PREDMET

Osnove Java Programiranja

Cas 15-16

NIZOVI

Oktobar 2011.

SADRŽAJ

Čas 15-16

Nizovi

Šta je to niz

Niz je skup uređenih podataka. U slučaju Jave kao strogo tipskog programskog jezika niz je skup uređenih podataka istog tipa. Uređenost niza znači da svi elementi niza imaju svoje mesto u niz. Prvi elementu u nizu je nulti element, odnosno taj elementi ima index 0. Sledeći element u nizu ima index 1, ... Poslednji element u nizu ima index n-1 pri čemu je n broj elemenata niza. Index je redni broj elementa u nizu. Java je preuzela sistem indexiranja nizova od C-a i C++-a

Deklaracija niza

Niz se deklariše tako što se: prvo definiše tip elemenata niza, zatim ide otvorena kockasta i zatvorena kockasta zagrada (što određuje da je niz), i nakon toga stoji naziv niza. Na ovaj način se definiše i deklariše niz. Kada se pravi niz, tip niza može da bude bilo koji prost tip ili bilo koji objekat.

```
int[] niz;
long[] nizDugih;
```

Konstrukcija niza

Nakon deklaracije niza, postoji samo rezervisan naziv promenljive za taj niz, ali niz u tom trenutku ne postoji. Da bi niz postojao, mora da se konstruiše. Konstrukcija niza može da se izvrši u istom redu naredbe kao i deklaracija. To je prikazano u prvoj naredbi. Na ovaj način se deklarše niz celih brojeva i konstruiše pomoću ključne reči **new.** Ključna reč new vrši konstrukciju, tom prilikom se dostavlja i informaca o broju elemenata tog niza, u ovom slučaju niz će imati 8 elemenata. Na ovaj način se izvrši deklaracija i konstrukcija niza, u jednoj naredbi. Deklaracija i konstrukcija niza može da budu i odvojene u dve naredbe. To je prikazano u drugom primeru. Dakle, nizDugih je naziv promenljive rezervisane za taj niz, i tek nakon nekog broja naredbi, se vrši konstrukcija (izvršava se new long[25]; , što znači da se konstruiše konkretan niz sa 25 elemenata tipa **long**). Kod vršenja konstrukcije, broj elemenata niza može da se definiše i pomoću neke varijabile, odnosno ne mora da bude broj, nego može da bude neka promenljiva, koja će biti definisana u vremenu izvršavanja aplikacije. U poslednjem redu primera je prkazana dinamička deklaracija niza, i konstrukcije, gde, kao što se vidi, veličina niza nije definisana u vremenu kompajliranja već u vremenu izvršavanja.

```
int[] niz = new int[8];
long[] nizDugih;
...
nizDugih = new long[25];
int[] nizDinamicki = new int[vInt.size()];
```

Inicijalizacija niza

Nakon izvršene deklaracije i konstrukcije niza, elementi tog novonastalog niza imaju svoje **default** vrednosti. Inicijalna **default** vrednost za brojeve, bez obzira da li su celi ili realni, je 0. Inicijalna realna vrednost za **String**-ove je prazan **String**. Inicijalna vrednost za objekte je **null**, što znači da objekat ne postoji. U mnogim situacijama je potrebno da, se definišu konkretne vrendosti niza na početku rada programa. Definisanje vrednosti elemenata niza se zove inicijalizacija, i to se može uraditi kao što je

prikazano na primeru. Prvi red je deklaracija i konstrukcija niza, a nakon toga prva 4 elementa niza se pune konkretnim vrednostima. Na ovaj način je izvršena inicijalizacija niza.

```
int[] niz = new int[8];
niz[0]=2;
niz[1]=4;
niz[2]=6;
niz[4]=8;
```

Deklaracija konstrukcija i inicijalizacija

Moguće je, u jednoj jedinoj naredbi, spojiti ove tri stvari: izvršiti deklaraciju niza (dakle, definisati kog su tipa elementi niza i rezervisati promenljivu za taj niz), izvršiti konstrukciju niza (odrediti koliko elemenata taj niz treba da ima), i izvršiti inicijalizaciju (te elemente napuniti početnim vrednostima). To se radi kao što je prikazano na primeru. Ovde se vidi da sa desne strane znaka jednakosti u okviru vitičastih zagrada, su pobrojene vrednosti tog niza. Onoliko vrednosti koliko postoji u vitičastim zagradama, nad toliko vrednosti će biti izvršena konstrukcija niza. U ovom slučaju biće izvršena konstrukcija niza sveZajedno, koji će imati 9 elemenata, a vrednost tih elemenata će biti 2, 4, 6, 8, 10, 12, 14, 16 i 18. Na ovaj način je, u jednoj jedinoj naredbi, spojena deklaracija, konstrukcija i inicijalizacija.

```
int[] sveZajedno = {2, 4, 6, 8, 10, 12, 14, 16, 18};
```

Dužina niza

U Javi su nizovi objekti, što znači da imaju svoje osobine i metode. Jedan od osnovnih podataka vezanih za objekat niza je njegova dužina. To je promenljiva **lenght**. U promenljivoj **lenght** se uvek nalazi kolika je dužina niza, odnosno koliko elemenata ima u tom nizu. Informacija o veličini niza je često potrebna. Kada god se, recimo, prolazi kroz petlju i ispituje svaki element niza ponaosob, treba da se zna koji element je poslednji to jest do kog indeksa da se vrti petlja. Ova informacija se u Javi nalazi u okviru objekta niz, za razliku od, recimo, C-a i C++-a, kod kojih se veličina niza mora da se vodi u posebnoj promenljivoj. Kada se, recimo, niz šalje u nekakvu metodu, u C-u i C++-u je bilo potrebno da se posebno pošalje i promenljiva koja definiše veličinu niza. U Javi to nije potrebno, jer sam niz nosi sa sobom i svoju veličinu.

```
int[] niz = new int[8];
int duzinaNiza = niz.length;
```

Kreiranje niza u petlji

Nizovi se često koriste u kombinaciji sa **for** petljom. **For** petlja je dobra za prolazak kroz nizove zato što ima inicijalizaciju brojača, a taj brojač može da služi i kao indeks elemenata niza. Zbog toga se for petlja često koristi u kombinaciji sa nizovima. U primeru se vidi jedna tipična upotreba **for** petlje u procesu punjenja niza određenim vrednostima. U primeru su napunjeni elementi niza kubom vrednosti indeksa. U primeru je prikazan niz od 8 elemenata, celih brojeva, i **for** petlja pomoću koje se puni taj niz. Deklariše se brojač, koji počinje od vrednosti 0. Brojač ide u krug, dokle god ne dođe do elementa koji je manji od veličine niza (u ovom slučaju dok i ne bude bilo 7 – to je poslednji korak u ovoj petlji), i, naravno, i povećava se brojač za 1 u svakom koraku. U samom telu petlje se pune elementi niza, tako što se u svaki element niza stavi vrednost kuba njegovog indeksa.

```
int[] niz = new int[8];
for (int i = 0; i < niz.length; i++)
{
      niz[i] = i * i * i;
}
```

Obrada elemenata niza u petlji

Ovo je primer obrade elemenata niza u okviru nove **for** petlje. Na početku se deklariše promenljiva suma, koja je tipa **ceo broj**, i postavlja se na početnu vrednost 0. Nakon toga se pravi nova **for** petlja koja ima lokalnu promenljivu t, tipa **int**, i niz **int** elemenata. U okviru petlje, svakom elementu niza se dodeljuje vrednost sume. Operatorom += se vrši dodavanje na već postojeću vrednost. Kada se prođe kroz sve elemente niza, u promenljivoj suma će se nalaziti suma svih elemenata niza. Na kraju se to prikazuje na ekranu pomoću naredbe **System.out.println** suma.

```
int suma=0;
for(int t : niz)
{
	suma+=t;
}
System.out.println("suma = " + suma);
```

Niz Objekata

Kao što je rečeno, moguće je praviti i nizove objekata. Kada se deklariše i konstruiše niz objekata, u suštini, nepostoji ni jedan objekat, odnosno svi elementi niza su tipa **null** (objekat ne postoji). Pored toga što se konstruiše niz, mora da se konstruiše i svaki objekat koji je element tog niza. Jedan primer konstruisanja niza objekata, i konstruisanja objekata u okviru tog niza, je ovde prikazan. Klasa Neprijatelji, je klasa koja je ranije kreirana. Postoji niz koji se zove đavoli, u kojem se nalaze elementi tipa Neprijatelj. Nakon što se izvrši deklaracija i konstrukcija tog niza i definiše se da će ovaj niz imati 10 neprijatelja u okviru niza koji se zove đavoli. Kreira se svaki pojedinačni objekta. Prolazi se kroz sve elemente niza, i za svaki element niza se kreira nov objekat tipa Neprijatelji, koji prima dva podatka. U ovom slučaju pravi se **random** vrednosti za štit i energiju klase Neprijatelji. Nakon toga se poziva metoda **setRedniBroj** klase Neprijatelji, i prikazuje se svaki od tih neprijatelja pomoću System.out.println. Ovde se primer kako se konstruiše niz. Prilikom konstrukcije niza mora da se da naziv niza, da se konstruiše niz pomoću reči **new**, i mora svaki objekat u okviru tog niza da se konstruiše pomoću reči **new**.

```
Neprijatelji[] djavoli = new Neprijatelji[10];
for(int i = 0; i<djavoli.length; i++)
{
	djavoli[i] = new Neprijatelji(Math.random()*100, Math.random()*100);
	djavoli[i].setRedniBroj(i+1);
	System.out.println(djavoli[i]);
}
```

Kloniranje niza – Stari način

Kada programer dobije zahtev da napravi kopiju niza, odnosno da napravi drugi niz koji u sebi ima iste elemente kao neki već prethodno definisani niz, onda on najčešće koristi kod kao što je prikazano na primeru. Deklariše se nov niz, koji se zove kopijaNizaOld kao **new int**, a veličina tog niza je, praktično, veličina niza koji se kopira (u ovom slučaju niz.length). Prikazana je **for** petlju, pomoću koje se prolazimo kroz svaki element starog niza niz, i svaki element tog niza se smešta u kopiju niza pod istim indeksom. Ovaj način mnogi programeri koriste, iako trenutno postoji mnogo efikasniji način za kloniranje niza.

```
int[] kopijaNizaOld = new int[niz.length];
for(int i = 0; i<niz.length; i++)
{
	kopijaNizaOld[i] = niz[i];
```

```
}
```

Kloniranje niza – Java

Kloniranje niza koje je prikazano u prethodnom primeru je karakteristično za C ili C++ programe, gde nizovi nisu klase, odnosno objekti, već gde su nizovi samo niz informacija u memoriji. U slučaju Java programskog jezika, niz je objekat. Samim tim on ima određene sposobnosti, npr. kopija niza može da se napravi tako što se uzme postojeći niz (koji se u ovom slučaju zove niz), i izvrši se njegovo kloniranje pozivajući se metoda **clone** objekta niz. Na kraju primera se prikazuje samo rezultat na ekranu. Kao što možete da se primeti, ovo je daleko jednostavniji način za kloniranje niza, a uz to je i brži i efikasniji.

```
int[] kopijaNiza = niz.clone();
for (int t : kopijaNiza)
{
      System.out.println("t = " + t);
}
```

Iz Vector-a u niz

Programeri često u hodu koriste vektor, ponekad se koristi niz, i potrebno je, s' vremena na vreme, da se podaci prepakuju iz vektora u niz i obrnuto. Na primeru se vidi kako može da se podaci iz vektora prebace u niz. Prvo se deklariše i konstruiše vektor, kolekcija, i napuni se sa nekim slučajnim vrednostima. Nakon toga se izvršili deklaracija i konstrukcija niza intovi, koji moraja da bude tipa **integer**, jer je vektor morao u sebi da sadrži objekte (ne može da sadrži proste tipove kao što je int). Iz tog razloga i niz je kreiran da bude istog tipa, odnosno tipa **integer**. Dimenzija niza je određena na osnovu veličine vektora. vInt.size nam vraća broj elemenata u vektoru, što ujedno treba da bude i veličina niza. Prebacivanje elemenata iz vektora u niz se vrši pomoću naredbe **toArray**. Svaka kolekcija ima metodu **toArray**, pomoću koje prebacuje svoje elemente u niz koji ima odgovarajući tip podataka.

```
Vector<Integer> vInt = new Vector<Integer>();
for (int i = 0; i < 80; i++)
{
      vInt.add((int) Math.round(1000 * Math.random()));
}

Integer[] intovi = new Integer[vInt.size()];
vInt.toArray(intovi);
```

Iz Niza u Vector

Prilikom rada sa kolekcijama i nizovima, često je potrebna i obrnuta konverzija – iz niza u vektor. Kolekcija tipa **list** može da se kreira pomoću **utility** klase **Arrays** i njene metode **asList**, čiji je parametar niz koji se konvertuje. Zahvaljujući **utility** klasi **Arrays** i njenoj metodi **asList**, mi moguće je bilo koji niz konvertovati u listu. Lista je određeni tip kolekcije, i svaka kolekcija, pa i vektor, može da kreira svoju instancu tako što kao parametar u konstruktoru primi listu. To se vidi u ovom primeru. Na ovaj način je moguće napraviti vektor pomoću već postojećeg niza, i tom prilikom će vektor da ima sve elemente koje ima i niz.

```
Vector<Integer> izNiza = new Vector<Integer>(Arrays.asList(intovi));
```

Punjenje niza nekom vrednošću

Utility klasa Arrays ima nekoliko veoma korisnih metoda. Na prošlom primeru je prikazana metoda **asList**, a na ovom primeru će biti prikazana uoptreba metode **fill**. Pomoću metode **fill** se puni niz određenom vrednošću. U primerz je deklarisan niz d, i konstruiše se tako da ima 50 elemenata. Zatim se korišćenjem **utility** klase **Arrays** i pozivom njene metode **fill**, napunili taj niz vrednostima 0,1. Dalje se vidi **for** petlja kroz koju se prolazi, i prikazuju se svi elementi. Tu može da se vidi da su svi elementi ovog niza zaista 0,1.

```
double[] nizD = new double[50];
Arrays.fill(nizD, 0.1d);
for (double d : nizD)
{
	System.out.println("Fill = " + d);
}
```

Povećavanje niza

Jedan od najvećih problema u radu sa nizovima je to što je niz fiksna veličina, odnosno ne može jednostavno, u toku rada sa nizom, da se promeni njegova veličina. Ono što može da se uradi je da se napravi novi niz koji je konstruisan kao veći niz, da on u sebi sadrži sve elemente prethodnog niza, i da ima još mesta za nove elemente. Ovo je dosta komplikovan proces, gde moramo da deklarišemo novi niz, pa da određujemo njegovu veličinu, pa da kopiramo svaki element iz prethodnog niza u taj novi niz, itd. I ovde nam pomaže **utility** klasa **Arrays**, koja ima metodu **copyOf**. Deklariše se novi niz (u ovom slučaju je deklarisan novi niz pod nazivom novi). Napravljen je tako što smo je uzet stari niz (niz d), i poveća se za 6 elemenata (novi niz ima 56 elemenata). To se uradi pomoću **copy** metode **Arrays utility** klase, i na taj način se napravi novi niz, koji ima 56 elemenata, od čega su prvih 50 elemenata elementi niza niz d.

```
double[] novi = Arrays.copyOf(nizD, 56);
System.out.println("novi.length = " + novi.length);
```

Sortiranje nizova

Prilikom programiranja često se dolazi u situaciju da je neophodno da se elementi niza sortiraju. Ovo je dosta naporan i komplikovan posao, odnosno zahteva dosta koda prilikom sortiranja. **Utility** klasa **Arrays** ima svoju metodu **sort**, pomoću koje može da se sortira niz. U ovom slučaju postoji niz intovi, koji u sebi sadrži niz intova, i može da se sortira pozivom **sort** metode **Utility** klase **Arrays**. Zatim se prolazi kroz sve elemente tog niza, i prikazuju se na ekranu, ovde se vidi da su oni zaista sortirani. Prilikom rada sa nizovima treba uvek imati u vidu ovu **Utility** klasu **Arrays**, koja značajno skraćuje posao u mnogim situacijama.

```
Arrays.sort(intovi);
for (int vT : intovi)
{
	System.out.println("Sortirani = " + vT);
}
```

Slanje niza u funkciju

U toku rada sa nizovima često treba da se celokupan niz pošalje u neku funkciju na obradu. To se radi tako se pozove naziv metoda (pozivMetoda), i prosledi joj se samo naziv niza. U primeru je prikazana celokupna metoda pozivMetoda, kako je deklarisana i implementirana. Dakle, private void pozivMetoda prima niz tipa integer (oznaka [] naznačava da je promenljiva intovi u suštini niz tipa integer). U samoj implementaciji metode je izvršeno sortiranje niza koji je prosleđen. S' obzirom da se nizovi šalju po referenci (to znači da se šalje pokazivač na sam niz), svaka izmena u okviru metode će

direktno uticati na sam niz, odnosno, u ovom slučaju, sortiranje elemenata niza u pozivMetode će da izvrši sortiranje samog niza, koji kasnije može da se koristi tako sortiran niz.

```
pozivMetoda(intovi);
private void pozivMetoda(Integer[] intovi) {
	Arrays.sort(intovi);
	for (int vT : intovi)
	{
		System.out.println("izVectora = " + vT);
	}
}
```

Slanje niza kao vararg

U Javu 1.5 je uvedena jedna mala novost. Moguće je deklarisati metodu da prima promenljiv broj parametara, tzv **vararg**. Promenljivi broj parametara se tumači kao i niz. U primeru je prikazanao da je implementacija ove metode identična implementaciji prethodne metode, iako su im same deklaracije različite. U predhodnoj metodi (bez vararg) se primao niz **integer**-a, a ovde se prima proizvoljan broj parametara. Kao što možete da se primeti, poziv ove metode je ostao isti u slučaju da imamo niz. Ako je niz intovi, možemo ga poslati u ovu metodu isto kao što smo ga pozivali u prethodnoj, odnosno, što se Jave tiče, ove dve metode su identične.

```
pozivMetoda(intovi);
private void pozivMetoda(Integer ... intovi) {
	Arrays.sort(intovi);
	for (int vT : intovi)
	{
		System.out.println("izVectora = " + vT);
	}
}
```

Slanje kopije niza

Kao što je rečeno u prethodnom primeru, kada se šalje niz u neku metodu ustvari se šalje njegova referenca. To znači da će sve izmene nad nizom u metodi uticati na originalni niz. Ako se neželi takvo ponašanje, ako se želi da izmene u metodi ne utiču na originalni niz, možemo da se pošalje klon tog niza. Zahvaljujući metodi **clone**, koja postoji u okviru objekta niza, može da se pošalje klon tog niza u pozivMetoda, tako da će sve izmene u metodi ostati samo na nivou te metode, odnosno neće uticati na originalni niz. Ovo bi, praktično, predstavljalo slanje niza po vrednosti.

```
pozivMetoda(intovi.clone());
private void pozivMetoda(Integer ... intovi) {
	Arrays.sort(intovi);
	for (int vT : intovi)
	{
		System.out.println("izVectora = " + vT);
	}
}
```

Dvodimenzionalni niz deklaracija

U Javi, višedimenzionalni nizovi su, u suštini, nizovi nizova. Kada je potrebno da se deklariše dvodimenzionalni niz, treba da se postave dve kockaste zagrade iza tipa koji niz treba da bude, i time je definisano da, u ovom slučaju promenljiva matrica, je deklarisan dvodimenzionalni niz.

```
int[][] matrica;
```

Dvodimenzionalni niz konstrukcija

Kao i običan niz, i dvodimenzionalni niz mora da se konstruiše, odnosno mora da se definiše koliko elemenata sadrži. Za razliku od jednodimenzionalnog niza, ovde postoje dve dimenzije, pa svaka od tih dimenzija mora da se definiše, odnosno mora da se odredi koliko ima kolona a koliko redova niz.

```
int[][] matrica = new int[5][8];
```

Dvodomenzionalni niz inicijalizacija

U primeru je prikazana inicijalizacija dvodimenzionalnog niza početnim vrednostima. Prvo je izvršena deklaracija niza - matrica, a nakon toga i njegova konstrukcija, kojom je definisano da niz ima 5*8 elemenata. Nakon toga je kreirana dvostruka **for** petlja, pri čemu unutrašnja **for** petlja prolazi kroz kolone, a spoljna kroz redove. Na taj način je obezbeđen mehanizam da se prođe kroz svaki elemenat ovog dvodimenzionalnog niza. Svaki elemenat ovog dvodimenzionalnog niza se puni proizvodom njegovih indeksa (tako je u ovom primeru).

```
int[][] matrica = new int[5][8];
for (int i = 0; i < 5; i++)
      for (int j = 0; j < 8; j++)
      {
            matrica[i][j] = i * j;
      }
```

Deklaracija konstrukcija i inicijalizacija

Kao i u slučaju jednodimenzionalnih nizova, i u slučaju višedimenzionalnih nizova može se izvršiti deklaracija, konstrukcija i inicijalizacija u jednoj jedinoj naredbi. U primeru je prikazana deklaracija niza matricaS, koja se puni sa 2*3 brojem elemenata. Ovde je prikazano da, prilikom inicijalizacije, mora da se upotrebi još jedna vitičasta zagrada da bi se odvojili elementi podniza od elemenata osnovnog niza. Kao što je prikazano, zahvaljujući ovim vitičastim zagradama odvaja se niz koji se nalazi u okviru niza. Rečeno je da su u Javi matrice u suštini nizovi nizova, što se i vidi u ovom primeru inicijalizacije. Da, kojim slučajem, imamo trodimenzionalni niz, onda bi umesto svake ove reči (umesto svakog ovog konkretnog elementa) imali još jednu vitičastu zagradu, u okviru koje bi se nalazio konkretan niz trećeg nivoa.

```
String[][] matricaS = {{"mi", "vi", "oni"}, {"ja", "ti", "on"}};
```