DIZAJN I RAZVOJ SOFTVERA

Nizovi i liste

Nizovi

- Niz predstavlja skup entiteta određenog tipa. Promjenljive se nazivaju elementima niza.
- U Javi, niz je objekat, pa se stoga smatra referencijskim tipom promjenljive.
- Elementi niza mogu biti primitivnog ili referencijskog tipa.
- Elementi niza mogu biti i nizovi.
- Za pristupanje pojedinačnim elementima niza, potrebno je navesti ime niza i poziciju elementa (indeks) u uglastim zagradama. Indeksiranje počinje od 0. Na primjer, ako imamo niz a od 10 elemenata, pojedinačnim elementima niza pristupamo sa `a[0]`, `a[1]`, ..., `a[9]`.
- Indeks niza mora biti vrijednost tipa `int` ili tipa koji se može unaprijediti u `int` (`byte`, `short` ili `char`). U suprotnom, dolazi do greške.

Deklarisanje i kreiranje nizova

- Nizovi se kreiraju, kao i ostali objekti, pomoću ključne riječi `new`. Pri kreiranju niza, u kombinaciji sa ključnom riječju `new`, potrebno je navesti ime, tip i broj elemenata niza, na primjer:

```
int[] x = new int[10];
int x[] = new int[10];
```

- Nakon ovakvog kreiranja niza, elementi imaju podrazumijevane vrijednosti, a to su 0 za primitivne numeričke tipove, `false` za `boolean` tip i `null` za reference.
- Prethodno kreiranje niza se moglo razbiti na dva koraka:

```
int[] x;   // ili int x[];
x = new int[10];
```

gdje prva naredba predstavlja deklaraciju reference na niz, dok druga predstavlja kreiranje objekta niza i dodjelu reference na taj objekat deklarisanoj promjenljivoj `x`.

Deklarisanje i kreiranje nizova

- Više nizova se može kreirati u jednom koraku. Na primjer, dva niza stringova, `a` i `b`, se mogu kreirati kao:

  ```
  String[] a = new String[10], b = new String[10];
  ```

- Kada se uglaste zagrade navedu odmah nakon tipa, sve promjenljive u deklaraciji će biti nizovi, kao u prethodnoj naredbi.
- Ukoliko ne želimo da u jednoj naredbi budu sve nizovi, onda se uglaste zagrade navode samo nakon imena promjenljivih koje su nizovi. Na primjer, u deklaraciji:

  ```
  String a[], b, c;
  ```

  će promjenljiva `a` biti niz stringova, dok će `b` i `c` biti stringovi.

- Nizovi mogu biti bilo kog tipa. Elementi niza primitivnog tipa sadrže primitivne vrijednosti, dok su elementi niza referencijskog tipa reference na objekte tog tipa. Na primjer, u prethodnoj deklaraciji, svaki element niza `a` će biti referenca na `String` objekat.

Inicijalizacija i dužina nizova

- Nizovi se u Javi mogu kreirati i inicijalizovati u jednom koraku:

```
int x[] = {1,3,4,11,27};
String s[] = {"prva","druga","treća"};
```

- Pri ovakvom kreiranju nizova ne navodi se ključna riječ `new`.
- Dužina niza se određuje na osnovu elemenata u vitičastim zagradama. Interno, kompajler prvo prebroji elemente u zagradi, pa onda izvrši operaciju `new`.
- Svaki niz ima dužinu, koja se čuva u promjenljivoj instance `length`. Dakle, dužina prethodna dva niza se dobija sa `x.length` i `s.length`.
- U Javi je dužina niza fiksna. Jednom kreiran niz na ovaj način ne može promijeniti dužinu, a `length` podatak mu je deklarisan ključnom riječju `final`. Promjena dužine niza podrazumijeva kreiranje novog niza.
- Ovaj se nedostatak može eliminisati korišćenjem kolekcija, kakva je `ArrayList`, koja dozvoljava dinamičku promjenu dužine niza.

Primjer – Pristupanje nepostojećem članu

- Tokom izvršavanja programa, provjeravaju se indeksi elemenata niza koji moraju biti veći ili jednaki od 0 i manji od dužine niza. Pokušaj da se indeksira element van ovih granica dovodi do greške izvršavanja (eng. *run-time error*) poznate kao `ArrayIndexOutOfBoundsException`.

```
public class Nizovi {

  public static void main(String[] args) {
    int x[] = {1,3,4,11,27};
    int suma = 0;

    for(int i = 0; i <= x.length; i++)
      suma += x[i];

    System.out.printf("Suma elemenata niza je %d.",suma);
  }
}
```

Ispis

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at Nizovi.main(Nizovi.java:9)
```

Malo o obradi izuzetaka

- Iz ispisa programa zaključujemo da je došlo do izuzetka (eng. *exception*) prilikom izvršavanja.
- Izuzetak predstavlja indikaciju da je došlo do neke greške pri izvršavanju programa.
- Dobro napisan program uzima u obzir mogućnost da može doći do izuzetka i sadrži odgovarajući kod koji obrađuje izuzetke.
- Upravljanje izuzecima (eng. *exception handling*) omogućava da se kreiraju programi koji ”hvataju” i razrješavaju izuzetke. U mnogim slučajevima, ovo će podrazumijevati neometan nastavak izvršavanja programa, kao da se ništa nije dogodilo.
- Kad dođe do nekog ozbiljnijeg problema, razrješavanje izuzetka može podrazumijevati i završetak rada aplikacije, uz određenu poruku korisniku. Kad dođe do greške prilikom izvršavanja, kaže se još da JVM ”baca” (eng. *throws*) izuzetak.
- Posmatrajmo primjer prethodne klase u kome ćemo obraditi izuzetak nepostojećeg indeksa niza.

Nepostojeći član – obrađen

```
public class Nizovi {
  public static void main(String[] args) {
    int x[] = {1,3,4,11,27};
    int suma = 0;

    for(int i = 0; i <= x.length; i++) {
      try{
        suma += x[i];
      }
      catch (ArrayIndexOutOfBoundsException e) {
        System.out.printf("Došlo je do greške %s\n", e);
        System.out.printf("Indeks je %d, a mora biti manji od %d.", i, x.length);
      }
    }

    System.out.printf("\nSuma elemenata niza je %d.",suma);
  }
}
```

```
Došlo je do greške java.lang.ArrayIndexOutOfBoundsException: 5
Indeks je 5, a mora biti manji od 5.
Suma elemenata niza je 46.
```

Ispis

try – catch

- Da bi se obradio izuzetak, dio koda koji može baciti izuzetak treba staviti u `try` blok.
- Sa druge strane, `catch` blok sadrži naredbe koje obrađuju izuzetak u slučaju da do njega dođe.
- Ako se naredbe u `try` bloku izvrše regularno, u `catch` blok se neće ni ulaziti.
- U slučaju da `try` blok sadrži više naredbi, prva naredba koja baci izuzetak uzrokuje da kontrola toka preskoči sve preostale naredbe u `try` bloku i pređe na izvršavanje `catch` bloka.
- Zagrade za `try` i `catch` blok su obavezne.
- Izuzeci imaju tipove. Uočimo da su u `catch` bloku u prethodnom primjeru deklarisani tip izuzetka, `ArrayIndexOutOfBoundsException`, i parametar izuzetka `e`.
- Tip izuzetka ukazuje na to do kakve greške je došlo. `catch` blok može da obradi izuzetke određenog tipa. Unutar `catch` bloka možemo koristiti parametar izuzetka `e` da interagujemo sa objektom izuzetka.

Generisanje slučajnih brojeva

- Za generisanje slučajnih brojeva u Javi, može nam poslužiti klasa `Random`, koja se nalazi u paketu `java.util`.
- `Random` objekat može dati slučajne vrijednosti tipa `boolean`, `byte`, `int`, `long`, `float`, `double`, kao i vrijednosti koje podliježu Gausovoj raspodjeli (poznate i kao normalne slučajne promjenljive).
- Slučajni brojevi se još mogu generisati pomoću statičke metode `random` iz klase `Math`, ali ona daje samo `double` slučajne brojeve iz opsega [0,1).
- Slučajni `int` broj se može kreirati pomoću objekta klase `Random` sa:

```
Random objRandom = new Random();
int slucBroj = objRandom.nextInt();
```

- Metoda `nextInt()` generiše slučajne cijele brojeve iz opsega tipa `int`, a to je od –2,147,483,648 do 2,147,483,647.
- Za sjeme (eng. *seed*) generatora se, u slučaju metoda klase `Random`, koristi trenutno vrijeme, tako da se svaki put kad se pozove neka metoda, recimo `nextInt`, generiše druga sekvenca brojeva.

Generisanje slučajnih brojeva

- Preklopljena `nextInt` metoda koja sadrži jedan `int` parametar vraća slučajne nenegativne brojeve manje od vrijednosti parametra. Na primjer,

  ```
  objRandom.nextInt(5);
  ```

  vraća slučajan cio broj iz skupa {0,1,2,3,4}.

- Slučajan cio broj iz skupa {A, A+1, A+2, …, B} se može dobiti sa:

  ```
  A + objRandom.nextInt(B-A+1);
  ```

  dok se slučajan cio broj iz skupa {A, A+K, A+2K, …, A+CK} dobija sa:

  ```
  A + K*objRandom.nextInt(C+1);
  ```

- Pri kreiranju `Random` objekta, konstruktoru možemo proslijediti argument tipa `int` koji predstavlja sjeme generatora, kao u

  ```
  Random objRandom = new Random(10);
  ```

- Takav `Random` objekat će pri svakom izvršavanju aplikacije kreirati istu pseudoslučajnu sekvencu. Ovo može biti korisno pri testiranju aplikacije.

Generisanje slučajnih brojeva - Primjer

```
import java.util.Random;
public class SlucajniBrojevi {
  public static void main(String[] args) {
    Random objRandom1 = new Random();
    Random objRandom2 = new Random(10);

    System.out.print("Konstruktor bez argumenta: ");
    for(int i = 1; i < 10; i++)
      System.out.printf("%2d ", objRandom1.nextInt(20));

    System.out.print("\nKonstruktor sa argumentom: ");
    for(int i = 1; i < 10; i++)
      System.out.printf("%2d ", objRandom2.nextInt(20));
  }
}
```

```
Konstruktor bez argumenta: 18  4 14  4  6 18  8  4  3
Konstruktor sa argumentom: 13  0 13 10  6 16 17  8  1

Konstruktor bez argumenta: 16 11  4 10  1  0  9 14 15
Konstruktor sa argumentom: 13  0 13 10  6 16 17  8  1
```

Prvo izvršenje

Drugo izvršenje

Primjer – Miješanje i dijeljenje špila karata

Klasa `Karta` ima dva člana podatka, `rang` ili lice karte (As, 2, 3, ..., Pub, Dama, Kralj) i `znak` (Pik, Tref, Karo, Herc), jedan konstruktor i metodu `toString` koja predstavlja `String` reprezentaciju karte.

```
public class Karta {

  private String rang;   // rang karte - "As","2",...,"Dama","Kralj"
  private String znak;   // znak karte - "Pik","Tref","Karo","Herc"

  public Karta(String r, String z){
    rang = r;
    znak = z;
  }

  public String toString(){
    return rang + " " + znak;
  }

}
```

Klasa SpilKarata

```
import java.util.Random;
public class SpilKarata {
  private Karta spil[];   // niz objekata Karta
  private int tekKarta;   // broj karte koja se trenutno dijeli (0 do 51)
  public static final int BROJ_KARATA = 52; // broj karata u špilu (konstanta)
  private static final Random slucBroj = new Random(); // generator sluč. brojeva

  public SpilKarata(){
    String rangovi[] = {"As","2","3","4","5","6","7",
        "8","9","10","Pub","Dama","Kralj"};
    String znakovi[] = {"Pik","Tref","Karo","Herc"};

    spil = new Karta[BROJ_KARATA]; // kreiranje novog špila
    tekKarta = 0;

    for(int i = 0; i < BROJ_KARATA; i++)
      spil[i] = new Karta(rangovi[i % 13], znakovi[i / 13]);
  }
  public void mijesaj(){ } // Sljedeći slajd
  public Karta dijeli(){ } // Sljedeći slajd
}
```

Klasa SpilKarata

```
public void mijesaj(){
  tekKarta = 0;

  for(int i = 0; i < BROJ_KARATA; i++){
    // generisanje slučajne pozicije u nizu Spil
    int j = slucBroj.nextInt(BROJ_KARATA);

    // zamjena i-te i j-te karte koristeći kartu privremena
    Karta privremena = spil[i];
    spil[i] = spil[j];
    spil[j] = privremena;
  }
}

public Karta dijeli(){
  if(tekKarta < BROJ_KARATA)
    return spil[tekKarta++];
  else
    return null;
}
```

Klasa SpilKarataTest

```
public class SpilKarataTest {

  public static void main(String[] args) {
    SpilKarata spil = new SpilKarata();

    spil.mijesaj();

    for(int i = 0; i < SpilKarata.BROJ_KARATA; i++){
      System.out.printf("%-12s", spil.dijeli());
      if((i+1) % 6 == 0)
        System.out.println();
    }
  }

}
```

Statički način pristupa podatku klase

```
3 Karo      2 Tref      Dama Pik    Pub Tref    7 Karo      As Karo
3 Herc      Pub Pik     2 Karo      5 Pik       6 Tref      Dama Herc
10 Karo     9 Pik       10 Tref     9 Tref      5 Tref      2 Pik
Kralj Tref  8 Herc      3 Pik       9 Karo      6 Karo      Kralj Karo
Kralj Pik   Pub Karo    4 Pik       Pub Herc    3 Tref      4 Karo
4 Tref      2 Herc      6 Herc      10 Herc     4 Herc      7 Tref
Dama Tref   5 Karo      6 Pik       9 Herc      5 Herc      As Tref
8 Tref      Kralj Herc  8 Pik       10 Pik      As Herc     7 Herc
As Pik      7 Pik       Dama Karo   8 Karo
```

Ispis

Unaprijeđena for petlja

- Java sadrži verziju `for` petlje koja omogućava da se prođe kroz bilo koju kolekciju podataka, uključujući niz, bez korišćenja brojačke promjenljive.
- Ova `for` petlja se naziva unaprijeđena `for` petlja (eng. *enhanced for loop*) a u drugim jezicima, npr. C#, Objective-C, Perl, PHP, Python, Ruby, Smalltalk, Visual Basic .NET, VBA, se naziva `for-each` petlja.
- Sintaksa ove petlje je:

```
for(parametar: imeNiza)
     tijelo petlje
```

gdje `parametar` ima `tip` i `ime`, kao i svaka druga deklarisana promjenljiva, a `imeNiza` predstavlja ime niza, ili kolekcije, kroz koju se prolazi.

- Tip parametra mora biti konzistentan sa tipom elemenata niza.
- Kod unaprijeđene `for` petlje se ne može indeksirati nepostojeći element.
- Unaprijeđena `for` petlja ne može da promijeni vrijednost elemenata niza, već samo može da ga pročita.

Unaprijeđena for petlja – Primjer

```
public class UnaprijedjenaForPetlja {

  public static void main(String[] args) {
    double[] niz = {3.1, 4.5, 7.12, 4.55, 2.1, 6, 11.8, 4.13, 8.61};
    double sredVrijednost = 0.0;

    for(double x: niz) {
      sredVrijednost += x;
      x *= 2;
    }

    sredVrijednost /= niz.length;
    System.out.printf("Srednja vrijednost niza je %f\n",sredVrijednost);
    System.out.print("Niz je: ");
    for(double x: niz)
      System.out.printf("%.2f ",x);
  }

}
```

```
Srednja vrijednost niza je 5.767778
Niz je: 3.10 4.50 7.12 4.55 2.10 6.00 11.80 4.13 8.61
```

Ispis

Prosljeđivanje niza metodi

- Metode koje za parametre imaju nizove se deklarišu na sljedeći način:

```
tip imeMetode(tip[] imeNiza)
```

- Kada se poziva metoda koja za parametar ima niz, samo se ime niza prosleđuje kao argument. Ne mora se prosljeđivati dužina niza, jer je dužina svakog niza upisana u njegovom `length` podatku.
- Prilikom prosljeđivanja niza metodi, metoda dobija kopiju njegove reference, pa se proslijeđeni niz može mijenjati u okviru metode.
- Sa druge strane, ako se prosljeđuju pojedinačni elementi niza primitivnog tipa, metoda dobija kopiju vrijednosti elementa, pa se elementi ne mogu mijenjati u okviru metode. Ovo je ilustrovano na sljedećem slajdu.

Prosljeđivanje niza metodi – Primjer

```
public class IzmjenaNiza {

  public static void main(String[] args) {
    int[] niz = {3, 5, 2, 4, 2, 7};
    izmijeniNiz(niz);
    System.out.println("Niz nakon izmjene: ");
    for(int x: niz)
      System.out.printf("%d ", x);
    izmijeniElement(niz[0]);
    System.out.printf("\nPrvi element niza je %d", niz[0]);
  }

  public static void izmijeniNiz(int[] y) {
    for(int i = 0; i < y.length; i++)
      y[i] *= 2;
  }

  public static void izmijeniElement(int z) {
      z *= 3;
  }
}
```

Ispis

```
Niz nakon izmjene:
6 10 4 8 4 14
Prvi element niza je 6
```

Prosljeđivanje argumenata metodi

- U programiranju postoje dva načina prosljeđivanja argumenata metodi, **prosljeđivanje po vrijednosti** (eng. *call-by-value*) i **prosljeđivanje po referenci** (eng. *call-by-reference*).
- Kod prosljeđivanja po vrijednosti, kopija vrijednosti argumenta se prosljeđuje metodi. Metoda radi isključivo sa kopijom i svaka modifikacija kopije ne utiče na originalnu vrijednost.
- Kad je argument proslijeđen po referenci, metoda može pristupiti originalnom podatku u memoriji i izmijeniti ga.
- Prosljeđivanjem po referenci, dajemo metodi adresu podatka u memoriji, i ne postoji nikakva zabrana da metoda izmijeni podatak na toj adresi.
- U slučaju da je modifikacija zaista ono što nam treba, prosljeđivanje po referenci može značajno poboljšati performanse aplikacije jer se eliminiše potreba da se kopiraju moguće velike količine podataka.

Prosljeđivanje argumenata metodi

- Java ne dozvoljava prosljeđivanje argumenata po referenci, tj. svi argumenti su proslijeđeni po vrijednosti.
- Dva tipa vrijednosti se mogu proslijediti metodama – kopije primitivnih vrijednosti (npr. `int`, `long`, `double`) i kopije referenci na objekte.
- Objekte ne možemo prosljeđivati metodama.
- Iz prethodnog primjera smo vidjeli da metoda ne može da promijeni originalnu vrijednost primitivnog podatka čija je kopija proslijeđena metodi. Slično važi i za reference. Kada se kopija reference proslijedi metodi, metoda može da promijeni vrijednost parametra u koji je upisana vrijednost reference tako da on pokazuje na drugi objekat. Ipak, izlaskom iz metode, izlazi se iz opsega tog parametra, tj. on biva dealociran. Originalna vrijednost reference ostaje nepromijenjena.
- Iako ne može da promijeni originalnu vrijednost reference, metoda može da promijeni vrijednost objekta na koji pokazuje ta referenca. Upisivanjem vrijednosti reference u parametar metode, originalna referenca i parametar pokazuju na isti objekat u memoriji. Metoda može da manipuliše objektima preko reference upisane u parametru metode.

Višedimenzioni nizovi

- Java ne podržava višedimenzione nizove direktno, ali podržava rad sa nizovima čiji su elementi nizovi, čime se postiže isti efekat.
- Najčešće korišćeni su dvodimenzioni nizovi (matrice), kod kojih su elementi grupisani u vrste (redove) i kolone. Za pristup pojedinačnim elementima matrice, koriste se dva indeksa, indeks vrste i indeks kolone.
- Kao i jednodimenzioni nizovi, i dvodimenzioni se mogu kreirati ključnom riječju `new` i pomoću inicijalizatora u deklaraciji. Na primjer, dvodimenzioni niz `a` sa dvije vrste i tri kolone se može kreirati na sljedeće načine:

```
int a[][] = new int[2][3];
int a[][] = {{1,2,3},{4,5,6}};
```

- Pošto su višedimenzioni nizovi u Javi zapravo nizovi nizova, pojedinačne vrste mogu imati različit broj elemenata. Na primjer, deklaracijom

```
int a[][] = {{1},{2,3},{4,5,6}};
```

  se kreira niz sa tri vrste, prva sa jednim, druga sa dva i treća sa tri elementa.

- Zapravo, `a` je niz od tri elementa, a svaki element niza je referenca na jednodimenzioni niz `int` promjenljivih.

Višedimenzioni nizovi

- Drugi način kreiranja niza sa nejednakim dužinama vrsta je dat sa

```
int a[][] = new int[3][];
a[0] = new int[1];
a[1] = new int[2];
a[2] = new int[3];
```

Primjer:

```
public class VisedimNizovi {

  public static void main(String[] args) {
    int a[][] = {{8},{5,3},{4,1,2}};

    for(int i=0; i < a.length; i++) {
      for(int j=0; j < a[i].length; j++)
        System.out.printf("%d ", a[i][j]);
      System.out.println();
    }
  }

}
```

```
8
5 3
4 1 2
```

Ispis

Klasa Arrays

- Klasa `Arrays` (paket `java.util`) sadrži veliki broj metoda za manipulaciju nizovima. Tu imamo:
  - metodu **`sort`** za sortiranje nizova, pri čemu postoje preklopljene verzije ove metode za sortiranje dijela niza. Na primjer, ako imamo niz `a`, sortiranje čitavog niza i prvih 5 njegovih elemenata bi se izvršilo sa `Arrays.sort(a)` i `Arrays.sort(a,0,5)`.
  - metodu **`binarySearch`** za pretragu niza. Predmetni niz mora biti sortiran. U suprotnom, metoda vraća nedefinisane rezultate. Metoda vraća indeks traženog elementa, ako on postoji u nizu, i vrijednost (**– tačkaUmetanja – 1**) u suprotnom. Tačka umetanja je tačka gdje bi se traženi element ubacio tako da ne naruši sortiranost niza. Dodatno oduzimanje 1 obezbeđuje da metoda uvijek vraća negativnu vrijednost ako traženi element ne postoji u nizu, dok nenegativna vrijednost ukazuje da je pronađen traženi element. Na ovaj način, vraćena vrijednost se može iskoristiti za eventualno umetanje elementa u sortirani niz. Postoji i preklopljena verzija metode koja radi na određenom podintervalu niza. Na primjer, traženje broja 3 u sortiranom nizu a se vrši sa `Arrays.binarySearch(a,3)`.
  - metodu **`equals`** za poređenje nizova. Ova metoda vraća `true` ako su poređeni nizovi isti i `false` u suprotnom. Metoda se poziva kao `Arrays.equals(a,b)`, gde su `a` i `b` reference na nizove.
  - metodu **`fill`** za upisivanje vrijednosti u elemente niza. Vrijednost možemo upisati u čitav niz, ili samo određene elemente niza. Na primjer, sa `Arrays.fill(a,4)` upisujemo broj 4 u sve elemente niza `a`, dok sa `Arrays.fill(a,0,5,4)` upisujemo broj 4 u prvih pet elemenata niza.

Metoda arraycopy

- Vrlo korisna može biti i metoda `arraycopy` iz klase `System` (paket `java.lang`) kojom se mogu kopirati nizovi. Opšti oblik ove metode je:

```
System.arraycopy(a, aPoc, b, bPoc, brElem)
```

gde je:

- `a` niz iz kog se kopira,
- `b` niz u koji se kopira,
- `aPoc` pozicija od koje počinje kopiranje iz niza `a`,
- `bPoc` pozicija od koje počinje upisivanje u niz `b`, i
- `brElem` je broj elemenata koji se kopiraju.

Osnovno o kolekcijama. Klasa ArrayList

- Kolekcije predstavljaju strukture podataka u kojima se čuvaju srodni objekti.
- Kolekcije kao klase pružaju efikasne metode za rad sa podacima, bez potrebe da se zna na koji način su podaci smješteni u klasama.
- Problem fiksne dužine niza se može riješiti koristeći kolekcionu klasu `ArrayList`. Klasa `ArrayList` predstavlja niz promjenljive dužine koji sadrži reference objekata.
- `ArrayList` kolekcija se deklariše na sljedeći način:

```
ArrayList <T> ime;
```

gdje `T` predstavlja tip promjenljive. Na primjer, naredbama

```
ArrayList <Integer> kolInt;
ArrayList <String> kolString;
```

se deklarišu kolekcije cijelih brojeva i stringova, respektivno.

- Važna činjenica vezana za kolekcije, a samim tim i za `ArrayList` kolekciju, je da one mogu skladištiti samo reference, a ne i vrijednosti primitivnih tipova.

Osnovno o pakovanju primitivnih tipova

- U tom smislu, iako naredba

  ```
  ArrayList <Integer> kolInt;
  ```

  deklariše kolekciju `int` tipova, ovaj tip je prethodno **spakovan** (eng. *wrapped*) u omotač njegovog tipa.
- Na ovaj način, sa primitivnim vrijednostima možemo raditi kao sa objektima.
- Svaki primitivni tip u Javi ima odgovarajuću **omotačku klasu** (eng. *type-wrapper class*) u paketu `java.lang`.
- Iz prethodne naredbe vidimo da klasa `Integer` odgovara prostom tipu `int`.
- **Pakovanje** i **raspakivanje** (eng. *boxing* i *unboxing*), kao operacije konvertovanja primitivnog tipa u odgovarajući objekat i obrnuto, se u Javi vrši automatski. O ovome će biti više riječi kada se budu detaljnije radile kolekcije.
- Klase koje u deklaraciji imaju `<T>`, pri čemu se umjesto `T` može koristiti bilo koji neprimitivni tip se nazivaju **generičkim klasama** (eng. *generic classes*), i o njima će biti više riječi u nastavku.

Neke metode i osobine klase ArrayList

Metoda	Opis
`add`	Dodavanje elementa na kraj kolekcije ili na određenu poziciju.
`clear`	Brisanje kolekcije (uklanjanje svih elemenata).
`contains`	Vraća true ako kolekcija sadrži traženi element i false u suprotnom.
`get`	Vraća element sa specificiranim indeksom.
`set`	Postavlja vrijednost elementu sa specificiranim indeksom.
`indexOf`	Vraća indeks prve pojave specificiranog elementa kolekcije.
`remove`	Uklanja prvu pojavu specificirane vrijednosti ili elementa sa specificiranim indeksom.
`size`	Vraća broj elemenata smještenih u ArrayList kolekciji.
`toArray`	Konvertovanje kolekcije u niz. Ovo se radi da bi se određene operacije sa nizom izvele efikasnije, kao i da bi se konvertovani niz mogao proslijediti metodama koje ne rade sa kolekcijama ili obraditi naslijeđenim starim kodom koji ne poznaje kolekcije.
`trimToSize`	Smanjenje kapaciteta kolekcije na trenutni broj elemenata.

Klasa ArrayList – Primjer

```
import java.util.ArrayList;

public class PrimjerArrayList {

  public static void main(String[] args) {
    ArrayList< String > kolString = new ArrayList< String >();

    System.out.printf("Početna veličina: %d\n", kolString.size());

    kolString.add("Prvi");
    kolString.add("Drugi");
    kolString.add(0,"Treći");
    kolString.add(1,"Četvrti");

    System.out.printf("Veličina nakon dodavanja: %d\n", kolString.size());
    System.out.printf("Kolekcija nakon dodavanja: ");
    for(String s: kolString)
      System.out.printf("%s ", s);

    String[] nizString = new String[kolString.size()];
    kolString.toArray(nizString);
    System.out.printf("\nNiz nakon kopiranja kolekcije u niz: ");
    for(String s: nizString)
      System.out.printf("%s ", s);

    // Nastavak na narednom slajdu...
```

```
    System.out.printf("\nU kolekciji %s elementa %s",
        kolString.contains("Treći") ? "ima" : "nema", "Treći");
    System.out.printf("\nU kolekciji %s elementa %s\n",
        kolString.contains("Peti") ? "ima" : "nema", "Peti");

    kolString.remove("Treći");
    System.out.printf("Kolekcija nakon uklanjanja elementa Treći: ");
    for(String s: kolString)
      System.out.printf("%s ", s);
    kolString.remove(0);
    System.out.printf("\nKolekcija nakon uklanjanja elementa sa indeksom 0: ");
    for(String s: kolString)
      System.out.printf("%s ", s);

    kolString.clear();
    System.out.printf("\nVeličina nakon brisanja kolekcije: %d", kolString.size());
  }

}
```

```
Početna veličina: 0
Veličina nakon dodavanja: 4
Kolekcija nakon dodavanja: Treći Četvrti Prvi Drugi
Niz nakon kopiranja kolekcije u niz: Treći Četvrti Prvi Drugi
U kolekciji ima elementa Treći
U kolekciji nema elementa Peti
Kolekcija nakon uklanjanja elementa Treći: Četvrti Prvi Drugi
Kolekcija nakon uklanjanja elementa sa indeksom 0: Prvi Drugi
Veličina nakon brisanja kolekcije: 0
```

Ispis

Kapacitet ArrayList kolekcije

- Prvom naredbom

```
ArrayList< String > kolString = new ArrayList< String >();
```

kreira se prazan `ArrayList` koji ima podrazumijevani početni kapacitet od 10 elemenata.

- **Kapacitet** nam kazuje koliko elemenata može sadržati `ArrayList` bez uvećanja. U pozadini se `ArrayList` implementira koristeći niz.
- Prilikom uvećanja `ArrayList` kolekcije, kreira se veći niz od postojećeg i svaki element starog niza se kopira u novokreirani niz. Ovo može oduzeti značajno vrijeme u slučaju velikih kolekcija i čestog dodavanja elemenata.
- U tom smislu, povećanje dužine `ArrayList` kolekcije svaki put kad se dodaje novi element može biti neefikasno.
- Umjesto toga, lista se implementira tako da raste samo kad se dodaje novi element i kada je tekući broj elemenata liste jednak kapacitetu liste, tj. kad nema prostora da se smjesti novi element.

Klasa ArrayList – Primjer

- Kapacitet `ArrayList` kolekcije možemo eksplicitno zadati metodom `ensureCapacity`, što se radi u slučaju kada očekujemo da kolekcija sadrži veliki broj elemenata, čime se eliminiše potreba za čestom promjenom kapaciteta.
- Smanjenje kapaciteta kolekcije na tekući broj elemenata se vrši metodom `trimToSize`.
- `ArrayList` kolekcija ima preklopljeni konstruktor koji omogućava definisanje kapaciteta kolekcije prilikom njenog kreiranja. Na primjer, naredbom

  ```
  ArrayList< String > kolString = new ArrayList< String >(50);
  ```

  se kreira kolekcija čiji je kapacitet 50 elemenata.
- Unaprijeđena `for` petlja predstavlja vrlo elegantno rješenje za prolazak kroz sve elemente kolekcije, bilo da je to `ArrayList` ili neka druga kolekcija.
- Drugi način da se obiđu svi elementi kolekcije, kao i da se manipuliše njima, je pomoću klase `Iterator`, o čemu će više riječi biti u nastavku.