4 Nizovi, enumeracije i liste

Niz predstavlja skup promenljivih istog tipa. Promenljive se nazivaju elementima niza. U Javi, niz je objekat, pa se stoga smatra referencijskim tipom promenljive. 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 unutar uglastih zagrada. Pozicija elementa se još naziva i indeks. Indeksiranje počinje od 0. Na primer, ako imamo niz a od 10 elemenata, pojedinačnim elementima niza pristupamo sa `a[0]`, `a[1]`, ..., `a[9]`. Indeks poslednjeg elementa je za jedan manji od dužine niza. Dakle, za sad je sve potpuno isto kao u C i C++.

Što se tiče indeksa niza, to mora biti vrednost tipa `int` ili tipa koji se može unaprediti u `int` (`byte`, `short` ili `char`). U suprotnom, dolazi do greške.

4.1 Deklarisanje i kreiranje nizova

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

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

Nakon ovakvog kreiranja niza, elementi imaju podrazumevane vrednosti, a to su 0 za primitivne numeričke tipove, `false` za tip `boolean` i `null` za reference. Prethodno kreiranje niza se moglo izvršiti u dva koraka na sledeći način:

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

gde prva naredba predstavlja deklaraciju referencijske promenljive za niz, što je označeno uglastim zagradama nakon ključne reči `int`, dok druga predstavlja kreiranje objekta niza i dodelu reference na taj objekat promenljivoj `x`.

Više nizova se može kreirati u jednom koraku. Na primer, 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 promenljive u deklaraciji će biti nizovi, kao u prethodnoj naredbi. Ako ne želimo da u jednoj deklaraciji budu sve nizovi, navodimo uglaste zagrade samo nakon imena promenljivih koje su nizovi. U deklaraciji

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

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

Možemo deklarisati nizove bilo kog tipa. Elementi niza primitivnog tipa sadrže primitivne vrednosti, dok su elementi niza referencijskog tipa reference na objekte tog tipa. U prethodnoj deklaraciji, svaki element niza a će biti referenca na `String` objekat.

4.2 Inicijalizacija i dužina nizova

Nizovi se u Javi mogu kreirati i inicijalizovati u jednom koraku na sledeći način:

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

Pri ovakvom kreiranju nizova, ne navodi se ključna reč `new`. Dužina niza se određuje na osnovu elemenata u vitičastim zagradama. Interno, kompajler prvo prebroji elemente unutar zagrada, pa onda izvrši operaciju `new`.

Ključna reč `new` se može koristiti pri inicijalizaciji na sledeći način:

```
int x[] = new int[]{1, 3, 4, 11, 27};
```

Svaki niz ima dužinu, koja se čuva u promenljivoj instance `length`. Dužina prethodna dva niza se onda dobija sa `x.length` i `s.length`. Specifičnost Jave je da je dužina niza fiksna. Jednom kreiran niz na ovaj način ne može promeniti dužinu, jer mu je `length` podatak deklarisan ključnom rečju `final`. Promena dužine niza podrazumeva kreiranje novog niza. Ovaj se nedostatak može eliminisati korišćenjem kolekcija, kakva je `ArrayList`, koja dozvoljava dinamičku promenu broja elemenata. O ovome će biti reči u poglavlju 4.11.

Dužina niza u Javi je fiksna. Promena dužine niza podrazumeva kreiranje novog niza.

4.3 Pristupanje nepostojećem elementu niza. Osnovno o obradi izuzetaka

Tokom izvršavanja programa, proveravaju se indeksi elemenata niza koji moraju biti veći od ili jednaki 0 i manji od dužine niza. Svaki pokušaj da se pristupi elementu niza čiji je indeks van ovih granica dovodi do greške izvršavanja (eng. *run-time error*) poznate kao `ArrayIndexOutOfBoundsException`. Posmatrajmo klasu `NizBezObradeIzuzetka` u čijoj metodi `main` se kreira niz i određuje suma elemenata tog niza. Namerno ćemo pogrešiti i staviti da poslednji element kom se pristupa ima indeks jednak dužini niza.

Pozivanje konstruktora sa argumentom nam daje tu funkcionalnost. Ovo je ilustrovano sledećim programom, ispod koga se nalaze ispisi za dva uzastopna izvršenja.

```
import java.util.Random;

public class SlucajniBrojevi {

  // Ilustracija rada sa klasom Random

  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
```

4.5 Primer sa špilom karata

Ovde ćemo kreirati aplikaciju koja simulira mešanje i deljenje špila karata. U tu svrhu, kreiraćemo tri klase: `Karta`, `SpilKarata` i `SpilKarataTest`. Ove klase su date u nastavku, a nakon njih je dat ispis za jedno pozivanje klase `SpilKarataTest`.

```
public class Karta {

  // Klasa koja predstavlja jednu kartu

  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;
```

```
  }
}
-------------------------------------------------------------------------

import java.util.Random;

public class SpilKarata {

  // Klasa koja predstavlja špil karata

  private Karta Spil[];     // niz objekata Karta
  public static final int BROJ_KARATA = 52;      // broj karata u špilu
  private int tekKarta;     // tekuća karta (0 do BROJ_KARATA-1)
  private static Random slucBroj = new Random(); // gener. 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]);
  }

  // metoda mesaj meša (nasumično raspoređuje) karte
  public void mesaj() {
    tekKarta = 0;

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

      // zamena i-te i j-te karte koristeci kartu privremena
      Karta privremena = Spil[i];
      Spil[i] = Spil[j];
      Spil[j] = privremena;
    }
  }

  // metoda deli koja deli karte (vraća tekuću kartu)
  public Karta deli() {
    if(tekKarta < BROJ_KARATA)
      return Spil[tekKarta++];
    else
      return null;
  }
}
```

```
-------------------------------------------------------------------------
```

```
public class SpilKarataTest {

  // Klasa koja testira rad sa špilom karata

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

    spil.mesaj();

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

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

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`.

Klasa `SpilKarata` ima četiri člana podatka:

- niz `Karta` objekata `Spil`,
- `static int` konstantu `BROJ_KARATA`,
- `int` promenljivu `tekKarta` koja predstavlja redni broj karte koja se deli (od 0 do `BROJ_KARATA-1`) i
- `static Random` promenljivu `slucBroj`, koja predstavlja generator pseudo-slučajnih brojeva.

Podaci `BROJ_KARATA` i `slucBroj` su deklarisani kao statički jer ta dva podatka mogu da dele svi objekti klase `SpilKarata`. Svi špilovi imaju isti broj karata, i svi mogu da koriste isti generator slučajnih brojeva.

U konstruktoru `SpilKarata` kreiramo dva niza stringova `rangovi` i `znakovi`, čije elemente prosleđujemo konstruktoru klase `Karta` u petlji

```
  for(int i = 0; i < BROJ_KARATA; i++)
      Spil[i] = new Karta(rangovi[i % 13], znakovi[i / 13]);
```

Niz `rangovi` ima 13 elemenata, čije indekse (0 do 12) možemo dobiti izrazom `i % 13`, dok niz `znakovi` ima 4 elemenata, čije indekse možemo dobiti izrazom `i / 13`. Svakom broju `0 ≤ i < 52` odgovara jedinstvena kombinacija (`i % 13, i / 13`). Takođe, konstruktor inicijalizuje promenljivu `tekKarta` iako to nismo morali uraditi. Podaci klase se inicijalizuju na podrazumevane vrednosti, a ta vrednost je 0 za numeričke primitivne tipove.

Metoda `mesaj` meša špil karata. Mešanje karta se simulira slučajnim razmeštanjem referenci u okviru niza `Spil`. Ovo se postiže `for` petljom u metodi u kojoj se tekuća referenca (pozicija `i`) menja sa slučajno odabranom referencom (pozicija `j`). Način generisanja slučajne pozicije je ranije objašnjen. U svrhu zamene pozicije dve reference, uveli smo pomoćnu promenljivu `privremena`. Takođe, u metodi `mesaj` se inicijalizuje promenljiva `tekKarta`.

Metoda `deli` vraća tekuću kartu, tj. onu sa pozicijom `tekKarta`. U slučaju da su sve karte podeljene, metoda vraća `null` referencu.

Klasa `SpilKarataTest` testira rad sa špilom karata. U njenoj metodi `main` se kreira novi špil karata, meša pozivom metode `mesaj` i deli pozivom metode `deli`. Metoda `deli` vraća samo jednu kartu, pa je potrebno pozvati N puta za deljenje N karata, što je u klasi `SpilKarataTest` postignuto pomoću `for` petlje, koja deli čitav špil. U jednom redu štampamo šest karata, pa prelazimo u novi red. Obratite pažnju da ovde nismo vršili proveru vraćene vrednosti metode `deli`.

4.6 Unapređena `for` petlja

Unapređenom `for` petljom (eng. *enhanced for loop*) ćemo zvati verziju `for` petlje koja ne koristi brojačku promenljivu da obiđe niz, odnosno kolekciju podataka, čime se sprečava mogućnost da se indeksira nepostojeći element. Sintaksa ove petlje je:

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

gde `parametar` ima tip i ime, kao i svaka druga deklarisana promenljiva, a `imeNiza` predstavlja ime niza, ili kolekcije, koja se obilazi. Tip parametra mora biti konzistentan sa tipom elemenata niza.

Unapređena `for` petlja se u nekim jezicima (npr. C#, Objective-C, Perl, PHP, Python, Ruby, Smalltalk, Visual Basic .NET, VBA) naziva `for-each` petlja.

Na primeru klase `UnapredjenaForPetlja` ilustrovaćemo rad sa ovom petljom.

```
public class UnapredjenaForPetlja {

  // Ilustracija rada sa unapređenom for petljom

  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};
```

4.11 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 smešteni u kolekciji. Ranije smo videli da su nizovi u Javi fiksne dužine, tj. da im se ne može menjati jednom postavljena dužina tokom izvršenja programa. Ovo dalje znači da se unapred mora znati dužina niza. Međutim, dužinu ponekad ne znamo sve do trenutka izvršavanja programa. Ovaj se problem može rešiti koristeći kolekcionu klasu `ArrayList`. Klasa `ArrayList` predstavlja niz promenljive dužine koji sadrži reference objekata.

`ArrayList` kolekcija se deklariše na sledeći način:

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

gde T predstavlja tip promenljive. Na primer, naredbama

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

se deklarišu kolekcije celih brojeva i stringova, respektivno. Važna činjenica vezana za kolekcije, a samim tim i za klasu `ArrayList`, je da one mogu skladištiti samo reference, a ne i vrednosti primitivnih tipova. U tom smislu, iako prva naredba deklariše kolekciju `int` tipova, ovaj tip je prethodno *spakovan* (eng. *wrapped*) u omotač njegovog tipa. Na ovaj način, sa primitivnim vrednostima možemo raditi kao sa objektima. Svaki primitivni tip u Javi ima odgovarajuću *omotačku klasu* (eng. *type-wrapper class*) u paketu `java.lang`. Iz prve naredbe vidimo da primitivnom tipu `int` odgovara omotačka klasa `Integer`. *Pakovanje* i *raspakivanje* (eng. *boxing* i *unboxing*), operacije konvertovanja primitivnog tipa u odgovarajući objekat i obrnuto, se u Javi vrši automatski. Više reči o ovome u glavi 10.1.

Kolekcije mogu sadržati samo referencijske tipove. Primitivne tipove je, pre smeštanja u kolekciju, potrebno spakovati u odgovarajuću omotačku klasu (npr. tipu `int` odgovara klasa `Integer`), što se vrši automatski.

Klase koje u deklaraciji imaju <T>, pri čemu se umesto T može koristiti bilo koji neprimitivni tip, se nazivaju *generičkim klasama* (eng. *generic classes*).

Neke osnovne operacije koje pruža kolekcija `ArrayList` su date u tabeli 4.2.

Metoda	Opis
`add`	Dodaje element na kraj kolekcije.
`clear`	Briše kolekciju (uklanja sve elemente).
`contains`	Vraća `true` ako kolekcija sadrži traženi element i `false` u suprotnom.
`get`	Vraća element sa specificiranim indeksom.
`indexOf`	Vraća indeks prve pojave specificiranog elementa kolekcije, odnosno -1 ako element ne postoji.