Visoka tehnička škola Niš

Studijski program:

Savremene računarske tehnologije

Internet programiranje

(10)

Generički tipovi i kolekcije u Javi ver. 2.2

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

Decembar, 2018.

Generički tipovi (1)

- **GENERIČKI** (engl. generics) tipovi omogućavaju **pravljenje 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** obezbeđuju "**bezbedan rad**"sa svim tipovima podataka.

Generički tipovi (2)

- **GENERIČKI** (ili **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**, **INTERFEJSIMA** ili **METODAMA**.
- Iako se sve ovo moglo uraditi sa objektom tipa **Object**, greške mogu nastati prilikom pokušaja rada sa **NEKOMPATIBILNIM tipovima podataka**.
- Sa generičkim tipovima **SVE KONVERZIJE TIPOVA** se odvijaju **AUTOMATSKI** i time obezbeđuju **BEZBEDAN** i **LAK** rad.
- Dekleracija **GENERIČKE KLASE** ima sledeći oblik:

```
class ime_klase <lista_parametara_tipa> {

    // telo klase

}
```

Generički tipovi (2)

// 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());
    }
}
```

Generička klasa **Gen**, realizovana za tip T

Dekleracija **ob** objekta tipa T

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

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

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

Primena generičkih tipova

```
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);
 }
}
```

Kreiranje Gen reference na tip Integers

Kreiranje objekta Gen<Integer>

Prikaz objekta

Kreiranje Gen objekta tipa String

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 param.

```
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 Getob1{
    return ob1;
}
V Getob2{
    return ob2;
} }
```

Generička komponenta TwoGen sa 2 tipa PARAMETARA.

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

Dodate metode showTytes(), getob1() i getob2()!

Primena Generičke klase sa 2 tipa param.

```
class SimpGen {
public static void main(String args[]) {
TwoGen<Integer, String> tgObj =
                        new TwoGen<Integer, String>(88, "Generics");
// Prikaz tipova
   tgObj.showTypes();

// Pribavi i prikaži vrednosti
   int v = tgObj.getob1();
   System.out.println("value: " + v);
   String str = tgObj.getob2();
   System.out.println("value: " + str);
   }
}
```

```
Tip T je java.lang.Integer
Tip V je java.lang.String
```

```
value: 88
value: Generics
```

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**.
- Prilikom zadavanja parametara, može se napraviti **GORNJA GRANICA** koja deklariše **NATKLASU** od koje svi argumenti **MORAJU BITI IZVEDENI**.
- Ključna reč **extends** prilikom zadavanja parametra tipa se koristi za ove svrhe na sledeći način:

<T extends natklasa>

- Na ovaj način, **T** može zameniti **SAMO NATKLASA** ili **NJENE POTKLASE**.

Ograničeni tipovi (2)

```
class Statis<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;
   }
}
```

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

Primer: ograničeni tipovi

```
class IlustracijaGranice {
public static void main(String args[]) {
    Integer ibrojevi[] = {1, 2, 3, 4, 5};
    Statis<Integer> iob = new Statis<Integer>(ibrojevi);
    Double v = iob.average();
    System.out.println("Sr. vr. iob je " + v);
    Double dbrojevi[] = {1.1, 2.2, 3.3, 4.4, 5.5};
    Statis<Double> dob = new Statis<Double>(dbrojevi);
    Double w = dob.average();
    System.out.println("Sr. vr. dob je " + w);
  }
}
```

Ovo je OK

```
Sr. vr.  iob je 3.0
Sr. vr.  dob je 3.3
```

Ako bi se prosledio tip **String**, program se neće dobro kompajlirati jer String nije potklasa tipa **Number**!

Ograničeni tipovi (3)

- 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) {

    ...

}
```

Generičke metode (1)

- Već je pokazano da klase unutar **GENERIČKIH KLASA** koriste **GENERIČKE PARAMETRE**.
- Može se deklarisati **GENERIČKA METODA** koja upotrebljava jedan ili više **SOPSTVENIH** parametara tipa.
- Može se kreitati **GENERIČKA METODA** koja je zatvorena **UNUTAR NEGENERIČKE KLASE**.
- Parametri tipa se deklarišu **PRE** tipa rezultata metode.
- Sintaksa svake **GENERIČKE METODE** je:

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

Generičke metode (2)

```
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;
       }
```

Negenerička klasa

Ovo je satatična **GENERIČKA** metoda koja utvrđuje da li je objekat član niza.

Parametri

Ime metode

Tip rezultata

Lista parametara tipova se deklariše pre tipa rezultata.

V mora biti istog tipa kao i T ili potklasa klase T.

Ako niz sadrži objekte kompatibilne sa traženim tipom, metoda se može upotrebiti sa svim tipovima objekata i nizova

Generičke metode (3)

```
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");
    }
}
```

Parametri:
1. parametar je Integer
2. parametar je u osnovi Integer

Drugi poziv koristi String tip, te se T i V zamenjuju tipom String

Generički konstruktori

```
class GenKonstrukt {
    private double broj;
    <T extends Number>GenKonstrukt(T arg) {
    broj = arg.doubleValue();
  }
…
}
```

Metoda GenKonstrukt se MOŽE POZVATI sa SVIM numeričkim tipovima

Poziv sa Integer i Float tipovima

```
…
GenKonstrukt test_1 = new GenKonstrukt(100);
GenKonstrukt test_2 = new GenKonstrukt(123.5F);
```

Generički interfejsi

- Pored **GENERIČKIH KLASA** i metoda mogu se formirati i **GENERIČKI INTEFEJSI**.
- Detaljniji prikaz interfejsa je prikazan na prethodnom predavanju.
- 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;
    }
```

Implementacija interfejsa

Dekleracija interfejsa

Primer generičkoig interfejsa (2)

```
// 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()); } }
```

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

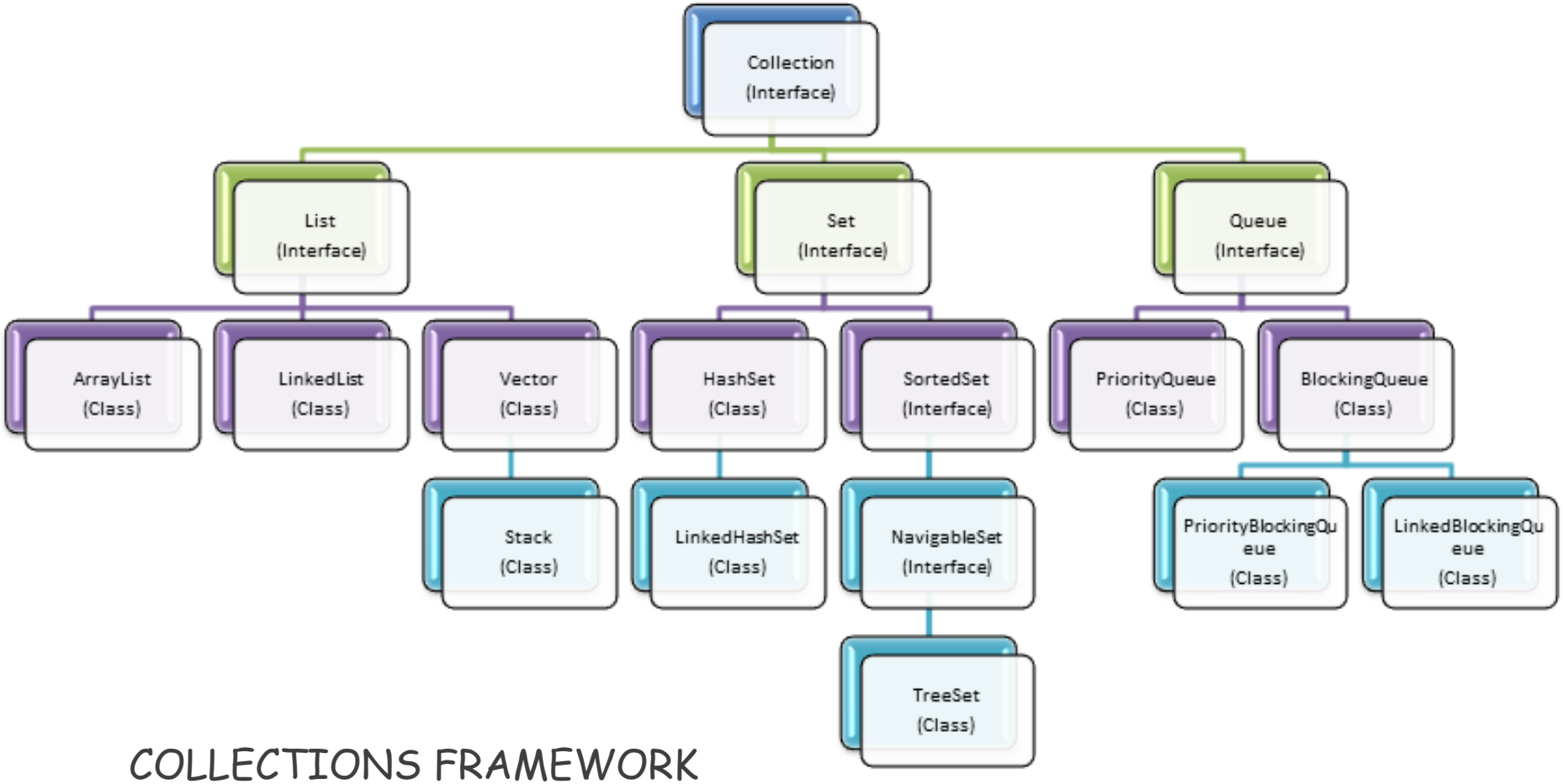
Kolekcije

- **KOLEKCIJA** – u Javi predstavlja **ZBIRKU OBJEKATA** kao što su: **dinamički nizovi**, **povezane liste**, **stabla** i **heš tabele**.
- Kolekcije u Javi koriste **JEDINSTVENI PRINCIP RADA** (definisan interefejsima) i služe za organizovanje **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.
- **INTEFEJSI za rad kolekcijama** su: **Colection**, **List**, **Queue**, **Set** i **SortedSet**.

Javine kolekcije

- Javin "**COLLECTIONS FRAMEWORK**" se sastoji od **hijerarhije KLASA** i **INTERFEJSA**.
- Obrađene su **OSNOVNE KOLEKCIJE**:
  - **Dinamičkih nizova**;
  - **Povezanih listi**;
  - **Stabla**;
  - **Hash-tabela**.
- U Javi su obezbeđene **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 klasa i interfejsa u Javi



COLLECTIONS FRAMEWORK

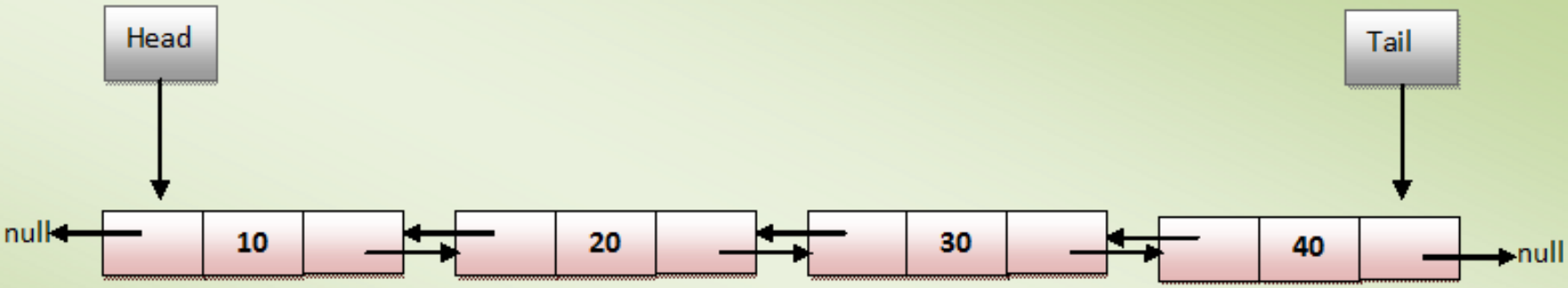
Interfejsi iz java.util

- **INTERFEJSI** koji podržavaju rad sa kolekcijama su (JSE 7.0):

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.

- U upotrebi su i sledeći interfejsi: **Comparator**, **RandomAccess**, **Iterator** i **ListIterator**.

Javine kolekcije: LinkedList

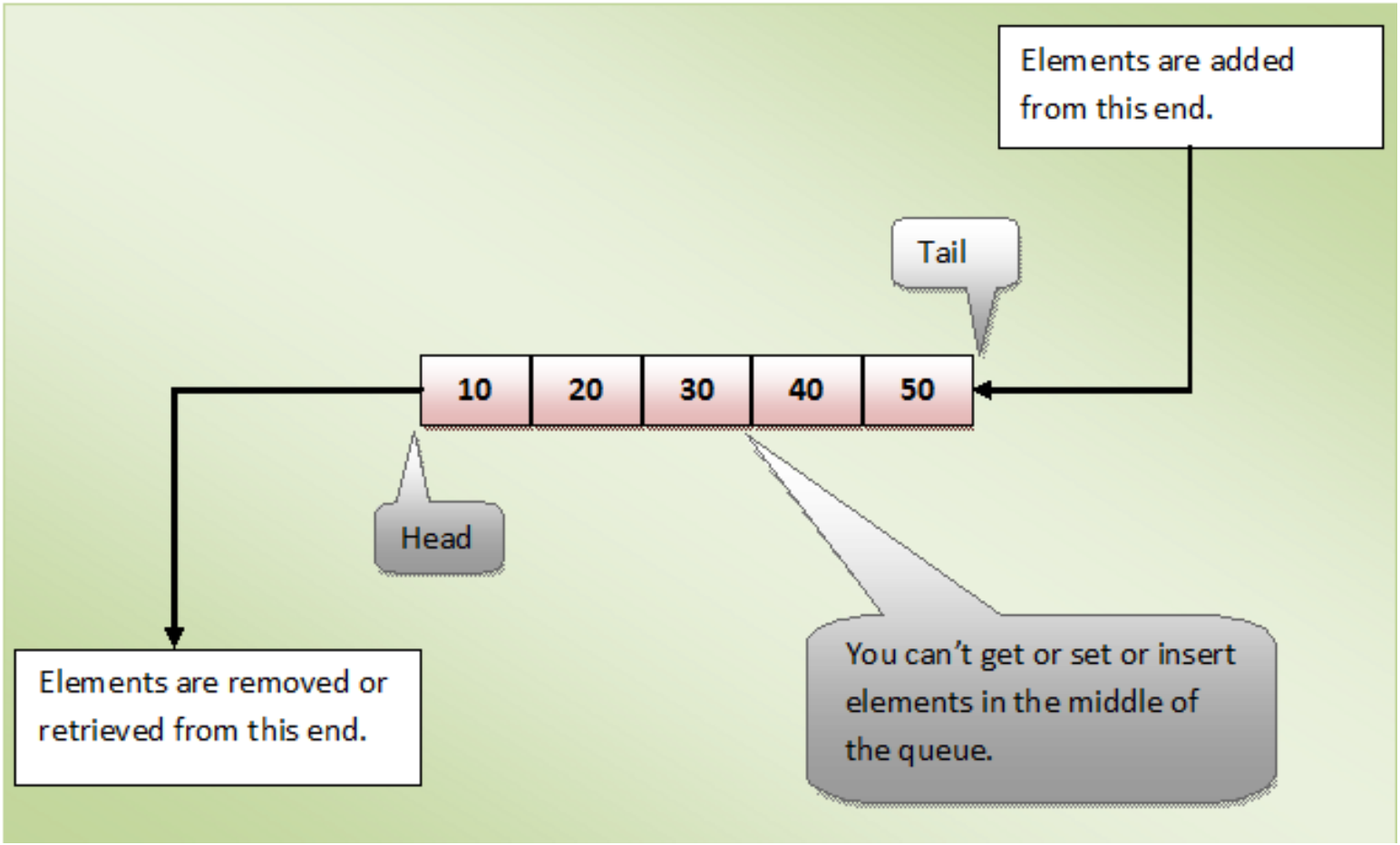


Insertions and Removals in LinkedList are faster than ArrayList. Because, You don't need to resize LinkedList after each insertions and removals.

Retrieval operations in LinkedList are slow compared to ArrayList. Because, it needs traversing from the beginning or from end to reach the element.

LinkedList/Povezane liste

Javine kolekcije: Queue



Queue/Red

Interfejs Collection

- Interfejs **Collection** mora implemetirati **SVAKA KLASA** koja definiše kolekciju na sledeći način:

interface Collection<E>

- **E** označava **tip** objekta koji će kolekcija sadržavati.
- **Collection** nasleđuje interfejs **iterable** tako da se za prolaz kroz kolekciju može koristiti **for-each** verzija for petlje.
- Interfejs **Collection** definiše **OSNOVNE METODE** koje poseduje **SVAKA KOLEKCIJA**.
- Više ovih metoda može da ispali izuzetak **UnsupportedOperationException** kada kolekcija ne može da se promeni.
- Metode koje definiše **Collection** interfejs su date u tabeli.

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 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**.

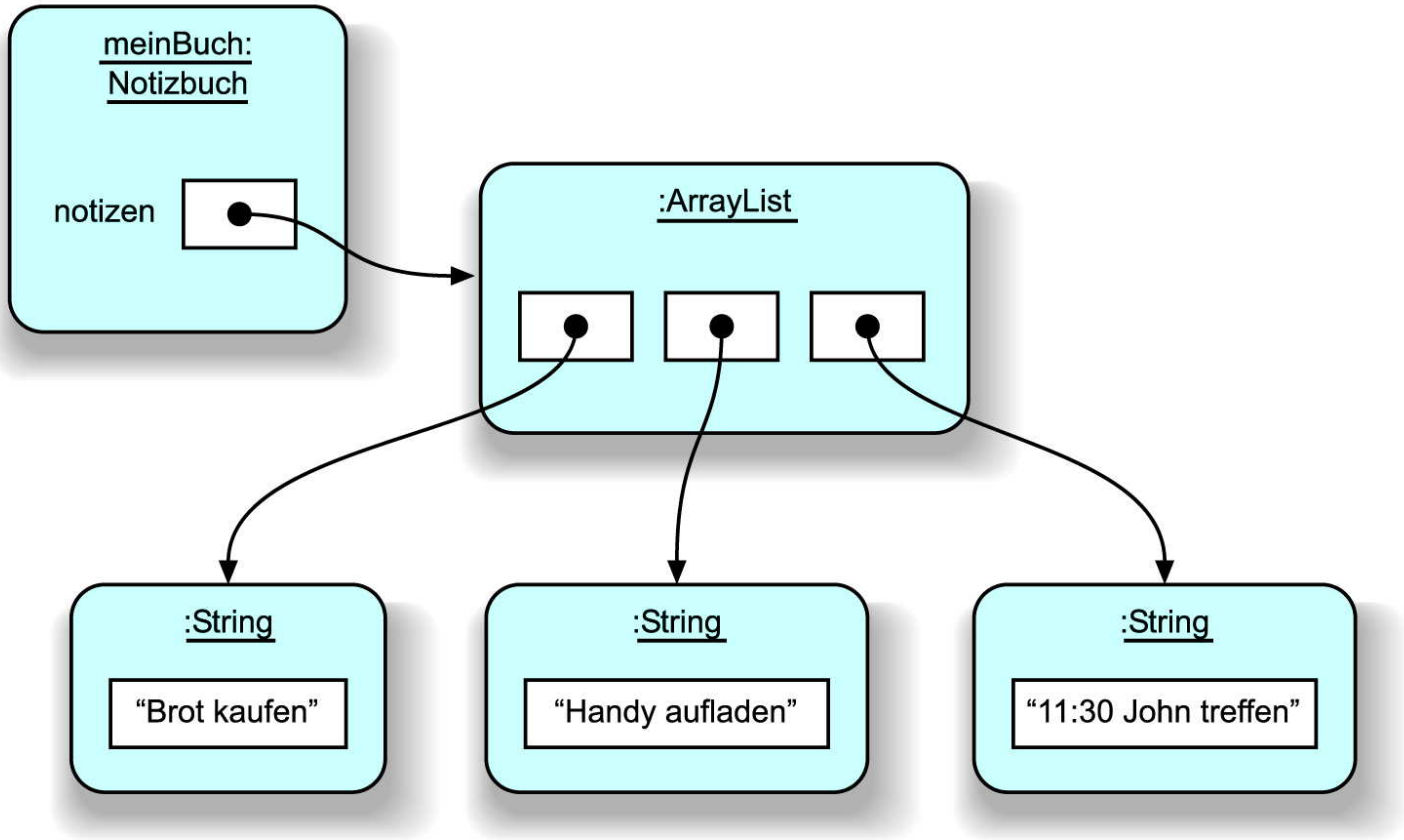
Klasa ArrayList

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

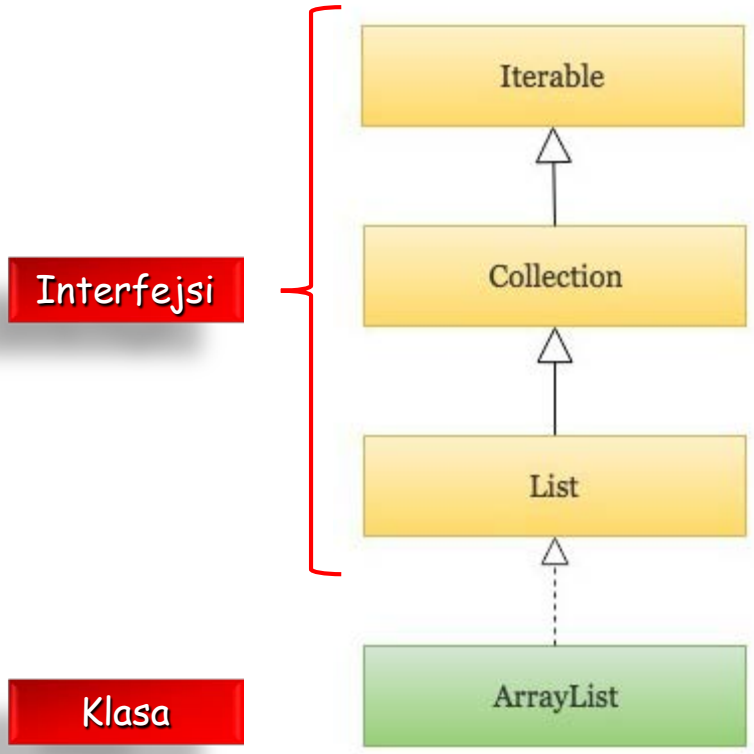
  class ArrayList<E>

- **E** označava tip objekta koje će lista sadržavati.
- Klasa **ArrayList** podržava **DINAMIČKE NIZOVE** koji po potrebi mogu da **RASTU** (već znamo da su u Javi standardni nizovi 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**.

Primer: ArayList



Hijerarhija klase ArayList



- **ArraiList** je dinamički niz, povećava svoju veličinu kako bi bili pridodati novi elementi, odnosno smanjuje veličinu kada se elementi uklonjaju.
- ArraiList interno koristi **niz** koji čuva elemente i omogućava da se preuzmu elementi po indeksu.
- Java ArraiList **dozvoljava duplirane** i null vrijednosti.
- Java ArraiList je **uređena kolekcija** - održava redosled umetanja elemenata.
- Ne može se kreirati ArraiList primitivnih tipova kao što su int, char, tako da se moraju koristiti **"boxed" tipovi** kao što su **Integer**, **Character**, **Boolean** itd.

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]
```