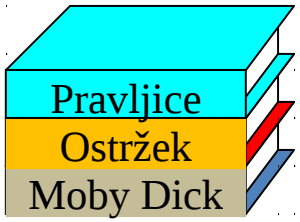
Stack je izpeljan iz razreda **Vector.** Poleg vseh njegovih metod ima dodani še **push()** in **pop()**.

Metoda **empty()** vrne **true,** če je sklad prazen.

Metoda **search()** pove, če je nek objekt že na skladu in kako globoko je

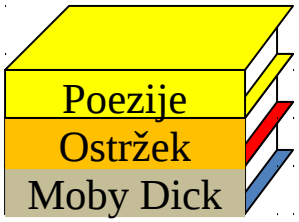
Sklad knjig

Začetna vsebina :

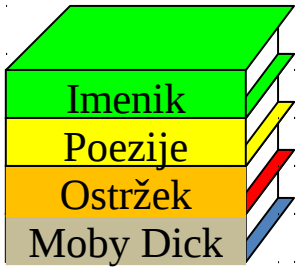


Odstranimo eno knjigo:

Dodamo "Poezije":



Dodamo "Imenik":



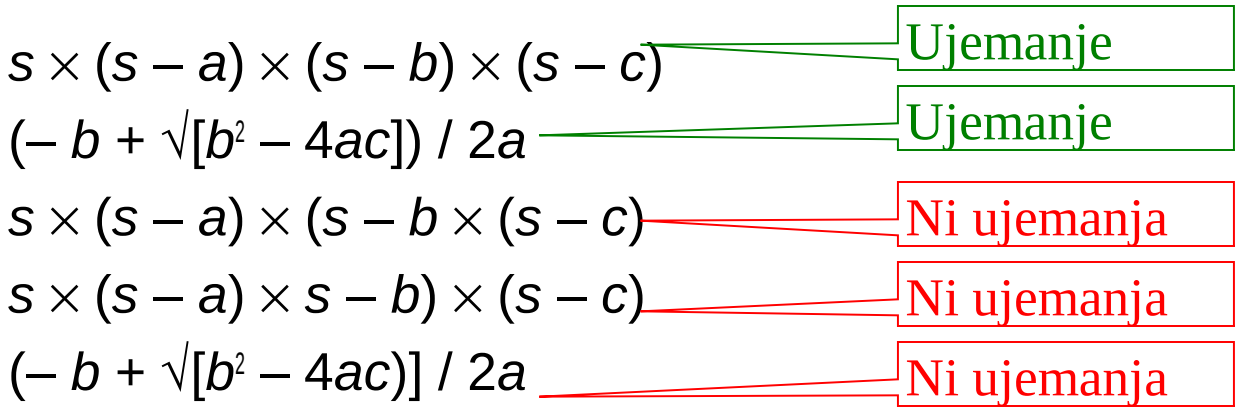
Primer s skladom

```
import java.util.*;

public class StackDemo{
   public static void main(String[] args) {
      Stack stack=new Stack();
      stack.push(new Integer(10));
      stack.push("a");
      System.out.println("Vsebina sklada je: " + stack);
      System.out.println("Velikost sklada je : " + stack.size());
      System.out.println("Iz sklada vzamemo: " + stack.pop());
      System.out.println("Iz sklada vzamemo: " + stack.pop());
      //System.out.println("Iz sklada vzamemo: " + stack.pop());
      System.out.println("Vsebina sklada je: " + stack);
      System.out.println("Velikost sklada je: " + stack.size());
   }
}
```

Demo

Primer: preverjanje oklepajev

- **Izraz ima pravilno število oklepajev, če:**
  - Za vsak levi oklepaj imamo kasneje ustrezen desni oklepaj
  - Za vsak desni oklepaj imamo prej ustrezni levi oklepaj
  - Ima vsaka podfraza med parom ujemajočih se oklepajev tudi sama pravilno število oklepajev (*matematični izrazi*):



Primer z "lastno napisanim" skladom Demo

Pregled zbirk

- Zbirka (kolekcija, Collection)
  - Podatkovna struktura (objekt), ki pomni reference na druge objekte
- Ogrodje kolekcij (Collections framework)
  - Vmesniki (interfaces) deklarirajo operacije za različne tipe kolekcij
  - V java.util najdemo
    - Collection
    - Set
    - List
    - Map

Java Collections Framework

Algoritmi:
Metode, ki na objektih implementirajo vmesnike zbirk, izvajajo uporabna rokovanja, kot na primer iskanje in sortiranje.

Algoritmi so polimorfni, ker lahko enako metodo uporabljamo na različnih implementacijah ustreznega vmesnika zbirke.

Imamo ponovno uporabljivo funkcionalnost.

Sortiranje
Mešanje (shuffling)
Rokovanje s podatki (obračanje vrstega reda, dodajanje)
Iskanje (na pr. Binarno iskanje)
Kompozicija (na pr. Frekvenca oziroma pogostnost)
iskanje ekstremnih vrednosti (na primer maksimuma)

Kaj je “Collections framework”?

- Ogrodje (framework) je množica vmesnikov, abstraktnih razredov in konkretnih razredov skupaj s podpornimi orodji.
- Ogrodje je podano v paketu java.util in ga sestavljajo trije deli:
    1. Osnovni vmesniki
    2. Množica implementacij.
    3. Uporabne metode

“**Java Collections Framework “**

•Ponuja vmesnike: abstraktne podatkovne tipe, ki predstavljajo zbirke.

•Omogoča rokovanje s zbirkami neodvisno od podrobnosti njihovih predstavitev.

•Ponuja implementacije: konkretne implementacije vmesnikov zbirk.

•Ponuja ponovno uporabljive podatkovne strukture

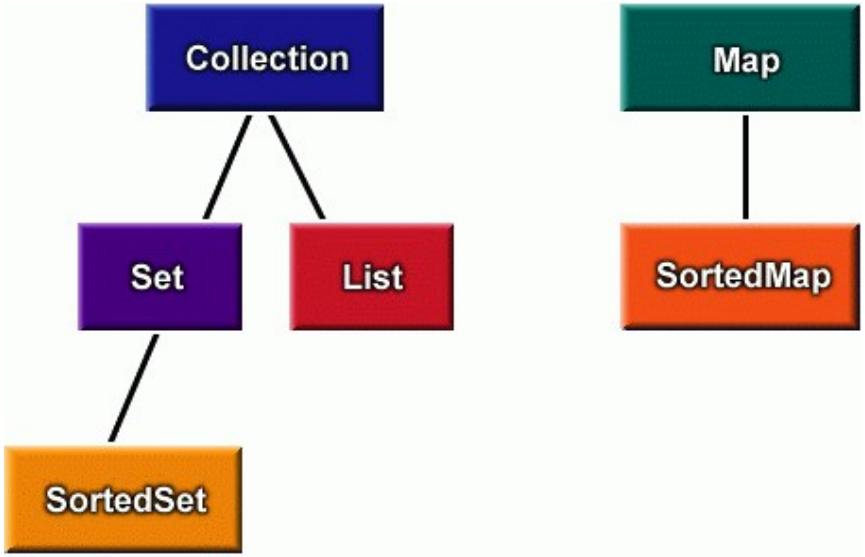
Le zakaj bi rabili tako ogrodje?

- Pred Javo SDK1.2, smo imeli povsem uporabne podatkovne strukture:
  - Hash Table
  - Vector
  - Stack

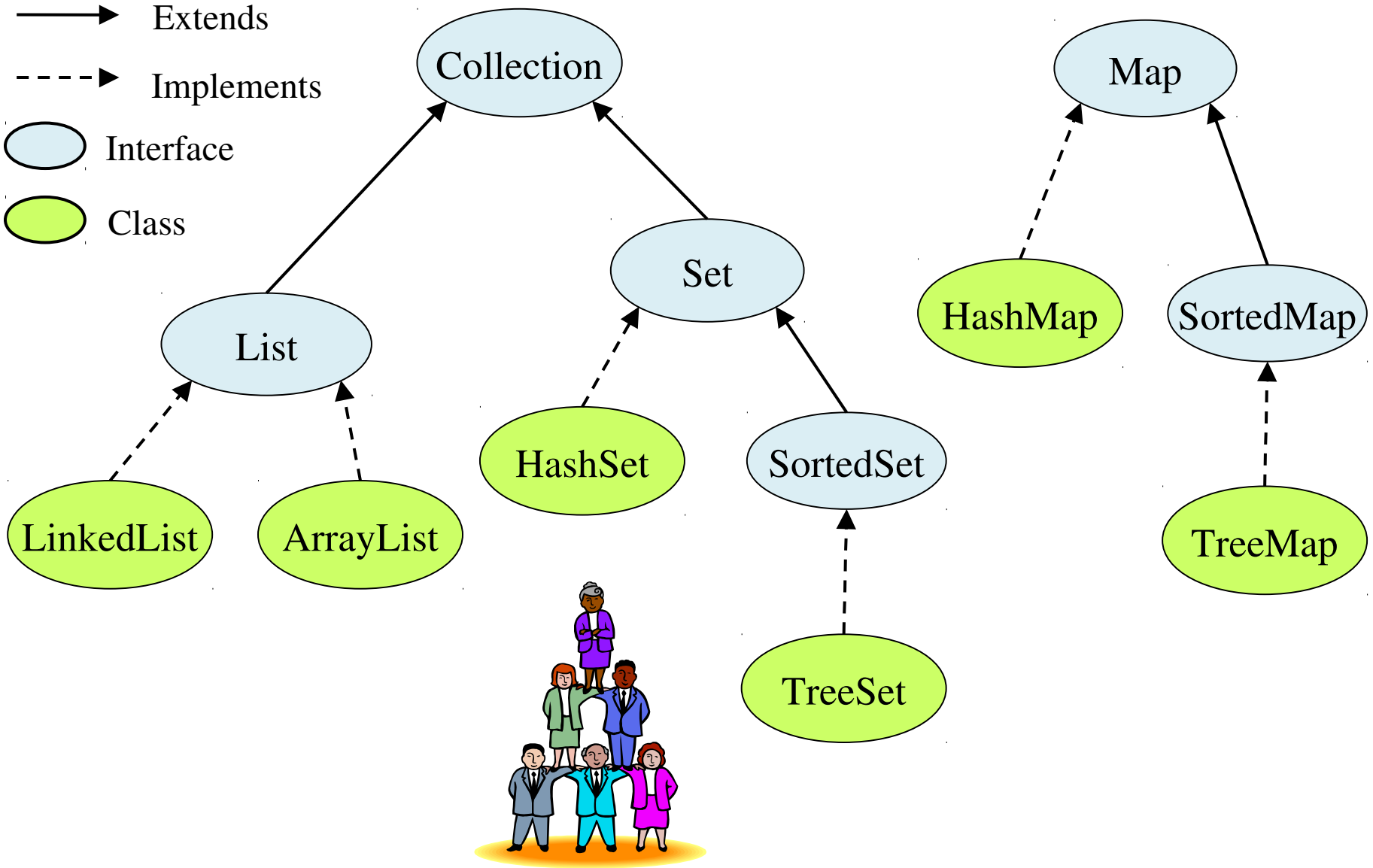
- Bile so dobre, njihova uporaba je bila enostavna. Vendar niso bile organizirane v bolj splošno ogrodje.
- Pomanjkljiva je bila tudi interoperabilnost.

Značilnosti kolekcijskega ogrodja

- Zmanjša napor pri programiranju.
- Poveča učinkovitost.
- Zagotavlja interoperabilnost med nevezanimi API-ji (Application Programming Interfaces).
- Hitrejša ponovna uporaba programov.



Vmesniki in razredi (implementacije)

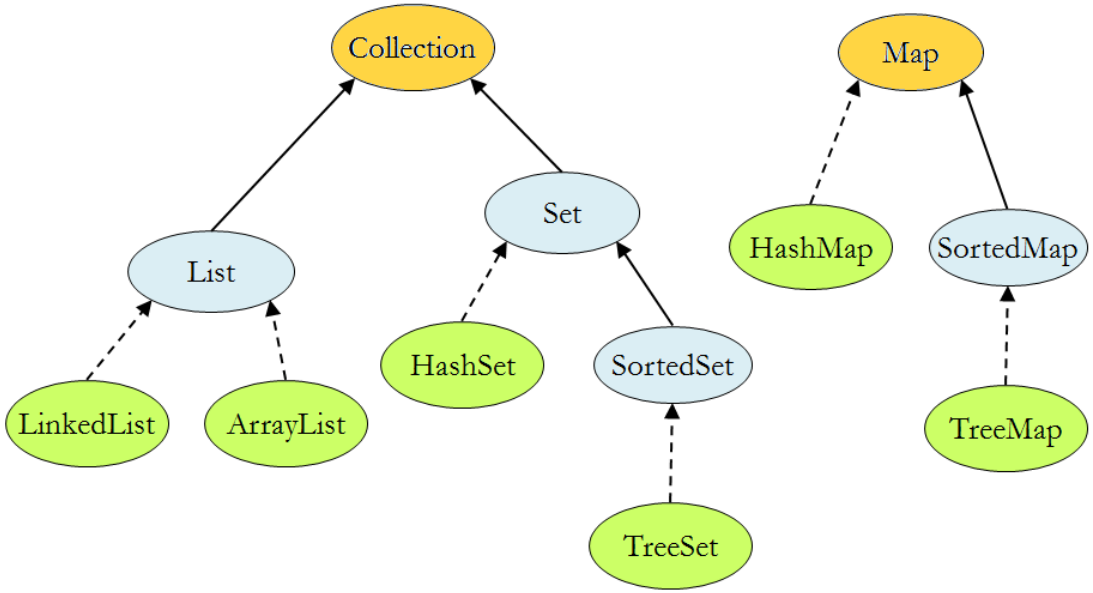


Razlika med kolekcijami in slovarji

Kolekcija (collection)
Lahko dodajamo, brišemo, gledamo izolirane postavke v kolekciji.

Slovar (map)
•Na voljo imamo operacije na kolekcijah, ki pa delujejo na parih ključ-vrednost namesto na izoliranih elementih.
•Tipična uporaba slovarjev je dostop do vrednosti, shranjenih s ključem.

Opomba:
V Javi so definirali razred Collection, ki se razlikuje od razreda Map, Oboje pa sta v bistvu kolekciji. Zaradi dvoumnosti uporabimo kot skupen pojem ***zbirka***

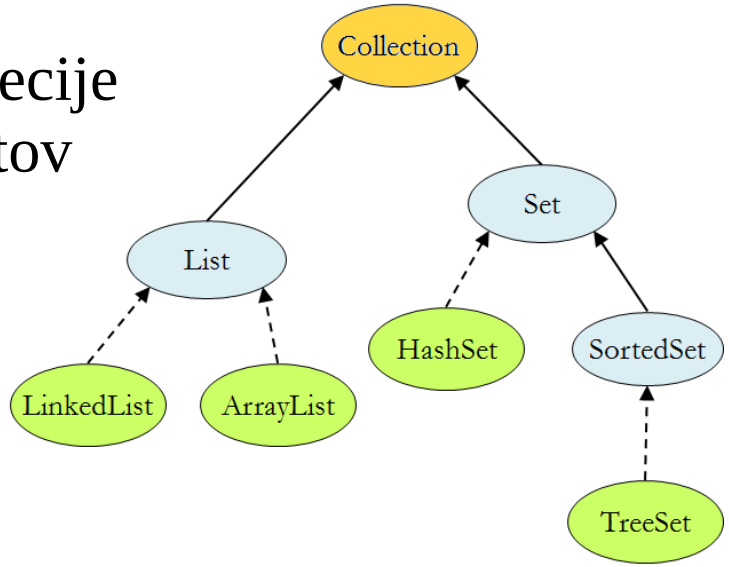


Zbirke so vmesniki

- Zbirka je pravzaprav vmesnik
- Vsaka vrsta zbirke ima eno ali več **implementacij**
- Tvorimo lahko nove vrste zbirk
- Ko implementiramo vmesnik, obljubimo, da bomo zagotovili zahtevane metode
- Nekatere metode zbirke so opcijske (neobvezne)

Metode kolekcij

- `add(o)` Dodajanje elementa
- `addAll(c)` Dodajanje kolekcije
- `clear()` Brisanje vseh elementov
- `contains(o)` Ali kolekcija vsebuje element.
- `containsAll(c)` Ali kolekcija vsebuje kolekcijo
- `isEmpty()` Ali je kolekcija prazna
- `iterator()` Vrne iterator
- `remove(o)` Odstrani podani element
- `removeAll(c)` Odstrani kolekcijo
- `retainAll(c)` Obdrži elemente kolecije
- `size()` Vrne število elementov



Metode kolekcij (bolj podrobno)

boolean	**add**(Object o)	Doda element v kolekcijo
boolean	**addAll**(Collection c)	Doda elemente podane kolekcije v to kolekcijo
void	**clear**()	Odstrani vse elemente iz te kolekcije
boolean	**contains**(Object o)	Vrne true, če ta kolekcija vsebuje podani element
boolean	**containsAll**(Collection c)	Vrne true, če ta kolekcija vsebuje vse elemente podane kolekcije
boolean	**equals**(Object o)	Primerja podani element s to kolekcijo
int	**hashCode**()	Vrne "hash" kodo te kolekcije
boolean	**isEmpty**()	Vrne true, če je ta kolekcija prazna, brez elementov
Iterator	**iterator**()	Vrne iterator nad elementi te kolekcije
boolean	**remove**(Object o)	Odstrani objekt iz te kolekcije, če tak element v njej obstaja.
boolean	**removeAll**(Collection c)	Odstrani vse elemente podane kolekcije, če ti obstajajo v tej.
boolean	**retainAll**(Collection c)	Ohrani v tej kolekciji le elemente podane kolekcije.
int	**size**()	Vrne število elementov v tej kolekciji.
Object[]	**toArray**()	Vrne polje, ki vsebuje vse elemente te kolekcije.
Object[]	**toArray**(Object[] a)	Vrne polje, ki vsebuje vse elemente te kolekcije. Tip vrnjenega polja je tip podanega polja.

Iteratorji

Iteratorji nudijo splošen način prehoda po vseh elementih v kolekciji

```
ArrayList<String> list = new ArrayList<String>();
list.add("1-prvi");
list.add("2-drugi");
list.add("3-tretji");
Iterator<String> itr = list.iterator();
while (itr.hasNext()) {
  System.out.println(itr.next().toLowerCase());
}
```

Izpis

```
1-prvi
2-drugi
3-tretji
```

DEMO

Metode iteratorjev

- hasNext() – test, če imamo še kaj elementov
- next() – vrne naslednji objekt in napreduje
- remove() – odstrani trenutno kazani objekt

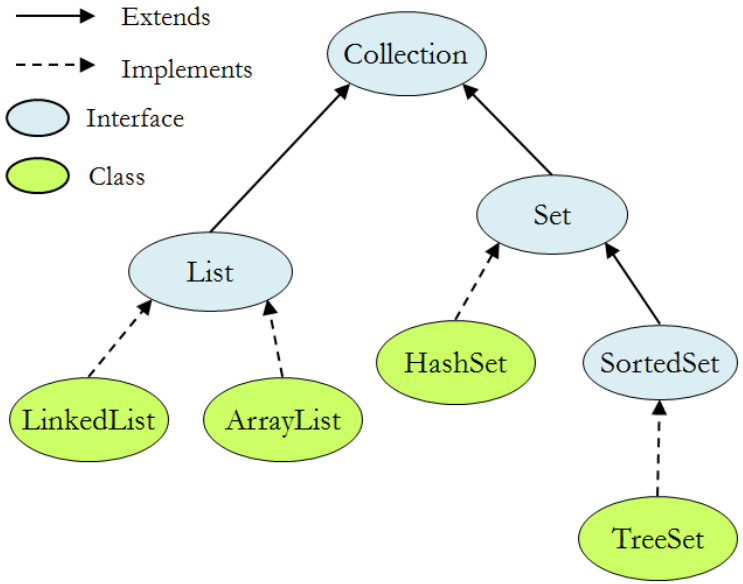
Še en primer uporabe iteratorja

```
import java.util.*;

  public class IteratorDemo {
     public static void main(String args[ ]) {

        List list = new ArrayList();
        for (int i = 1; i <= 10; i++)
          list.add(i + " * " + i + " = " + i * i);

        Iterator iter = list.iterator();
        while (iter.hasNext())
          System.out.println(iter.next());
    }
  }
```

Demo

Konkretne kolekcije

konkretna kolekcija	implements	Opis
`HashSet`	`Set`	hash table (zgoščena tabela)
`TreeSet`	`SortedSet`	balanced binary tree (uravnovešeno binarno drevo)
`ArrayList`	`List`	resizable-array (razširljivo polje)
`LinkedList`	`List`	linked list (povezan seznam)
`Vector`	`List`	resizable-array (razširljivo polje)
`HashMap`	`Map`	hash table (zgoščena tabela)
`TreeMap`	`SortedMap`	balanced binary tree (uravnovešeno binarno drevo)
`Hashtable`	`Map`	hash table (zgoščena tabela)

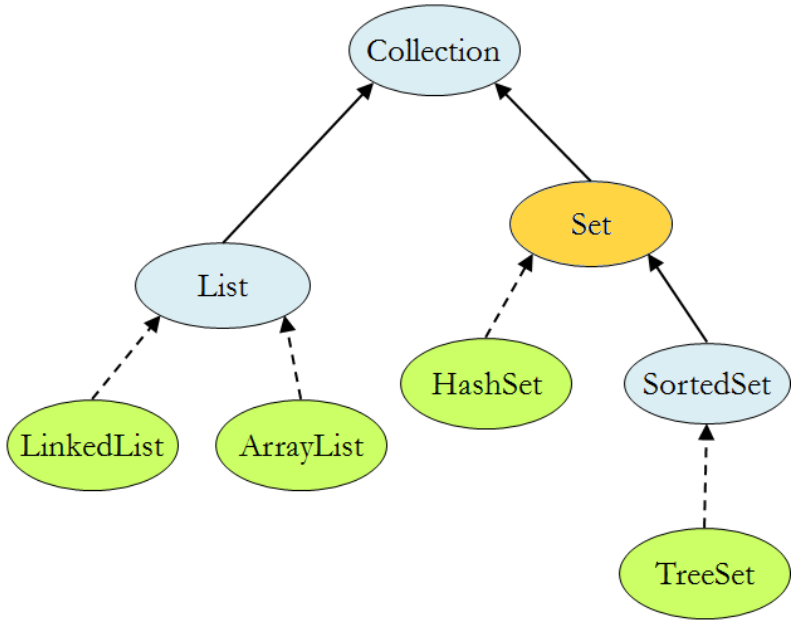


Množice (Sets)

Skupina unikatov, ki **nima duplikatov**

Nekaj primerov
- Množica velikih črk ‘A’ do ‘Z’
- Množica nenegativnih celih števil { 0, 1, 2, … }
- Prazna množica{ }

Osnovne lastnosti množic
- Ima le po en primerek vsake postavke
- Lahko je končna ali neskončna
- Lahko definira abstraktne pojme
- Lahko je urejena ali neurejena

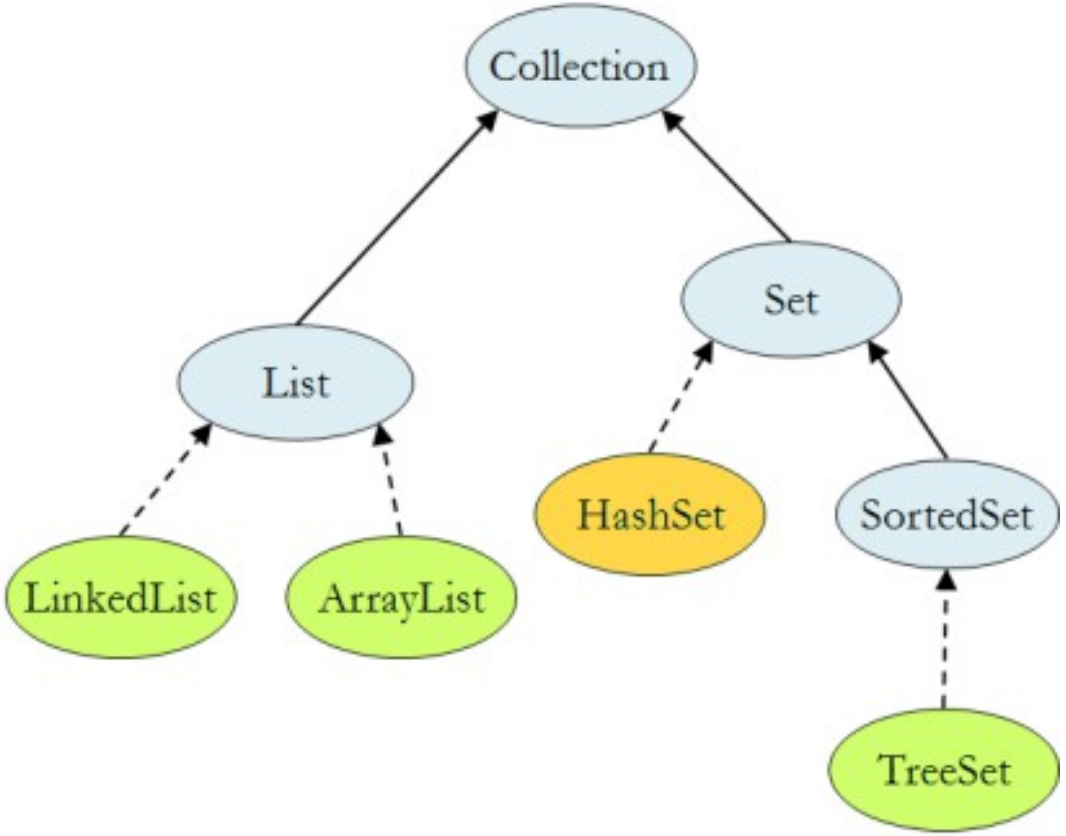


Kdaj sta dve množici enaki?

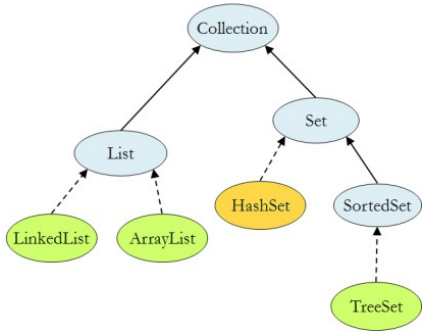
{1, 2, 3} == {1, 3, 4} ?
{A, a, c} == {A, a, c} ?

HashSet

HashSet predstavlja unikatno kolekcijo elementov, ki so urejeni v skladu z njihovo hash kodo.



Uporaba HashSet




```
import java.util.*;
public class HashSetDemo  {
    public static void main(String[] args) {

        // Napolnimo HashSet s tremi celimi stevili.
        HashSet hs = new HashSet();
        hs.add(new Integer(1));    hs.add(new Integer(2));
        hs.add(new Integer(2));    hs.add(new Integer(3));
        // Ugotovimo velikost HashSet.
        System.out.println("HashSet vsebuje " +  hs.size() + " elementov.");

        // Iteriramo po vsebini HashSet.
        Iterator iter = hs.iterator();
        while (iter.hasNext())  System.out.println(iter.next().toString());
        // Preverjamo obstoj elementa. Ce je, ga odstranimo.
        if (!hs.isEmpty()) {
            Integer toRemove = new Integer(2);
            if (hs.contains(toRemove)) {
                if (hs.remove(toRemove)) {
                    System.out.println("Odstranjen element " + toRemove.toString());
                }
 } } }}
```

Demo

Primer: štetje različnih besed

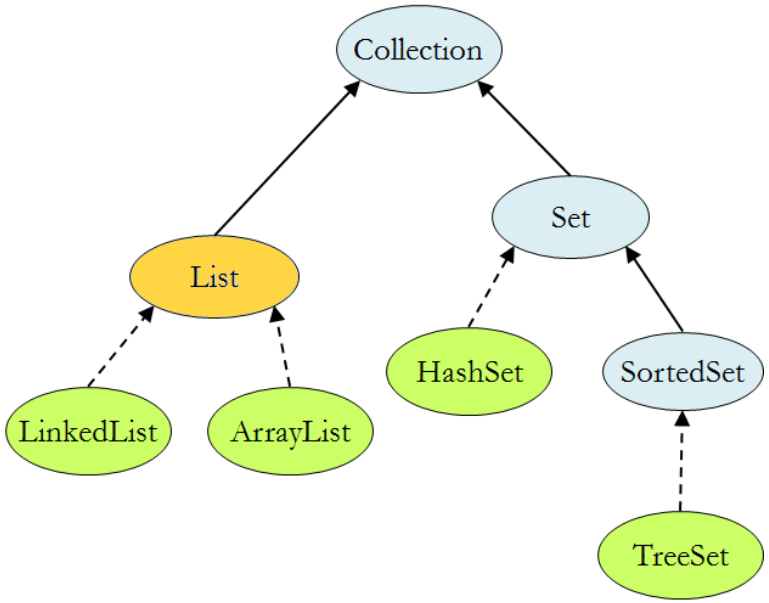
```
public class CountWords {
    static public void main(String[] args) {
        Set words = new HashSet();
        BufferedReader in =  new BufferedReader( new  InputStreamReader (System.in));
        String delim = " \t\n.,:;?!-/()[]\"\'";
        String line;
        int count = 0;
        try {
            while ((line = in.readLine()) != null) {
                StringTokenizer st =  new StringTokenizer(line, delim);
                while (st.hasMoreTokens()) {
                    count++;
                    words.add(  st.nextToken().toLowerCase());
                }
            }
         } catch (IOException e) {}
        System.out.println("Stevilo besed : "  + count);
        System.out.println("Stevilo razlicnih besed: " + words.size());
    }
}
```

Demo

List (seznam)

Urejena kolekcija elementov (pravimo ji tudi zaporedje):

- Lahko vsebuje duplikate.
- Do elementov dostopamo glede na njihov položaj v seznamu.
- Prvi element ima indeks nič.



metode List

- add(i,o) — vstavi o na položaj i
- *add(o)* — dodaj o na konec
- get(i) — vrni i-ti element
- remove(i) — odstrani i-ti element
- set(i,o) — zamenjaj i-ti element z o
- indexOf(o)
- lastIndexOf(o)
- listIterator()
- sublist(i,j)

Komentar:
o... pomeni objekt

List lahko pomni **le en tip** elementov

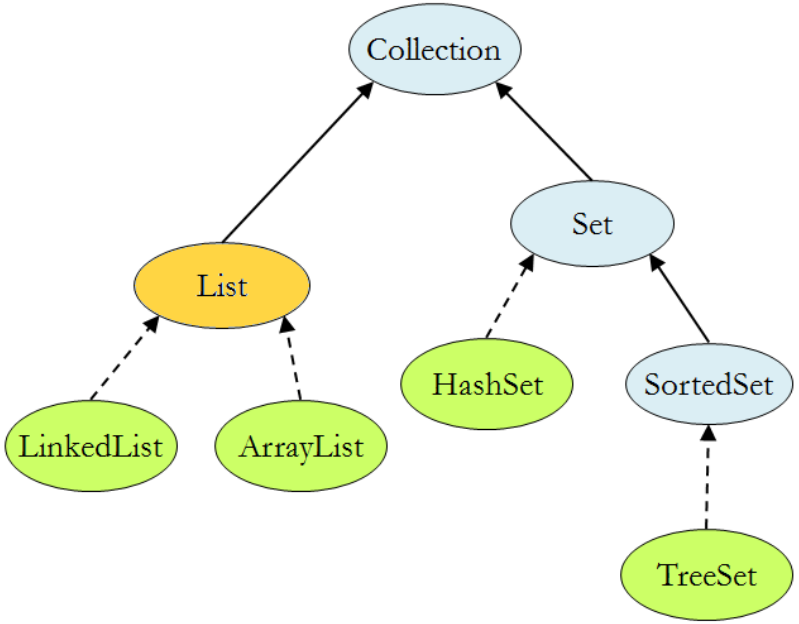
Tip elementa

```
List<BankAccount> accountList =
                new ArrayList<BankAccount>();

accountList.add(new BankAccount(“EUR", 111.11));
accountList.add(new BankAccount(“USD", 222.22));
accountList.add(new BankAccount(“CHF", 333.33));
accountList.add(new BankAccount(“HRK", 444.44));
System.out.println(accountList.toString());
```

[EUR 111.11, USD 222.22, CHF 333.33, HRK 444.44]

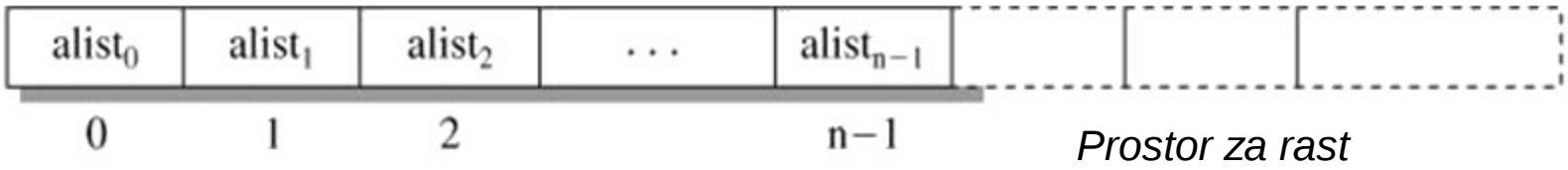
Konkretne implementacije List

- Imamo dve konkretni implementaciji vmesnika List
  - LinkedList
  - ArrayList
- Katero naj bi uporabili, je odvisno od naših potreb.

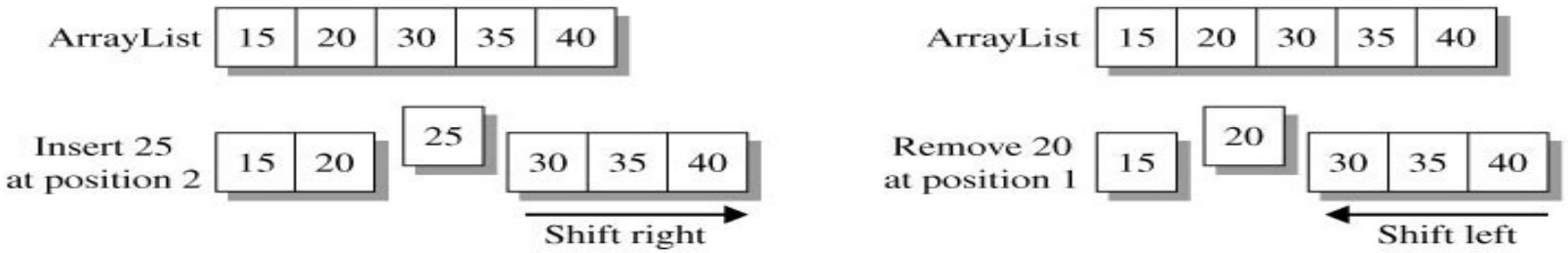


Array List

- Shranjuje elemente v celovit kos pomnilnika, ki ga lahko avtomatično podaljšujemo.



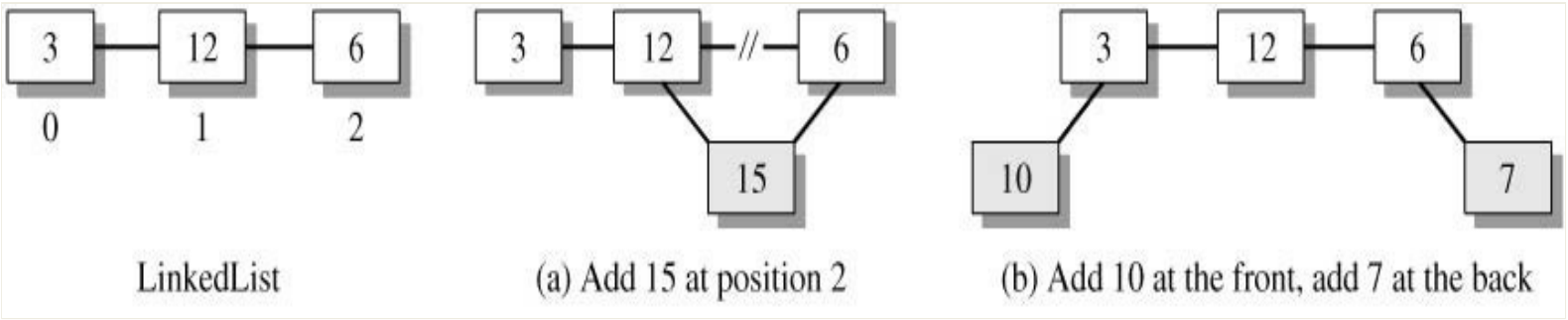
- Kolekcija učinkovito dodaja in briše elemente na koncu seznama.
- Operacije na vmesnih pozicijah so manj hitre.



Shifting blocks of elements to insert or remove an ArrayList item.

Link List

- Elementi imajo vrednost in povezave, ki identificirajo sosednje elemente v zaporedju.
- Vrivanje in brisanje elementov ni zelo hitro.



Primer z ArrayList

```
import java.awt.Color;
import java.util.*;

public class CollectionTest {
   private static final String colors[] = { "red", "white", "blue" };

  public CollectionTest()  {
     List list = new ArrayList();   // tvorimo ArrayList,
     // v seznam dodajamo objekte
     list.add( Color.MAGENTA );

     for ( int count = 0; count < colors.length; count++ )
        list.add( colors[ count ] );

     list.add( Color.CYAN );

     // izpis vsebine seznama
     System.out.println( "\nArrayList: " );
     for ( int count = 0; count < list.size(); count++ )
        System.out.print( list.get( count ) + " " );
```

Z metodo add dodajamo objekte v ArrayList

Metoda get vrača posamezni element v List

Primer z ArrayList (nadaljevanje 1)

```
   // odstranimo vse objekte String
   removeStrings( list );

   // izpis vsebine seznama
   System.out.println( "\n\nArrayList po klicu removeStrings: " );

   for ( int count = 0; count < list.size(); count++ )
      System.out.print( list.get( count ) + " " );

} // end constructor CollectionTest

// odstranimo objekte String iz kolekcije...
private void removeStrings( Collection collection ) {
   Iterator iterator = collection.iterator(); // get iterator

   // zanka, dokler je še kaksen element kolekcije
   while ( iterator.hasNext() )

      if ( iterator.next() instanceof String )
         iterator.remove();  // odstranimo objekt String
}
```

Metoda removeStrings ima kot argument kolekcijo

Dobimo iterator kolekcije

Metoda hasNext ugotovi, ali ima Iterator še več elementov

Metoda next vrne naslednji objekt iz Iterator

Z metodo remove iteratorja iterator odstranjujemo objekte String

Primer z ArrayList (nadaljevanje 2)

```
    public static void main( String args[] )   {
       new CollectionTest();
    }

} // end class CollectionTest
```

Izpis

```
ArrayList:
java.awt.Color[r=255,g=0,b=255] red white blue java.awt.Color
[r=0,g=255,b=255]

ArrayList after calling removeStrings:
java.awt.Color[r=255,g=0,b=255] java.awt.Color[r=0,g=255,b=255]
```

Primer z LinkedList

```
import java.util.*;
public class LinkedListExample{
    public static void main(String[] args) {
        LinkedList <Integer>list = new LinkedList<Integer>();
        int num1 = 11, num2 = 22, num3 = 33, num4 = 44;
        int size;
        Iterator iterator;
        //Adding data in the list
        list.add(num1); list.add(num2); list.add(num3); list.add(num4);
        size = list.size();
        System.out.print( "Linked list data: ");
        iterator = list.iterator(); //Create a iterator
        while (iterator.hasNext())  System.out.print(iterator.next()+" ");
```

Primer z LinkedList (nadaljevanje)

```
//Check list empty or not
        if (list.isEmpty()) System.out.println("Linked list is empty");
        else System.out.println( "Linked list size: " + size);
        list.addFirst(55);  //Adding data 55 at 1st location
        System.out.print("Now the list contain: ");
        iterator = list.iterator();
        while (iterator.hasNext())    System.out.print(iterator.next()+" ");
        System.out.println();
        list.clear();  // remove all
        if (list.isEmpty())    System.out.println("Linked list is empty");
    }
}
```

Demo

Primerjava Link list : Array List

Link List

Ni naključnega dostopa

Hitro rokovanje

Array List

Hiter naključni dostop

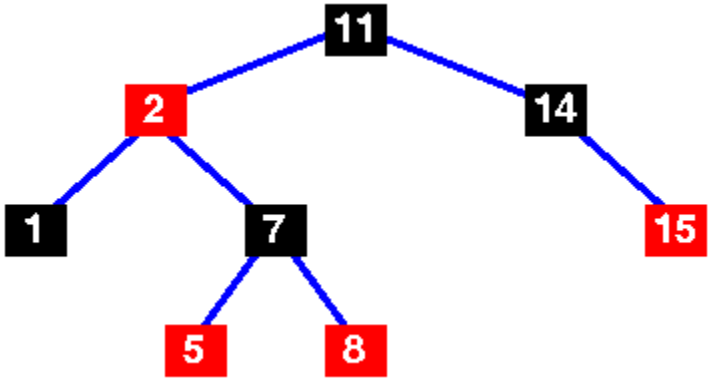
Počasno rokovanje

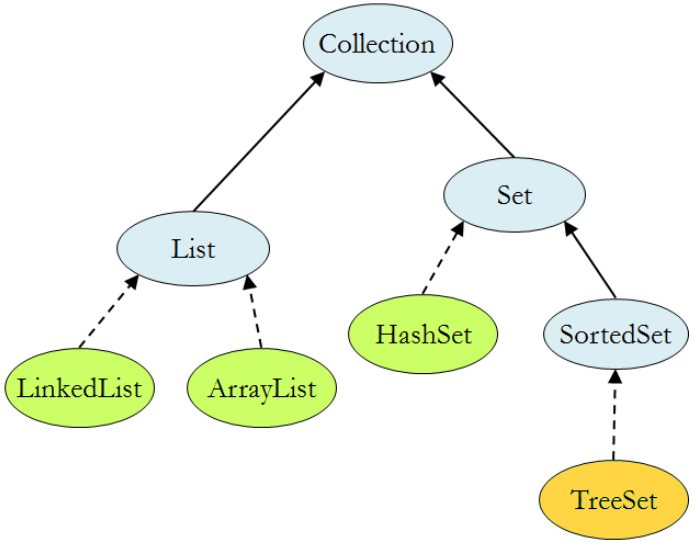
TreeSet

TreeSet implementira vmesnik Set in je podprt z instanco TreeMap:

•Zagotavlja, da bo urejena množica z elementi v naraščajočem vrstnem redu.

•TreeMap je rdeče-črno drevo, ki implementira vmesnik SortedMap.





Demo

Primer z drevesom TreeSet

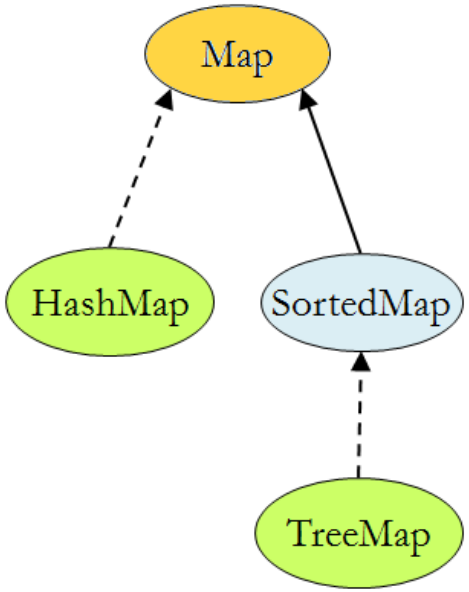
```
import java.util.*;
import java.util.Scanner;

public class TreeSetExample2{
    public static void main(String[] args) {

        Scanner src = new Scanner(System.in);
        TreeSet <Integer>tree = new TreeSet<Integer>();

        while (src.hasNext()) {
            if (src.hasNextInt())   tree.add(src.nextInt());
         }
        Iterator iterator;
        iterator = tree.iterator();
        System.out.print("Podatki v drevesu: ");
        while (iterator.hasNext())     System.out.print(iterator.next() + " ");
    }
}
```

Popoln primer

Map (slovar)

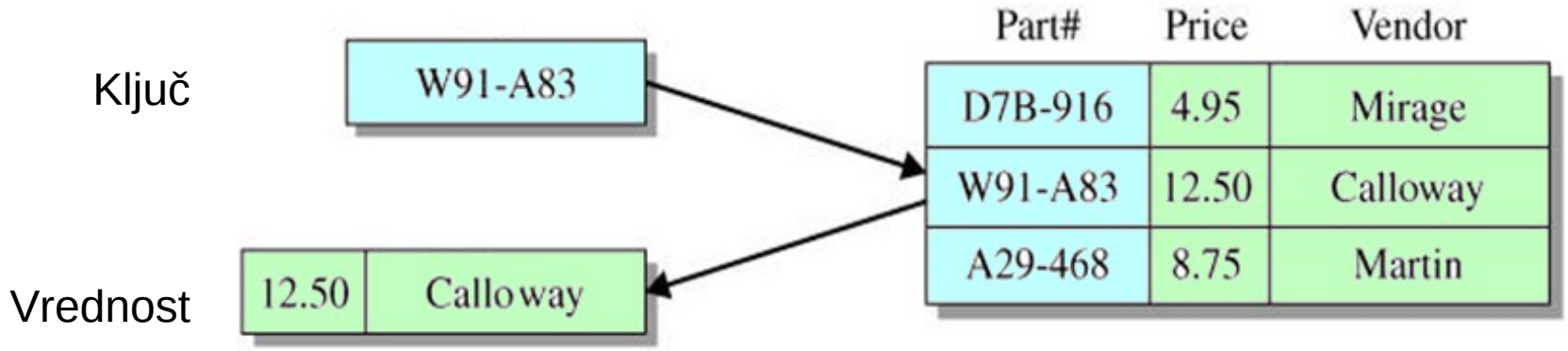
Map je posebna oblika množice.

- Množica parov ključ- vrednost.
- Vsak ključ preslika v največ eno vrednost.
- Le enkratni ključi vendar večkratne vrednosti



Nekaj primerov:

- Preslikava ključev v zapise podatkovne baze
- Slovar (besede preslikane v pomen)



Metode Map

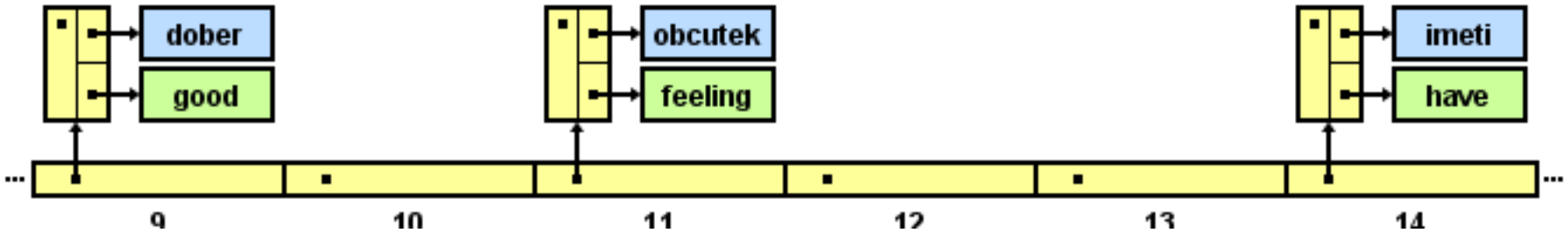
`clear()`	Odstrani vse preslikave
`containsKey(k)`	Ali vsebuje preslikavo za k
`containsValue(v)`	Ali vsebuje preslikavo v `v`
`entrySet()`	Postavi pare ključ-vrednost
`get(k)`	Vrne vrednost, ki ustreza k
`isEmpty()`	Ali je slovar prazen
`keySet()`	Množica ključev
`put(k,v)`	Asociacija vrednosti v ključu k
`remove(k)`	Odstrani preslikavo za `k`
`size()`	Število parov
`values()`	Kolekcija vrednosti

Demo

Uporaba Map

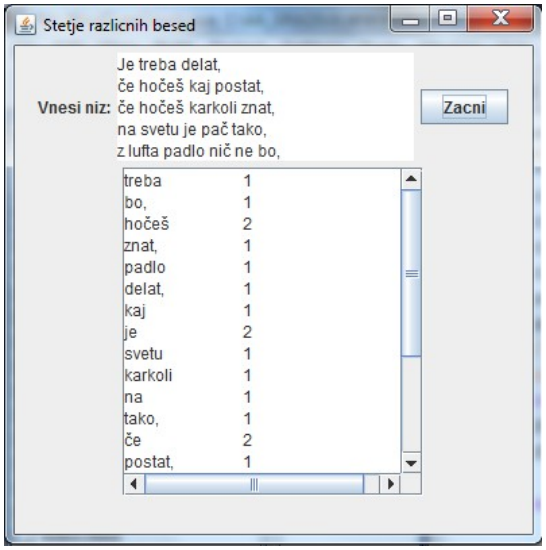
Demo

```
import java.util.*;

public class HashMapExample  {
   static String s;
   public static void main(String[] args) {
      HashMap map = new HashMap();
      map.put("Jaz", "I");
      map.put("imeti", "have");
      map.put("dober", "good");
      map.put("obcutek", "feeling");

      s=  (String) map.get("imeti");      System.out.println(s);
      s = (String) map.get("obcutek"); System.out.println(s);
     }
}
```



Primer uporabe Map

```
// Program steje stevilo nastopov vsake besede v nizu
    import java.awt.*;
    import java.awt.event.*;
    import java.util.*;
    import javax.swing.*;

    public class WordTypeCount extends JFrame {
        private JTextArea inputField;
       private JLabel prompt;
       private JTextArea display;
      private JButton goButton;
       private Map map;

      public WordTypeCount()      {
         super( "Stetje razlicnih besed" );
        inputField = new JTextArea( 3, 20 );

       map = new HashMap();

        goButton = new JButton( "Zacni" );
        goButton.addActionListener(
```

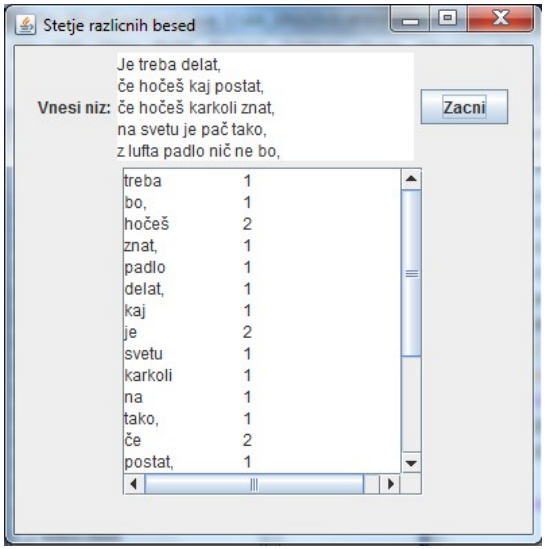


Tvorba HashMap

DEMO

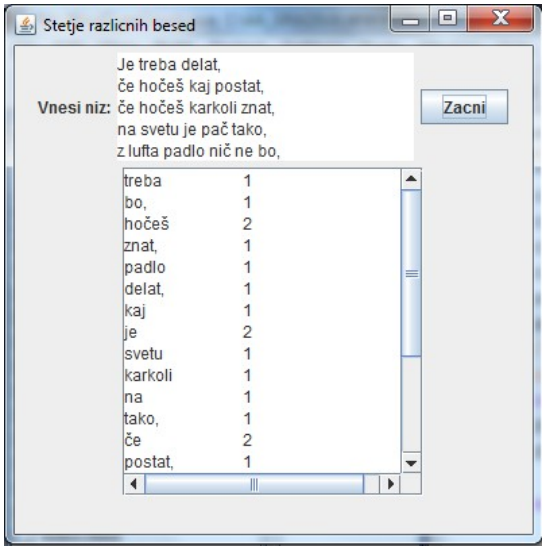
Primer uporabe Map (nadaljevanje 1)

```
        new ActionListener()   { // inner class
           public void actionPerformed( ActionEvent event )   {
              createMap();
              display.setText( createOutput() );
           }
        } // end inner class
     ); // end call to addActionListener

     prompt = new JLabel( "Vnesi niz:" );
     display = new JTextArea( 15, 20 );
     display.setEditable( false );
     JScrollPane displayScrollPane = new JScrollPane( display );
     // add components to GUI
     Container container = getContentPane();
     container.setLayout( new FlowLayout() );
     container.add( prompt );
     container.add( inputField );
     container.add( goButton );
     container.add( displayScrollPane );
    setSize( 400, 400 );
     show();
  } // end constructor
```



Primer uporabe Map (nadaljevanje 2)

```
// create map from user input
private void createMap()       {
   String input = inputField.getText();
   StringTokenizer tokenizer = new StringTokenizer( input );

    while ( tokenizer.hasMoreTokens() ) {
      String word = tokenizer.nextToken().toLowerCase(); // get word

      // if the map contains the word
      if ( map.containsKey( word ) ) {

         Integer count = (Integer) map.get( word ); // get value

         // increment value
         map.put( word, new Integer( count.intValue() + 1 ) );
      }
      else // otherwise add word with a value of 1 to map
         map.put( word, new Integer( 1 ) );

   } // end while
} // end method createMap
```



Z metodo `get` dobimo znak iz `HashMap`

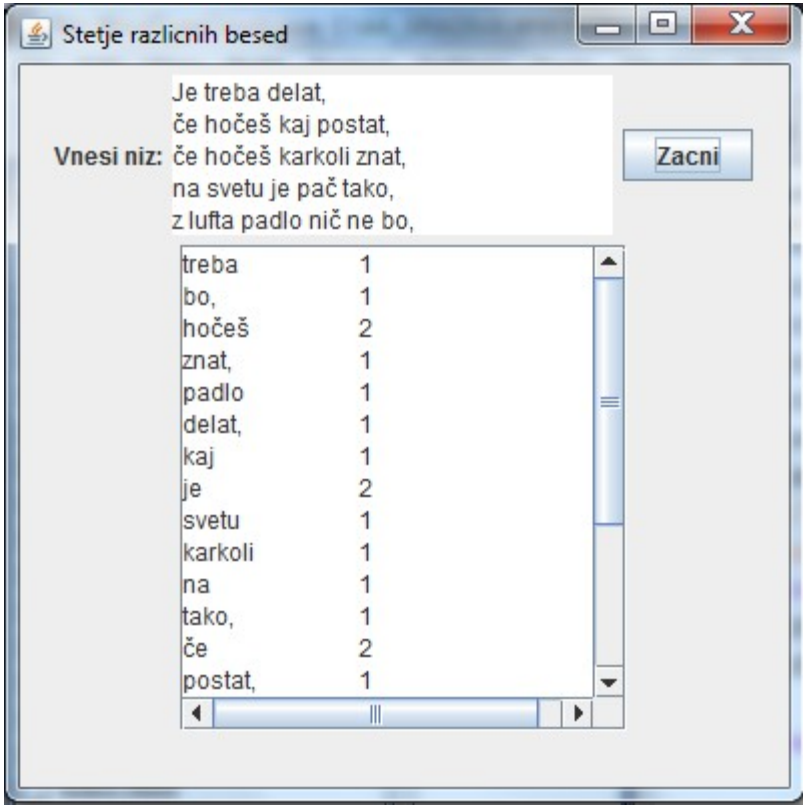
Z metodo `put` shranimo znak s ključem `Integer` v `HashMap`

Primer uporabe Map (nadaljevanje 3)

```
// create string containing map values
private String createOutput() {
    StringBuffer output = new StringBuffer( "" );
    Iterator keys = map.keySet().iterator();

    // iterate through the keys
    while ( keys.hasNext() ) {
       Object currentKey = keys.next();

      // output the key-value pairs
      output.append( currentKey + "\t" +    map.get( currentKey ) + "\n" );
    }

    output.append( “velikost: " + map.size() + "\n" );
    output.append( “Prazen niz: " + map.isEmpty() + "\n" );

    return output.toString();
} // end method createOutput
```

Primer uporabe Map (nadaljevanje 3)

```
    public static void main( String args[] )     {
        WordTypeCount application = new WordTypeCount();

        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLO
        SE );
    }
} // end class WordTypeCount
```



Primer: frekvenca besed v datoteki

```
public class FrekvencaBesed {
  static public void main(String[] args) {
    String line, word;
    Count count;
    Map words = new HashMap();
    String delim = " \t\n.,:;?!-/()[]\"\'";
    try {
       BufferedReader in =  new BufferedReader(new FileReader("jeTrebaDelat.txt"));
        while ((line = in.readLine()) != null) {
           StringTokenizer st = new StringTokenizer(line, delim);
           while (st.hasMoreTokens()) {
              word = st.nextToken().toLowerCase();
              count = (Count) words.get(word);
              if (count == null) {
                 words.put(word,  new Count(word, 1));
              }else  count.i++;
        }
      }
    } catch (Exception e) {}
```

Popoln primer

Primer: frekvenca besed (nadaljevanje 1)

```
import java.util.*;
import java.io.*;
//*************************************************
  class Count {
      public String word;
      public int i;
      public Count(String word, int i) {
          this.word = word;
          this.i = i;
      }
}
```

Primer: frekvenca besed (nadaljevanje 2)

```
Set set = words.entrySet();
   Iterator iter = set.iterator();
   while (iter.hasNext()) {
       Map.Entry entry = (Map.Entry) iter.next();
       word = (String) entry.getKey();
       count = (Count) entry.getValue();
      System.out.println(word + (word.length() < 8 ? "\t\t" : "\t") +
       count.i);
   }
```

Primer besedila

Je treba delat, če hočeš kaj postat,
če hočeš karkoli znat,
na svetu je pač tako,
z lufta padlo n'č ne bo,
je treba delat, ja delat, delat, kol'kr se le da.

Je treba delat, če hočeš nekam prit',
če hočeš pameten bit',
po svojih najboljših močeh,
brezdelje zapelje te u greh,
je treba delat, ja delat, delat, kol'kr se le da.

Ko pogledam u vesolje, se zdi mi, da pr' mer' stoji,
če pa pogledaš malo bolje, se kar širi in rotira,
oscilira in vibrira, čisto nič pri miru ni.

Je treba delat, …..

Primer: frekvenca besed: izpis

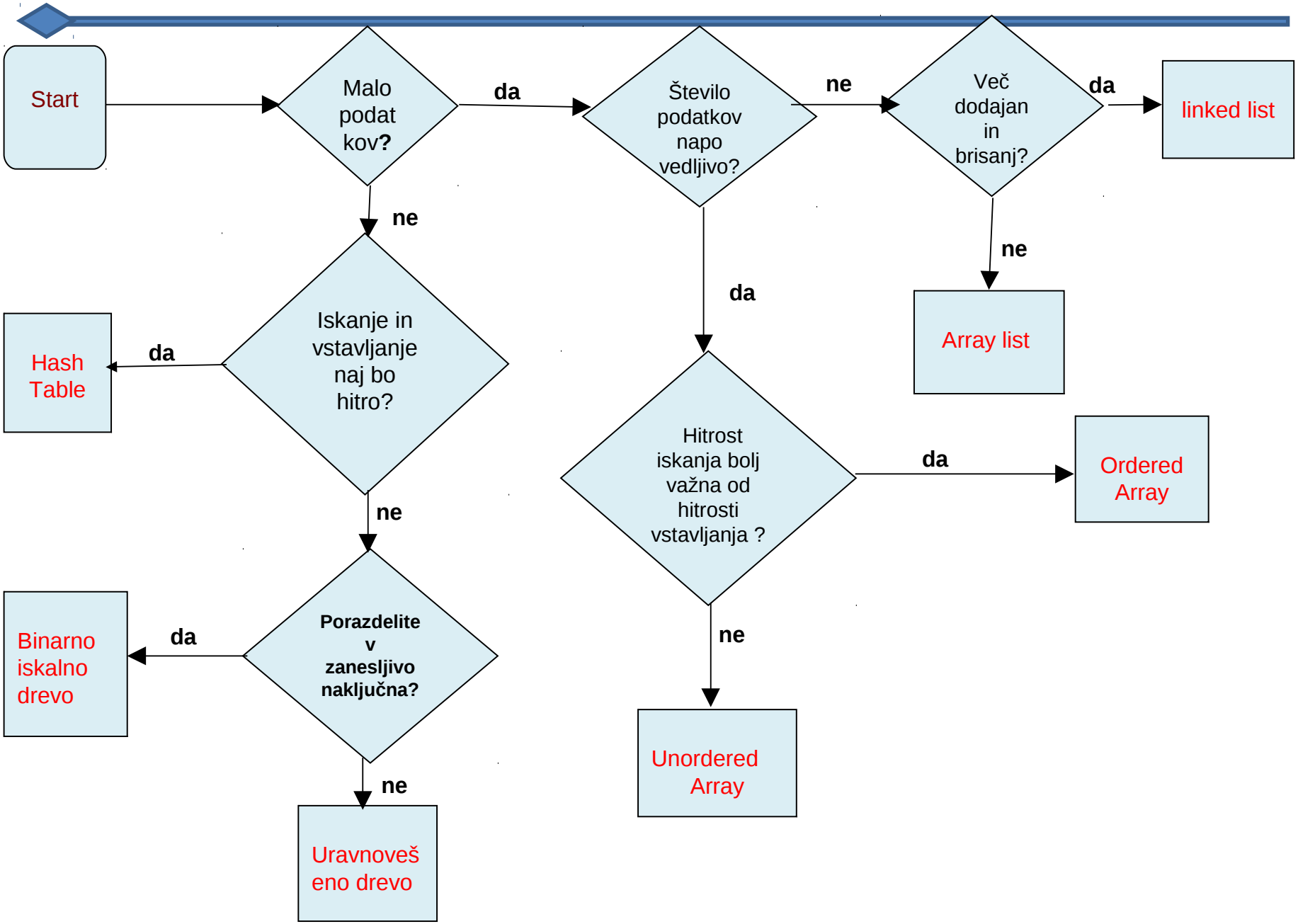
Smiselno	1	pa	12	kol	4
hočeš	4	bat	1	rotira	1
ga	1	klase	1	nič	2
žitno	1	živel	1	žal	1
zapelje	1	po	1	kok	2
znat	1	celica	1	deli	1
karkoli	1	rine	1	pomislim	1
lufta	1	živa	1	mi	5
pošteno	1	čisto	2	telo	1
polni	1	šteta	1	zemljo	1
boljše	1	vibrira	1	odšteva	1
č	1	vsakdo	1	n	1
naredil	1	živ	1	oscilira	1
kar	3	padlo	1	v	1
imel	1	greh	1	u	2
delo	1	smrt	1	na	1
širi	1	dela	1	lepo	1
vrti	1	svetu	1	r	2
kaj	2	krast	1	zapomni	1
bit	1	pač	1	ne	6

najpogostejša

delat	25

od 137 besed

Izbira primerne podatkovne strukture



Comparable in Comparator

- Štiri metode podpirajo mnoge tipe kolekcij: equals, compare in compareTo ter hashCode
  - Če vstavljamo lastne objekte v kolekcijo, moramo zagotoviti, da so te metode pravilno definirane
  - Vsaka kolekcija, ki ima tudi kakšen način preverjanja članstva, uporablja equals (ki je v mnogih primerih kar ekvivalenten ==)
  - Vsaka kolekcija, ki temelji na sortiranju, potrebuje primerjave večji/enak/manjši (compare ali compareTo)
  - Vsaka kolekcija, ki temelji na razprševanju (*hashing*), potrebuje tako preverjanje enakosti kot hash kode (equals in hashCode)
  - Kadarkoli implementiramo hashCode, moramo implementirati tudi equals
- Nekateri Javanski razredi, na primer String, že imajo implementirane vse te lastnosti
  - Za objekte, ki si jih sami izmislimo, moramo to narediti sami

Primerjava naših objektov

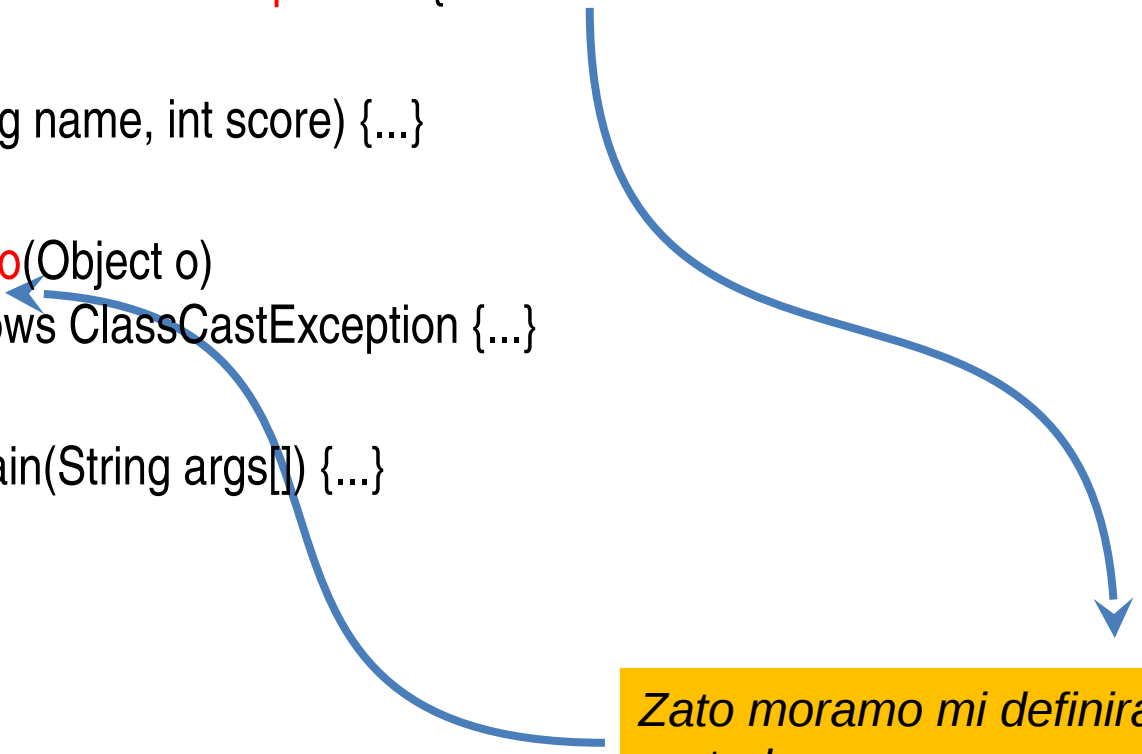
- Razred Object nudi javni metodi

  boolean equals(Object obj) in public int hashCode()

  - Za naše objekte, ki jih mi definiramo, uporabljata podedovani metodi equals in hashCode naslov objektov v pomnilniku
  - Ti metodi lahko prekrijemo
  - Če prekrijemo metodo equals, moramo prekriti tudi hashCode
  - Če prekrijemo hashCode, moramo prekriti tudi equals
- Razred Object sam po sebi ne nudi metod tipa “manjši” ali “večji”— vendar ...
  - Imamo vmesnik Comparable, definiran v java.lang
  - Imamo vmesnik Comparator definiran v java.util

Primer: Razred Student

```
public class Student implements Comparable {

    public Student(String name, int score) {...}

    public int compareTo(Object o)
                    throws ClassCastException {...}

    public static void main(String args[]) {...}
}
```



Zato moramo mi definirati to metodo

Konstruktor za razred Student

- Ta je za oba pristopa enak, nič ni tu novega:

```
public Student(String name, int score) {
      this.name = name;
      this.score = score;
   }
```

- Študente želimo urediti glede na njihovo oceno
- V tem primeru bomo sicer uporabili množice. Pomembno pa je, da primerjamo po dva objekta ne glede na tip uporabljene zbirke

DEMO

Metoda main, verzija 1

```
public static void main(String args[]) {
    TreeSet<Student> set = new TreeSet<Student>();

    set.add(new Student(“Ana", 87));
    set.add(new Student(“Robert", 83));
    set.add(new Student(“Katja", 99));
    set.add(new Student("Danilo", 25));
    set.add(new Student("Eva", 76));

    Iterator<Student> iter = set.iterator();
    while (iter.hasNext()) {
      Student s = iter.next();
      System.out.println(s.name + "  " + s.score);
    }
}
```

Uporabimo TreeSet

- V metodi *main* imamo vrstico

  TreeSet set = new TreeSet();

- Nato uporabimo iterator za zaporedni izpis vseh vrednosti. Dobimo naslednji izpis:

  Danilo 25
  Eva 76
  Robert 83
  Ana 87
  Katja 99

- Kako je iterator vedel, da je moral urediti študente po oceni namesto morda po imenih?

Implementirajmo Comparable

```
public class Student implements Comparable
```

- To pomeni, da moramo implementirati metodo

```
public int compareTo(Object o)
```

- Opazimo, da imamo za parameter Object
- Če želimo implementirati ta vmesnik (Comparable) , mora biti tudi naš parameter tipa Object, čeprav tega ne bi želeli

```
public int compareTo(Object o) throws ClassCastException {
    if (o instanceof Student)
      return score - ((Student)o).score;
    else
      throw new ClassCastException("Ni student!");
  }
```

- Do izjeme ClassCastException mora priti, če kot parameter navedemo nekaj, kar ni tipa Student

Izboljšana metoda

- Ker pretvorba (cast) poljubnega objekta v tip Student lahko za nas povzroči izjemo classCastException, ni potrebno, da to predvidevamo mi eksplicitno:

```
public int compareTo(Object o) throws ClassCastException {
        return score - ((Student)o).score;
}
```

- Še več, ker je classCastException podrazred RuntimeException, nam sploh ni potrebno, da to mi deklariramo:

```
public int compareTo(Object o) {
        return score - ((Student)o).score;
}
```

Uporabimo ločen “Comparator”

- V pravkar razvitem programu je Student implementiral Comparable
  - Zato je imel metodo compareTo
  - Študente smo lahko razvrščali le glede na njihovo oceno
  - Če bi hoteli razvrstiti študente po kakšnem drugem kriteriju (na primer po imenu), bi bili v težavah!
- Zato vstavimo metodo primerjave v ločen razred, ki implementira Comparator namesto Comparable
  - To bo precej bolj fleksibilno (uporabili bomo lahko različen Comparator za razvrščanje študentov po imenu ali po oceni), je pa tudi bolj okorno
  - Comparator je v java.util, ne v java.lang
  - Comparable zahteva definicijo compareTo , Comparator zahteva definicijo compare
  - Comparator zahteva tudi neke vrste equals

Osnutek StudentComparator

```
import java.util.*;

public class StudentComparator
                    implements Comparator<Student> {

    public int compare(Student s1, Student s2) {...}


    public boolean equals(Object o1) {...}
}
```

- Opomba: Ko uporabljamo ta Comparator, v razredu Student ne potrebujemo metode compareTo
- Zaradi generikov lahko naša metoda compare uporablja argumente tipa Student namesto argumentov tipa Object

Metoda compare

```
public int compare(Student  s1, Student s2) {
        return s1.score – s2.score;
}
```

- To se razlikuje od compareTo(Object o) v Comparable:
  - Ime je drugačno
  - Kot parametra navajamo oba objekta, ne le enega
  - Ali uporabimo generike, ali pa preverjamo tip obeh objektov
  - Če sta parametra tipa Object, jih moramo pretvoriti (cast) v tip Student

Metoda *someComparator*.equals

- Ignorirajmo to metodo!
  - Te metode ne uporabljamo za primerjavo dveh študentov. Rabi se pri primerjavi dveh komparatorjev
  - Čeprav je del vmesnika Comparator ,je ni potrebno prekriti
    - Implementiranje vmesnika sicer zahteva, da definiramo VSE metode vmesnika (ali je to izjema?)
    - Zato, ker IMAMO definicijo, podedovano iz razreda Object !
  - Pravzaprav je bolj varno, da to metodo ignoriramo
  - Cilj je učinkovitost – en Comparator lahko zamenjamo z drugim enakim ali hitrejšim

Metoda main

- Metoda main je taka kot prej, le namesto

```
TreeSet<Student> set = new TreeSet<Student>();
```

imamo sedaj

```
Comparator<Student> comp = new StudentComparator();
TreeSet<Student> set = new TreeSet<Student>(comp);
```

Kdaj kaj uporabimo?

- Vmesnik Comparable je bolj preprost in dela je manj
  - Naš razred implements Comparable
  - Mi zagotovimo metodo public int compareTo(Object o)
  - V naših konstruktorjih TreeSet ali TreeMap ne uporabljamo argumentov
  - Vedno bomo uporabljali isto primerjalno metodo
- Vmesnik Comparator je bolj fleksibilen, je pa nekaj več dela
  - Tvorimo toliko različnih razredov, ki implementirajo Comparator , kolikor jih hočemo
  - Zbirke TreeSet ali TreeMap lahko z vsakim različno sortiramo
    - Tvorimo TreeSet ali TreeMap s tistim komparatorjem, ki ga želimo
  - Tako lahko študente razporejamo po oceni ali po imenu

Primer:razvrščanje delavcev po starosti in imenu

```
class Delavec{
   private int starost;
   private String ime;

   public void setStarost(int st1){
      this.starost = st1;
   }

   public int getStarost(){
      return this.starost;
   }

   public void setIme(String ime1){
      this.ime = ime1;
   }

   public String getIme(){
      return this.ime;
   }
}
```

Imejmo razred Delavec

Razvrščanje delavcev: dva komparatorja

```
class StarostComparator implements Comparator{
   public int compare(Object delavec1, Object delavec2){
      int starost1 = ((Delavec)delavec1).getStarost();
      int starost2 = ((Delavec)delavec2).getStarost();
      if (starost1 > starost2)       return 1;
      else if (starost1 < starost2)  return  - 1;
      else    return 0;
   }
}
```

Za primerjanje po starosti

```
class ImeComparator implements Comparator{
   public int compare(Object delavec1, Object delavec2){
      String ime1 = ((Delavec)delavec1).getIme();
      String ime2 = ((Delavec)delavec2).getIme();
      //uporabimo kar metodo za primerjanje nizov
      return ime1.compareTo(ime2);
   }
}
```

Za primerjanje po imenu

Razvrščanje delavcev: testni primer

```
public class ComparatorExample{

   public static void main(String args[]) {

      Delavec delavec[] = new Delavec[3]; // imamo polje treh delavcev

      //Tvorimo posamezne delavce in njihove lastnosti..
      delavec[0] = new Delavec();
      delavec[0].setStarost(40);   delavec[0].setIme("Janez");
      delavec[1] = new Delavec();
      delavec[1].setStarost(20);  delavec[1].setIme("Miha");
      delavec[2] = new Delavec();
      delavec[2].setStarost(30);  delavec[2].setIme("Micka");
```

Razvrščanje delavcev: po starosti in po imenu

```
Arrays.sort(delavec, new StarostComparator());
System.out.println("\n\nZaporedje delavcev, urejenih po starosti:");
for (int i = 0; i < delavec.length; i++){
   System.out.println("Delavec " + (i + 1) + " ime :: " +
      delavec[i].getIme() + ", starost :: " + delavec[i].getStarost());
}

Arrays.sort(delavec, new ImeComparator());
System.out.println("\n\nZaporedje delavcev, urejenih po imenih:");
for (int i = 0; i < delavec.length; i++){
   System.out.println("Delavec " + (i + 1) + " ime :: " +
      delavec[i].getIme() + ", starost :: " + delavec[i].getStarost());
}
```

Demo

Algoritmi

- “Collections Framework” nudi množico algoritmov
  - Implementirane kot statične metode
    - Algoritmi List
      - sort
      - binarySearch
      - reverse
      - shuffle
      - fill
      - copy
    - Algoritmi Collection
      - min
      - max

Algoritem sort

- `sort`
  - Uredi elemente `List`
    - Vrstni red je določen z naravnim redom tipa elementov
    - Relativno hiter

Primer uporabe metode sort

```
import java.util.*;

 public class Sort1 {
   private static final String suits[] =
     { "Hearts", "Diamonds", "Clubs", "Spades" };

   // display array elements
  public void printElements()      {
    // create ArrayList
    List list = new ArrayList( Arrays.asList( suits ) );

    // output list
    System.out.println( "Unsorted array elements:\n" + list );

    Collections.sort( list ); // sort ArrayList

    // output list
    System.out.println( "Sorted array elements:\n" + list );
  }
```

Create ArrayList

Use Collections method sort to sort ArrayList

Primer sort (nadaljevanje)

```
   public static void main( String args[] )     {
       new Sort1().printElements();
   }

} // end class Sort1
```

- Unsorted array elements:
- [Hearts, Diamonds, Clubs, Spades]
- Sorted array elements:
- [Clubs, Diamonds, Hearts, Spades]

Primer Sort2

```
// Using a Comparator object with algorithm sort.
import java.util.*;

public class Sort2 {
   private static final String suits[] =
      { "Hearts", "Diamonds", "Clubs", "Spades" };

   // output List elements
  public void printElements()  {
     List list = Arrays.asList( suits ); // create List

     // output List elements
     System.out.println( "Unsorted array elements:\n" + list );

     // sort in descending order using a comparator
     Collections.sort( list, Collections.reverseOrder() );

     // output List elements
     System.out.println( "Sorted list elements:\n" + list );
  }
```

Metoda reverseOrder razreda Collections vrne objekt Comparator, ki predstavlja obratno zaporedje kolekcije

Metoda sort razreda Collections lahko uporabi objekt Comparator za urejanje List

Primer Sort2 (nadaljevanje)

```
public static void main( String args[] )      {
   new Sort2().printElements();
}

29  } // end class Sort2
```

- Unsorted array elements:
- [Hearts, Diamonds, Clubs, Spades]
- Sorted list elements:
- [Spades, Hearts, Diamonds, Clubs]

Primer Sort3

```
import java.util.*;

public class Sort3 {
   public void printElements()       {
      List list = new ArrayList(); // create List

      list.add( new Time2(  6, 24, 34 ) );
      list.add( new Time2( 18, 14, 05 ) );
      list.add( new Time2(  8, 05, 00 ) );
      list.add( new Time2( 12, 07, 58 ) );
      list.add( new Time2(  6, 14, 22 ) );

      // output List elements
      System.out.println( "Unsorted array elements:\n" + list );
      // sort in order using a comparator
     Collections.sort( list, new TimeComparator() );

     // output List elements
     System.out.println( "Sorted list elements:\n" + list );
}
```

Sort in order using a custom comparator `TimeComparator`.

Sort3 (nadaljevanje 1)

```
public static void main( String args[] )      {
    new Sort2().printElements();
}

private class TimeComparator implements Comparator {
   int hourCompare, minuteCompare, secondCompare;
   Time2 time1, time2;

   public int compare(Object object1, Object object2)      {
      // cast the objects
      time1 = (Time2)object1;
      time2 = (Time2)object2;
      hourCompare = new Integer( time1.getHour() ).compareTo(
                    new Integer( time2.getHour() ) );
      // test the hour first
      if ( hourCompare != 0 )    return hourCompare;

      minuteCompare = new Integer( time1.getMinute() ).compareTo(
                      new Integer( time2.getMinute() ) );
```

Custom comparator TimeComparator implements Comparator interface.

Implement method compare to determine the order of two objects.

Sort3 (nadaljevanje 2)

```
         // then test the minute
          if ( minuteCompare != 0 )
             return minuteCompare;

          secondCompare = new Integer( time1.getSecond() ).compareTo(
                          new Integer( time2.getSecond() ) );

          return secondCompare; // return result of comparing seconds
       }

   } // end class TimeComparator

} // end class Sort3
```

- Unsorted array elements:
- [06:24:34, 18:14:05, 08:05:00, 12:07:58, 06:14:22]
- Sorted list elements:
- [06:14:22, 06:24:34, 08:05:00, 12:07:58,

Algoritem shuffle

- `shuffle`
  - Naključno razvrsti elemente `List`

Primer s kartami

```
import java.util.*;
    // karta v paketu kart
    class Karta {
      private String podoba;
      private String vrednost;

     public Karta( String podob1, String vred1 )    {
       podoba = podob1;  vrednost = vred1;
     }

     public String getPodoba()      {  return podoba;   }

     public String getVrednost() {   return vrednost;    }

     public String toString() {
          return podoba + " " + vrednost;
     }

  } // end class Karta
```

Primer s kartami (nadaljevanje 1)

```
// deklaracija razreda IgraKart
   public class IgraKart {
      private static final String podobe[] =    { "Srce", "Pik", "Karo", "Kriz" };
      private static final String vrednosti[] = { "As", "Dva", "Tri",  "Stiri", "Pet", "Sest",
           "Sedem", "Osem", "Devet", "Deset",   "Fant", "Dama", "Kralj" };
      private List list;

      // vzpostavimo paket kart
      public IgraKart()  {
           int trenutnaBarva = -1;
         Karta paket[] = new Karta[ 52 ];
        for ( int count = 0; count < paket.length; count++ ){
            if((count%13)== 0) trenutnaBarva++;
            paket[ count ] = new Karta( podobe[ trenutnaBarva ], vrednosti[ count %
    13 ]     );
         } // for
        list = Arrays.asList( paket );   // get List
        Collections.shuffle( list );    // paket zamesamo
      }
```

Z metodo shuffle zmešamo List

Primer s kartami (nadaljevanje 2)

```
// izpis paketa
      public void printCards() {
         for ( int i = 0; i < 52; i++ )
            System.out.println ((i+1) + ": " + list.get( i ).toString());
      }

      public static void main( String args[] )     {
         new IgraKart().printCards();
      }

   } // end class IgraKart
```

Ahh ?!

Demo

Primer s kartami (izpis)

1: Kriz Kralj
2: Kriz Sest
3: Kriz Deset
4: Kriz Devet
5: Karo Dama
6: Srce As
7: Karo Stiri
8: Karo Sest
9: Pik Stiri
10: Kriz Dva
11: Pik Osem
12: Kriz Osem
13: Srce Osem
14: Kriz Sedem
15: Karo Fant
16: Pik Sedem
17: Kriz Tri

Algoritmi reverse, fill, copy, max in min

- reverse
  - Obrne vrstni red elementov List
- fill
  - Napolni elemente List z vrednostmi
- copy
  - Tvori kopijo List
- max
  - Poišče in vrne največji element v List
- min
  - Poišče in vrne najmanjši element v List

Primer z algoritmi

```
import java.util.*;

   public class Algorithms1 {
      private String letters[] = { "P", "C", "M" }, lettersCopy[];
      private List list, copyList;

      // create a List and manipulate it with methods from Collections
     public Algorithms1()     {
        list = Arrays.asList( letters );     // get List
        lettersCopy = new String[ 3 ];
        copyList = Arrays.asList( lettersCopy );
           System.out.println( "Initial list: " );
        output( list );

        Collections.reverse( list );         // reverse order
        System.out.println( "\nAfter calling reverse: " );
        output( list );

        Collections.copy( copyList, list );  // copy List
        System.out.println( "\nAfter copying: " );
        output( copyList );
```

Use method reverse of class Collections to obtain List in reverse order

Use method copy of class Collections to obtain copy of List

Primer z algoritmi (nadaljevanje1)

```
      Collections.fill( list, "R" );       // fill list with Rs
      System.out.println( "\nAfter calling fill: " );
      output( list );
   } // end constructor

   // output List information
   private void output( List listRef )    {
      System.out.print( "The list is: " );

      for ( int k = 0; k < listRef.size(); k++ )
         System.out.print( listRef.get( k ) + " " );

      System.out.print( "\nMax: " + Collections.max( listRef ) );
      System.out.println( "  Min: " + Collections.min( listRef ) );
   }

   public static void main( String args[] )     {
      new Algorithms1();
   }
} // end class Algorithms1
```

List napolnimo s črkami "R"

Dobimo maksimum v List

Dobimo minimum v List

DEMO

Primer z algoritmi (izpis)

```
Initial list:
The list is: P C M
Max: P  Min: C

After calling reverse:
The list is: M C P
Max: P  Min: C

After copying:
The list is: M C P
Max: P  Min: C

After calling fill:
The list is: R R R
Max: R  Min: R
```

Algoritem binarySearch

- binarySearch
  - Poišče objekt v List
    - Vrne indeks objekta v List, če objekt tu obstaja
    - Vrne negativno vrednost, če objekta ni v List

Primer z BinarySearch

```
import java.util.*;
 public class BinarySearchTest {
    private static final String colors[] = { "red", "white",  "blue", "black", "yellow", "purple",
  "tan", "pink" };
    private List list;        // List reference
  // create, sort and output list
  public BinarySearchTest()    {
     list = new ArrayList( Arrays.asList( colors ) );
     Collections.sort( list );   // sort the ArrayList
     System.out.println( "Sorted ArrayList: " + list );
  }
  // search list for various values
  private void printSearchResults()   {
     printSearchResultsHelper( colors[ 3 ] ); // first item
     printSearchResultsHelper( colors[ 0 ] ); // middle item
     printSearchResultsHelper( colors[ 7 ] ); // last item
     printSearchResultsHelper( "aardvark" );  // below lowest
     printSearchResultsHelper( "goat" );      // does not exist
     printSearchResultsHelper( "zebra" );     // does not exist
  }
```

List uredimo v naraščajočem zaporedju

Primer z BinarySearch (nadaljevanje)

```
   // helper method to perform searches
   private void printSearchResultsHelper( String key )      {
      int result = 0;

      System.out.println( "\nSearching for: " + key );
      result = Collections.binarySearch( list, key );
      System.out.println( ( result >= 0 ? "Found at index " + result :
         "Not Found (" + result + ")" ) );
   }

   public static void main( String args[] )      {
      new BinarySearchTest().printSearchResults();
   }

} // end class BinarySearchTest
```

Use method binarySearch of class Collections to search List for specified key

DEMO

```
Sorted ArrayList: black blue pink purple red tan white
Searching for: black
Found at index 0

Searching for: red
Found at index 4

Searching for: pink
Found at index 2

Searching for: aardvark
Not Found (-1)

Searching for: goat
Not Found (-3)

Searching for: zebra
Not Found (-9)
```

Uvod v generike

Javanska kolekcija je fleksibilna podatkovna struktura, ki lahko vsebuje heterogene objekte in imajo lahko elementi različen tip reference.

Skrb programerja je, da vodi evidenco o tem, kakšne tipe objektov vsebuje kolekcija.

Primer: V kolekcijo želimo dodajati cela števila. Vendar vanjo lahko vstavljamo le objekte. Zato moramo vsako celoštevilčno spremenljivko prej pretvoriti v ustrezen referenčni tip (torej Integer). Ko pa uporabljamo elemente iz kolekcije splošnih objektov, moramo spet zagotoviti pravi tip (torej uporabiti kasto (Integer))

Zato so taki javanski programi težje berljivi in bolj pogosto pride do napak v času izvajanja.

Uporabimo generike

Z uporabo generikov kolekcije ne obravnavamo več kot seznam referenc na objekte in lahko razločujemo med kolekcijami referenc na Integer , referenc na Byte itd.

Kolekcija z generičnim tipom ima parameter tipa, ki specificira tip elementov, ki so v taki kolekciji pomnjeni.

Primer

Imejmo povezani seznam najprej brez uporabe generikov. Dodajmo mu celoštevilčno vrednost in jo nato preberimo nazaj:

```
LinkedList list = new LinkedList();
list.add(new Integer(1));
   . . . . .
Integer num = (Integer) list.get(0);
```

To je sicer zaradi eksplicitne pretvorbe (kaste) varno, vendar bi v času izvajanja lahko prišlo do napake, če bi brali objekt nekega drugega tipa.

Z uporabo generikov pa bi to zgledalo tako:

```
LinkedList<Integer> list = new LinkedList<Integer>();
list.add(new Integer(1));
Integer num = list.get(0);
```

Primer (nadaljevanje)

Nered zmanjšamo, če primer zapišemo malo drugače, z uporabo opredmetenja (autoboxing):

```
LinkedList<Integer> list = new
LinkedList<Integer>();
list.add(1);
int num = list.get(0);
```

Več o autoboxing

Še en primer brez generikov

Poglejmo še primer, ko tvorimo kolekcijo z dvema nizoma in enim celoštevilčnim objektom ter nato kolekcijo izpišemo:

```
import java.util.*;
public class Ex1 {
    private void testCollection() {
        List list = new ArrayList();
        list.add(new String(“Dober dan!"));
        list.add(new String(“Na svidenje!"));
        list.add(new Integer(95));
        printCollection(list);
    }
    private void printCollection(Collection c) {
        Iterator i = c.iterator();
        while(i.hasNext()) {
            String item = (String) i.next(); System.out.println("Item: "+item);
    } }
    public static void main(String argv[]) {
    Ex1 e = new Ex1(); e.testCollection(); } }
```

Potrebna je eksplicitna pretvorba.

Pri izpisu tretjega elementa pride do napake

DEMO

Primer z generiki

```
import java.util.*;
public class Ex2 {
    private void testCollection() {
        List<String> list = new ArrayList<String>();
        list.add(new String(“Dober dan!"));
        list.add(new String("Good bye!"));
        list.add(new Integer(95));
        printCollection(list);
    }
    private void printCollection(Collection c) {
        Iterator<String> i = c.iterator(); while(i.hasNext()) {
            System.out.println("Item: "+i.next());
    } }

    public static void main(String argv[]) {
        Ex2 e = new Ex2(); e.testCollection();
} }
```

O napaki nas bo obvestil že prevajalnik (ne moremo Integer elementa vstaviti v kolekcijo elementov String)

DEMO

Generiki in parametrizirani razredi

Javanske generike uporabljamo v poljih in metodah. Lahko pa jih uporabljamo tudi za tvorbo parametriziranih razredov

Primer : Imejmo razred Member, katerega polje Id lahko parametriziramo v String, Integer ali kaj drugega:

Glej primer na naslednji strani

DEMO

Več o generikih v Javi

Generiki in parametrizirani razredi (nadaljevanje)

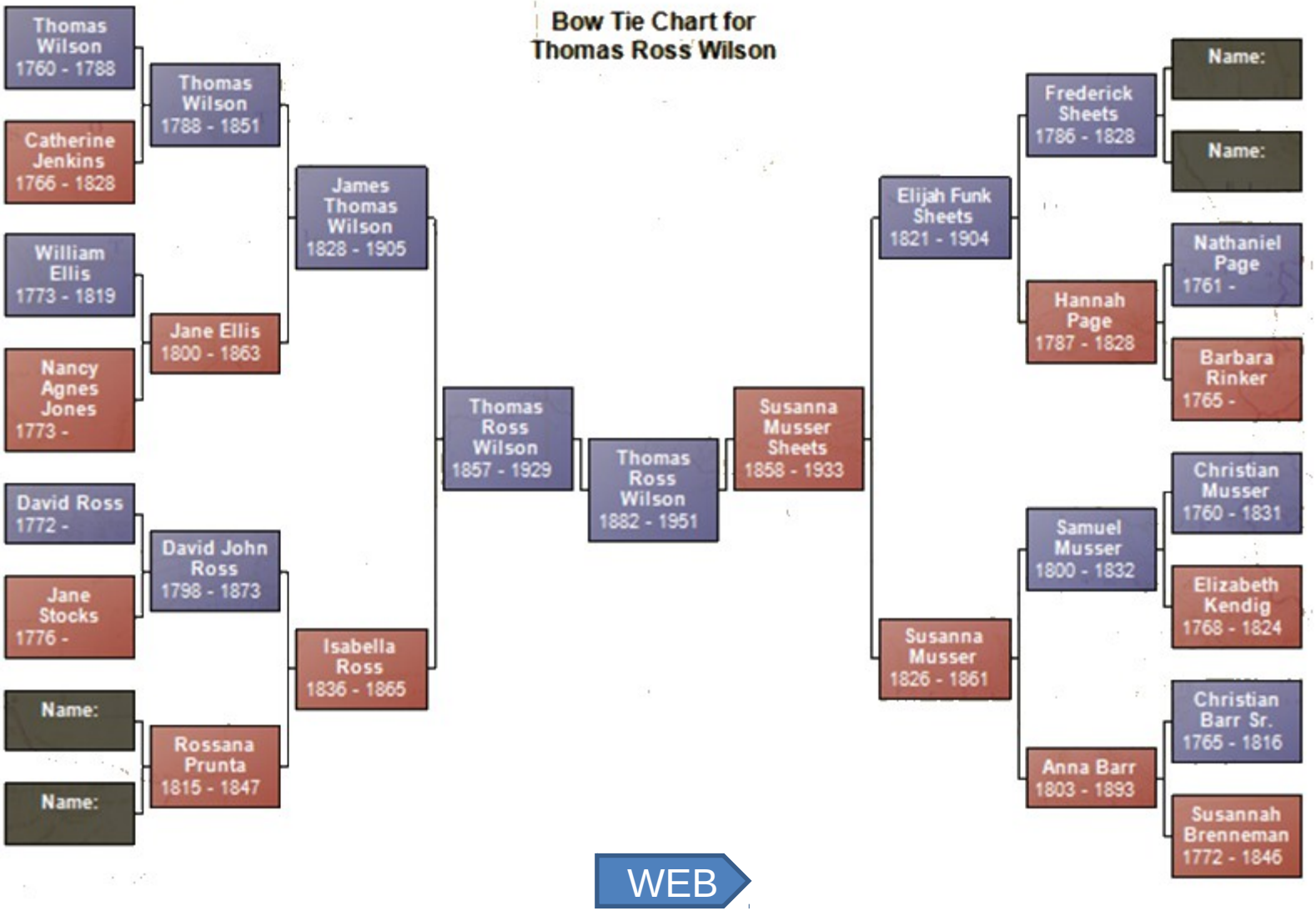
```
public class Member<T> {
    private T id;
    public Member(T id) {
      this.id = id;
 }
 public T getId() {
    return id;
}
 public void setId(T id) {
    this.id = id;
 }

 public static void main(String[] args) {
     Member<String> mString = new Member<String>(“tralala");
     mString.setId(“hopsasa");
     System.out.printf("id  je: %s%n", mString.getId());  //izpis:  id je: hopsasa

   Member<Integer> mInteger = new Member<Integer>(1);
   mInteger.setId(2);
   System.out.printf("id je: %d%n", mInteger.getId());  //izpis:  id je: 2
 }
 }
```

Za konec

“Uporaba kolekcij je stil programiranja, ki omogoča ponovno uporabljivost algoritmov z različnimi tipi podatkov”

“Objektno usmerjeno programiranje je dalo večjo moč tipom... Generično programiranje daje večjo moč algoritmom”

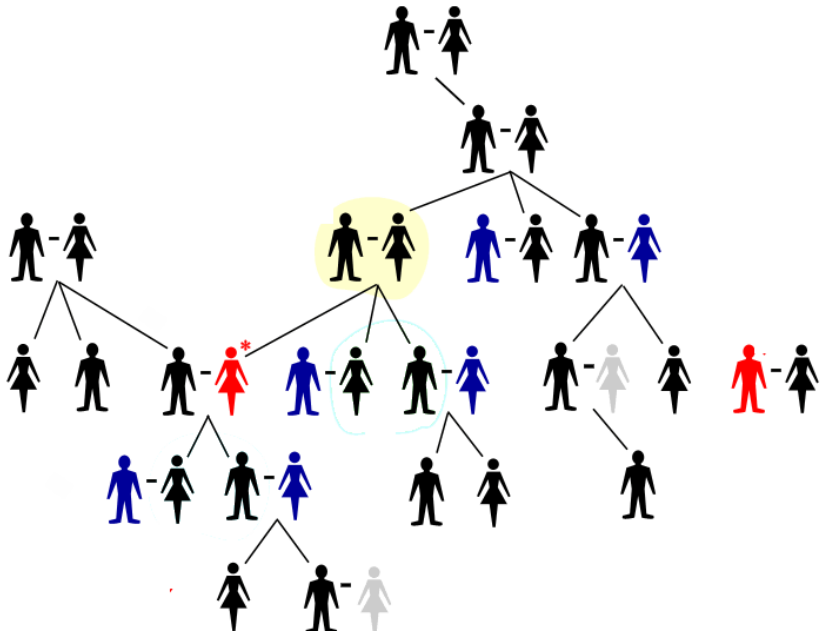
Študijski primer: družinsko drevo



WEB

Študijski primer: družinsko drevo

```
class Person {
    private Person mother;
    private Person father;
     String name;
     int birthYear;
}
```



```
 class Person {
     private ArrayList<Person> parents;
     private ArrayList<Person> children;
      String name;
      int birthYear;
 }
```

Demo