Akademija tehničko-vaspitačkih strukovnih studija
odsek NIŠ

Savremene računarske tehnologije **SRT**

OBJEKTNO ORIJENTISANO PROGRAMIRANJE - OOP

Prof. dr Zoran Veličković, dipl. inž. el.

2019/2020.

Prof. dr Zoran Veličković, dipl. inž. el.

OBJEKTNO ORIJENTISANO PROGRAMIRANJE - OOP

Generički tipovi i kolekcije u OOP

(12)

Sadržaj

- **GENERIČKI TIPOVI**
  - Konverzija tipova
  - Bezbedan rad sa objektima
- **GENERIČKE KLASE**
  - Jednostavna generička klasa Gen<T>
  - Primena generičke klase: Gen<T>
  - Generičke klase sa više tipova parametara
  - Ograničeni tipovi
- **GENERIČKE METODE**
  - Primer generičke metode
- **GENERIČKI INTERFEJSI**
  - Primer generičkog interfejsa
- **KOLEKCIJE**
  - Javine kolekcije
  - Primer: klasa ArrayList

Generički tipovi (1)

- **GENERIČKI** (engl. *generics*) **TIPOVI** omogućavaju kreiranje:
  - klasa,
  - interfejsa i
  - metoda

  koje **BEZBEDNO RADE** sa podacima **RAZLIČITIH TIPOVA**.
- Tako na primer, način na koji radi **STEK** je **ISTI** bez obzira na **TIP PODATAKA** (Integer, String ili Object) sa kojima je realizovan.
- Kada se razviju algoritmi za rad sa **GENERIČKIM TIPOVIMA**, oni se mogu primeniti na **RAZLIČITE TIPOVE PODATAKA** (dakle nezavisni su od tipa podataka sa kojima se radi)!
- Java poseduje **RADNO OKRUŽENJE** za rad sa **SKUPOM OBJEKATA** (engl. *Collections framework*) koje se u OO programiranju nazivaju **KOLEKCIJE** (engl. *Collections*).
- **KOLEKCIJE** su bazirane na **GENERIČKIM TIPOVIMA**, i definisane su klase za upravljanje kolekcijama.
- **KOLEKCIJE** omogućavaju **BEZBEDAN RAD** sa **SVIM TIPOVIMA** podataka jer koriste generičke klase i metode.

Generički tipovi (2)

- **GENERIČKI** (kaže se i parametarizovani) **TIPOVI** omogućavaju da se klasama, metodama i interfejsima **PROSLEDI TIP PODATAKA** sa kojima treba da rade.
- Dakle, klase, interfejsi ili metode koje rade sa parametarizovanim tipovima nazivaju se **GENERIČKIM KLASAMA**, **GENERIČKIM INTERFEJSIMA** i **GENERIČKIM METODAMA**.
- Iako se sve ovaj cilj može postići radom sa objektima tipa **`Object`**, greške koje mogu nastati prilikom pokušaja rada sa **NEKOMPATIBILNIM TIPOVIMA** podataka su glavni limitirajući faktor.
- Sa generičkim tipovima **SVE KONVERZIJE TIPOVA** se odvijaju **AUTOMATSKI** i time obezbeđuju **BEZBEDAN** i **EFIKASAN** rad.
- Krenimo od klase, deklaracija **GENERIČKE KLASE** ima sledeći oblik:

```
class ime_klase <lista_parametara_tipa> {

    // telo klase

}
```

Oznake < i > se odnose na generički tip parametara

Jednostavna generička klasa

// Jednostavna OPŠTA KLASA. Parametar **T** predstavlja tip koji će biti
// zamenjem realnim tipom kada se objekt tipa **Gen** kreira

```
class Gen<T> {
  T ob;
  Gen(T o) {
     ob = o;
  }
  T getob() {
    return ob;
   }
     void showType() {
     System.out.println("Tip T je " + ob.getClass().getName());
    }
} // Gen
```

Generička klasa **Gen**, radi sa parametraskim tipom podataka tipa **T**

Dekleracija ob objekta tipa **T**

Konstruktor klase **Gen**, prosleđen mu je objekt tipa **T**

Deklaracija metode **getob()** koja vraća rezultat tipa **T**

Metoda **showType()**, prikazuje tip objekta **T**

Oracle-ov help za metoda **getName()** prikazan je na sledećem slajdu

Metoda GetName()

getName

```
public String getName()
```

Returns the name of the entity (class, interface, array class, primitive type, or void) represented by this `Class` object, as a `String`.

If this class object represents a reference type that is not an array type then the binary name of the class is returned, as specified by *The Java™ Language Specification*.

If this class object represents a primitive type or void, then the name returned is a `String` equal to the Java language keyword corresponding to the primitive type or void.

If this class object represents a class of arrays, then the internal form of the name consists of the name of the element type preceded by one or more '`[`' characters representing the depth of the array nesting. The encoding of element type names is as follows:

Element Type	Encoding
boolean	Z
byte	B
char	C
class or interface	L*classname*;
double	D
float	F
int	I
long	J
short	S

The class or interface name *classname* is the binary name of the class specified above.

Examples:

```
String.class.getName()
    returns "java.lang.String"
byte.class.getName()
    returns "byte"
(new Object[3]).getClass().getName()
    returns "[Ljava.lang.Object;"
(new int[3][4][5][6][7][8][9]).getClass().getName()
    returns "[[[[[[[I"
```

Returns:

the name of the class or interface represented by this object.

Metoda **`GetName()`** upotrebljena u prethodnom primeru koristi se za dobijanje imena entiteta (klase, interfejsa, ...)

Primena generičke klase: Gen<T>

```
class GenDemo {
    public static void main(String args[]) {
    Gen<Integer> iOb;
    iOb = new Gen<Integer>(88);
    iOb.showType();
    int v = iOb.getob();
      System.out.println("Vrednost: " + v);
      System.out.println();
    Gen<String> strOb = new Gen<String>("Testiranje Gen");
    strOb.showType();
    String str = strOb.getob();
     System.out.println("Vrednost: " + str);
 }
}
```

1. Kreiranje **Gen** reference na tip **Integers**

2. Kreiranje objekta **Gen<Integer>**

Prikaz objekta

Kreiranje **Gen** objekta tipa **String** u jednom koraku

Prikaz objekta

1. Izlaz:
Tip T je java.lang.Integer
Vrednost: 88

2. Izlaz:
Tip T je java.lang.String
Vrednost: Testiranje Gen

Generičke klase sa 2 tipa parametara

```
class TwoGen<T, V> {
 T ob1; V ob2;
  TwoGen(T o1, V o2) {
    ob1 = o1;
    ob2 = o2;
  }
   Void showTypes(){
     System.out.println("Tip T je:" + ob1.getClass().getName());
     System.out.println("Tip V je:" + ob2.getClass().getName());
  }
   T getobj1() {
      return ob1;
  }
   V getobj2() {
      return ob2;
  }
}
```

Dodati metode **showTytes()**, **getob1()** i **getob2()**

Konstruktor klase **TwoGen,** prosleđen mu je objekt tipa **T** i **V**

Povratni **tip T**

Povratni **tip V**

Generičke klase sa 2 tipa parametara

```
class SimpGen {
public static void main(String args[]) {
TwoGen<Integer, String> tgObj = new TwoGen<Integer, String>(88,"Generics");
    tgObj.showTypes();
    int v = tgObj.getob1();
    System.out.println("value: " + v);
    String str = tgObj.getob2();
 System.out.println("value: " + str);
   }
 }
```

Ograničeni tipovi (1)

- Često je veoma korisno **OGRANIČITI IZBOR TIPOVA** koji se mogu **PROSLEDITI** kao PARAMETRI TIPA.
- **OGRANIČENI TIPOVI** (engl. *Bounded types*) se koriste za ove svrhe, tako što se napravi **GORNJA GRANICA** koja **DEKLARIŠU NATKLASU** od koje **SVI** argumenti **MORAJU BITI NASLEĐENI**.
- Ključna reč **extends** se tada koristi na sledeći način:

```
<T extends natklasa>
```

- Zarad bezbednosti tipova ponekad se mogu **SPREČITI** potpuno legalni zahtevi.
- Da bi se ovo izbeglo, uvedeni su **DŽOKERSKI ARGUMENTI** ("?") koji mogu **ZAMENITI** samo **ISPRAVNE TIPOVE**:

```
boolean sameAvg (Stats<?> ob) {
                ...
}
```

? - zamenjuje bilo koji ispravni tip

Ograničeni tipovi (2)

```
class Stats<T extends Number> {
    T[] nums;
    Stats(T[] o) {
      nums = o;
    }
    double average() {
       double sum = 0.0;
       for(int i=0; i < nums.length; i++)
         sum += nums[i].doubleValue();
      return sum / nums.length;
    }//average
}//Status
```

Argument tipa za **T** mora biti **natklasa Number** ili neka njena **potklasa**

Niz tipa **Number**

Konstruktor

Tip rezultata je **double** u svim slučajevima!

Izračunavanje **SREDNJE VREDNOSTI** niza

Ograničeni tipovi (2)

java.lang

Class Number

java.lang.Object
    java.lang.Number

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

AtomicInteger, AtomicLong, BigDecimal, BigInteger, Byte, Double, Float, Integer, Long, Short

---

```
public abstract class Number
extends Object
implements Serializable
```

The abstract class `Number` is the superclass of classes `BigDecimal`, `BigInteger`, `Byte`, `Double`, `Float`, `Integer`, `Long`, and `Short`.

Subclasses of `Number` must provide methods to convert the represented numeric value to `byte`, `double`, `float`, `int`, `long`, and `short`.

Since:

JDK1.0

See Also:

`Byte`, `Double`, `Float`, `Integer`, `Long`, `Short`, Serialized Form

Constructor Summary

Constructors
Constructor and Description
`Number``()`

Generičke metode (1)

- Već je pokazano da metode unutar **GENERIČKIH KLASA** koriste **GENERIČKE PARAMETRE**.
- Može se deklarisati **GENERIČKA METODA** koja se upotrebljava sa jednim ili više **SOPSTVENIH PARAMETARA TIPA**.
- PARAMETRI TIPA se deklarišu **PRE** tipa rezultata metode.
- Konstruktori **MOGU BITI GENERIČKI** iako klasa **NIJE GENERIČKA**.
- Sintaksa svake generičke metode je:

```
<lista_parametara_tipova> tip_rezultata ime_metode(lista param.)
  {
      // telo metode
      ...
  }
```

Generička metode `isIn()`

PRIMER:
Ispitati da li je objekat **x** tipa **T** član niza **y** ?

Negeneričkа klasa

Ime metode

Parametri metode

Niz

```
class GenMethDemo {

 static <T, V extends T> boolean isIn(T x, V[] y)
{
 for(int i=0; i < y.length; i++)
    if(x.equals(y[i]))
        return true;
     return false;
}
```

Tip povratnog rezultata

Lista parametara tipova - parametar **V** je ograničen sa gornje strane parametrom **T**

Generička metoda

Primena generičke metode `isIn()`

```
public static void main(String args[]) {
// Upotreba metode sa celim brojevima
Integer nums[] = { 1, 2, 3, 4, 5 };
    if(isIn(2, nums))
     System.out.println("2 is in nums");
    if(!isIn(7, nums))
     System.out.println("7 is not in nums");
     System.out.println();
// Upotreba metode na znakovnim nizovima
String strs[] = { "one", "two", "three", "four", "five" };
  if(isIn("two", strs))
    System.out.println("two is in strs");
  if(!isIn("seven", strs))
    System.out.println("seven is not in strs");
   }
 } //class GenMethDemo
```

Primer sa celobrojnim nizon

Primer sa nizom Stringova

Generički interfejsi

- Pored **GENERIČKIH KLASA** i **METODA** mogu se formirati i **GENERIČKI INTEFEJSI**.
- Generički intefejsi se zadaju **ISTO** kao i **GENERIČKE KLASE**.
- Opšta sintaksa **GENERIČKOG INTEFEJSA** je:

```
interface ime_interfejsa <lista_parametara_tipova> {
    // ...
}
```

- **LISTA PARAMETARA TIPOVA** predstavlja listu parametara tipova odvojenih zapetama.
- Prilikom realizacije generičkog interfejsa treba zadati ARGUMENTE TIPOVA na sledeći način:

```
class ime_klase <lista_parametara_tipova>
                    implements ime_interfejsa <lista_argumenata_tipova> {
    // ...
}
```

Primer: generičkog interfejsa (1)

```
// Primer generičkog interfejsa MinMax, sa metodama min() i max()
// koje vraćaju najmanju i najveću vrednost određenog skupa objekata.
interface MinMax <T extends Comparable<T>>  {
    T min(); T max();
  }
class MyClass <T extends Comparable<T>> implements MinMax <T> {
    T[] vals;
    MyClass(T[] o) {
      vals = o;
  }
// Vrati minimum vrednosti u vals.
      public T min() {
      T v = vals[0];
   for(int i=1; i < vals.length; i++)
       if(vals[i].compareTo(v) < 0) v = vals[i];
   return v;
 }
```

Dekleracija generičkog interfejsa

Implementacija interfejsa

Realizacija metode min ()

Primer: generičkog interfejsa (2)

Realizacija metode `max ()`

```
// Vrati maximum vrednost u vals.
public T max() {
 T v = vals[0];
 for(int i=1; i < vals.length; i++)
    if(vals[i].compareTo(v) > 0) v = vals[i];
   return v;
  }
}
class GenIFDemo {
   public static void main(String args[]) {
   Integer inums[] = {3, 6, 2, 8, 6 };
   Character chs[] = {'b', 'r', 'p', 'w' };
   MyClass<Integer> iob = new MyClass<Integer>(inums);
   MyClass<Character> cob = new MyClass<Character>(chs);
   System.out.println("Max value in inums: " + iob.max());
   System.out.println("Min value in inums: " + iob.min());
   System.out.println("Max value in chs: " + cob.max());
   System.out.println("Min value in chs: " + cob.min());
 }
}
```

Primer generičkog interfejsa

```
Max value in inums: 8
Min value in inums: 2
Max value in chs: w
Min value in chs: b
```

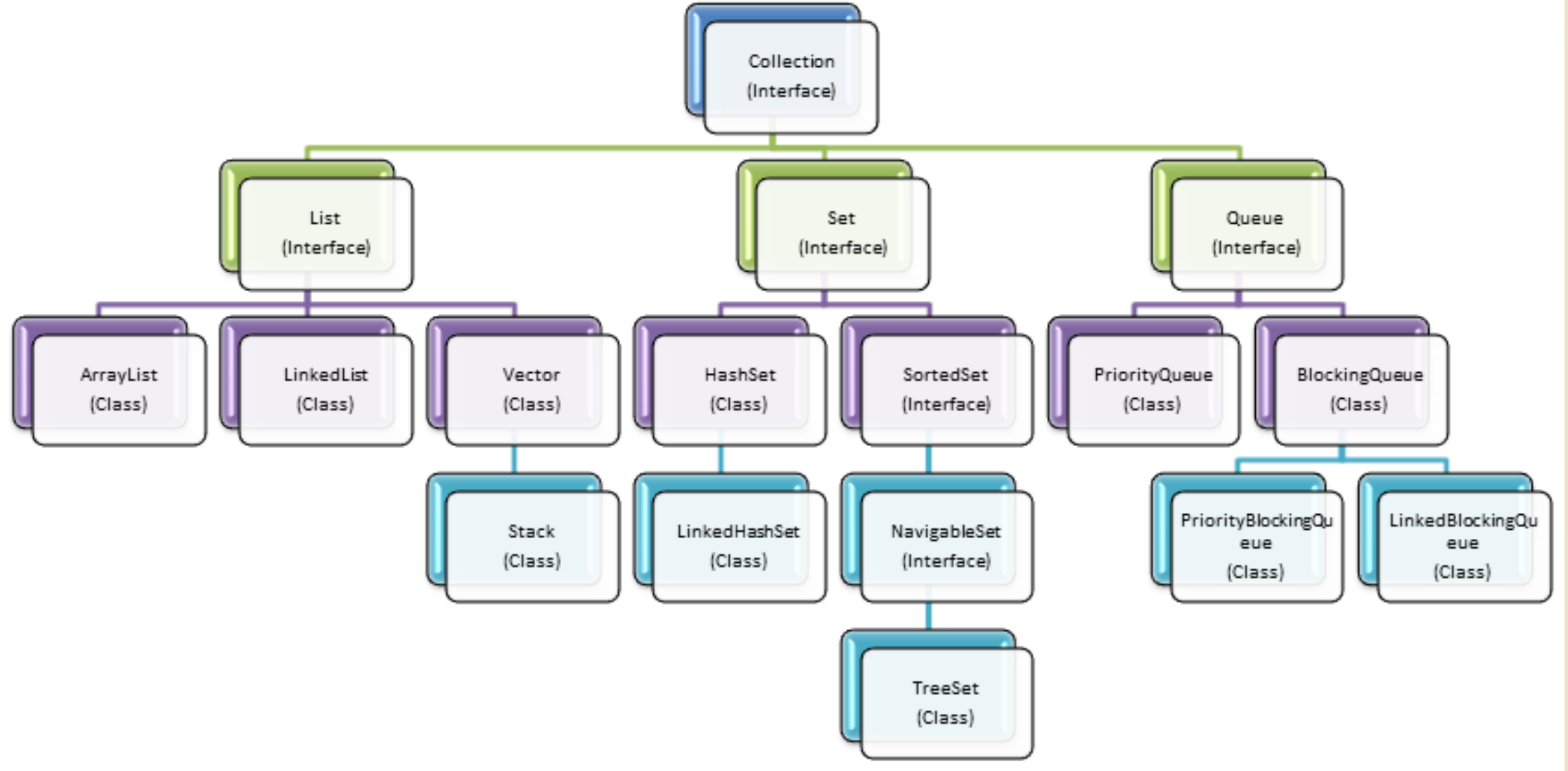
Kolekcije (1)

- **KOLEKCIJA** predstavlja **ZBIRKU OBJEKATA** u kojoj **NE POSTOJE DUPLIKATI**.
- **KOLEKCIJE** služe za organizovanje nekog **SKUPA OBJEKATA** na ODREĐENI NAČIN.
- Kolekcije zapravo služe kao **OKVIR** za **ORGANIZACIJU DRUGIH OBJEKATA**.
- U Javi se to čini **FORMIRANJEM OBJEKATA** koji služe za organizaciju **DRUGIH OBJEKATA** DEFINISANOG TIPA u **KOLEKCIJE** na odgovarajući način.
- Paket **`java.util`** poseduje sitem za **RAD SA KOLEKCIJAMA** (engl. *Collections Framework*).
- „JAVA COLLECTIONS FRAMEWORK“ je **API** koji omogućava mapiranje objekata u **GRUPU** sa kojom je **LAKO MANIPULISATI** nezavisno od implementacije.
- U Javi su realizovani sledeći **INTEFEJSI** za rad **KOLEKCIJAMA**:
  - **`Colection`**,
  - **`List`**,
  - **`Queue`**,
  - `Set` i
  - `SortedSet`.

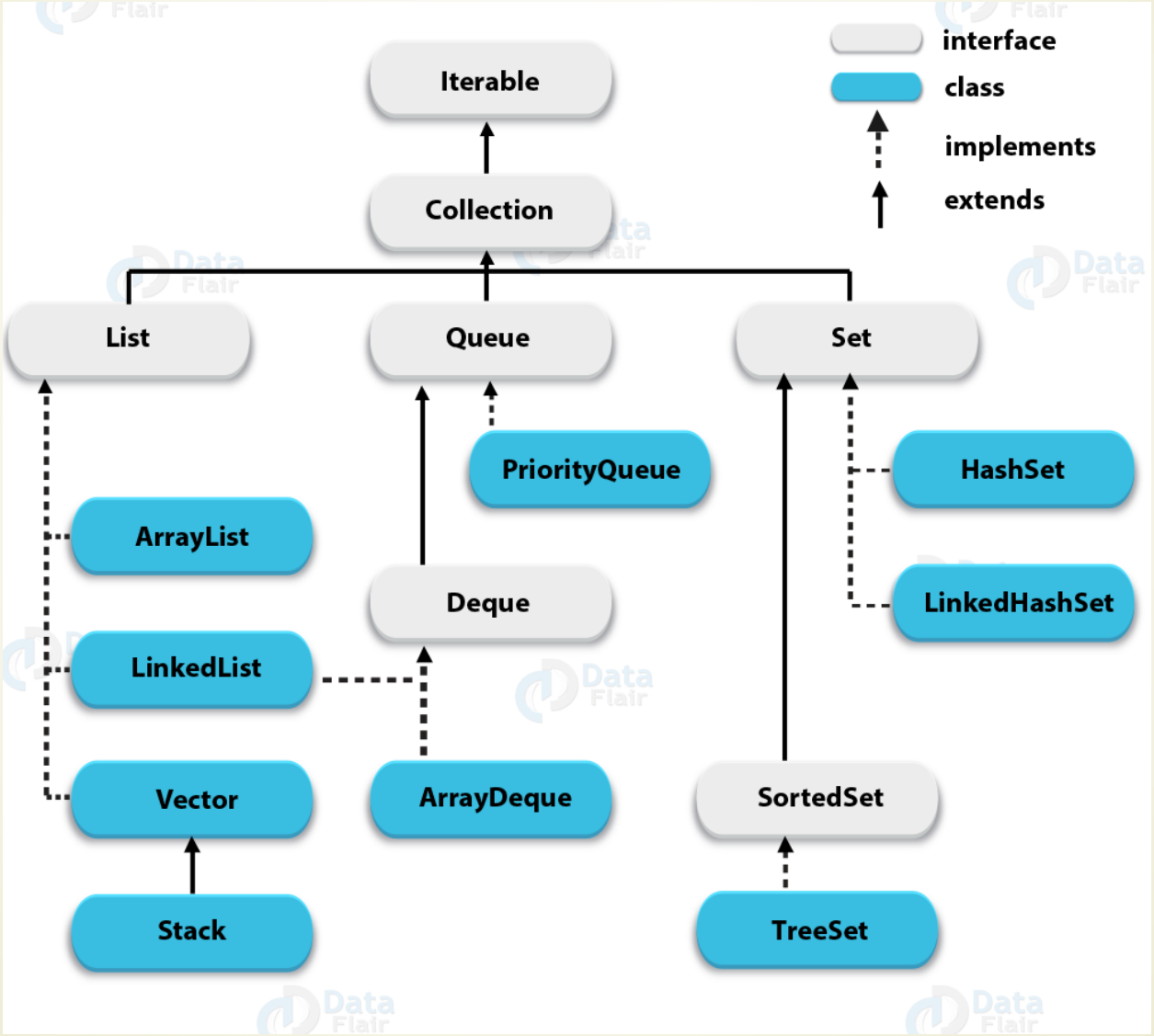
Kolekcije (2)

- „**COLLECTIONS FRAMEWORK**“ se sastoji od **HIJERARHIJE KLASA** i **INTERFEJSA**.
- U ovom radnom okruženju obrađene su **OSNOVNE KOLEKCIJE**:
  - Dinamičkih nizova;
  - Povezanih listi;
  - Stabla;
  - Hash-tabela.
- Obezbeđene su **STANDARDNE REALIZACIJE INTEFEJSA**: `LinkedList`, `HashSet` i **TreeSet**.
- Sa kolekcijama je povezan **INTERFEJS ITERATOR** koji omogućava **STANDARDIZOVAN NAČIN** za **PRISTUP** POJEDINIM ČLANOVIMA KOLEKCIJE.
- **MAPE** čivaju parove **ključ/vrednost** i mogu se posmatrati kao **KOLEKCIJE**.

Hijerarhija kolekcija i intefejsa (1)



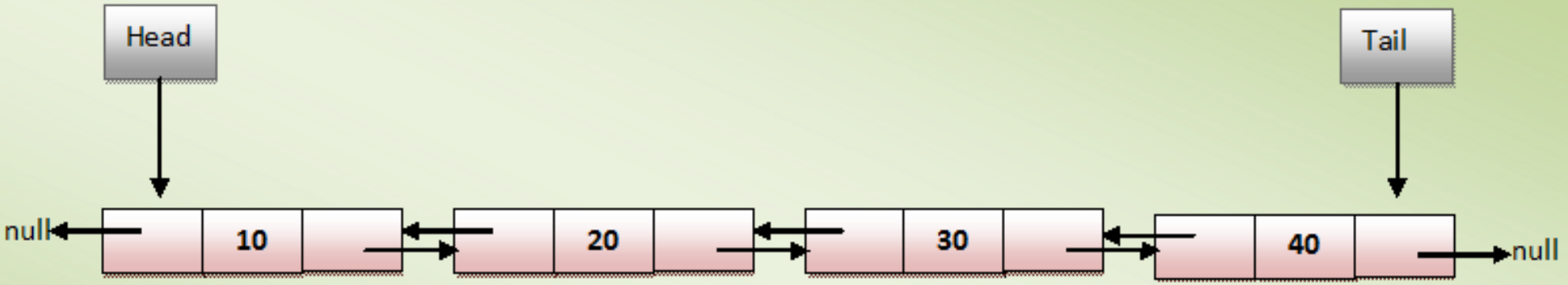
Hijerarhija kolekcija i intefejsa (2)



Interfejsi kolekcija

INTERFEJS	OPIS
`Collection`	Dozvoljava rad sa grupama objekta, nalazi se na **vrhu hijererhije** kolekcija.
`List`	Proširuje interfejs Collection za rad sa **listama objekata**.
`Queue`	Proširuje interfejs Collection za rad sa specijalnim tipovima listi čiji su elementi uklonjeni samo iz zaglavlja.
`Deque`	Proširuje Queue za rad sa double-ended queue.
`Set`	Proširuje interfejs Collection za rad sa skupovima koji moraju da sadrže **jedinstvene elemente**
`SortedSet`	Proširuje interfejs Set za rad sa **uređenim skupovima**
`NavigableSet`	Proširuje interfejs SortedSet za opsluživanje elmenata baziranih na closest-match pretraživanju.

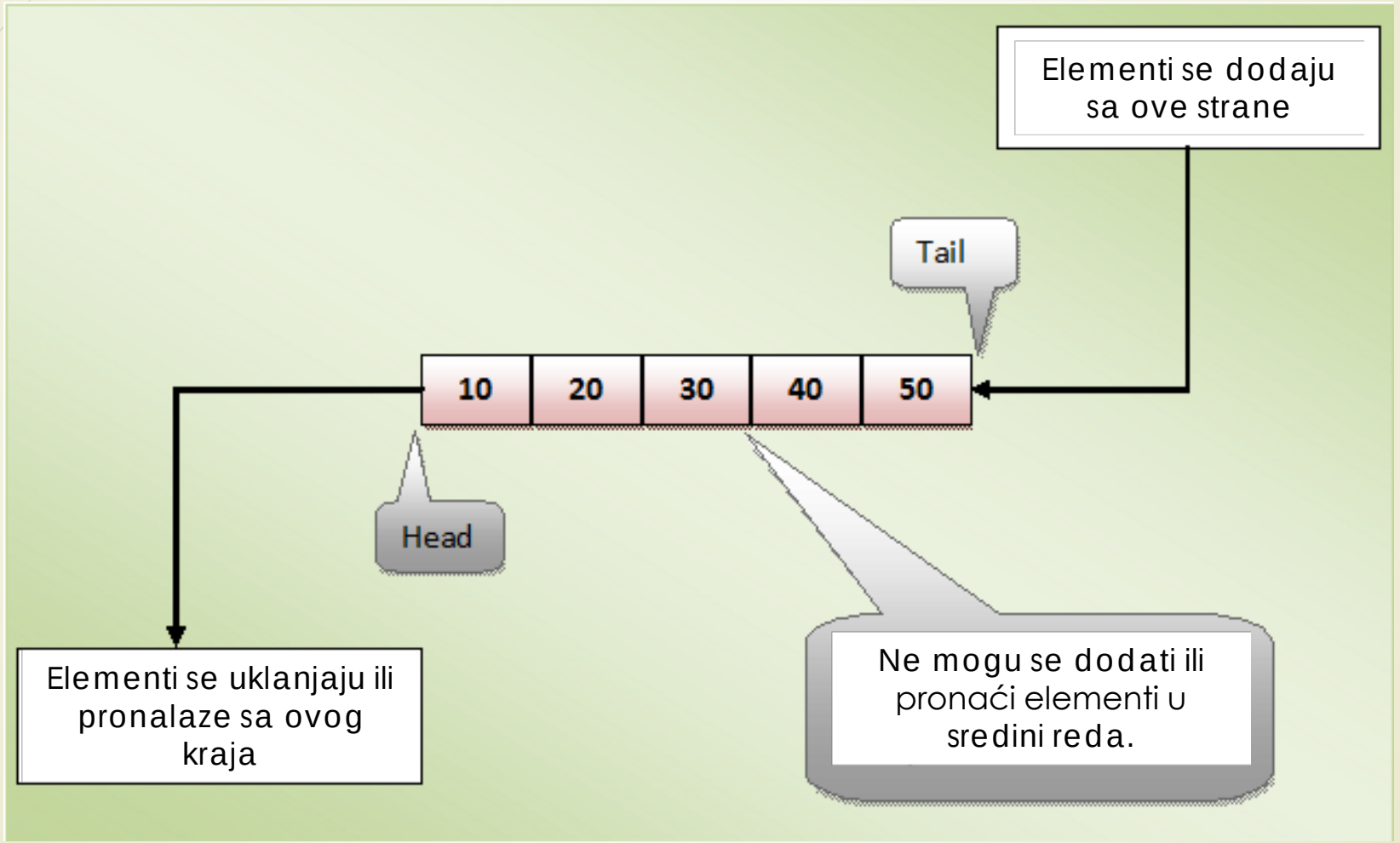
Javine kolekcije: LinkedList



Insertovanje i uklanjanje članova iz **LinkedList** kolekcije je brže u odnosu na **ArrayList** jer nije potrrbno menjati veličinu **LinkedList** – e posle svakog insertovanja ili uklanjana.

Pronalaženje u **LinkedList** - i su sporije u poređenju sa **ArrayList** – e jer je neophodmo „proputovati“ listu od početka do kraja dok se ne pronađe element.

Javine kolekcije: Queue



Queue/Red

Metode interfejsa Collection (1)

Method	Description
boolean add(E *obj*)	Adds *obj* to the invoking collection. Returns **true** if *obj* was added to the collection. Returns **false** if *obj* is already a member of the collection and the collection does not allow duplicates.
boolean addAll(Collection<? extends E> *c*)	Adds all the elements of *c* to the invoking collection. Returns **true** if the operation succeeded (i.e., the elements were added). Otherwise, returns **false**.
void clear( )	Removes all elements from the invoking collection.
boolean contains(Object *obj*)	Returns **true** if *obj* is an element of the invoking collection. Otherwise, returns **false**.
boolean containsAll(Collection<?> *c*)	Returns **true** if the invoking collection contains all elements of *c*. Otherwise, returns **false**.
boolean equals(Object *obj*)	Returns **true** if the invoking collection and *obj* are equal. Otherwise, returns **false**.
int hashCode( )	Returns the hash code for the invoking collection.
boolean isEmpty( )	Returns **true** if the invoking collection is empty. Otherwise, returns **false**.

Metode interfejsa Collection (2)

Iterator<E> iterator( )	Returns an iterator for the invoking collection.
boolean remove(Object *obj*)	Removes one instance of *obj* from the invoking collection. Returns **true** if the element was removed. Otherwise, returns **false**.
boolean removeAll(Collection<?> *c*)	Removes all elements of *c* from the invoking collection. Returns **true** if the collection changed (i.e., elements were removed). Otherwise, returns **false**.
boolean retainAll(Collection<?> *c*)	Removes all elements from the invoking collection except those in *c*. Returns **true** if the collection changed (i.e., elements were removed). Otherwise, returns **false**.
int size( )	Returns the number of elements held in the invoking collection.
Object[ ] toArray( )	Returns an array that contains all the elements stored in the invoking collection. The array elements are copies of the collection elements.
<T> T[ ] toArray(T *array*[ ])	Returns an array that contains the elements of the invoking collection. The array elements are copies of the collection elements. If the size of *array* equals the number of elements, these are returned in *array*. If the size of *array* is less than the number of elements, a new array of the necessary size is allocated and returned. If the size of *array* is greater than the number of elements, the array element following the last collection element is set to **null**. An **ArrayStoreException** is thrown if any collection element has a type that is not a subtype of *array*.

Standardne klase kolekcija

- Standardne klase koje realizuju standardne interfejse se mogu koristiti BEZ IZMENA.

Class	Description
AbstractCollection	Implements most of the **Collection** interface.
AbstractList	Extends **AbstractCollection** and implements most of the **List** interface.
AbstractQueue	Extends **AbstractCollection** and implements parts of the **Queue** interface.
AbstractSequentialList	Extends **AbstractList** for use by a collection that uses sequential rather than random access of its elements.
LinkedList	Implements a linked list by extending **AbstractSequentialList**.
ArrayList	Implements a dynamic array by extending **AbstractList**.
ArrayDeque	Implements a dynamic double-ended queue by extending **AbstractCollection** and implementing the **Deque** interface. (Added by Java SE 6.)
AbstractSet	Extends **AbstractCollection** and implements most of the **Set** interface.
EnumSet	Extends **AbstractSet** for use with **enum** elements.
HashSet	Extends **AbstractSet** for use with a hash table.
LinkedHashSet	Extends **HashSet** to allow insertion-order iterations.
PriorityQueue	Extends **AbstractQueue** to support a priority-based queue.
TreeSet	Implements a set stored in a tree. Extends **AbstractSet**.

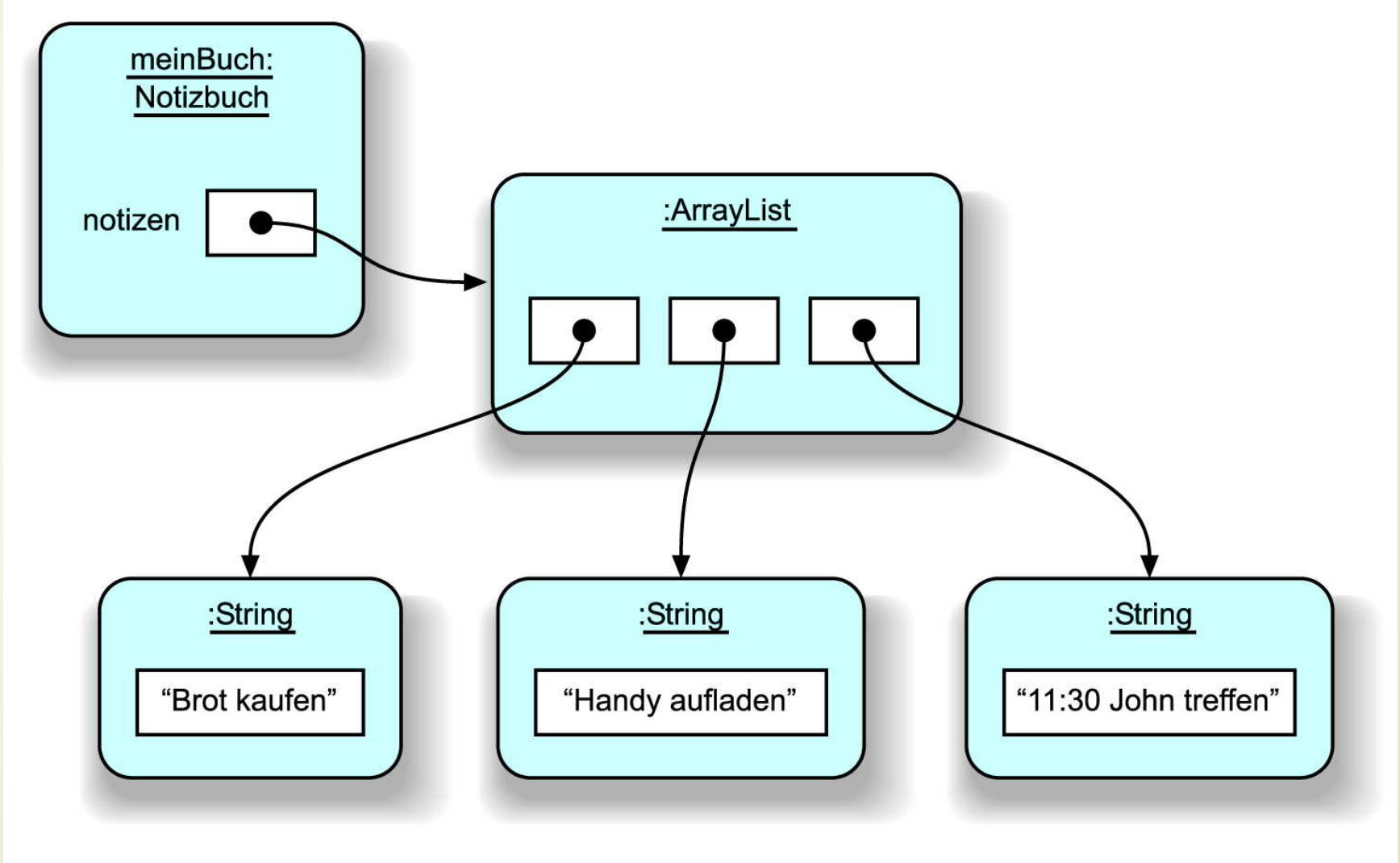
Klasa ArrayList

- Klasa **ArrayList** proširuje klasu **AbstractList** i realizuje interfejs **List**:

```
class ArrayList<E>
```

- **E** označava **TIP OBJEKTA** koje će lista sadržavati.
- Klasa **ArrayList** podržava **DINAMIČKE NIZOVE** koji po potrebi mogu da rastu (u Javi standardni nizovi su fiksne dužine).
- Kada se objekat UKLONI IZ LISTE, kolekcija se smanjuje.
- Konstruktori klase **ArrayList** su:
  - `ArrayList();`
  - `ArrayList(Collection<? Extends E>kolekcija);`
  - `ArrayList(int kapct);`
- Prvi konstruktor pravi **PRAZNU LISTU**, drugi pravi listu i **INICIJALIZUJE** elementima kolekcije, treći definiše listu sa početnim **KAPACITETOM**.

ArrayList kolekcija



Primer: klasa ArrayList (1)

```
// Demonstracija ArrayList kolekcije
import java.util.*;
class ArrayListDemo {
    public static void main(String args[]) {
// Kreiranje Array liste.
    ArrayList<String> al = new ArrayList<String>();
    System.out.println("Initial size of al: " + al.size());
// Dodavanje elemenata u Array list.
    al.add("C");
    al.add("A");
    al.add("E");
    al.add("B");
    al.add("D");
    al.add("F");
    al.add(1, "A2");
```

Primer: klasa ArrayList (2)

```
System.out.println("Size of al after additions: " + al.size());

// Prikaži array list.
    System.out.println("Contents of al: " + al);

// Ukloni elemente iz array list-e.
    al.remove("F");
    al.remove(2);
    System.out.println("Size of al after deletions: " + al.size());
    System.out.println("Contents of al: " + al);
   }
}
```

```
Initial size of al: 0
Size of al after additions: 7
Contents of al: [C, A2, A, E, B, D, F]
Size of al after deletions: 5
Contents of al: [C, A2, E, B, D]
```