PREDMET

Osnove Java Programiranja

Cas 13-14

KONTROLNE STRUKTURE

Oktobar 2011.

SADRŽAJ

Čas 7 - 8

Kontrolne strukture - Petlje

Petlje

Petlje su kontrolne strukture pomoću kojih možemo da izvršimo višestruko ponavljanje jednog dela koda. Petlje se koriste u nekoliko karakterističnih situacija:

- kontrola toka
- rad sa nizovima
- rad sa kolekcijama
- iteracije
- čekanje na ispunjenje uslova
- itd...

Petlje – tipovi

Postoji nekoliko tipičnih petlji: **for**, nova for (predstavljena u Javi 1.5), **while** i **do-while** petlja. Svaka od ovih petlji je optimizovana za određenu primenu.

- for
- for (nova)
- while
- do-while

For petlja

U ovom primeru je prikazana **for** petlja. **For** petlja se sastoji od naredbe **for**, tri dela **for** naredbe u običnoj zagradi, a u vitičastoj zagradi se nalaze naredbe koje se vrte u okviru petlje. Tri dela koja **for** petlja ima su odvojena pomoću oznake ; . Prvi deo **for** petlje od otvorene obične zagrade do prve; služi za inicijalizaciju brojača. Drugi deo, koji se nalazi između dve oznake ; , je uslov. Treći deo, od ; do zatvorene obične zagrade, je uvećavanje, odnosno umanjenje brojača. U prvom delu petlje, koji služi za inicijalizaciju brojača, se nalazi deklaracija brojača (u ovom slučaju **int** i) i njegova inicijalizacija na početnu vrednost (u ovom slučaju na 0). U drugom delu je uslov. Dokle god je uslov ispunjen, odnosno dokle god je izraz između dve oznake ; **true**, petlja će da se izvrsava (vrti). U trenutku kada uslov postane netačan napustiće se petlja. U ovom slučaju uslov je da je brojač manji od dužine niza. To znači da će u ovom slučaju **for** petlja proći kroz sve elemente niza. Treći deo petlje služi za uvećavanje, odnosno umanjenje brojača. U ovom slučaju i++ označava da će u svakom koraku petlje brojač da se poveća za jedan.

Tok izvršenja for petlje:

U prvom trenutku se definiše vrednost brojača na nulu, zatim se proverava da li je ispunjen uslov (u ovom slučaju da li je i manje od dužine niza). Ako jeste manji od dužine niza prolazi se kroz telo petlje. To su sve naredbe od otvorene do zatvorene vitičaste zagrade. Kada se dođe do kraja, odnosno do zatvorene vitičaste zagrade, izvršava se treći deo petlje koji povećava, odnosno umanjuje brojač (u ovom slučaju i++ koji uvećava brojač za 1). Ponovo se proverava uslov, ponovo se prolazi kroz telo petlje, povećava se za 1 brojač petlje, proverava uslov, prolazi kroz telo petlje... To se ponavlja dokle

god je uslov tačan. U trenutku kada izraz iz uslova postaje netačan, odnosno **false**, napušta se petlja i izvršava se prva naredba nakon zatvorene vitičaste zagrade.

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

For petlja bez bloka naredbi

Specijalni slučaj **for** petlje je kada se u bloku naredbi nalazi samo jedna naredba koja se ponavlja. U tom slučaju mogu da se izostave oznake za blok (otvorena i zatvorena vitičasta zagrada).

Napomena:

Izostavljanje vitičastih zagrada (bloka naredbi) je loša programerska praksa. Dobra programerska praksa nalaže da se uvek stavljaju vitičaste zagrade, jer ako naknadno bude potrebno da se dodaje još naredbi u okviru bloka, neće se desiti nepredviđena situacija. U slučaju da nema vitičastih zagrada, dodavanje još jedne naredbe može da prouzrokuje grešku, jer se ta naredba neće ponavljati.

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

For petlja – bez inicijalizacije brojača

U slučaju da je brojač neka promenljiva koja je prethodno definisana, mi moguće je da se u okviru **for** petlje, izostavi deo za definiciju brojača.

```
int j = niz.length;
for (; j >= niz.length; j--)
{
      niz[j] = j * j;
}
```

For petlja – bez povećanja brojača

Brojač može da se uvećava u delu bloka naredbi (ili da se umanjuje), i u toj situaciji nije neophodno imati uvečivač, odnosno umanjivač brojača u delu **for** naredbe.

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

For petlja – bez uslova

Specijalna verzija **for** petlje je izostavljen uslov. U datom primeru, u delu **if** (i < niz.length) postoji naredba **break**, **break** će da napravi nasilni prekid rada **for** petlje i da prekine tok programa ivodeci ga van petlje. Dakle, uslov postoji, ali nije u delu koji je predviđen u **for** petlji, već se nalazi u samom telu petlje.

```
for (int i = 0; ; i++)
{
      if (i < niz.length)
            break;
```

```
        niz[i] = i * i;
}
```

For petlja – bez dva dela

Moguće je u okviru **for** petlje izostaviti i više od jednog dela. Dati primer je za situaciju kada je izostavljno dva dela: inicijalizacija brojača i inkrementacija brojača.

```
int j = niz.length;
for (; j >= niz.length;)
{
        niz[j] = j * j--;
}
```

Beskonačna for petlja

Specijalni slučaj **for** petlje je i kada se petlja vrti u beskonačno. Moguće je kreirati beskonačnu **for** petlju tako što se izostave sva tri dela **for** petlje. Odnosno, u okviru običnih zagrada postojaće samo dve oznake ; . U tom slučaju petlja se vrti u beskonačno.

Napomena:

Beskonačne petlje su specijalna varijanta petlji, koje se često karakterišu kao negativna pojava jer ne želimo da nam se program zaglavi u nekoj beskonačnoj petlji. Međutim, u Javi postoje situacije kada je dozvoljena beskonačna petlja, odnosno kada su one izuzetno korisne. Beskonačne petlje imaju svoju ulogu u delu višenitnog programiranja.

```
for(;;)
{
        //beskonačna petlja
}
```

Nova for petlja (java 5)

Nova **for** petlja je predstavljena tek u verziji Jave 1.5, i ona je idealna petlja za prolaženje kroz nizove i kolekcije. Nova **for** petlja se sastoji iz dva dela. U prvom delu se definiše kog je tipa element niza ili kolekcije, a u drugom delu petlje (posle znaka :) se navodi instanca niza ili kolekcije kroz koji se prolazi.

Funkcionisanje petlje:

Nova **for** petlja funkcioniše na sledeći način: promenljiva a uzima vrednost prvog elementa niza, prolazi kroz telo petlje, zatim promenljiva a uzima vrednost drugog člana niza, prolazi kroz telo petlje, zatim uzima vrednost trećeg člana niza, i prolazi kroz petlju dokle god ne uzme vrednost i poslednjeg člana niza. Nakon toga izlazi van petlje. Na isti način **for** petlja prolazi i kroz kolekcije. Nova **for** petlja je specijalno optimizovana za prolazak kroz nizove i kolekcije, i idealna je za te poslove.

```
for(int a : niz)
{
        a=a*a;
}
```

while petlja

While petlja je petlja koja je specijalno optimizovana za prolazak kroz petlju dokle god je neki uslov ispunjen. **While** petlja može da se svede na **for** petlju, i obrnuto: **for** petlja može da se svede na **while** petlju. Na primeru, u primeru gde je izostavljeno dva dela **for** petlje praktično je **for** petlja svedena na

while petlju, odnosno prolaženje kroz nizove pomoću indeksa niza. Ovo ne bi trebalo raditi. Java ima specijalno optimizovan kod svake petlje za određenu vrstu poslova, tako da treba petlje koristiti u zavisnosti od toga šta treba uraditi. Ako postoji samo ispunjenje uslova, onda je idealno koristiti **while** petlju dok, ako postoji brojač, onda je idealno koristiti **for** petlju. **While** petlja ima samo jedan deo, i to je uslov. Dokle god je uslov ispunjen (dokle god je izraz ili ono što se nalazi između otvorene i zatvorene zagrade iza naredbe **while**, je **true**), dotle se petlja izvršava (vrti). U trenutku kada taj izraz ili vrednost postane **false**, izlazi se van petlje.

```
int k=0;
while(k<niz.length)
{
	niz[k]=k*k;
	k++;
}
```

Beskonačna while petlja

Kao što **for** petlja ima svoju beskonačnu varijantu, tako i **while** petlja ima svoju beskonačnu varijantu. S' obzirom na to da je **while** petlja uslova, potrebno je da uslov bude uvek ispunjen da bi se petlja vrtela u beskonačno. Naravno, stavljanjem reči **true** u **while** petlju, postiže se da se petlja vrti u beskonačno.

```
while(true)
{
	//beskonačna petlja
}
```

do – while petlja

Do – **while** petlja je specijalna verzija **while** petlje gde se uslov proverava pri dnu. Kao i kod **while** petlje, i ovde postoji samo provera uslova. Za razliku od **while** petlje, u **do** – **while** petlji taj uslov se proverava na kraju petlje. Razlika između **do** – **while** petlje i svih ostalih petlji je u tome što se **do** – **while** petlja uvek izvršava bar jednom. Kod svih ostalih petlji se uslov proverava na početku, tako da, ako uslov nije ispunjen, može da se desi da kod ni jednom ne bude izvršen. U slučaju **do** – **while** petlje ta situacija ne može da se desi, jer kod koji se nalazi u okviru petlje se uvek bar jednom izvršava. To je ujedno i glavna razlika između **do** – **while** petlje i ostalih petlji. Treba obratiti pažnju da na kraju **while** petlje (nakon naredbe **while** i uslova) postoji ; . Razlog za postojanje ; na ovom mestu je oznaka da je tu kraj petlje. Kod svih ostalih petlji (**while** i **for**) ne postoji ; nakon petlje, zbog toga što nakon njih ide blok naredbi koje se izvršavaju. U slučaju **do** – **while** petlje taj blok se nalazi iznad naredbe **while**, i samim tim je neophodno da se označi gde je kraj. U primeru je prikazan ulaz brojeva i sabiranje tih brojeva dokle god klijent unosi brojeve. U trenutku kada klijent napiše **quit**, prekida se sa daljim unosom. Ovo je tipičan primer **do** – **while** petlje, gde je prvo neophodno da klijent unese broj da bi moglo da se ustanovi da li je korisnik uneo broj ili je uneo naredbu za kraj. Dakle, ne može to da se proverava pre nego što korisnik unese informaciju, i zbog toga je (konkretno u ovom primeru) idealno iskoristiti **do** – **while** petlju. **Do** – **while** petlja se često koristi za iteraciju u inžinjerskim poslovima i za slične situacije gde je neophodno prvo nešto izračunati, pa tek nakon toga proveriti uslov. **Do** – **while** petlja može da se simulira pomoću **for** ili **while** petlje, ali onda mora da se duplira kod iz petlje još jednom pre same petlje, a dupliranje koda nije dobro. Kada treba da se koristi **do** – **while** petlja nepoželjno je svoditi je na neku drugu petlju, već treba koristiti takvu kakva jeste. Generalno pravilo za Javu je da svaku petlju koristite za ono za šta je ona idealna.

```
String unos;
int sumaBrojeva;
do
{
```

```
        sumaBrojeva += Integer.parseInt(unos);
        unos = JOptionPane.showInputDialog("Uneti neki broj ili QUIT za
        kraj:");
}while(!unos.equals("QUIT"));
```

Ugnježdene petlje

Moguće je napraviti petlje u petlji. U primeru su prikazane petlje u petlji u okviru kojih se računa početna vrednost matrice. Ugnježdavanje petlji nije preporučljivo, osim u situacijama kada to zadatak zahteva, kao što je to slučaj u ovom primeru:

```
int[][] matrica = new int[5][3];
for (int i = 0; i < 5; i++)
{
        for (int j = 0; j < 3; j++)
        {
                matrica[i][j] = i * j;
        }
}
```

Kontrola petlji

Postoji nekoliko karakterističnih naredbi koje se koriste u kombinaciji sa petljama kako bi se izvršila dodatna kontrola rada petlje. Te naredbe su:

- Pomoću naredbe **break** se prekida tok petlje i izlazi se iz petlje.
- Naredba **continue** prekida petlju u trenutku kada dođe do naredbe **continue** i vraća je na početak sa uzimanjem nove vrednosti.
- Naredba **return** je naredba bezuslovnog skoka, odnosno vraćanje vrednosti pozivaru. Pomoću naredbe **return** prekidamo trenutni rad petlje i vraćamo se u metodu koja je pozvala ovu petlju.
- **System.exit(0)** prekida rad programa i gasi virtuelnu mašinu i program u njoj. Ovo je najdrastičniji način prekida petlje. Naredbe za prekid rada petlji nisu preporučljive. Treba uvek težiti boljem dizajnu koda tako da ne postoji potreba za ovim naredbama. Upotreba ovih naredbi ukazuje na loš dizajn koda.
- Jedini izuzetak je naredba return.

Naredbe za kontrolu toka petlje:

- break
- continue
- return
- System.exit(0);

Primer za break

Kao što vidi u primeru, pomoću **break** naredbe je prekinut tok petlje i petlja je u tom trenutku napuštena. Naravno, ovu istu petlju je moguće drukčije dizajnirati, tako da **break** naredba uopšte ne bude potrebna.

```
for (int i = 0; ; i++)
{
```

```
        if (i < niz.length)
              break;
        niz[i] = i * i;
}
```

Primer za return

U prikazanom primeru metode izračunava se suma. Napravljena je petlja u kojoj se računa suma elemenata niza. Pomoću naredbe **return** je prekinut tok ove petlje i vraćena je suma u trenutku kada se došlo do kraja niza. Naravno, kao i u prethodnom slučaju, ovo je moguće drukčije izvesti, bolje dizajnirati kod tako da se ne javi potreba za upotrebom naredbe **return**, odnosno naredba **return** bi se nalazila samo jednom na kraju metode.

```
publi int izracunajSumu(int[] niz)
{
int sumabrojeva;
for (int i = 0; ; i++)
{
        if (i < niz.length)
              return sumabrojeva;
        sumabrojeva += niz[i];
        }
        return 0;
}
```

Primer za continue

U datom primeru se vidi upotreba naredbe continue za kontrolu toka petlje. Na početku postoji deklaracija početnih podataka koji su potrebni: unosa kao stringa i sume brojeva kao ceo broj. Za potrebe ovog primera korišćena je **do** – **while** petlja u okviru koje postoji dijalog gde korisnik unosi broj ili naredbu **quit**. U slučaju da korisnik ne unese ni naredbu **quit** ni nekakav broj, već unese neko slovo ili reč koja nije **quit**, biće prekinut rad programa u tom trenutku, i ponovo će se zahtevati unos. Dakle, neće se ići na izračunavanje sume i na parsiranje, već će automatski biti prekinut rad programa. Kao i u prethodnim slučajevima, i ovde je moglo da se sve to bolje dizajnira tako da naredba **continue** ne bude potrebna.

```
String unos;
int sumaBrojeva;
do
{
        unos = JOptionPane.showInputDialog("Uneti neki broj ili QUIT za
        kraj:");
        if (Character.isLetter(unos.charAt(0)) && unos.charAt(0) != 'Q')
              continue;
        sumaBrojeva += Integer.parseInt(unos);
} while (!unos.equals("QUIT"));
```