3 Grananje

Grananje

Do sada smo videli kako da prikupimo podatke od korisnika, kako da ih obradimo i ispišemo na standardni izlaz. Primetili smo da se izvršavanje našeg koda vrši linija-po-linija odozgo nadole. Sa dosadašnjim znanjem probajmo da rešimo naredni problem (napravimo program):

Zadatak 3.1 (Punoletstvo)**.**

Napisati program koji prima broj godina korisnika i ispisuje odgovarajuću poruku:

- "Punoletni ste"; ukoliko korisnik ima više od 18 godina
- "Niste punoletni"; ukoliko korisnik ima strogo manje od 18 godina

Ne preostaje nam ništa drugo nego da krenemo redom:

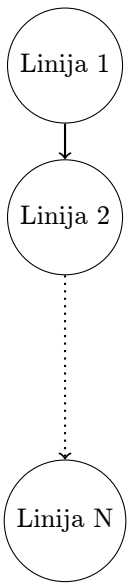
Punoletsvo

```
1  package neki.paket;
2
3  //Importujemo potrebnu biblioteku
4  import java.util.Scanner;
5
6  public class Punoletsvo {
7
8      public static void main(String[] args) {
9          //Kreiramo skener
```

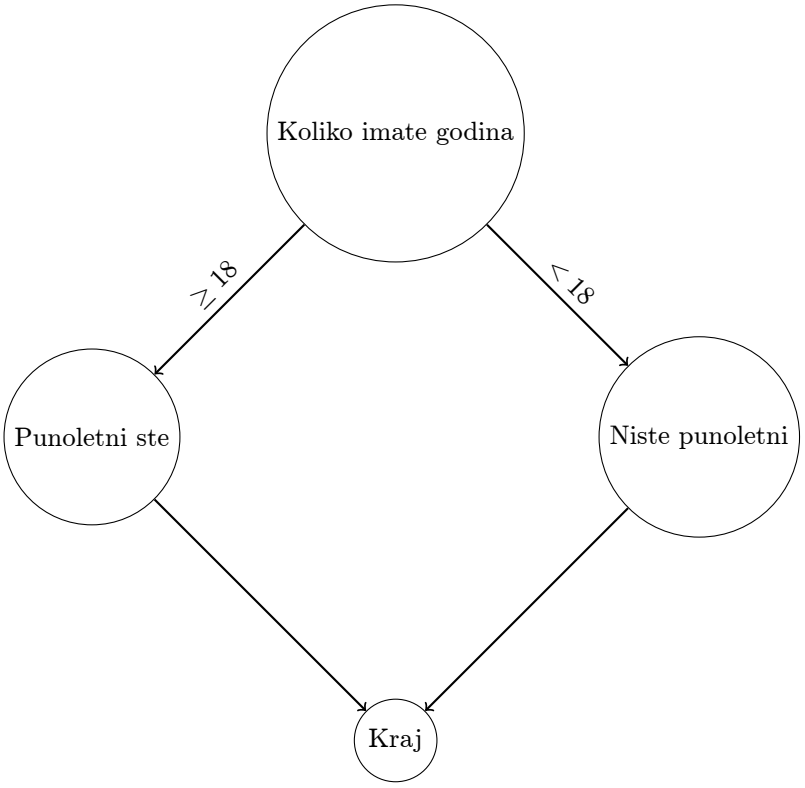
```
10          Scanner sc = new Scanner(System.in);
11
12          //Ispisujemo zahtev
13          System.out.print("Unesite broj godina: ");
14          //Prikupljamo podatak
15          int brojGodina = sc.nextInt();
16          //Ispisujemo poruku
17          System.out.println("Punoletni ste");
18          System.out.println("Niste punoletni");
19      }
20  }
```

```
Input: 25
Output:
Punoletni ste
Niste punoletni
---
Input: 5
Output:
Punoletni ste
Niste punoletni
```

Kako se program izvršava red-po-red, to ispisuje obe poruke, prvo onu u redu 17, a zatim onu u redu 18, nezavisno od toga koji broj godina je unet. Grafički prikazano naš kod se izvršava:



A u našem slučaju želimo nešto poput:



Da bi smo ovaj efekat postigli, potrebno je da se naš kod **grana**, tj. da se delovi koda izvršavaju samo kada su odredjeni uslovi ispunjeni. Naredbe koje nam ovo omogućavaju se nazivaju **naredbe grananja** i u Javi se mogu videti u više oblika.

Blokovi koda

Pre nego što predjemo na proces grananja, potrebno je da definišemo novi pojam - **blok koda**.

Definicija 3.1 (Blok koda)**.**

Blok koda je skup naredbi učauren izmedju vitičastih zagrada `{ }`

Primer:

```
1  package ime.paketa;
```

```java

public class NekaKlasa { //Ovo je početak glavnog bloka koda za ovu klasu
/*
On označava da sve što se nalazi izmedju ovih vitičastih zagrada pripada klasi 'NekaKlasa'
*/

    public static void main(String[] args) { //novi blok kod

        { //Blok koda
            int x = 5;
            int y = 10;
            int z = x + z;
        }

        { //Jos jedan blok koda
            String s = "Zdravo Svete!";
        }

        { //U ovom bloku koda ...
            double r = 2;
            { //... postoji još jedan blok koda
                double povrsinaKruga = 3.14 * r * r;
                System.out.println(povrsinaKruga);
            }
        }
    }
}
```

Primetimo da jedan blok koda može u sebi da sadrži druge blokove koda (koji u sebi mogu da sadrže nove blokove koda itd..)
Za dva blok koda možemo da kažemo da su **paralelni** ili **ugnježdeni** u zavisnosti od njihovog medjusobnog poretka:

```
{ //Blok kod 1
 ....
}

{ //Blok kod 2
 ....
 { //Blok kod 3

 }
}
```

- Blok kod 1 je paralelan sa blokom koda 2
- Blok kod 3 je ugnježden u blok kod 2
- Blok kod 1 i blok kod 3 nisu ni u kakvom poretku

Još jedan važan pojam je **vidljivost** (promenjivih, objekata itd.).

Definicija 3.2 (Vidljivost)**.**

Vidljivost predstavlja sposobnost blok koda da pristupi odredjenoj promenjivoj, objektu, funkciji ...
Promenjiva x je vidljiva datom bloku koda akko:

1. x je definisana (deklarisana) unutar tog blok koda
2. x je definisana (deklarisana) unutar nekog blok koda koji sadrži zadati blok kod, pre početka zadatog blok koda

Pored ovoga, vrlo je važno napomenuti da jedna poremnjiva može biti deklarisana najviše jedanput unutar jednog bloka, uključujući i sve njegove pod-blokove, ali sa druge strane, u paralelnim blokovima je moguće deklarisati istu promenjivu (promenjivu sa

istim imenom) više puta.
Primer:

vidljivost

```
1  package neki.paket;
2
3  public class Vidljivost {
4
5      public static void main(String[] args) {
6
7          { //Blok koda 1
8              int x = 5;
9              int x = 3; //Greska, x je vec deklarisana unutar bloka 1
10         }
11
12         { //Blok koda 2
13             System.out.println(x); //Greska, x nije vidljiva u bloku 2 jer je ona paralelan sa blokom 1
14             int x = -17; //Dozvoljeno jer u ovom bloku ne postoji promenjiva sa imenom 'x'
15             { //Blok koda 3
16                 double x = 3.14; //Greska, promenjiva sa imenom 'x' vec postoji u bloku koda koji sadrži ovaj blok koda
17             }
18         }
19     }
20 }
```

IF naredba

Sa svim ovim stvarima dobro definisanim, možemo konačno krenuti sa prvom naredbom toka, `if` naredbom. `if` naredba je prva naredba sa kojom se susrećemo koja se ne pridržava pravila

da se završava sa simbolom `;`, vec ona zahteva definisanje novog kod bloka. Njen šablon pisanja je sledeći:

```
if (<uslov>) {
    //Linija koda 1
    //Linija koda 2
    ...
    //Linija koda N
}
```

Dakle, komanda počinje sa ključnom rečju `if`, prateći sa uslovom čije krajnje rešenje mora biti logički literal upisanim u obične zagrade i zatim otvaranje novog kod bloka. Unutar tog kod bloka, navode se sve komande čije izvršavanje će se desiti ako i samo ako je navedeni uslov ispunjen, tj. ako je vrednost tog logičkog literala `true`.
Pogledajmo dva praktična primer:

pozitivanbroj1

```
package neki.paket;

public class PozitivanBroj {

    public static void main(String[] args) {
        int x = 3;
        System.out.println("Zdravo");
        if (x > 0) {
            System.out.println(x + " je pozitivan broj");
        }
        System.out.println("Dovidjenja");
    }
}
```

```
Output:
```

```
Zdravo
3 je pozitivan broj
Dovidjenja
```

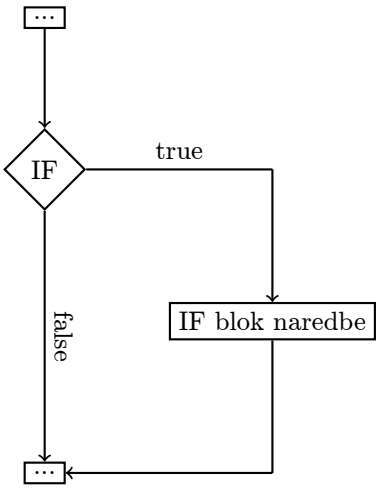
pozitivanbroj2

```java
package neki.paket;

public class PozitivanBroj {

    public static void main(String[] args) {
        int x = -100;
        System.out.println("Zdravo");
        if (x > 0) {
            System.out.println(x + " je pozitivan broj");
        }
        System.out.println("Dovidjenja");
    }
}
```

```
Output:
Zdravo
Dovidjenja
```

U prvom primeru vrednost broja 'x' je 3, te kada program dodje do linije 8 on evaluira izraz $3 > 0$, što je tačno i dodeljuje mu vrednost `true`, što znači da je uslov ispunjen pa se izvršava naredba u liniji 9. U tom momentu, izvršene su sve konde iz if bloka i kod nastavlja dalje sa svojim izvršavanjem, tj. izvršava intrukciju sa linije 11.
U drugom primeru vrednost boja 'x' je -100, evaluiranje izraza $-100 > 3$ je netačno, dakle dodeljena vrednost je `false`, pa program ne ulazi u if kod blok, tj. preskače liniju 9 i nastavlja dalje sa normalnim izvršavanjem koda, u našem slučaju, skače na liniju 11, izvršava je i zaustavlja se program.

Grafički prikazano, `if` naredba izgleda ovako:



Pokušajmo ponovo da rešimo zadatak za punoletstvo:

punoletstvo

```
1  package neki.paket;
2
3  //Importujemo potrebnu biblioteku
4  import java.util.Scanner;
5
6  public class Punoletsvo {
7
8      public static void main(String[] args) {
9          //Kreiramo skener
10         Scanner sc = new Scanner(System.in);
11
12         //Ispisujemo zahtev
13         System.out.print("Unesite broj godina: ");
14         //Prikupljamo podatak
15         int brojGodina = sc.nextInt();
16         //Pitamo se da li je unet broj godina veci od 18
17         if (brojGodina >= 18) {
18             System.out.println("Punoletni ste");
19         }
20         //Pitamo se da li je unet broj godina strogo
   manji od 18
```

```
21          if (brojGodina < 18) {
22              System.out.println("Niste punoletni");
23          }
24      }
25  }
```

Sada, naš kod neće izvršiti liniju 18 ukoliko je unet broj godina strogo manji od 18, ali u tom slučaju će izvršiti liniju koda 22. U suprotonom, ukoliko je unet broj godina veći ili jednak 18, izvršava se linija koda 18 a linija koda 22 se preskače.

U programiranju, postoji pojam koji se naziva **syntaxical sugar** (olakšana sintaksa). Olakšana sintaksa je najčešće kraći način da se napiše isti kod. Tako na primer, ukoliko unutar if bloka se nalazi samo jedna instrukcija, onda možemo zanemariti vitičaste zagrade skroz (ovo će važiti i za ostale naredbe sličnog izgleda). Primer:

olaksice

```
1  if (n > 5)
2      System.out.println("n je vece od 5");
3
4  if (d >= 2 * n) System.out.println("d je barem duplo vece
   od n");
```

Zadatak 3.2 (Legitimacija)**.**

Policajac zeli da legitimise sumnjivo lice. Da bi to učinio potrebno je da sumnjivo lice kod sebe ima ličnu kartu ili pasoš.
Napisati program koji od korisnika prima dva boolean-a, jedan koji označava da li sumnjivo lice sa sobom ima ličnu kartu a drugi koji označava da li sumnjivo lice sa sobom ima pasoš. Ispisati odgovarajuću poruku:

- "Sumnjivo lice je legitimisano"; ukoliko sumnjivo lice ima ličnu kartu ili pasoš
- "Sumnjivo lice nije legitimisano"; inače

Rešenje ovog zadatka, koristeći se dosadašnjim znanjem bi izgledao ovako:

sumnjivoLice

```
1  package neki.paket;
2
3  import java.util.Scanner;
4
5  public class SumnjivoLice {
6
7      public static void main(String[] args) {
8          Scanne sc = new Scanner(System.in);
9          boolean imaLicnuKartu = sc.nextBoolean();
10         boolean imaPasos = sc.nextBoolean();
11         if (imaLicnuKartu || imaPasos)
12             System.out.println("Sumnjivo lice je
   legitimisano");
13         if (!imeLicnuKartu && !imaPasos)
14             System.out.println("Sumnjivo lice nije
   legitimisano");
15     }
16 }
```

Primetimo da u govornom jeziku često koristimo reči poput "inače" ili "u suprotnom" da naznačimo slučaj kada nijedan od prethodnih uslova nije ispunjen (to smo uradili u formulaciji ovog zadatka), dok u realizaciji ovog programa morali smo eksplicitno da napišemo šta je taj "inače" slučaj - nema ličnu kartu i nema pasoš. Ovu spetku možemo zaobići sa narednim oblikom `if` grananja.

Else naredba

Ukoliko želimo da istaknemo slučaj kada uslov u `if` neredbi nije ispunjen koristimo ključnu reč `else`. Kako `else` znači 'inače', ona ne dolazi sa uslovom kao što je to slučaj za `if` naredbu, već samo definiše svoj poseban blok koda. Uopšteno, šablon izgleda:

```
if (<uslov>) {
    //Linija koda 1
    //Linija koda 2
    ...
    //Linija koda N
}
else {
    //Linija koda 1
    //Linija koda 2
    //Linija koda M
}
```

Gornji pseudokod se izvršava na sledeći način:

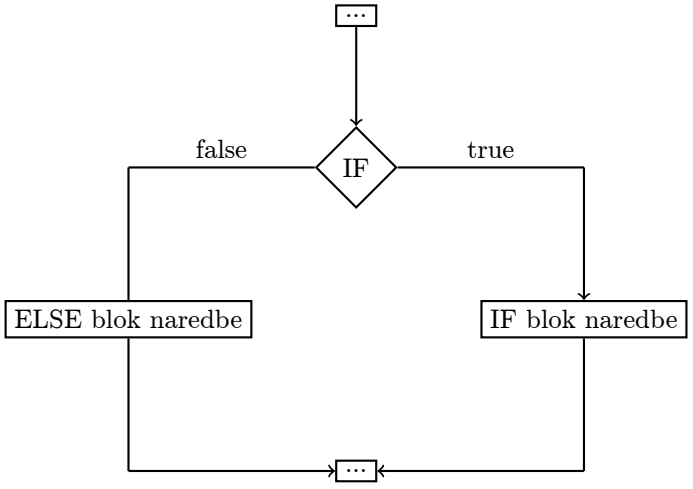
1. Ukoliko je zadati uslov ispunjen, program ulazi u if blok i izvršava linije koda 1 ... N, a zatim preskače čitav else blok.
2. Ukoliko zadati uslov nije ispunje, program preskače if blok kao i do sada, ali u ovom slučaju ulazi u else blok i izvršava linije koda 1 ... M.

Pre nego što krenemo na praktične primere, primetimo par činjenica:

- `if` blok koda će se izvršiti ako i samo ako `else` blok koda se neće izvršiti, odnosno `if` blok koda se neće izvršiti ako i samo ako će se izvršiti `else` blok koda.

- Za ma koji uslov, izvršiće se tačno jedan blok koda: `if` blok isključivo ili `else` blok

Grafički prikazano:



Odradimo ponovo zadatak sa punoletstvom:

Punoletsvo

```
1  package neki.paket;
2
3  import java.util.Scanner;
4
5  public class Punoletsvo {
6
7      public static void main(String[] args) {
8          Scanner sc = new Scanner(System.in);
9
10         System.out.print("Unesite broj godina: ");
11         int brojGodina = sc.nextInt();
12
13         //Pitamo se da li je unet broj godina veci od 18
14         if (brojGodina > 18) //tada je osoba punoletna
15             System.out.println("Punoletni ste");
16         else //Inace, broj je sigurno strogo manji od 18,
   pa osoba nije punoletna
17             System.out.println("Niste punoletni");
```

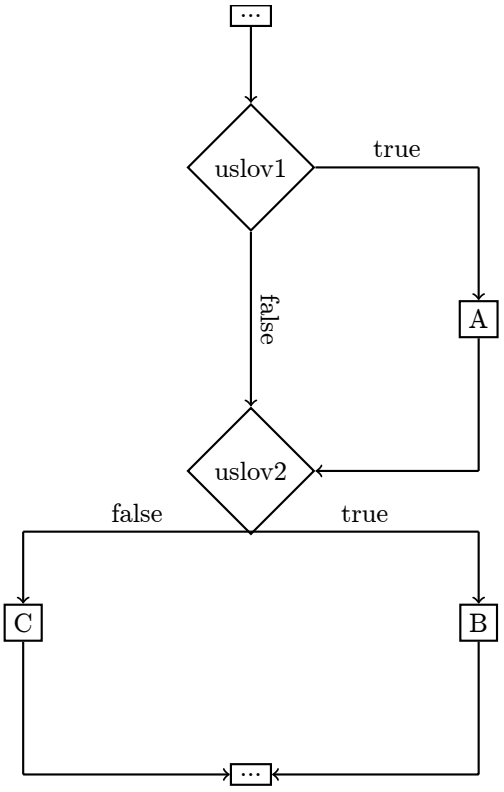
```
18      }
19  }
```

Slično ovome, možemo korigovati i zadatak 3.2.

Važno je napomenuti da se `else` naredba odnosi samo na *poslednju* `if` naredbu i mora se nalaziti odmah nakon kraja njegovog bloka koda!
Primer:

```
1  if (<uslov1>) {
2      A
3  }
4  if (<uslov2>) {
5      B
6  }
7  else {
8      C
9  }
```

U poslednjem primeru, `else` naredba se odnosi na onu `if` naredbu sa uslovom 2 i izvršiće se ako i samo ako nije zadovoljen uslov 2. Ova `else` naredba nema nikakve veze sa prvom `if` naredbom! Grafički prikazano to izgleda:



Ugnježdene IF naredbe

Blokove koda možemo posmatrati kao doredjene celine programa, ali osim par paravila oko restrikcije vidljivosti, oni su deo celokupnog programa kao i bilo šta drugo; ništa nas ne sprečavada da u njima imamo nove blokove koda, naredbe grananja ... Vodjeni tom logikom, možemo da zaključimo da unutar jednog if bloka, možemo umetnuti jos takvih if blokova. Ovaj postupak se naziva **ugnježdavanje if naredbi**. Možda je najjednostavnije videti njegovu svrhu na praktičnom primeru:

Zadatak 4.3 (Pozitivni-Neutralni-Negativni)**.**

Napisati program koji prima jedan realan broj i ispisuje odgovarajuću poruku:
"Pozitivan"; ukoliko je broj pozitivan
"Neutralan"; ukoliko je broj 0
"Negativan"; inače

Jedan od mnogih (ali podjednako validnih) rešenja ovog zadatka je:

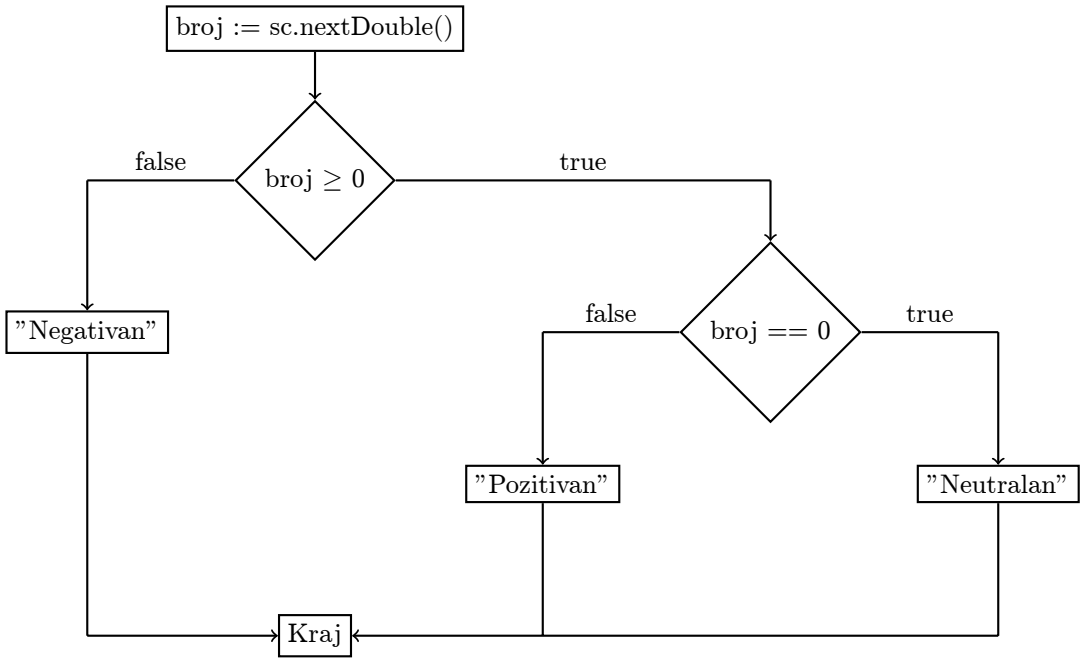
Pozitivni-Neutralni-Negativni

```java
package neki.paket;

import java.util.Scanner;

public class PozitivniNeutralniNegativni {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Unesite realan broj: ");
        double broj = sc.nextDouble();

        //Prvo se pitamo da li je broj veci ili jednak 0
        if (broj >= 0) {
            /*
            Ukoliko se nadjemo u ovoj grani koda, znamo sigurno da je broj 0 ili veci od toga
            Dakle, dalje se pitamo da li je bas 0 ili je veci od toga
            */
            if (broj == 0)
                System.out.println("Neutralan");
            else //Znamo da je broj >=0, a nije 0, dakle u pitanju je pozitivan broj
                System.out.println("Pozitivan");
        }
        else //Inace, broj je negativan
            System.out.println("Negativan");
    }
}
```

```
Input: 3.3
```

```
Output: "Pozitivan"
Input: 0.0
Output: "Neutralan"
Input: -0.001
Output: "Negativan"
```

Grafički prikaz prethodnog koda je:



IF-ELSE IF naredbe

Iz prethodnog zadatka lako se vidi da suštinski imamo 3 različita slučaja koja treba da obradimo. Jedan način da se ovo razreši je da objedenimo dva uslova u jedan, pa da ih naknadno zasebno razmatramo, kao što smo i uradili u prikazanom rešenju. Drugi ,mnogo jednostavniji i prirodniji način, je pomoću **if-else if** naredbi. Kao i sa ukljanjanjem nepotrebnih vitičastih zagrada, ovo je samo jedna sintaksička olakšica. Naime, moguće je ulančati više `if` naredbi tako da se više različitih slučaja razmatraju u zasebnim blokovima koda, bez potrebe za ugnježdavanjem. Uopšteno, **if-esle if** naredbe izgledaju nešto poput ovoga:

```
if (<uslov1>) {
```

```
    ....
}
else if (<uslov2>) {
    ....
}
...
else if (<uslovN>) {
    ....
}
else {
    ....
}
```

Dakle, ona se sastoji od jedne `if` naredbe praćenje sa `else if` naredbama koje idu sa svojim uslovima. Program ovakvu strukturu interpretira na sledeći način:

- Kreni od uslova 1. Ukoliko je on ispunjen, udji u taj blok, izvrši ga celog, a zatim skoči nakon `else` bloka ukoliko on postoji, odnosno poslednjeg `if else` bloka.
- Ukoliko uslov 1 nije ispunje, ponovi isti postupak postupak za uslov 2.
- Ukoliko nijedan uslov nije ispunje i ukoliko postoji `else` blok, izvrši njega. U suprotnom nastavi dalje.
  Primetimo da je poslednji `else` blok opcion i ne mora da postoji, slično kao što smo mogli da navedemo `if` komandu bez prateće `else` komande.

Grafički prikazano:

Sada možemo da korigujemo naše rešenje prethodnog zadatka u nešto čitkiji oblik:

Pozitivni-Neutralni-Negativni

```java
package neki.paket;

import java.util.Scanner;

public class PozitivniNeutralniNegativni {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Unesite realan broj: ");
        double broj = sc.nextDouble();

        //Prvo se pitamo da li je broj strogo veci od 0
        if (broj >= 0)
            System.out.println("Pozitivan");
        else if (broj == 0) //Ukoliko nije, pitamo se da li je bas 0
                System.out.println("Pozitivan");
        else //Inace, znamo da je broj negativan
            System.out.println("Negativan");
    }
}
```

Dakle, da rezimiramo:

- Za N `if else-if` naredbi bez `else` bloka izvršiće se *najviše* jedan blok koda.
- Za N `if else-if` naredbi sa `else` blokom izvršiće se *tačno* jedan blok koda.

Switch naredbe, break i default

Pored familije `if` naredbi grananja postoji još jedna familija naredbi koja je specializovana za slučajeve kada imamo veliki broj

jednakosnih uslova, tj. uslova poput `A = B`. Ta naredba se naziva `switch` naredba grananja tok, odnosno `switch - case`.

Zadatak 4.3 (Opisna ocena)**.**

Korisnik unosi ocenu koju je dobio na testu [1, 5]. Ukoliko korisnik unese ocenu strogo veću od 5, postaviti dobijenu ocenu na 5, a u koliko unese ocenu strogo manju od 1, postaviti je na 1. Ispisati na standardnom izlazu odgovarajuću poruku:

- "Odlican"; ukoliko je dobijena ocena 5
- "Vrlo dobar"; ukoliko je dobijena ocena 4
- "Dobar"; ukoliko je dobijena ocena 3
- "Dovoljan"; ukoliko je dobijena ocena 2
- "Nedovoljan"; ukoliko je dobijena ocena 1

opisnaOcena

```
1  package neki.paket;
2
3  import java.util.Scanner;
4
5  public class OpisnaOcena {
6
7      public static void main(String[] args) {
8          Scanner sc = new Scanner(System.in);
9          int ocena = sc.nextInt();
10         if (ocena > 5)
11             ocena = 5;
12         else if (ocena < 1)
13             ocena = 1;
14
15         if (ocena == 5)
```

```
16              System.out.println("Odlican");
17          else if (ocena == 4)
18              System.out.println("Vrlo dobar");
19          else if (ocena == 3)
20              System.out.println("Dobar");
21          else if (ocena == 2)
22              System.out.println("Dovoljan");
23          else //Sada znamo sigurno da je ocena 1
24              System.out.println("Nedovoljan");
25      }
26  }
```

Primetimo da u redovima 15-24 izdvajamo odredjene slučajeve pomoću operatora poredjenja, za koje vršimo odredjene radnje. Upravo je u ovakvim situacijama `switch` naredba jako pogodna. `Switch` naredba prima jednu promenjivu kao argument i dalje definiše po blokovima ponašanje koda u zavisnosti od vrednosti te promenjive. Njen šablon se zasnivna korišćenju ključnih reči `switch, case` a po potrebi i `default i break` i izgleda ovako:

```
1   switch (<identifikator_promenjive>) {
2       case <literal1>:
3           ....
4       case <literal2>:
5           ....
6       .
7       .
8       .
9       case <literalN>:
10          ....
11  }
```

gde su je svaki literal_i tipa koji odgovara tupu promenjive zadate na početku `switch` naredbe.
Program interepretira gornji šablon na sledeći način:

- Proverava vrednost zadate promenjive sa navedenim literalima po slučajevima redom.
- U momentu kada naidje na podudaranje, tj. jednakost izmedju vrednosti promenjive i literala, kod ulazi u taj `case` blok (naznačen sada sa dvotačkom `:`) i izvršava **SVE** naredne linije koda do kraja završetka `switch` bloka!
  Grafički prikazano, to bi izgledalo ovako:

Važno je još jednom skrenuti pažnju na to da ovakva konstrukcija `switch` naredbe propagira izvršavanje instrukcija kroz `case` blokove. Kada bi smo na ovaj način prepravili rešenje prethodnog zadatka, za npr. unetu ocenu 4, dobili bi smo sledeći ispis:

```
"Vrlo dobar"
"Dobar"
"Dovoljan"
"Nedovolja"
```

To bi se desilo upravo zato što kada program udje u `case` koji se odnosi na "Vrlo dobar" poruku, krenuće da propagira kroz sve ostale niže navedene slučajeve. Da bi se ovakvo ponašanje sprečilo, koristi se ključna reč `break` koja se navodi **na kraj** `case` bloka za koji želimo da sprečimo dalje propagiranje. U momentu kada program naidje na `break` instrukcijo, on iskače iz celog `switch` bloka i nastavlja sa radom dalje. Za šablon:

```
switch (<identifikator_promenjive>) {
    case <literal1>:
        ....
        break;
    case <literal2>:
        ....
        break;
```

```
        .
        .
        .
        case <literalN>:
            ....
            break;
    }
```

grafik bi izgledao:

Kombinovanje `case` blokova koji sadrže `break` naredbe i onih koji je ne sadrže u jednom `switch` bloku je sasvim dozvoljeno. U tom slučaju, kada program udje u neki `case` blok, izvršavaće sve linije koda dok ne naidje na `break` komandu, ili do kraja `switch` bloka. Samim tim, pisanje `break` komande u poslednjem `case` bloku nije neophodno, ali se često navodi radi (simetrične) estetike i čitkosti koda.
Još jednom napominjemo da unutar jednog `case` bloka, ukoliko postoji `break` naredba, ona mora biti **poslednja** naredba u tom bloku!

Modifikujmo rešenje zadatak 3.3 tako da inkorporiramo novostečeno znanje:

opisnaOcena

```
package neki.paket;

import java.util.Scanner;

public class OpisnaOcena {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int ocena = sc.nextInt();
```

```
10          /*
11          Posto u ovom delu radimo sa relacionim
    operatorima
12          koji nisu '==', MORAMO da koristimo obicano if
    grananje!
13          */*
14          if (ocena > 5)
15              ocena = 5;
16          else if (ocena < 1)
17              ocena = 1;
18
19          //Prevodimo sve u switch-case oblik:
20          switch (ocena) {
21              case 5:
22                  System.out.println("Odlican");
23                  break;
24              case 4:
25                  System.out.println("Vrlo dobar");
26                  break;
27              case 3:
28                  System.out.println("Dobar");
29                  break;
30              case 2:
31                  System.out.println("Dovoljan");
32                  break;
33              case 1:
34                  System.out.println("Nedovoljan");
35                  break;
36          }
37      }
38  }
```

Ukoliko bi smo želeli da objedinimo više slučajeva u jedan, to bi smo mogli da izvedemo pomoću praznih `case` blokova sa propagacijom:

```
switch (<identifikator_promenjive>) {
    case <literal1>:
    case <literal2>:
    case <literal3>:
        ...
}
```

ali za nešto ovako postoji sintaksička olakšica:

```
switch (<identifikator_promenjive>) {
    case <literal1>, <literal2>, <literal3>
        ...
}
```

Već smo videli moć `else` naredbe kada smo se bavili prvim tipom grananja; slično tome, ukoliko želimo da nagalasimo slučaj koji treba da se izvrši 'inače', koristimo ključnu reč `default`. Slično `else` naredbi, `default` naredba ne uzima nikakav argument i ispunjava se ako i samo ako se ne ispunjava nijedan drugi slučaj (nezavisno od toga u kom poretku je `default` u odno su na ostale slučajeve)!
Kao i kod `if-else`, ključna reč `default` nam garantuje da će se izvršiti barem jedan blok koda unutar `switch` naredbe!

Enhanced switch

Pored ovoga, `switch` naredba se već nekoliko puta 'modernizovala' i od Java verzije jezika 21 podržava razne mogućnosti poput pattern-matching-a (što znači da se i niske mogu porediti na ovaj način) i **enhanced** varijante. Unapredjena varijanta `switch` naredbe se zasniva na simbolu strelice koji se često vidja kod funkcionalnih jezika (poput Haskell-a) `->` i vitičastih zagrada `{ }`. Njen šablon je:

```
switch (<identifikator_promenjive>) {
    case <literal1> -> {
        ....
    }
    case <literal2> -> {
        ....
    }
    case <literal3>, <literal4> -> {
        ....
    }
    .
    .
    .
    default -> {
        ....
    }
}
```

Kao i kod obične `switch` naredbe, `default` naredba može a i ne mora da postoji i njen položaj unutar `switch` naredbe nije bitan. Slično, više slučajve se mogu objediniti sa zarezima `,`.
Takodje, ukoliko slučaj ima samo jednu naredbu u svom bloku, kao i kod `if else` naredbi, vitičaste zagrade se mogu zanemariti.
Jedina suštinska razlika izmedju obične i unapredjene varijante je to što unapredjena varijanta podrazumeva `break` komandu u **svakom** `case` bloku i nemora se eksplicitno navoditi!

---

*Konjuktivna normalna forma

Vratimo se nazad na deo u kome smo pričali o ungježdenim `if` naredbama. Pogledajmo naredni zadatak:

Zadatak.

Napisati program koji prima pol kao niz karaktera "musko" ili "zensko" i broj godina. Ispisati poruku:
"Spremni za starosnu penziju"; ako je osoba sprena za starosnu penziju
"Niste spremni za starosnu penziju"; ukoliko osoba nije sprena za starosnu penziju
Osoba je sprena za starosnu penziju:

- Ako je musko i ima barem 65 godina
- Ako je zensko i ima barem 63 godina
  (Ostale uslove poput staža zanemarujemo)

Prikazimo deo pseoudo-koda koji rešava deo zadatka:

```
String pol := input
int brojGodina := input
if (pol.equalsIgnoreCase("musko")) {
    if (brojGodina >= 65)
        print("Spremni")
    if (!(brojGodina >= 65))
        print("Niste spremni")
}
```

Jasno je da je potrebno da razlikujemo muskarca od zene i u zavisnosti od pola odredimo donju granice godina, te se ugnježdavanje `if` naredbi nalaže prirodno. No, ovakvo ugnježdavanje može da pretstavi veliki problem ukoliko se produbi u više nivoa. Takav kod je nepregledan i olako se može napraviti neka logička greška koju je posle teško identifikovati. Rešenje ove nezgode leži u činjenici da uslove iz unutrašnje `if` naredbe

možemo 'podići' u spoljašnju tako što ćemo taj uslov konjuktovati sa prethodnim. Na našem primeru to bi izgledalo:

```
String pol := input
int brojGodina := input
if (pol.equalsIgnoreCase("musko") && brojGodina >= 65)
    print("Spremni")
else if (pol.equalsIgnoreCase("musko") && !(brojGodina >= 65))
    print("Niste spremni")
```

Na ovaj način kreiramo nešto što se zove Konjuktivna Normalna Forma (KNF). KNF je ništa drugo nego konjukcija jedne ili više disjunkcije literala. Matematički zapisano, KNF ima oblika:

$$\text{KNF} = \bigwedge_{i \in I} p_i$$

$$p_i = \bigvee_{j \in J} q_j \text{ ,gde je } q_j \text{ neki logički literal ili negacija logičkog literala}$$

ili sve zajedno (konjukcija disjunkcije):

$$\text{KNF} = \bigwedge_{i \in I} \bigvee_{j \in J} q_{ij}$$

Kako je svako rešenje logičkih operacija logički literal, to svaki izraz (poput `d >= 3.14`, `"abc".equalsIgnoreCase("xyz")` ...) možemo posmatrati kao logički iskaz, a naš uslov kao iskaznu formulu. Da bi smo se uverili da gore pomenuti postupak radi za svaku iskaznu formulu ϕ, a ne da smo samo imali sreće i naišli na dobar primer, potrebno je da pronadjemo, tj. opišemo, uopšten postupak koji konvertuje formulu ϕ u KNF oblik.
Nešto poput ovog zadatka prevazilazi granice ovog kursa, te ostavljamo za znatiželjne da sami istraže taj deo.

Primer KNF formule u javi bi bio:

```
boolean KNF = true && (5 == 2) && (!(x >= y)) && (a || b || !(c)) && (s1.equalsIgnoreCase(s2));
```

KNF formule ne igraju pretežnu ulogu u pisanju programa i zato su ostavljenje kao apendiks poglavlju. Izvan programa koji se specifično bave ovom tematikom, vide svrhu samo kao uništavanja ugnježdenih `if` naredbi i njihovo dublje razumevanje nije neophodno za savladavanje kursa.