Kontrola toka programa

Autori:
Goran Savić
Milan Segedinac

1. Relacioni operatori

Relacioni operatori se koriste za poređenje vrednosti. U tabeli su navedeni Java relacioni operatori.

Simbol operatora	Opis
==	Poredi da li su operandi jednaki
>	Poredi da li je levi operand veći od desnog
<	Poredi da li je levi operand manji od desnog
>=	Poredi da li je levi operand veći ili jednak desnom operandu
<=	Poredi da li je levi operand manji ili jednak desnom operandu
!=	Poredi da li su operandi različiti

Rezultat primene relacionog operatora je logička (boolean) vrednost. Dakle, tačno ili netačno. Dat je primer korišćenja jednog relacionog operatora.

```
int a = 3;
int b = 4;
boolean x = a > b;
```

U promenljivu x će biti upisana vrednost false jer vrednost promenljive a nije veća od vrednosti promenljive b.

2. Logički operatori

Nad logičkim promenljivima moguće je vršiti operacije korišćenjem logičkih operatora. Rezultat primene logičkog operatora je takođe logička vrednost. Osnovne logičke operacije su

- Konjukcija - zove se i logičko I (eng. *AND*). Odgovarajući Java operator je &&
- Disjunkcija - zove se i logičko ILI (eng. *OR*). Odgovarajući Java operator je ||
- Negacija – zove se i logičko NE (eng. *NOT*). Odgovarajući Java operator je !

U tabeli su prikazane vrednosti koje konjukcija vraća u zavisnosti od vrednosti operanada.

Prvi operand	Operator	Drugi operand	Rezultat
true	&&	true	true
true	&&	false	false
false	&&	true	false
false	&&	false	false

Kao što vidimo, logičko I vraća istinitu vrednost samo ako su oba operanda istinita.

U narednoj tabelu su prikazane vrednosti koje disjunkcija vraća u zavisnosti od vrednosti operanada.

Prvi operand	Operator	Drugi operand	Rezultat
true	||	true	true
true	||	false	true
false	||	true	true
false	||	false	false

Dakle, logičko ILI vraća istinitu vrednost kada je bilo koji od operanada istinit.

Konačno, prikazan je rezultat negacije. Treba primetiti da je logičko NE unarni operator i da se primenjuju nad jednim operandom.

Operand	Operator	Rezultat
true	!	false
false	!	true

Ne postoji prepreka da izraz sadrži više logičkih operatora. Na primer izraz

```
false && (true || false)
```

vraća vrednost false.

Važno je napomenuti da operatori && i || vrše takozvanu skraćenu konjukciju, odnosno disjunkciju. To podrazumeva da program evaluira vrednost logičkog izraza samo dotle dok jednoznačno ne utvrdi njegov rezultat. Tako u prethodno prikazanom primeru, kada se utvrdi da je prvi operand false i da je nad njim primenjen operator &&, program neće preuzimati naredne vrednosti u izrazu jer bez obzira na naredne vrednosti, rezultat ovog izraza će sigurno biti false (jer false u konjukciji sa bilo kojom vrednošću daje rezultat false).

3. Grananje programa

U do sada pokazanim primerima, naredbe programa su se izvršavale redom kojim su napisane. Moguće je kontrolisati tok programa, tako da u zavisnosti od podataka u programu, program izvršava odgovarajuće delove koda. Jedan način kontrole toka programa je grananje programa. U programu se može definisati više grana, pri čemu tok programa prelazi u određenu granu u zavisnosti od podataka programa. Pokazaćemo dva načina za realizaciju grananja programa u Javi – korišćenjem if, odnosno switch naredbi.

Naredba if

Naredba if, u zavisnosti od ispunjenosti određenog uslova izvršava određeni deo koda programa. Uslov je predstavljen izrazom koji vraća logičku (boolean) vrednost. Ova naredba je ilustrovana na narednom primeru koji koristi neku celobrojnu promenljivu *broj*.

```
if (broj > 0) {
      System.out.println("Broj je: " + broj);
      System.out.println("Broj je pozitivan.");
}
```

U prikazanom primeru, kod koji ispisuje tekst na ekran se ispisuje samo ako logički izraz u if naredbi vrati rezultat true. Nema prepreke da u if naredbi stoji i složeni logički izraz koji sadrži prethodno objašnjene logičke operatore.

Treba primetiti da su stavljene vitičaste zagrade nakon if naredbe, jer je reč o novom bloku koda. Sve naredbe u tom bloku koda će se redom izvršiti ako je uslov unutar if naredbe istinit. Ako blok sadrži samo jednu liniju koda, onda nije obavezno pisati vitičaste zagrade.

Prethodni primer pokazuje jednu granu programa u koju program ulazi ako je uslov u if naredbi ispunjen. Moguće je u if naredbi definisati i granu u koju će program preći ako uslov nije ispunjen. Ovo se vrši dodavanjem else dela if naredbi. U nastavku je prethodni primer proširen else naredbom.

```
if (broj > 0) {
      System.out.println("Broj je: " + broj);
      System.out.println("Broj je pozitivan.");
} else {
      System.out.println("Broj nije pozitivan.");
}
```

Dakle, else grana se izvršava ako uslov nije ispunjen (ako je broj manji ili jednak nuli u prikazanom primeru).

Moguće je definisati i dodatne uslove za prelazak u grane ako uslov unutar zaglavlja if naredbe nije ispunjen. To se vrši dodavanjem else if delova u if naredbu. Moguće je dodati proizvoljan broj else if delova. Ovo je ilustrovano proširenjem prethodnog primera posebnom granom za slučaj kada je broj nula.

```
if (broj > 0) {
      System.out.println("Broj je: " + broj);
      System.out.println("Broj je pozitivan.");
} else if (broj == 0) {
      System.out.println("Broj je nula.");
}else {
      System.out.println("Broj je negativan.");
}
```

Naredba switch

Ako u zavisnosti od vrednosti neke promenljive, program prelazi u određenu granu, ovo se može realizovati nizom if, else-if naredbi. Ako je broj ovakvih grana relativno veliki, čitljiviji način za realizaciju iste funkcionalnosti je korišćenje naredbe switch. Na primer kod prikazan u narednom primeru može biti zamenjen switch naredbom.

```
if (brDana == 1) {
      System.out.println("Ponedeljak");
} else if (brDana == 2) {
      System.out.println("Utorak");
} else if (brDana == 3) {
      System.out.println("Sreda");
} else if (brDana == 4) {
      System.out.println("Cetvrtak");
} else if (brDana == 5) {
      System.out.println("Petak");
} else {
      System.out.println("Neradni dan");
}
```

Ista funkcionalnost realizovana putem naredbe switch data je u narednom primeru.

```
switch (brDana) {
      case 1:
            System.out.println("Ponedeljak");
            break;
      case 2:
```

```
            System.out.println("Utorak");
            break;
      case 3:
            System.out.println("Sreda");
            break;
      case 4:
            System.out.println("Cetvrtak");
            break;
      case 5:
            System.out.println("Petak");
            break;
      default:
            System.out.println("Neradni dan");
}
```

Dakle, naredba switch ispituje vrednost prosleđene promenljive (brDana u prikazanom primeru). U zavisnosti od vrednosti te promenljive, program ulazi u jednu od grana. Svaka grana je definisana case blokom. Za svaki case blok se navodi vrednost koju promenljiva mora da ima da bi program ušao u tu granu. U okviru case bloka, može da bude više naredbi. Važno je naglasiti da kada program uđe u odgovarajući case blok, on redom izvršava naredbe iz tog bloka i ide dalje prelazeći i na naredbe narednog case bloka sve dok ne naiđe na naredbu break ili na kraj switch naredbe. Pošto najčešće ne želimo da nakon izvršavanja jednog case bloka program pređe na naredni case blok, na kraju svakog case bloka se piše naredba break. Konačno, ako vrednost promenljive ne odgovara nijednom case bloku, program će ući u blok definisan labelom default. Ovaj deo switch naredbe nije obavezan.

Ako koristimo switch naredbu, promenljiva čiju vrednost ispitujemo mora da bude tipa byte, short, integer, char ili String. Pored navedenih, moguće je da vrednost bude tipa enumeracija. Ovaj tip podatka je objašnjen u nastavku.

4. Enumeracija

Enumeracija je još jedan složeni tip u programskom jeziku Java. Ovaj tip se najčešće koristi u situacijama kada promenljiva može imati određenu vrednost iz skupa diskretnih vrednosti. Da bi se ovo realizovalo, potrebno je najpre definisati novi tip i navesti moguće vrednosti. Ovo se radi korišćenjem rezervisane reči enum. Primer pokazuje uvođenje novog tipa koji predstavlja skup mogućih vrednosti za ime dana u nedelji.

```
enum Dan {Ponedeljak, Utorak, Sreda, Cetvrtak, Petak, Subota, Nedelja}
```

Za ovako uveden tip, moguće je definisati promenljivu ovog tipa. Promenljiva može da dobije jednu od diskretnih vrednosti definisanih u tipu.

```
Dan d = Dan.Sreda;
```

Zašto koristiti enumeraciju? Iz dva razloga. Enumeracija povećava čitljivost koda. Bez enumeracije, za oznaku dana u nedelji bismo mogli izabrati brojnu vrednost. Tada bi prethodni primer ovako izgledao.

```
int d = 3;
```

Sada je smanjena čitljivost koda jer nije očigledno da se broj 3 odnosi baš na sredu. Drugi razlog zašto se koristi enumeracija je da bi se promenjiva koja može imati samo određeni skup vrednosti ograniči upravo na taj skup. U primeru sa korišćenjem int promenljive, nema prepreke da se za oznaku dana upiše bilo koji broj (npr. 43), što nije validan dan u nedelji.