5

Složeni tipovi podataka

- Niz
- String

Niz

Često ćemo prilikom programiranja trebati u varijable spremati i nekoliko desetaka pa čak i stotina podataka. Deklarirati 100 varijabli i onda još njima uspješno manipulirati bilo bi poprilično komplicirano. U takvim slučajevima u pomoć nam stižu **indeksirane varijable** tzv. **nizovi**. Prisjetimo se matematike i npr. sustava linearnih jednadžbi. Ako imamo sustav od npr. 3 linearne jednadžbe s 3 nepoznanice, tada ćemo najčešće nepoznanice označavati s x, y i z. Međutim imamo li sustav od 5 jednadžbi s 5 nepoznanica, nepoznanice ćemo najčešće označavati s x_1, x_2, x_3, x_4, i x_5. Slično je i s nizovima kod programiranja. Jedina razlika je što elemente nizova nećemo označavati s: x_1, x_2, ..., x_n nego s: *x[1], x[2],...x[n]*.

Slika 5 – 1: Niz

//ovdje može npr. slika vlaka s vagonima na kojima piše *x[1], x[2],..., x[n]*

Neke od prednosti korištenja su:

- unos/ispis elemenata moguć je pomoću jedne petlje;
- ne deklariramo svaki element niza posebno, već ih deklariramo sve istovremeno;
- lakše manipuliranje podacima (sortiranje, pretraživanje,...);
- ...

U Javi je moguće kreirati niz bilo kojeg tipa podataka (*int*, *char*, *String*,...), poslije ćemo vidjeti da je moguće kreirati čak i niz objekata. Elementi niza u Javi počinju se indeksirati od 0. To znači da prvi element niza ima indeks 0, drugi 1,... Npr. ako je x niz od n elemenata, tada će elementi toga niza biti: *x[0], x[1], x[2],..., x[n-1]*.

Niz ćemo u Javi deklarirati na jedan od sljedećih načina:

```
tip[] ime_niza;
```

ili

```
tip ime_niza[];
```

Primjer 5 – 1:
Deklarirajmo niz *a* čiji će elementi biti cijeli brojevi:

Rješenje:
Kao što smo rekli niz možemo deklarirati na dva načina:

```
int[] a;
```

odnosno:

```
int a[];
```

Mi ćemo od sada pa na dalje niz uvijek deklarirati na prvi način:

```
tip[] ime_niza;
```

Prilikom deklaracije niza, kao i kod deklaracije ostalih jednostavnih tipova podataka, elementima niza možemo pridružiti inicijalne vrijednosti:

```
tip[] ime_niza = {vr_1, vr_2, ..., vr_n};
```

Primjer 5 – 2:
Deklarirajmo niz *c* čiji će elementi biti znakovnog tipa, a inicijalno će vrijednosti elemenata niza biti samoglasnici:

Rješenje:
```
char[] c = {'a', 'e', 'i', 'o', 'u'};
```

Nakon ovakve deklaracije niza vrijednosti elemenata niza bit će redom:
```
c [0] = 'a';
c [1] = 'e';
c [2] = 'i';
c [3] = 'o';
c [4] = 'u';
```

Ukoliko deklariramo neki niz i nakon deklaracije prvom elementu niza pokušamo pridružiti neku vrijednost:

```
int[] a;
a [0] = 2;
```

te pokušamo kompajlirati taj dio koda, doći će do greške. Kompajler će javiti sljedeću grešku:

```
variable a might not have been initialized
      a[0] = 2;
```

U čemu je problem? Izgleda sve korektno napisano. Da uistinu smo sve korektno napisali. Međutim, ako nismo elementima niza inicijalno pridružili vrijednosti, tada prije pridruživanja vrijednosti nekom elementu niza trebamo još "***rezervirati prostor***" za taj niz, tj. trebamo **alocirati memoriju**.

Memoriju za elemente nekog niza alocirat ćemo na sljedeći način:

```
ime_niza = new tip[broj_elemenata];
```

pri čemu je ***tip*** onaj isti tip podataka koji smo naveli kod deklaracije niza, dok je ***broj_elemenata*** maksimalni broj elemenata koje će niz sadržavati.

Često se deklaracija niza spaja s alociranjem prostora za niz, pa u tom slučaju deklaracija niza ima oblik:

```
tip[] ime_niza = new tip[broj_elemenata];
```

Primjer 5 – 3:
Deklarirajmo znakovni niz od 20 elemenata te elementima niza pridružimo sva velika slova engleske abecede.

Rješenje:
```
...
char[] c;
c = new char [26];
for (int i = 65; i <= 90; i++)
      c[i - 65] = (char) i;
...
```

Pokušamo li u prethodnom primjeru npr. 30-tom elementu niza pridružiti neku vrijednost, prilikom izvršavanja programa doći će do sljedeće greške:

```
ArrayIndexOutOfBoundsException
null
```

Ulazni parametar metode isto tako može biti i niz. U tom slučaju elemente niza pišemo unutar vitičastih zagrada - {}, međusobno odvojene znakom zareza - , (isto kao kad inicijaliziramo elemente niza prilikom deklaracije niza).
Broj elemenata nekog niza (za koje je alociran prostor) vratit će nam naredba:

```
ime_niza.length;
```

Primjer 5 – 4:
Napišimo metodu čiji će ulazni parametri biti niz *a* te prirodni broj *n*, *n* manji od broja elemenata niza. Metoda treba sortirati niz *a* te vratiti element niza *a* čiji je indeks *n*.

Rješenje:
Niz ćemo sortirati najjednostavnijom metodom razmjene koju ovdje nećemo specijalno objašnjavati. Princip sortiranja niza metodom razmjene je sljedeći:
- općenito *i*-ti element niza (*i* ide od prvog do pretposljednjeg elementa niza) uspoređujemo sa svim elementima iza njega. Ukoliko naiđemo na element, s indeksom *j*, koji je manji od *i*-tog elementa *i*-tom i *j*-tom elementu niza zamjenjujemo vrijednosti.

```
public static int pr54 (int[] a, int n)
{
      int tmp;
      for (int i = 0; i < a.length - 1; i++)
            for (int j = i + 1; j < a.length; j++)
                  if (a [i] > a [j])
                  {
                        tmp = a [i];
                        a [i] = a [j];
                        a [j] = tmp;
                  }
      return a[n];
}
```

Višedimenzionalni nizovi

Nizovi s kojima smo se upoznali u prethodnom poglavlju imaju jedan indeks te ih stoga zovemo **jednodimenzionalni nizovi**. Osim takvih jednodimenzionalnih nizova u praksi se vrlo često koriste i višedimenzionalni, najčešće dvodimenzionalni nizovi. **Dvodimenzionalni nizovi** će imati dva indeksa i zvat ćemo ih **matrice**. Princip korištenja matrica analogan je principu korištenja jednodimenzionalnih nizova.
Elementi matrice će imati dva indeksa, te će opći oblik elementa matrice biti:

```
ime_niza [i][j];
```

Deklaracija matrice biti će sljedećeg oblika:

```
tip[][] ime_niza;
```

odnosno, deklaracija s inicijalizacijom:

```
tip[][] ime_niza = {{vr_11, vr_12,..., vr1m},...,{vr_n1,...,vr_nm}}
```

odnosno, deklaracija s alokacijom memorije:

```
tip[][] ime_niza = new tip[broj_el1][broj_el2]
```

Na matricu možemo gledati kao na tablicu od ***broj_el1*** redaka i ***broj_el2*** stupaca.

ime_niza[0][0]	**ime_niza[0][1]**	**...**	**ime_niza[0][broj_el2]**
ime_niza[1][0]	**ime_niza[1][1]**	**...**	**ime_niza[1][broj_el2]**
...	**...**	**...**	**...**
ime_niza[broj_el1][0]	**ime_niza[broj_el1][1]**	**...**	**ime_niza[broj_el1][broj_el2]**

Slika 5 – 2: Matrica

Dimenzija matrice je broj redaka i stupaca te matrice. Općenito ćemo dimenziju matrice koja ima n redaka i m stupaca označavati s $n \times m$.

Primjer 5 – 5:
Napiši metodu čiji će ulazni parametar biti dimenzija matrice (n i m) te prirodan broj $k <= n*m$. Metoda treba popuniti matricu prirodnim brojevima do $n * m$ i to tako da redom popunjava elemente matrice počevši od gornjeg lijevog kuta do donjeg desnog kuta matrice.
Npr. ako je unesena dimenzija matrice 4×3, tada metoda treba kreirati sljedeću matricu:

1 2 3
4 5 6
7 8 9
10 11 12

Nadalje metoda treba vratiti zbroj svih susjeda broja k u matrici. Npr. susjedi broja 8 su: 4, 5, 6, 7, 9, 10, 11 i 12, pa je onda njihov zbroj jednak 64.

Rješenje:

```
//metoda kreira traženu matricu dimenzije nxm
public static int[][] matrica (int n, int m)
{
	int k = 1;
	int[][] a = new int[n][m];
	for (int i = 0; i < n; i++)
		for (int j = 0; j < m; j++)
		{
			a[i][j] = k;
			k++;
		}
	return a;
}

public static int pr55(int n, int m, int t)
{
	int[][] a = matrica (n, m);
	int x = 0, y = 0, s = 0;
	//tražimo poziciju broja t
	for (int i = 0; i < n; i++)
		for (int j = 0; j < m; j++)
			if (a[i][j] == t)
			{
				x = i;
				y = j;
			}
	if (x > 0)
	{
		if (y > 0)
			s += a [x - 1][y - 1] + a [x][y - 1];
		s += a [x - 1][y];
		if (y < m - 1)
			s += a [x - 1][y + 1] + a [x][y + 1];
	}
	if (x < n - 1)
	{
		if (y > 0)
			s += a [x + 1][y - 1];
		s += a [x + 1][y];
		if (y < m - 1)
			s += a [x + 1][y + 1];
	}
	return s;
}
```

Primjer 5 – 6:
Svima nam je dobro poznata igra križić-kružić. Napiši metodu čiji će ulazni parametar biti matrica koja predstavlja jednu odigranu igru križić-kružić i ispisivati pobjednika (križić ili kružić) odnosno da nema pobjednika ako u odigranoj igri nema pobjednika.
Npr. kombinaciju:

x	o	**x**
o	**x**	x
x	o	o

ćemo unijeti kao: {{'x', 'o', 'x'},{'o', 'x', 'x'},{'x', 'o', 'o'}}
i pobjednik je očito *x*.

Rješenje:

```
//metoda koja na osnovu učitane matrice vraća true ako je c pobjednik
```

```
public static boolean pobjednik (char[][] a, char c)
{
	//provjeravamo postoji li redak sastavljen samo od znakova c
	for (int i = 0; i < 3; i++)
	if (a[i][0] == c && a[i][1] == c && a[i][2] == c)
			return true;
	//provjeravamo postoji li stupac sastavljen samo od znakova c
	for (int i = 0; i < 3; i++)
	if (a[0][i] == c && a[1][i] == c && a[2][i] == c)
			return true;
	//postoji li dijagonala sastavljena samo od znakova c
	//glavna dijagonala
	if (a[0][0] == c && a[1][1] == c && a[2][2] == c)
		return true;
	//sporedna dijagonala
	if (a[0][2] == c && a[1][1] == c && a[2][0] == c)
		return true;
	return p;
}

public static String pr56 (char[][] a)
{
	if (pobjednik (a, 'x') && pobjednik (a, 'o'))
		return "Neriješeno";
	else if (pobjednik (a, 'x'))
		return "Pobjednik je x";
	else if (pobjednik (a, 'o'))
		return "Pobjednik o";
	else
		return "Nema pobjednika";
}
```

String

O tipu podataka **String** smo već nešto govorili u prethodnim poglavljima. Ovdje ćemo nastojati kroz nekoliko jednostavnih primjera u potpunosti usvojiti tip podataka **String**, ukazati na mjesta na kojima ćemo ga koristiti, prednosti korištenja ovog tipa podataka, a s druge strane ćemo dati uvod u sljedeće poglavlje u kojem će biti riječi o klasama.
Za razliku od jednostavnog tipa podataka **char**, koji može sadržavati samo jedan znak, koji se u tom slučaju navodi unutar jednostrukih navodnika ('), tip podataka **String** može sadržavati više znakova, tj. u tip podataka **String** ćemo pohranjivati riječi pa čak i čitave rečenice, a takve riječi odnosno rečenice ćemo u tom slučaju navoditi unutar dvostrukih navodnika (").
Za razliku od svih tipova podataka koje smo do sada naučili, nad kojima smo mogli izvršavati neke jednostavnije operacije (najčešće matematičke), tip podataka **String** je jedan znatno složeniji tip podataka sa mnoštvom radnji koje možemo nad njim izvršavati. Takve složene tipove podataka nad kojima su definirane mnoge radnje općenito ćemo zvati klase. Dakle tip podataka **String** je u stvari jedna klasa u Javi. O klasama još nismo do sada govorili, o njima ćemo više govoriti u sljedećem poglavlju. Dok još ne znamo puno o klasama, klase možemo zamišljati kao složene tipove podataka. Nad takvim složenim tipovima podataka moći ćemo izvršavati određene radnje, tzv. metode.
Neke od "radnji" koje bi mogli izvršavati nad riječima (stringovima) bile bi:

- prebrojati broj znakova u riječi;
- provjeriti koji se znak nalazi na nekom mjestu u riječi;
- ...

Pa upravo su za takve nad stringovima definirane metode.
No vratimo se na početak. Varijablu u koju ćemo moći spremati riječi, odnosno varijablu tipa **String** deklarirat ćemo na sljedeći način:

```
String ime_varijable;
```

odnosno, deklaracija s inicijalizacijom:

```
String ime_varijable = "vrijednost";
```

Najčešće radnje (metode) koje ćemo izvršavati nad tipom podataka **String** su:

int length () – vraća broj znakova u stringu
boolean equals (String s) – uspoređuje dva stringa i vraća true ako su jednaka, inače vraća false
int indexOf (String s) – vraća poziciju prvog pojavljivanja stringa *s* u stringu u kojem se nalazimo
int lastIndexOf (String s) – vraća poziciju zadnjeg pojavljivanja stringa *s* u stringu u kojem se nalazimo
void replaceAll (String s1, String s2) – svako pojavljivanje stringa *s1* u stringu u kojem se nalazimo zamjenjuje sa stringom *s2*
String substring (int poc, int kraj) – vraća dio stringa od znaka s rednim brojem *pos* do znaka s rednim brojem *kraj* stringa u kojem se nalazimo
String toUpperCase () – sva mala slova stringa zamjenjuje s odgovarajućim velikim slovima, ostala slova ostavlja nepromijenjenim
String toLowerCase () – sva velika slova stringa zamjenjuje s odgovarajućim malim slovima, ostala slova ostavlja nepromijenjenim

Ilustrirajmo sve rečeno na nekoliko jednostavnih primjera:

Primjer 5 – 7:
Napišimo metodu čiji će ulazni parametar biti ime i prezime neke osobe (ime i prezime su jedan string i međusobno su odvojeni točno jednim razmakom), a vraćati će inicijale te osobe (prvo slovo imena i prvo slovo prezimena).

Rješenje:
```
public static String pr57 (String ime)
{
      /*uzima prvo slovo imena, dodaje mu točku i razmak te sve to sprema
      u string s */
      String s = ime.substring (0, 1) + ". ";
      //traži razmak u stringu
      int n = ime.indexOf (" ");
      /*na kraju stringa s dodaje prvi znak nakon razmaka te na kraju još
      jednu točku */
      s = s + ime.substring (n + 1, n + 2) + ".";
      return s;
}
```

Napomena:
Kod pokretanja metoda čiji su ulazni parametri stringovi, stringove trebamo navesti unutar dvostrukih navodnika.

Primjer 5 – 8:
Napišimo metodu čiji će ulazni parametar biti jedna riječ. Metoda treba rastaviti riječ na slogove (u ovom slučaju ćemo riječ rastavljati na slogove tako da iza svakog samoglasnika stavimo znak -, osim u slučaju kada je zadnji znak samoglasnik, u tom slučaju iza njega nećemo stavljati znak -).

Rješenje:
```
public static String pr58 (String rijec)
{
      String s = "";
      for (int i = 0; i < rijec.length() - 1; i++)
      {
            String z = rijec.substring (i, i + 1);
            s = s + z;
            z = z.toUpperCase ();
            if (z.equals ("A") || z.equals ("E") || z.equals ("I") ||
(z.equals ("O")) || (z.equals ("U")))
                  s = s + "-";
      }
      s = s + rijec.substring (rijec.length () - 1, rijec.length ());
      return s;
}
```