Matematički fakultet, Univerzizet u Beogradu
Katedra za računarstvo i informatiku

Objektno orijentisano programiranje

vežbe
školska 2016/2017

Biljana Stojanović
Nemanja Mićović
Nikola Milev

1 Klasa java.util.Scanner

Klasa kojom se implementira jednostavni čitač (skener) koji izdvaja (parsira) primitivne tipove i stringove koristeći regularne izraze.
Ulaz se deli na tokene, a kao delimiter se podrazumevano koriste beline (' ', '\n', '\t', …).
Dobijeni tokeni se mogu konvertovati u vrednosti različitih tipova koristeći razne *next...* metode:

Metodi next() i hasNext*()*

next()* i *hasNext*()* metodi postoje za većinu primitivnih tipova (int, double, long, boolean,...).
Metodi *next*()* preskaču karaktere sa ulaza koji odgovaraju delimiterima i pokušavaju da vrate naredni token koji odgovara primitivnom tipu.

```
public boolean hasNext();
```
vraća true ako postoji naredni token na ulazu

```
public String next();
```
pronalazi i vraća ceo naredni token; ”ceo” token je okružen delimiterima (i ispred i iza njega su delimiteri)

```
public boolean hasNextLine();
```
vraća true ako postoji naredna linija na ulazu

```
public String nextLine();
```
vraća ostatak tekuće linije isključujući oznaku za kraj reda sa njenog kraja

Skup metoda *next*()* i *hasNext*()* gde se umesto * nalazi (*Int*, *Double*, *Float*, *Short*, *Long*, *Byte*, *Boolean*) - jasno je na osnovu imena šta rade.

Dokumentacija

https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

Primeri:
Čitanje celog broja sa standardnog ulaza (*System.in*):

```
Scanner sc = new Scanner(System.in);
int ceo = sc.nextInt();
```

Može se čitati i iz stringa:

```
Scanner sc = new Scanner("1 8 java jdk");
System.out.println(sc.nextInt()); // 1
System.out.println(sc.nextInt()); // 8
System.out.println(sc.next()); // java
System.out.println(sc.next()); // jdk
```

Osim belina, mogu se koristiti i drugi delimiteri koji se postavljaju metodom *useDelimiter():*

```
Scanner sc = new Scanner("1, 2, 3, 4, 5, 6, 7, 8, 9, 10").useDelimiter(",");
while(sc.hasNextInt()) {
  int broj = sc.nextInt();
  if (broj % 2 == 0) System.out.println(broj);
}
```

Neodgovarajući token

Kada neki od metoda *next*()* izbaci izuzetak tipa *InputMismatchException*, ne prosleđuje se token koji je izazvao izuzetak. Token se može dobiti ili preskočiti primenom nekog drugog metoda.
Napomena: Izbacivanje izuzetka tipa *IOException* predstavlja kraj ulaza.

Metodom close() zatvara se skener. Kada se skener zatvori, obično se zatvara i ulazni tok.

Najčešće korišćeni konstruktori

```
public Scanner(InputStream source)
public Scanner(InputStream source, String charsetName)
```

Prave skener koji čita (izdvaja tokene) iz ulaznog toka, uključujući i standardni ulazni tok (*System.in).* Argument *charsetName* drugog konstruktora predstavlja ime karakterskog skupa na osnovu kojeg se sekvenca bajtova sa ulaza prevodi u sekvencu karaktera.

```
public Scanner(Path source)
public Scanner(Path source, String charsetName)
```

Prave skener koji čita iz datoteke. Argument *charsetName* ima prethodno opisano značenje.

```
public Scanner(String source)
```

Pravi skener koji čita iz stringa.

Čitanje iz datoteke će biti obrađeno na nekom od narednih časova.

Čitanje karaktera

Za čitanje karaktera **ne** postoji odgovarajući metod *nextChar()*, kao što je to slučaj sa ostalim primitivnim tipovima. Izdvajanje jednog karaktera vrši se naredbom:

```
char c = sc.next().charAt(0);
```

Najpre se metodom next() izdvaja sledeći token kao objekat klase String, a potom samo prvi karakter iz stringa.

2 Nizovi u Javi

Nizovi u Javi su objekti, dinamički se prave i mogu biti dodeljeni promenljivoj tipa *Object*. Iz tog razloga, svi metodi klase *Object* mogu biti primenjeni na niz.

Deklaracija niza

```
tip_elemenata[] a; // uobicajeno za Javu
tip_elemenata a[]; // uobicajeno za C
```

Tip elemenata niza može biti bilo koji primitivni tip ili referentni tip (elementi niza su reference na objekte tog tipa).
Gornji primeri deklarišu da je namena promenljive a da referiše na niz vrednosti datog tipa, a ne da čuva jednu vrednost tog tipa.
Nizovska promenjiva a zapravo sadrži referencu na niz (vrednost na osnovu koje JVM može da locira niz u memoriji).
Deklaracijom se NE alocira memorija za sam niz.
Veličina niza se NE navodi u deklaraciji.

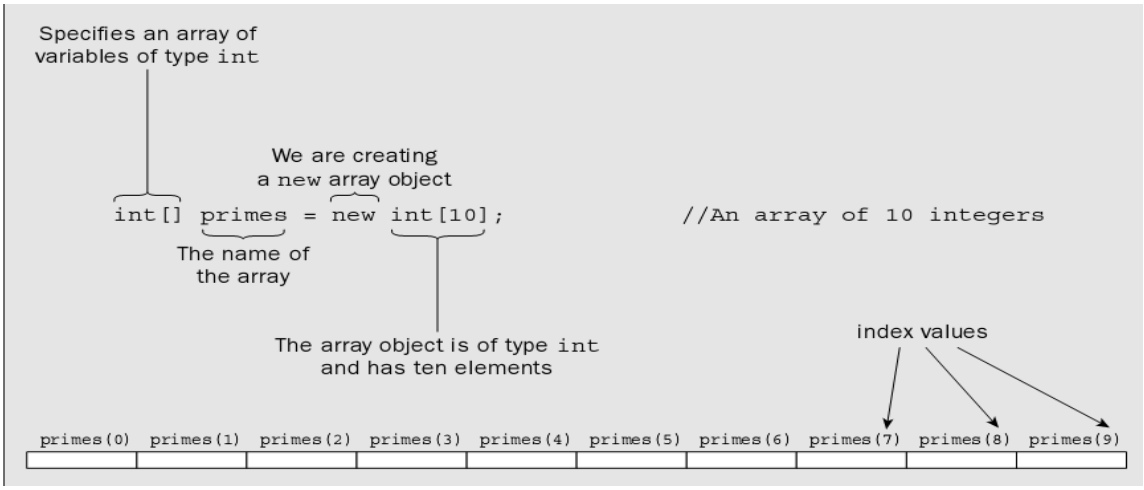
Primer:

```
int[] a;  // ili: int a[];
```
Nakon deklaracije nizovske promenljive, može se definisati niz na koji će ona referisati.

```
a = new int[10];
```
Operatorom new pravi se niz koji može da čuva 10 vrednosti tipa *int*, a referenca na niz smešta se u promenljivu a.

Moguće je u jednoj naredbi deklarisati nizovsku promenljivu i definisati niz na koji će ona referisati:

```
int[] a = new int [10];
```
Operatorom new alocira se memorija za niz i to 4*10 = 40 bajtova, kao i odgovarajući broj bajtova (obično 4, zavisno od platforme) za promenljivu a koja čuva referencu na niz.



Pravljenjem niza pomoću operatora new elementi niza se automatski inicijalizuju na podrazumevanu vrednost, koja zavisi od tipa elemenata:

- *numeričke vrednosti* – 0
- *boolean* – false
- *char* – '\u0000' (null karakter, odnosno, '\0'; decimalni ekvivalent je 0)
- *klasni tip* – null (*null* je specijalni literal, koji može da bude bilo kog referentnog tipa)

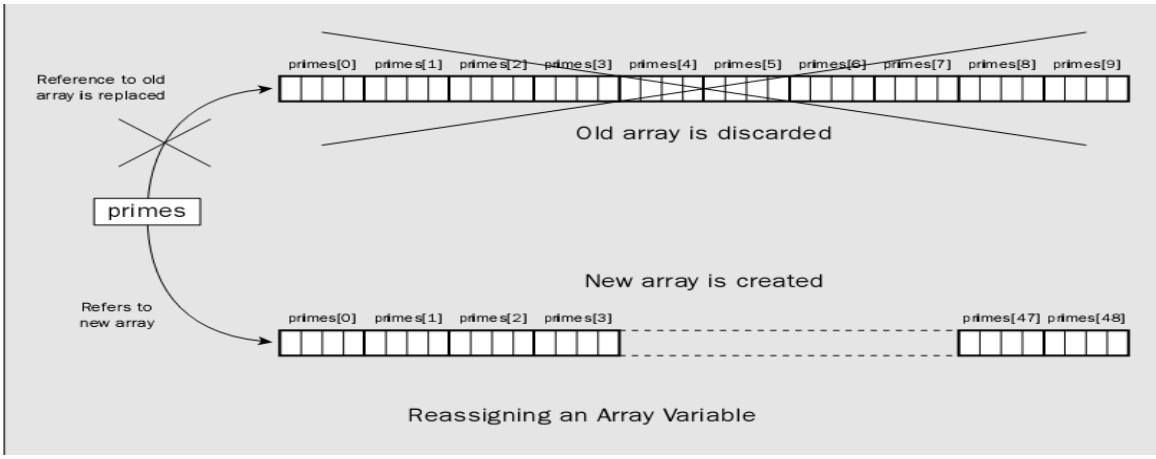
Pošto je niz objekat, *dužina niza* čuva se u samom nizu, tj. u njegovom polju *length* (tipa int), kome se može pristupiti u obliku *ime_niza.length*.

Moguće je jednu nizovsku promenljivu koristiti za referisanje *različitih* nizova u različitim delovima programa:

```
int a[] = new int[10];
```
a referiše na niz od 10 celih brojeva.

```
a = new int[50];
```
a sada referiše na novi niz od 50 celih brojeva.

Svi nizovi na koje promenljiva može da referiše u programu moraju da sadrže elemente onog tipa koji je zadat prilikom njene deklaracije. U gornjem primeru a može da referiše samo na celobrojne nizove.



Moguće je inicijalizovati elemente niza prilikom deklaracije niza.
Veličina niza se automatski određuje kao broj vrednosti za inicijalizaciju.

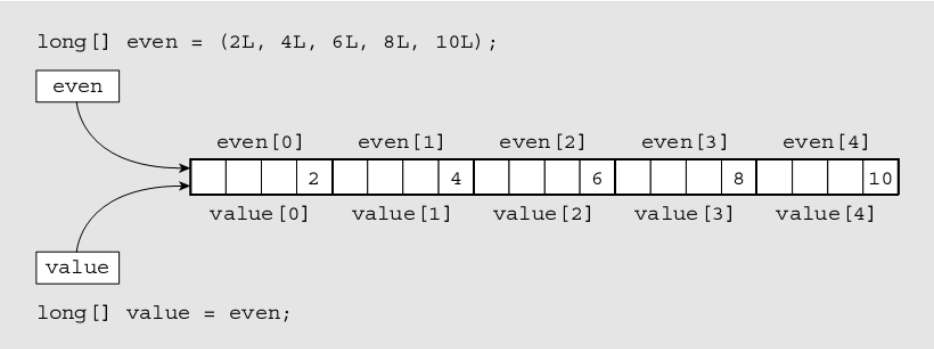
```
int[] niz = {2,3,4};  // niz.length se postavlja na 3
```

Da bi se svi elementi niza inicijalizovali istom vrednošću, moguće je koristiti statički metod *fill* iz klase *java.util.Arrays*:

```
import java.util.Arrays;
Arrays.fill(niz, 2);  // svi elementi inicijalizuju se na 2
```

Moguće je kreirati više promenljivih koje će referisati na jedan isti niz, što se može videti na narednom grafiku:

```
int[] parni = {2, 4, 6, 8, 10};
int[] novi = parni;
```
Obe promenljive, parni i novi referisu na isti niz.



Inicijalizacija elemenata niza može se izvršiti i upotrebom *for ciklusa*:

```
double[] niz = new double[50];
for( int i = 0; i < niz.length; i++)
     niz[i] = 100.0; // svi elementi se inicijalizuju na 100.0
```

Može se koristiti i oblik *for* ciklusa za kolekcije umesto klasične numeričke forme kada je potrebno pristupiti svim elementima niza:

```
double prosek = 0.0;
for(double element : niz)
     prosek += element;
prosek /= niz.length;
```

Ovakav oblik *for ciklusa* omogućuje **samo** iteraciju kroz sve elemente niza, ne može se upotrebiti za pristup elementima niza kako bi se promenila njihova vrednost. U tom slučaju treba koristiti numeričku formu *for* ciklusa.

Možemo praviti i nizove ostalih primitivnih tipova, npr. nizove karaktera:

```
char[] message = new char[50];
// svaki element zauzima 2 bajta, jer se koristi UNICODE.

java.util.Arrays.fill(message, ' ');
// svi elementi niza message su blanko karakteri.

char[] v = { 'a', 'e', 'i', 'o', 'u'};
// niz v se inicijalizuje datom listom karaktera.
```

3 String-reprezentacija elemenata niza

Za formiranje *String*-reprezentacije elemenata niza koristi se statički metoda *toString()* klase *java.util.Arrays*. String-reprezentacija se sastoji od liste String-reprezentacija elemenata, razdvojenih zapetama i ograđenih uglastim zagradama.

Primer:

```
int a[] = {2, -120, 15, 7, -10, 6};
System.out.println("a = " + Arrays.toString(a));
      // Arrays.toString(a) vraca String "[2, -120, 15, 7, -10, 6]"
```

4 Sortiranje nizova

Za sortiranje nizova koristi se statički metoda *sort()* klase *java.util.Arrays*.

Sortiranje u rastućem poretku

```
import java.util.Arrays;
// ...
long a[] = new long[30];
// ...
Arrays.sort(a); // referenca na niz prosledjuje se metodu kao argument
```

Sortiranje u opadajućem poretku

Sortiranje u opadajućem poretku moguće je samo za nizove čiji su elementi klasnog tipa, tj. reference na objekte.

Primer:
Neka je dat niz objekata klase *java.util.Integer* (omotač-klase primitivnog tipa *int*). Sortiranje u opadajućem poretku moguće je pomoću metoda *Arrays.sort()* tako što se primenom statičkog metoda *reverseOrder()* klase *java.util.Collections* generiše objekat komparator koji služi za poređenje objekata niza (u opštem slučaju kolekcije) u obrnutom poretku od prirodnog.

```
Integer aI[] = { new Integer(2), new Integer(-120), new Integer(15),
                 new Integer(7), new Integer(-10), new Integer(6) };
  System.out.println("aI = " + Arrays.toString(aI));
  Arrays.sort(aI, Collections.reverseOrder());
  System.out.println("Nakon sortiranja opadajuce: ");
  System.out.println("aI = " + Arrays.toString(aI));
```

Napomena:
Omotač-klase primitivnih tipova postoje za sve primitivne tipove u paketu *java.lang* (*Byte, Character, Short, Integer, Long, Float, Double, Boolean*).

5 Nizovi nizova (višedimenzioni nizovi)

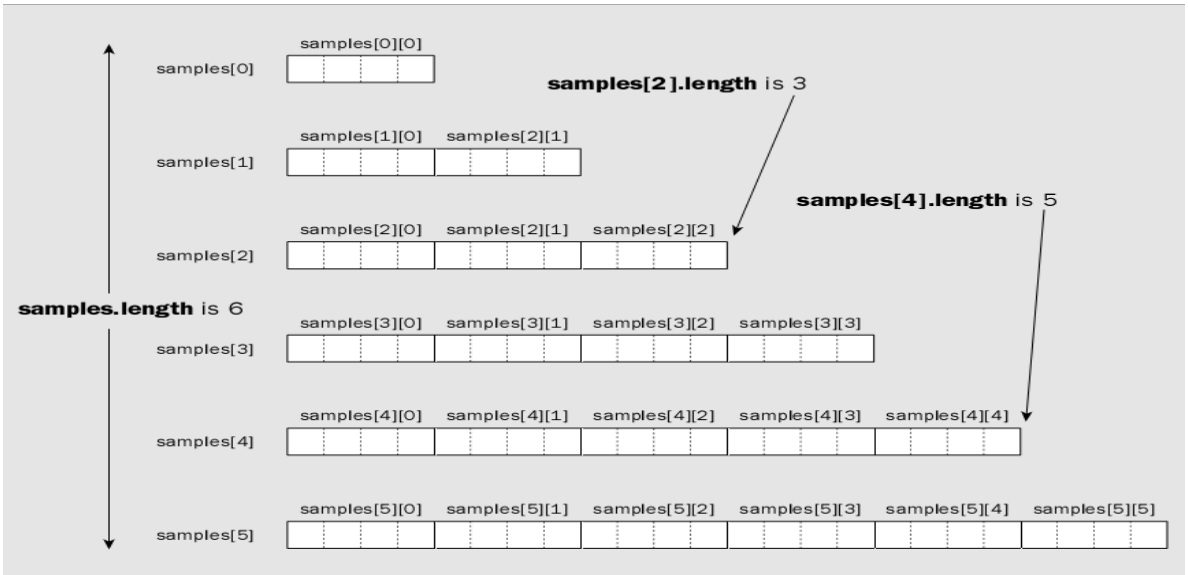
Primer:

```
  int[][] a = new int[3][5];
```

a je referenca na niz od 3 elementa, a svaki od njih je niz od 5 elemenata.

Mogu se kreirati i nizovi nizova *različitih dužina*:

```
  float[][] samples;
  samples = new float[6][];
```



Promenljiva samples sada referiše na niz od 6 elemenata a svaki od njih može da čuva *referencu* na jednodimenzioni niz.

```
samples[1] = new float[4]; // samples[1] je niz dužine 4
samples[3] = new float[10]; // samples[3] je niz dužine 10


for(int i = 0; i < samples.length; i++)
     samples[i] = new float[i+1];
            // i-ti element niza je niz od i+1 elemenata
```