

---

# **JAVA PROGRAMIRANJE**

# 1. Uvod - kako početi programirati u JAVA jeziku

Cilj ovog poglavlja je napisati i pokrenuti jednostavnije Java programe.

## SADRŽAJ

1. O predmetu.
2. Programi i programski jezici.
3. Neki jednostavni Java programi.
4. Objekti i metode.
5. Konstrukcija programa.
6. Metoda `System.out.println`.
7. Kako prevesti i pokrenuti java program.
8. Pisanje Java programa.
9. Kako raditi na računalu.
10. Zadaci za prvo poglavlje.

## 1. O predmetu

Sadržaj ovog predmeta je pisanje objektno orijentiranih programa u Java jeziku.

Java je :

- programski jezik
- vrlo velika biblioteka programskih rutina
- izvršna okolina za izvođenje programa.

Java biblioteka programskih rutina sastoji se od tisuća **klasa**. Postoje klase za rad s datotekama, klase za rad s 3D grafikom, klase za pristup bazama podataka, animaciju web stranica, itd. Moglo bi se reći da se u bibliotekama nalazi veći dio onoga što će vam ikada zatrebati u programiranju. Možda ste se dosad susreli s JavaScript jezikom za web stranice. JavaScript nije Java jezik ! Ovaj predmet se zbog ograničenog broja sati ipak mora ograničiti na osnove jezika te na manji dio osnovnih klasa.

Kao dodatnu literaturu možete koristiti knjige:

1. Beginning Java 2, SDK 1.4 Edition by, Ivor Horton ISBN:0764543652 Wrox Press 2003.
2. Java , Čukman, Tihomir , Alfej
3. Java: Programiranje za Internet i World Wide Web , Dario Sušan, Znak, Zagreb, 1997.

Ako dosad niste programirali bit će potrebno uložiti značajan trud u savladavanje gradiva. Java jezik zajedno s ogromnim bibliotekama namijenjen je profesionalnim programerima. Štoviše Java biblioteke se stalno proširuju i mijenjaju .

Naučiti programirati u bilo kojem jeziku nije moguće bez praktičnog rada na računalu.

Stoga je od vrlo velike važnosti da primjere sa satova pokušate realizirati na računalu tijekom vježbi.

Ako ste dosad programirali u nekom programskom jeziku mogao bi vam prvi uvodni dio biti vrlo jednostavan. Međutim kako lekcije budu išle broj novih sadržaja i znanja će se povećavati.

Potrudite se zadatke odraditi sami. Nemojte kopirati programski kod od kolega. Osim što varate nastavnika varate i sami sebe.

## 2. Programi i programiranje

U ovome poglavlje prikazat ćemo jednostavni Java program i način kako ga pokrenuti na računalu. Za početak ćemo ponoviti glavne dijelove računala i njihova svojstva:

1. **Memorija** je dio računala u koji pohranjujemo informacije. Pohranjene informacije možemo pisati, brisati, obnovljati,... Informacije u memoriji su niz bitova dok na višoj razini predstavljaju brojeve, tekst, slike, glazbu,...
2. **Uređaji za ulaz/izlaz (I/O)**. Informacije u računalo unosimo bilo tipkovnicom ili preko medija (disketa, CD-ROM ,mreža...). Informacije iz računala prikazujemo na ekranu ili pak šaljemo na neki od medija.
3. **Procesor** koji djeluje po instrukcijama **programa**. Program se sastoji od niza operacija koje procesor izvršava. To uključuje akcije poput izvršavanja proračuna, čitanja ili pisanja po memoriji, slanja podataka na izlazna sučelja procesora, ...

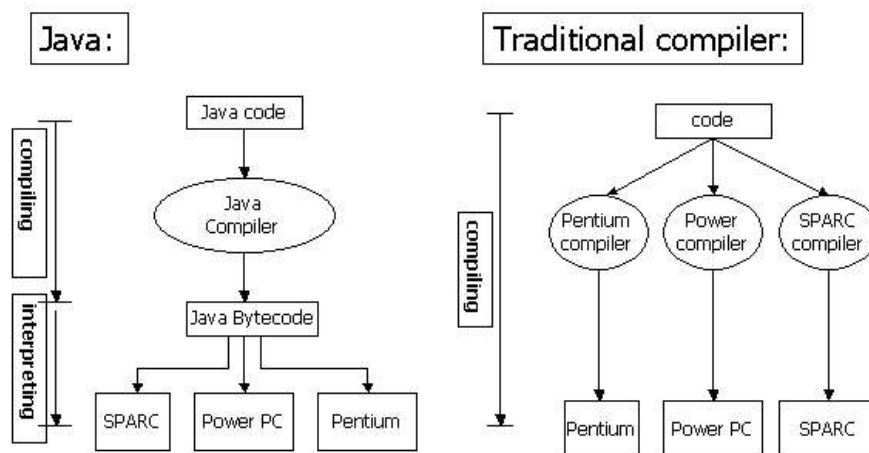
Program se piše u notaciji koja se naziva **programski jezik**. Svako računalo (procesor) ima svoj programski jezik koji nazivamo **strojni jezik** (machine code). Taj jezik je dizajniran s fokusom na elementarne operacije koje se obavljaju nad hardverom računala. Radi se o jednostavnim operacijama poput pisanja ili čitanja iz memorije, aritmetičkim operacijama nad registrima procesora, itd. Iako je teoretski svaki program moguće napisati koristeći strojni jezik to je vrlo teško čak i za jednostavne programe. U praksi se gotovo sve programiranje izvodi u jezicima koji su prilagođeni programeru. Takvi jezici se nazivaju **jezici visokog nivoa**.

Jezici koji se danas koriste u komercijalnoj upotrebi su C, C++,Java,C#,Perl,Phyton,Pascal(Delphi), Basic,Fortran....

Java jezik je jedan od najmlađih jezika. Prvi put se pojavio 1995. godine. Java 2 specifikacija jezika koju koristimo u ovom predmetu pojavila se 1998.

Rekli smo da programeri pišu programe uglavnom koristeći programske jezike visokog nivoa, a da računala izvršavaju instrukcije strojnog jezika. Pitanje je što računalo radi s programom napisanim u jeziku visokog nivoa. Najčešći način je koristiti računalni program koji nazivamo **prevodilac (compiler)**. Prevodilac prevodi program napisan u jeziku visokog nivoa u program sastavljen od strojnog jezika. Prevedeni program onda možemo pokrenuti na računalu. (U čemu se piše prevodilac ?)

U slučaju Java jezika korišten je malo drukčiji pristup u kojem se u procesu pisanja Java koda do transformacije u računalu razumljiv kod koriste dva programa. Prvo se program koji je programer napisao u Javi, pomoću prevodioca prevodi u **bytecode** program. Bytecode je sličan strojnom jeziku, ali je neovisan o bilo kojem računalu. Bytecode program nije više čitljiv od strane programera. Njega čita i izvršava program koji se naziva **Java virtual machine**. Prednosti pristupa u dva koraka jest da se tako proizvode programi koji se ipak izvršavaju zadovoljavajućom brzinom te se Java okolina može brzo realizirati na bilo kojem računalu. Originalni Java program koji piše programer i kojeg prevodi prevodilac naziva se **izvorni kod**. Bytecode koji proizvodi prevodilac i interpretira Java virtual machine naziva se **objektni kod**.



Slika 1.1 Usporedba izvršavanja Java programa s tradicionalnim postupkom prevođenja

## 1. Uvod

Postoje brojne razvojne okoline u kojima je moguće pisati, prevoditi i izvršavati programe. Za potrebe ovog predmeta zadržat ćemo se na najjednostavnijem sustavu koji je ujedno i podloga i za druge kompleksnije razvojne okoline. Sustav se naziva Software Development Kit (SDK). Može se naći na stranicama

`java.sun.com`

Ove stranice održava kompanija Sun Microsystems koja je odgovorna za razvoj Jave.

Verzija Java okoline u trenutku pisanja ovog teksta je 1.5. U ovom predmetu koristit ćemo prethodnu 1.4 verziju.

Instalacija Java okoline se u najkraćim crtama obavlja na slijedeći način:

- Pokreni Java 2 SDK installer (datoteka `j2sdk-1_***-win.exe`), odaberi mjesto instalacije i instaliraj.
- Dodaj u PATH varijablu operativnog sustava mjesto instalacije (npr. `C:\jdk1.4.2\bin`). Način dodavanja je ovisan korištenom operativnom sustavu.
- Provjeri (ukloni) CLASSPATH varijablu (`-classpath` command-line prekidač je bolji način).

Detaljniji (aktualniji) opis instalacije može se naći na stranicama `java.sun.com`.

## 3. Primjeri jednostavnih Java programa

U Javi program iste funkcionalnosti kao C program koji ispisuje "Hello World!" izgleda ovako:

```
public class Hello
{
    /* napiši jednostavnu poruku na ekran*/
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

Ako niste vidjeli dosada neki Java program ovaj jednostavni program izgledat će vam konfuzno. Cilj ovih početnih sati predavanja je da vam se objasni struktura ovako jednostavnih programa.

Svaki Java program sadrži **naredbe** (statements). Svaka naredba opisuje neku operaciju koju računalo treba izvršiti. Operacija može biti ispis neke informacije na ekranu, može biti neka računaska operacija, provjera položaja miša na ekranu, itd. Računalo jednostavno izvršava naredbu po naredbu.

U programu koji je gore napisan nalazi se samo jedna naredba: `System.out.println("Hello, World!");`

Kad se ta naredba izvrši na ekranu će se pojaviti slijedeći ispis: `Hello, World!`

Java ima različite načine pisanja poruka po ekranu bilo da pišemo po prozoru, na web stranicu, itd. U ovome slučaju koristimo jednu Java **metodu** koja se naziva `System.out.println`. Rezultat izvršavanja bit će ispis poruke u najjednostavnijem obliku prozora kojeg nazivamo **konzola** (ili DOS prozor u Windows OS). Konzola dopušta samo jednostavan ispis teksta, redak po redak.

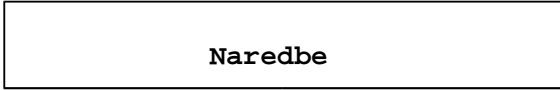
Slijedi program s dvije naredbe koje su tiskane podebljano (ne koristimo podebljane fontove u Java programima).

```
public class Hello
{
    /* napiši jednostavnu poruku na ekran*/

    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
        System.out.println("See you later.");
    }
}
```

## 1. Uvod

Kada pokrenemo ovaj program izvršit će se obje naredbe jedna za drugom. Prvo će se na ekranu ispisati u jednoj liniji `Hello, World!`, a nakon toga u drugoj liniji `See you later`. Oba programa imaju formu:

```
public class Hello
{ /* komentar. */
  public static void main(String[] args)
  {
    
  }
}
```

Ne postoji ograničenje na broj naredbi u programu. Može ih biti na tisuće. Ostatak ovog jednostavnog programa može se promatrati kao pakiranje. Ovo pakiranje ćemo objasniti poslije. Sad ćemo se zadržati na opisu **objekata** koji su neizostavni dio svakoga pa i najjednostavnijeg Java programa.

## 4. Objekti i Metode

Razmotrimo naredbu koja ispisuje poruku: `System.out.println("Hello, World!");`

Gdje su tu **objekti**? Poznavalac Java jezika vidjet će dva objekta. Prvi je objekt `System.Out`, a drugi sami niz znakova `"Hello, World!"`. Java cijelo vrijeme radi s objektima. U Java biblioteci definirano je mnogo vrsta različitih objekata koje možemo koristiti u svojim programima.

Možemo i kreirati objekte prema našim potrebama. Npr. pišemo program koji će pratiti koji su studenti na FESB-u prijavljeni na koji predmet. Tada ćemo napisati takav program koji će pokretanjem:

- za svakog studenta kreirati jedan objekt studenta
- za svaki predmet također jedan objekt predmeta.

Svaki objekt studenta sadržavat će određene podatke poput osobnih podataka studenta i liste upisanih predmeta. Objekt predmeta može sadržavati naziv predmeta i druge podatke vezane za predmet.

U isto vrijeme kad definiramo izgled objekata trebamo i definirati koje će se operacije izvršavati nad tim objektima. Što se tiče objekta `student`, bit će nam potrebne operacije kreiranja objekta studenta, ažuriranja liste predmeta koje je student upisao, operacije ispisa podataka o studentu na ekran, itd. Te operacije koje se izvršavaju nad objektom nazivaju se **metode**.

Dosad smo već vidjeli primjer metode. Njen puni naziv je: `System.out.println`

Ovaj naziv označava metodu `println` koja pripada objektu `System.out`. `System.out` je objekt čiji je zadatak da primi poruku koju treba prikazati na ekranu.

Zamislimo ga kao osobu kojoj dajemo što treba ispisati na ploči.

`Println` metoda je operacija koja se izvršava nad porukom. tako nam izraz:

```
System.out.println("Hello, World!");
```

kaže:

koristi `println` metoda za slanje poruke `"Hello, World!"` objektu `System.out`, koji će je prikazati na ekranu.

Svaki objekt pripada **klasi (class)** koja specificira od kojih podataka se objekt sastoji i koje metode posjeduje. Npr. svi nizovi znakova pripadaju klasi koja se naziva `String`. Klase `String` i `System.Out` definirane su u klasama koje pripadaju Java bibliotekama. Možemo kreirati i svoje klase npr. klasu `Student` i klasu `Predmet`.

## 1. Uvod

Koji je odnos klasa-objekt? Kažemo da je objekt instanca od klase. Jednostavno, klasa je opis objekta napisan u kodu. Možemo je promatrati kao kalup ili skicu prema kojoj se u tijeku izvršavanja programa kreiraju objekti. Program može kreirati više objekata, instanci iste klase. Java biblioteka je u potpunosti sastavljena od definicija klasa. Ako napišemo bilo koji program u Javi i on će se sastojati od klasa. Većina klasa definira tipove objekata. Postoje samo nekoliko klasa kojima se ne definira objekt već su sastavljene samo od samostalnih metoda.

Ipak nije sve u Javi objekt. Najjednostavniji tipovi podataka poput cjelobrojnih i brojeva u pokretnom zarezu tretiraju se nešto drugačije. Takvi podaci nazivaju se **primitivni tipovi podataka**.

## 5. Kako je konstruiran program iz primjera

Prethodni dio pokazao nam je da se Java programi sastoje od klasa. Programi koji su navedeni kao primjer uklapaju se u tu tvrdnju, ali ipak na vrlo primitivan način. Oba programa sastoje se od jedne klase (Hello) koja se sastoji samo od jedne statičke metode tj. metode koja ne pripada nijednom određenom objektu.

Svaki metoda bilo da pripada objektu ili ne, sadrži određeni broj naredbi koje izvršavaju neku korisnu operaciju. (U drugim računalnim jezicima imamo funkcije ili procedure)

Analizirat ćemo korak po korak kompletnu metodu iz zadnjeg primjera:

```
/* napiši jednostavnu poruku na ekran*/
public static void main(String[] args)
{
    System.out.println("Hello, World! ");
    System.out.println("See you later. ");
}
```

Sastoji se od slijedećih dijelova:

1. `/* napiši jednostavnu poruku na ekran*/`

Ovo je komentar koji opisuje što ni program trebao raditi. to je jednostavno poruka za onoga tko čita izvorni kod programa. Svaki tekst između `/*` i `*/` bit će tretiran kao komentar i Java će ga u potpunosti ignorirati.

2. `public static void main(String[] args)`

Ovo predstavlja **zaglavlje** (heading) metode. Svaka metoda ima svoj naziv. U ovom slučaju naziv metoda je riječ `main` koja se nalazi neposredno ispred zagrade. Riječi `public`, `static` i `void` pokazuju Java prevodiocu način korištenja metoda `main`. (Bit će objašnjeno kasnije).

Dio u zagradama, `String[] args`, opisuje informaciju koja će biti proslijeđena metodi svaki put kad bude pozvan. Naziva se lista **parametara**. U navedenom primjeru ta je informacija ignorirana, odnosno nije korištena u programu. (koko se koristi bit će objašnjeno poslije)

3. 

```
{
    System.out.println("Hello, World! ");
    System.out.println("See you later. ");
}
```

Ovo je **tijelo** (body) metoda. Uvijek se sastoji od niza naredbi zatvorenih u vitičaste zagrade, `{..}`. Pozivom ovog metoda izvršava se svaka od naredbi.

Sve metode sastoje se od tri navedena dijela. Strogo rečeno komentar je opcionalan. Međutim preporuča se uvijek početi s komentarom koji ukratko kaže što radi metoda koji slijedi. Taj dio nazivamo **specifikacijom**. Gornji primjer je prejednostavan da bi specifikacija bila od veće koristi, ali u većim programima to je najefikasniji način da pomognemo razumijevanju programa. Posebno je to bitno ako na programu radi više programera.

Svaki program sastoji se od određenog broja definicija klasa. U dva gornja primjera u programu je definirana samo jedna jedina klasa nazvana `Hello`.

Definicija klase započinje s zaglavljem: `public class Hello`

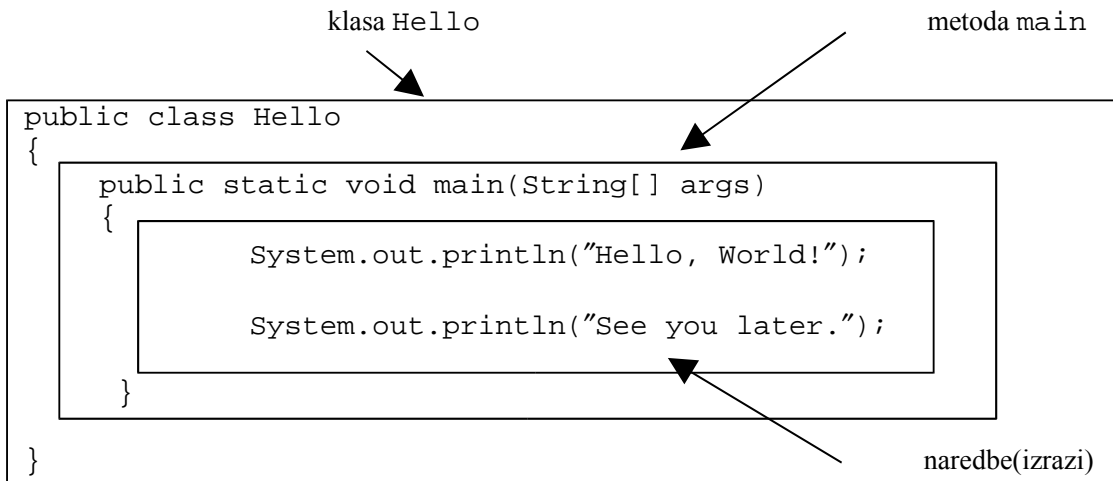
## 1. Uvod

Zaglavlje je praćeno elementima koji sačinjavaju klasu, zatvorenim u vitičaste zagrade.

U našim primjerima klasa se sastoji od samo jednog člana, metode nazvane `main`, koju smo već opisali. Općenito program se sastoji od jedne ili više definicija klasa od kojih svaka sadržava jednu ili više metoda.

Java programeri koriste konvenciju po kojoj naziv klase započinje velikim slovom, a naziv metode malim slovom. U ovome predmetu ćemo se nastojati strogo pridržavati navedene konvencije. Općenito bilo koji naziv u Javi (**identifikator**) sastoji se od slova, znamenki i mora početi sa slovom. Za potrebe ove definicije se simboli valuta poput £ i \$, i povlaka ('\_') računaju kao slova. Duljina naziva nije ograničena.

Na slijedećoj slici prikazan je naš program koji se sastoji od dvije naredbe razdvojen okvirima koji nam pokazuju što je klasa, što je metoda a što su naredbe.



## 6. Metoda `System.out.println`

`System.out.println` je prva metoda iz Java biblioteke koji smo upotrijebili. Ova metoda će ispisati niz znakova na konzolu (DOS prozor). Naziv `println` je skraćenica za 'print line'.

Mjesto na ekranu na kojemu će se ispisati slijedeći znak označeno je na ekranu s malom treptajućom linijom nazvanom **kursor**. `System.out.println` metoda ima tu osobinu da ispisom teksta pomiče kursor na početak slijedeće linije. Ako želimo da kursor ostane na prethodnoj liniji koristit ćemo metodu `System.out.print`.

Drugi način određivanja kad želimo ispis u novoj liniji je korištenje para znakova `\n` u nizu znakova svaki put kad je potrebno da ispis krene u novu liniju. Npr. :

```
System.out.print("Hello, World!\nSee you later\n")
```

će ispisati `Hello, World!`, nakon toga pomaknuti kursor u novu liniju gdje će ispisati `See you later.`, i nakon toga pomaknuti kursor na novu liniju. Ta će naredba imati isti rezultat kao i par naredbi:

```
System.out.println("Hello, World! ");
System.out.println("See you later. ")
```

## 7. Kako prevesti i pokrenuti Java program

Pretpostavimo da želite izvršiti program iz prvog primjera:

```
class Hello
{
    /* napiši jednostavnu poruku na ekran*/
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

}

Prvo morate biti pristupiti računalo na koje je instalirana podrška za Javu odnosno Java SDK.

### A. **Ukucajte program i pohranite ga u datoteku `Hello.java`.**

Možete koristiti bilo koji tekst editor za unos koda, npr. Notepad. Korisno je imati i neki sofisticiraniji Java editor npr. JedPlus. Takvi editori nam mogu omogućiti korisne funkcije poput sintaksnog naglašavanja ili automatskog uvlačenja teksta. Moguće je iz takvih editora pozvati i operacije prevođenja i izvršavanja.

Ako se vaš program sastoji od samo jedne klase potrebno ga je pohraniti u datoteku koja ima isti naziv kao i klasa uz dodanu ekstenziju '`.java`'. kako se naš jednostavni program sastoji od samo jedne klase nazvane Hello, datoteku **moramo** nazvati `Hello.java`.

### B. **Otvori DOS prozor i postavi trenutni direktorij na direktorij gdje je datoteka s programom.**

Sve naredbe koje slijede tipkaju se u ovaj prozor. Poruke prevodioca i sve što će program ispisati odvija se također u ovome prozoru. Postoje i drugi načini koje ćemo obraditi na vježbama, ali ovo je općeniti način koji radi na svim računalima.

U ovome trenutku bit će otvorena dva prozora. Jedan s editorom (npr. Notepad), a drugi s DOS prozorom koji služi za interakciju s Java sustavom. U tom DOS prozoru prevodimo i izvršavamo program.

### C. **Korištenje JDK za prevođenje programa u `Hello.java` datoteci.**

Za prevođenje programa treba utipkati:

```
javac Hello.java
```

Ako dobijete poruku da sustav ne može naći javac (Java prevodilac) znači da ili nije pravilno postavljena PATH varijabla sustava ili nije instaliran JDK.

Ako nema grešaka u programu prevodilac će proizvesti bytecode verziju vašeg programa u datoteci nazvanoj `Hello.class`.

Ako prevodilac nađe greške poput tipkanja `Class` umjesto `class` ili izostavljanja znaka `;` na kraju naredbe, odbit će prevođenje i izvijestiti o pronađenim greškama.

To nazivamo **greškom prevođenja (compiler error)**. Najčešće je poruka o grešci takva da je jednostavno naći mjesto u kodu gdje smo učinili pogrešku. Ponekad iz poruke nije jasno odakle potječe greška i tada je potrebno pažljivo pregledati kod. Početnici često pogriješe tražeći grešku baš u liniji gdje je to prevodilac javio. Međutim, greška se može nalaziti i negdje prije !

Ako postoje greške kod prevođenja potrebno ih je otkloniti u editoru. nakon otklanjanja grešaka ne zaboravite snimiti datoteku.

### A. **Korištenje JDK za pokretanje prevedene verzije programa koja se nalazi u `Hello.class` datoteci.**

Nakon što je prevodilac proizveo bytecode verziju programa `Hello.class`, možete ga pokrenuti u Java Virtual machine tipkanjem:

```
java Hello
```

Ne tipkajte `.class` ekstenziju. JDK pretpostavlja da ste mislili na `Hello.class`.

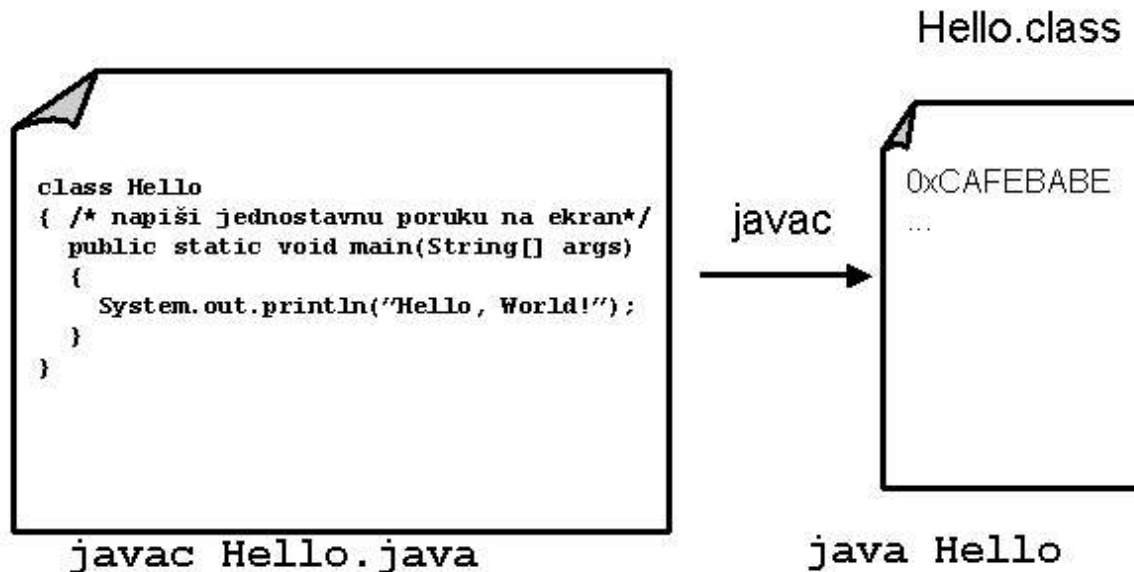


Ovaj se program nakon toga treba izvršiti i ispisati poruku u DOS prozoru. Ako želite modificirati ispis programa ili dodati još koju naredbu opet se trebate vratiti u prozor tekst editora, napraviti modifikacije i nakon toga u DOS prozoru ponoviti postupak prevođenja i izvršavanja.

U tijeku izvođenja programa može nastati greška. Takva greška naziva se **run-time greška** (run-time error) ili **greška u izvršavanju** (execution error).

U nekim slučajevima JVM neće moći izvršiti program do kraja. Tada će ispisati poruku o grešci s podatkom gdje je program došao u izvođenju prije nego što je prekinut.

Proces pisanja, prevođenja i izvođenja programa pokazan je i na sljedećim slikama:



Slika 1.2 Pisanje programa, prevođenje, izvršavanje.

## 8. Pisanje koda Java programa

Prvo pravilo pisanja bilo kojeg programa je: pišite pažljivije što god možete. Skoro svaka pogreška u pisanju će najvjerojatnije proizvesti grešku prilikom prevođenja ili prilikom izvođenja. Prvo će je trebati detektirati, a nakon toga korigirati. Npr. svaki izraz koji koristi naredbu `System.out.println` mora završiti s `;`. Ako izostavite ; dobit ćete poruku o grešci u fazi prevođenja.

U pisanju Java programa morate paziti na korištenje malih i velikih slova. Za razliku od nekih drugih programskih jezika Java razlikuje velika i mala slova (case sensitive). To znači da ne možete utipkati `Class` ili `CLASS` na početku programa. Potrebno je točno pisati `class`.

Riječ `class` ima u Java programima specijalno značenje i jedna je od **ključnih riječi** (keywords) Java jezika. `java` ima 47 ključnih riječi i sve se pišu malim slovima. kao što je prije napisano držat ćemo se konvencije po kojoj se nazivi klasa pišu s početnim velikim slovom, a nazivi metoda malim slovima.

Ako imamo nazive koji se sastoje od više spojenih riječi možemo početak sljedeće riječi pisati velikim slovom. Npr. klasu možemo nazvati `HelloWorld`, a metodu npr. `ispisTeksta`.

Koliko grešaka pisanja možete vidjeti u ovome kodu ? prevodilac će javiti svaku od njih.

```
public Class Hello
{ /* Write a simple message on the screen.
  public static viod main(string[] args)
```

## 1. Uvod

---

```
    { System.out.println("Hello, World!");  
      System.out.println("See you later.");  
    }  
  )
```

Jedina greška koju prevodilac neće javiti je pogrešno pisanje unutar niza znakova `See you later`. Ta greška će biti očigledna tek nakon izvršavanja programa.

Zbog svega navedenog nije se praktično oslanjati na prevodilac u otkrivanju grešaka. To je prije svega dugotrajan proces jer je potrebno svaki put nakon ispravljene greške proći kroz faze prevođenja i izvršavanja. Međutim to više vrijedi za iskusne programere nego za početnike kojima će prevodilac često biti neprocjenjiva pomoć u otklanjanju pogrešaka.

Osim što je Java prevodilac razlikovanjem velikih i malih slova zna biti dosta nezgodan u drugim pogledima je vrlo liberalan. Uopće mu nije bitan prostorni raspored vašeg koda tj. upotreba praznina i novih redova.

Slijedeći primjer bi se trebao prevesti bez greške iako je vrlo teško čitljiv:

```
public class Hello{public static void main(String[]  
args){System.out.println("Hello, World!");System.  
out.println("See you later.");}}
```

Iz ovoga je na prvi pogled teško ustanoviti da se program sastoji od jednog metoda koji sadrži dvije naredbe. Čak će se i slijedeći primjer pravilno prevesti:

```
public                                     class  
Hello  
public  
static  
void                                     main  
(String[] args)  
  { System.out.println  
( "Hello, World!" );  
                                     System.out.  
println( "See you later." );  
  } }
```

Sve dok ne spojite neke riječi poput `classHello`, ili ne razdvojite `class`, prevodilac se neće buniti..

Prevodilac prema svemu navedenom je vrlo fleksibilan u korištenju razmaka i novih redova. Međutim bilo tko bude čitao vaš program (i vi sami) poželjet će mnogo uredniji i jasniji kod. Zato je praznine uputno koristiti da bi se dobio jasniji kod.

Ako pogledate na originalni kod gore modificiranih programa, vidjet ćete da postoje neka pravila u pisanju. Npr. metoda `main` je pomaknuta tri mjesta unutar u odnosu na prethodni redak. To nazivamo **uvlačenje koda** (indentation). Ako ima više metoda unutar klase, stavljamo razmak između svake. Primijetite i da vitičaste zagrade imaju konvenciju o smještaju. Vidjet ćemo tijekom ovoga predmeta kako postupati s pojedinim elementima koda. Osim urednosti koja omogućava razumijevanje koda, drugi rezultat je pisanje koda s manje grešaka.

## 9. Kako rješavati zadatke

Zadaci koji su navedeni na kraju svakog poglavlja predstavljaju zadatke koji će se javiti i na ispitu. Stoga nastojte zadatke samostalno rješavati jer je to najefikasniji način za usvajanje gradiva. Podloga za rješavanje zadataka jeste sadržaj svakog poglavlja. Štoviše, tu ćete naći i dijelove izvornog koda koji će vam koristiti u rješavanju zadataka.

Na računalu na kojemu dobijete korisnički račun kreirajte direktorij `JProg`. Unutar tog direktorija za svako poglavlja otvorite direktorij `Pn` gdje je `n` broj poglavlja. Dakle za prvo poglavlje otvorite direktorij `P1`. Kada završite poglavlje 1 unutar direktorija `P1` trebali biste imati slijedeće datoteke:

```
Hello.java , Hello.class
ImeUOkviru.java, ImeUOkviru.class
Gresnik.class , Gresnik.java
Inicijali.class , Inicijali.java
```

## 10. Zadaci za prvo poglavlje

Za ovo poglavlje zadana su tri jednostavna problema tj. pisanje tri jednostavna programa..

1. Ukucaj, prevedi i pokreni slijedeći Hello program koji ispisuje dvije linije na ekran:

```
/* Autor: Ime Prezime */
class Hello
{
    /* jednostavan ispis na ekran. */
    public static void main(String[] args)
    { System.out.println("Ciao, Svite!");
      System.out.println("Vidimo se kasnije !");
    }
}
```

2. Napišite program s nazivom klase `ImeUOkviru`. Program treba ispisati vaše ime u okviru poput slijedećega:

```
+-----+
|  Ivan  |
+-----+
```

Uputa: nije svejedno kako ćete nazvati datoteku s kodom!

3. Ukucaj i pokreni slijedeći program te analiziraj pogreške koje će javiti prevodilac. ukucajte ga sa svim pogreškama !

```
public Class Gresnik {
    /* Write a simple message on the screen.
    public static void main(string[] args)
    { System.out.println('Alo, Svijete!');
      System.out.println("Vikimo se kasnije.");
    }
}
```

Zatim ispravite pogreška koliko možete i pokušajte natjerati program da radi kako bi trebao.

## Poglavlje 2. Brojevi i znakovni nizovi (strings)

U prvom poglavlju pisali smo programe koji su samo ispisivali poruke na ekran. U ovome poglavlju uvodimo nove izraze koje ćemo koristiti u metodama. To uključuje upotrebu dvije vrste vrijednosti: brojevi i znakovni nizovi (strings).

### SADRŽAJ

1. Cjelobrojne vrijednosti (integers).
2. Varijable (variables).
3. Dodjeljivanje(assignments).
4. Brojevi u pokretnom zarezu (floating-point values).
5. Čitanje ulaza korištenjem `ConsoleReader` objekta.
6. Znakovni nizovi (strings).
8. Zadaci za 2. poglavlje.

Java posjeduje slijedeće tipove primitivnih tipova podataka:

ključna riječ	opis	veličina
<i>cjelobrojni (integers)</i>		
<code>byte</code>	cijeli broj duljine jednog byte-a	8 bit-a bez predznaka
<code>short</code>	cijeli broj duljine dva byte-a	16 bit-a bez predznaka
<code>int</code>	<code>Integer</code>	32 bit-a bez predznaka
<code>long</code>	<code>Long Integer</code>	64 bit-a bez predznaka
<i>brojevi u pokretnom zarezu (real numbers)</i>		
<code>float</code>	jednostruka preciznost	32-bita
<code>double</code>	dvostruka preciznost	64-bita
<i>ostali tipovi</i>		
<code>char</code>	jedan znak	16-bita unicode znak
<code>boolean</code>	<code>boolean</code> (logička vrijednost)	true ili false

### 1. Cjelobrojne vrijednost (integers)

U prvom poglavlju smo spomenuli da nisu svi podaci u Javi objekti. Najjednostavniji tipovi podataka, poput cijelih brojeva ili brojeva u pokretnom zarezu odstupaju od logike da sve treba biti objekt. Takvi tipovi podataka nazivaju se *primitivni tipovi podataka*.

Ovo poglavlje počinjemo primjerom upotrebe cijelih brojeva (integers).

Pretpostavimo da idete kupovati na tržnici. Kupili ste 2 kg jabuka po 12 kuna, 3kg krumpira po 2kune i 50lipa i 2 kg bresaka po 14 kuna. Koliko ste kuna potrošili.

Nije potrebna snaga Jave na modernom računalu za izračunati navedeni primjer jer i napamet (ili bar uz pomoć papira i olovke) da se izračunati je ukupan račun 85.5 kune.

Međutim program će rasvijetliti nekoliko točaka u računanju s cijelim brojevima.

#### PRIMJER 1

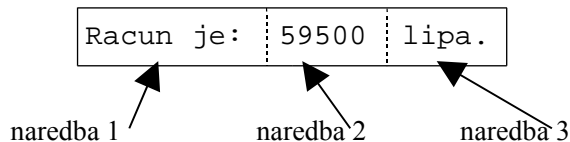
```
public class Racun1
{
    /* Izracunaj ukupni racun za 2 kg jabuka po 12 kn
    3kg krumpira po 2kn i 50lipa i 2 kg bresaka po 14 kn */
}
```

## 2. Brojevi i znakovni nizovi

```
public static void main(String[] args)
{
    System.out.print("Racun je: ");
    System.out.print(2*1200 + 3*250 + 2*1400);
    System.out.println(" lipa.");
}
}
```

Poput svih primjera za ovo poglavlje program se sastoji od jedne klase koja sadrži samo jednu metodu `main`. U ovome primjeru metoda sadrži tri naredbe. Kad se izvrši tri metode će proizvesti dijelove linije na ekranu kako je to prikazano na slici ispod.

Primijetite da prve naredbe koriste `print` umjesto `println` što znači da ne pomiču kursor u novi red nakon ispisa.



Kada Java izvrši naredbu 2, izračunat će vrijednost slijedećeg aritmetičkog izraza:

$$2*1200 + 3*250 + 2*1400$$

i prikazati rezultat 59500. Primijetite da možemo ispisivati brojeve koristeći `System.out.print` na isti način na koji smo koristili kod ispisivanja znakovnih nizova.

U Java programu slijedeći operatori mogu se koristiti u aritmetičkim izrazima:

+	zbrajanje (addition)
-	oduzimanje (subtraction) (koristi se i za minus predznak)
*	množenje (multiplication)
/	dijeljenje (division)
%	ostatak (remainder)

U Javi (i većini drugih programskih jezika višeg nivoa) brojevi su **cjelobrojni** ili **brojevi u pokretnom zarezu** (integers or floating-point numbers). Ove dvije vrste vrijednosti pohranjuju se u memoriji na različite načine. Ako vrijednost napišemo bez decimalne točke, npr. 4 ona će biti tretirana kao cjelobrojna vrijednost. Ako je napišemo s decimalnom točkom npr. 4.0 bit će tretirana kao broj u pokretnom zarezu. Uvijek trebamo koristiti cijele brojeve kad to možemo. Računanja s njima su brža, koriste manje memorije i nisu podložni greškama zaokruživanja poput brojeva s pokretnim zarezom.

Operator dijeljenja, `/`, u stvari ima dva značenja. Ako ga koristimo s cijelim brojevima tada znači **cjelobrojno dijeljenje**. Npr.  $11 / 4 = 2$ ,  $(-11) / 4 = -2$ . Ako se koristi s dva broja s pokretnim zarezom onda znači normalno dijeljenje.

Npr.  $10.5 / 2 = 5.25$ . Operator ostatka koristi se samo s cijelim brojevima. Npr.  $11\%4 = 3$  and  $(-11)\%4 = -3$ .

Ako Javi damo da izračuna izraz `m%n` gdje je `n` jednak nuli, Java će prijaviti run-time grešku.

kada procesira složenije izraze Java se drži standardnog prioriteta operatora pa npr. izraz `3+4*5` znači isto kao i `3+(4*5)`.

Kada se izračunava izraz sa više operatora istog prioriteta u nizu, izraz se računa s lijeva nadesno. Npr. `3/4/5` znači isto kao i `(3/4)/5`, na kao `3/(4/5)` što bi proizvelo grešku (Zašto?).

## 2. Varijable

Slijedeći primjer je druga verzija programa prvog primjera. Ovaj primjer koristi **varijable** tj. naziv koji znači vrijednost.

Program izračunava ukupan račun u dva odvojena koraka.

### PRIMJER 2

```
public class Racun2
{
    /* Izracunaj ukupni racun za 2 kg jabuka po 12 kn
    3kg krumpira po 2kn i 50lipa i 2 kg bresaka po 14 kn */

    public static void main(String[] args)
    {
        int total =2*1200 + 3*250 + 2*1400;
        System.out.print("Racun je: ");
        System.out.print(total/100);
        System.out.print(" kn i ");
        System.out.print(total%100);
        System.out.println(" lipa.");
    }
}
```

Izraz: `int total =2*1200 + 3*250 + 2*1400;`

kaže: kreiraj varijablu `total` koja će sadržavati vrijednost izraza  $2*1200 + 3*250 + 2*1400$  (=59500). Riječ `int` na početku pokazuje da je tip vrijednosti koju varijabla `total` može pohraniti je cjelobrojna vrijednost.

Ovaj izraz se naziva **deklaracija varijable (ili definicija varijable)**.

Izraz: `System.out.print(total/100);`

kaže: ispiši vrijednost naznačenu kao `total/100`. Radi se o cjelobrojnomo dijeljenju čija je vrijednost 85 i ta će vrijednost biti ispisana.

Izraz: `System.out.print(total%100);`

kaže: ispiši ostatak dijeljenja `total` sa 100. U ovome slučaju je to 50.

Ukupan ispis na ekran bi trebao biti:

Racun je:	59	kn i	50	lipa.
-----------	----	------	----	-------

Ovaj primjer smo mogli realizirati i bez varijable s upotrebom sljedeće dvije naredbe:

```
System.out.print((2*1200 + 3*250 + 2*1400)/100);
System.out.print((2*1200 + 3*250 + 2*1400)%100);
```

Upotrebom varijable izbjegli smo dvostruko računanje ukupnog broja lipa. U ovako malom promjeru to je mala ušteda, ali u realnim primjerima mogu se postići značajne uštede.

Drugi razlog upotrebe varijabli je da se jasno označe vrijednosti koje se koriste u računu tako da je jasnije što se u programu radi. Za nazive varijabli možete koristiti bilo koji naziv koji već nije upotrebljen za nešto drugo. U praksi upotrebljavaju se opisni nazivi koji odgovaraju ulozi varijable.

U sljedećoj verziji programa broju kilograma pojedinog artikla dani su očiti nazivi `jabuka`, `krumpir`, `breskva`. Usput, prihvaćena je konvencija da nazivi varijabli uvijek počinju s malim slovom.

### PRIMJER 3

```
public class Racun3
{
    /* Izracunaj ukupni racun za 2 kg jabuka po 12 kn
    3kg krumpira po 2kn i 50lipa i 2 kg bresaka po 14 kn */
```

## 2. Brojevi i znakovni nizovi

---

```
public static void main(String[] args)
{
    int jabuka = 2;
    int krumpir = 3;
    int breskva = 2;

    int total =
        jabuka*1200 + krumpir*250 + breskva*1400;

    System.out.print("Racun je: ");
    System.out.print(total/100);
    System.out.print(" kn i ");
    System.out.print(total%100);
    System.out.println(" lipa.");
}
}
```

Ovaj program daje izlaz isto kao i prethodni. Upotreba varijabli omogućava bolje razumijevanje koda bilo vama ili nekome drugome koji će nakon vas modificirati kod.

U programiranju postoji pojam "magični broj". To su razne konstante u programu za koje prije ili kasnije se zaboravi što znače.

Kada je varijabla kreirana za nju se alocira prostor u memoriji gdje se potom može spremi vrijednost varijable. Količina memorije koja se alocira ovisi o tipu varijable. Npr. za tip varijable `int`, kao što je to u slučaju varijable `total`, bit će alocirana 32 bita memorije.

Format pohrane je rijetko važan programeru. Bitniji je opseg vrijednosti koji se može pohraniti u zadanu varijable. Vrijednost `int` varijable može biti u području od  $-2147483648$  do  $2147483647$ . Ako su potreban veći opseg vrijednosti potrebno je umjesto `int` upotrijebiti `long` tip varijable. Taj tip zauzima 64 bita, a opseg je od  $-9223372036854775808$  do  $9223372036854775807$ . Postoje i kraći tipovi cjelobrojnih varijabli `short` (16 bita) i `byte` (8 bita).

## 3. Dodjeljivanje (Assignments)

Kao što smo vidjeli, svaka varijable ima naziv i odgovarajući dio memorije. Vrijednost koja je sadržana u memoriji je **dodijeljena** varijabli. Ta vrijednost ne mora biti ista u tijeku izvršavanja programa. Može biti zamijenjena drugom vrijednošću u tijeku izvršavanja programa. U slijedećem primjeru promijenit ćemo sadržaj varijable.

### PRIMJER 4

```
public class Mjenjac1
{
    /* Preracunaj eure u kune */

    public static void main(String[] args)
    {
        int eura = 110;
        int centa = 55;

        centa = 100*euru + centa;
        double kuna = centa * 0.0751;
        System.out.print("Kuna: ");
        System.out.print(kuna);
    }
}
```

## 2. Brojevi i znakovni nizovi

Program započinje kreiranjem dviju varijabli nazvanih `maraka` i `pfeniga` koje su postavljene na određene iznose.

Nakon toga se izvršava slijedeći izraz:

```
centa = 100*euru + pfeniga;
```

Izraz znači: izračunaj vrijednost izraza `100*euru + pfeniga`, i pohrani rezultat (11055) u varijablu `centa`. Drugim riječima, ovaj izraz će zamijeniti originalnu vrijednost pohranjenu u varijabli `centa` (=55) s ukupnim iznosom koji imamo izraženim u centima. Nova vrijednost se koristi u slijedećem izrazu:

```
double kuna = centa * 0.0751;
```

i predstavlja naše prvo korištenje brojeva u pokretnom zarezu. Izraz znači: kreiraj varijablu nazvanu `euru`, tipa `double`, i dodijeli joj vrijednost `centa * 0.0751`. Varijabla tipa `double` sadržava broj u pokretnom zarezu *duple preciznosti* i alocira 64 bita memorije.

Produkt će se izračunati u aritmetici pokretnog zareza i rezultat će biti 830.2305

Konačan ispis je:

```
Kuna: 830.2305
```

Izraz:

```
pfeniga = 100*maraka + pfeniga;
```

nazivamo **dodjeljivanje** vrijednosti. Ovaj izraz mijenja vrijednost *postojeće* varijable.

Kadgod Java obavlja dodjeljivanje ono se obavlja u dva koraka:

- (1) Prvo se računa vrijednost s desne strane od znaka '='.
- (2) Zatim se ta vrijednost sprema u varijablu naznačenu s lijeve strana izraza

Moguće je mijenjati vrijednost spremljenu u varijablu, ali nije moguće mijenjati tip (type) varijable.

Moguće je deklarirati varijablu bez specificiranja njene inicijalne vrijednosti. Ako Java izvrši slijedeći izraz:

```
int total;
```

kreirat će varijablu tipa `int` i dati joj naziv `total`, ali neće spremiti nikakvu vrijednost u istu. Vrijednost varijabli će biti potrebno dodijeliti kasnije u programu. U međuvremenu nije moguće znati što sadrži varijabla `total`. U njoj će biti neki bitovi koji su ostali u memoriji otprije. Kažemo da vrijednost varijable 'nije definirana'.

Opći oblik deklaracije varijable je dakle:

deklaracija (s inicijalnom vrijednosti)

```
TIP NAZIV = IZRAZ ;
```

Npr. `TIP` može biti `double`, `NAZIV` može biti `euru`, a `IZRAZ` može biti `64.45`.

dakle

```
double eur = 64.45;
```

deklaracija (bez inicijalne vrijednosti)



## 2. Brojevi i znakovni nizovi

TIP NAZIV;

dodjeljivanje

NAZIV = IZRAZ;

Java omogućava deklaracije varijabli u kojima se istovremeno deklarira više varijabli odjednom.

Primjer: kreiranje varijabli  $w$ ,  $x$ ,  $y$  i  $z$ , i dodjeljivanje inicijalnih vrijednosti varijablama  $x$  i  $z$ .

```
int w, x = 1, y, z = 2;
```

Često je nekoj varijabli potrebno samo nadodati neku vrijednost i postoji skraćeno pisanje za takav tip operacije. Npr. za dodati 3\*jabuka varijabli `total` pisat ćemo slijedeći izraz:

```
total += 3*jabuka;
```

Slično možemo koristiti simbol `--` kada želimo oduzeti vrijednost od varijable, `*=` kada želimo pomnožiti varijablu s izrazom, itd.

Općenito izraz:

$x \text{ OP} = y$ ; ekvivalentan je izrazu

$x = x \text{ OP } y$ ;

gdje je OP bilo koji binarni operator

Jedan od najčešćih slučajeva promjene vrijednosti varijable je povećavanje za jedan.

Možemo pisati:

```
x = x+1; ili kraće x += 1; ili još kraće x++;
```

Slično za oduzimanje 1 od  $x$ , pišemo `x--`.

### 4. Brojevi s pokretnom zarezom (Floating-point)

Naziv 'broj s pokretnim zarezom' proizlazi iz načina pohrane takvih brojeva u memoriji.

U Javi tip `double` se najčešće koristi za brojeve u pokretnom zarezu. Vrijednost tipa `double` zauzima 64 bita. to omogućava veoma velik opseg vrijednosti, približno  $\pm 1.8 \times 10^{308}$ , što izraženo preko preciznosti iznosi 15 značajnih znamenki. Naziv `double` je skraćenica za 'double precision floating-point'. Java posjeduje i drugi tip za rad s brojevima s pokretnim zarezom koji se naziva `float`. Taj tip podataka zauzima samo 32 bita tako da predstavlja ekonomičnije korištenje memorije, ali ima samo pola preciznosti tipa `double` te ekvivalentno manji opseg od oko  $\pm 3.4 \times 10^{38}$ .

Broj s pokretnim zarezom može se pisati s decimalnom točkom, npr. 14.56 ili 14.0. Može biti pisan i s eksponentom, npr. 14.56E-12.

Java sadržava niz metoda koje računaju širok opseg matematičkih funkcija koje koriste `double` vrijednosti. Sve su statičke metode (ne pripadaju nijednom objektu) iz klase `Math` koja se nalazi u Java biblioteci. Kadgod koristite statičku metodu koja pripada klasi biblioteke potrebno je kao prefiks naziva metode staviti i naziv klase.

<code>Math.sin(x)</code>	sinus od x
<code>Math.cos(x)</code>	cosinus od x
<code>Math.tan(x)</code>	tanges od x
<code>Math.exp(x)</code>	$e^x$
<code>Math.log(x)</code>	prirodni logaritam od x

## 2. Brojevi i znakovni nizovi

---

<code>Math.abs(x)</code>	apsolutna vrijednost od <code>x</code>
<code>Math.floor(x)</code>	najveći cijeli broj $\leq x$
<code>Math.ceil(x)</code>	najmanji cijeli broj $\geq x$

U svakoj od metoda parametar `x` je tipa `double`, a rezultat je tipa `double`.

Java se neće buniti ako nađe vrijednost tipa `int` na mjestu gdje je predviđen tip `double`. Kadgod se to dogodi Java će konvertirati cijeli broj u odgovarajući broj s pokretnim zarezom

Razmotrite slijedeće izraze:

```
int m = 3;
int n = 4;
double x = m*n;
```

Kada Java bude izvršavala treći izraz, izračunat će `m*n` koristeći cjelobrojnu aritmetiku te rezultat privremeno spremiti kao 32-bitnu vrijednost odnosno cijeli broj 6. Zatim će tu vrijednost konvertirati u 64-bitni broj u pokretnom zarezu (6) te ga spremiti u varijablu `x`.

Što će biti rezultat izvršavanja slijedećeg izraza:

```
double x = m/n;
```

Odgovor je da će `x` biti postavljen na vrijednost 0. Razlog tome je da će za izračunavanje izraza `m/n` biti upotrijebljena cjelobrojna aritmetika, jer su oba operanda cjelobrojna

Pretpostavimo da ipak želimo koristiti normalno dijeljenje brojeva s pokretnim zarezom. Potrebno je Javu uvjeriti da vrijednosti `m` i `n` tretira kao brojeve s pokretnim zarezom. Možemo to učiniti na slijedeći način:

```
double x = ((double) m) / n;
```

Izraz `(double)` naziva se **cast operator**. Stavljanjem cast operatora ispred `m` kažemo Javi da konvertira vrijednost od `m` u ekvivalentnu `double` vrijednost. Kad smo to učinili operator se sada odnosi na jednu `double` vrijednost i jednu `int` vrijednost. U tom slučaju Java će koristiti dijeljenje za brojeve s pokretnim zarezom.

Kada želimo konvertirati broj u pokretnom zarezu u cijeli broj postoje dva načina.

Prvi je način da od broja s pokretnim zarezom uzmemo cijeli dio, dakle dio koji ostane kad oduzmemo decimalni dio.

Npr., broj u pokretnom zarezu 4.7 tada postaje cijeli broj 4. Način kako to postizemo je upotreba `int` cast operatora:

```
double x = 4.7;
int i = (int) x;
```

Drugi izraz će konvertirati vrijednost od `x` u cijeli broj 4 i nakon toga pohraniti ga u varijablu `i`. Da je `x` bio postavljen na `-4.7`, `i` bi primio vrijednost `-4`.

Drugi način konvertiranja brojeva u pokretnom zarezu u cijele brojeve je zaokruživanje na najbliži cijeli broj. Npr. najbliži cijeli broj za 4.7 je 5.

Npr. slijedeći izraz obaviti će zaokruživanje pozitivnog broja `x` i pohraniti zaokruženu vrijednost u varijablu `i`:

```
int i = (int) (x + 0.5);
```

Npr., ako je `x` jednak 4.7, vrijednost od `x + 0.5` bit će 5.2, a vrijednost od `(int)(x + 0.5)` bit će 5.

(Upozorenje. Ne piši: `int i = (int)x + 0.5;`)

Kako bi bilo izvedeno zaokruživanje za negativne brojeve ?

U zaokruživanju broja u pokretnom zarezu (i pozitivnih i negativnih) na najbliži cijeli broj možete koristiti i metodu iz `Math` klase:

```
Math.round(x)
```

To je zgodno rješenje osim što `Math.round(x)` kao rezultat vraća vrijednost tipa `long`. Sve što je potrebno da bi se rezultat dodijelio varijabli tipa `int` je primjena `(int)` cast operatora :

```
int i = (int) (Math.round(x));
```

## 5. Čitanje ulaza korištenjem `ConsoleReader` objekta

Primjeri koji su dosada obrađivani koriste za svoj račun vrijednosti koje je programer upisao u sam kod programa. Ako u njima želimo neke druge vrijednosti moramo ih mijenjati u kodu programa.

Ljepša alternativa je da ponovo napišemo programe tako da njihove ulazne vrijednosti unosimo koristeći tipkovnicu tijekom izvršavanja programa. Npr. program koji konvertira valute mogao bi prvo ispisati poruku da korisnik upiše iznos u markama, zatim pročita ono što mu se upiše i na kraju ispiše rezultat.

U ovom dijelu napisati ćemo takav program.

Dosada smo u primjerima prikazivali informacije na ekranu koristeći `println` i `print` metode koje pripadaju objektu `System.out`.

Objekt `System.out` ima pristup ekranu i može proslijediti na njega bilo koje brojeve ili stringove koje mu damo kao parametre. Isto je točno kada čitamo s tipkovnice. U tom slučaju trebamo objekt koji je spojen na tipkovnicu i koji ima metode kojima možemo pristupiti znakovima koje kucamo na tipkovnici.

U Javi postoji takav objekt i naziva se `System.in`. Nažalost posjeduje veoma limitiran broj metoda. Posjeduje jednu metodu `read` koji će pročitati sve što korisnik upiše, ali ga je teško koristiti jer čita samo jedan karakter.

Nije teško konstruirati klasu s boljim setom metoda. U ovome tečaju koristit ćemo jednu klasu nazvanu `ConsoleReader`.

Kad je želimo koristiti u programu na početku programa napišemo:

```
ConsoleReader in = new ConsoleReader(System.in);
```

Primijetite da se radi o deklaraciji varijable. Ova deklaracija kreira novi objekt tipa klase `ConsoleReader` koji može pristupiti `System.in` objektu koji je spojen na tipkovnicu.

U isto vrijeme kreira se varijabla `in` tipa `ConsoleReader` koja označava taj objekt.

Kad se izvrši ta naredba objekt koji nam je potreban odsad je prisutan u memoriji i njegove metode možemo pozivati koristeći njegov naziv `in`. (Možemo ga i drukčije nazvati)

`ConsoleReader` objekt posjeduje tri korisne metode:

```
in.readInt() vraća int vrijednost.
```

To će biti cijeli broj koji korisnik utipka. Npr. izraz :

```
int broj = in.readInt(); //će kreirati varijablu broj i pridružiti joj vrijednost koja se ukuca s tastature.
```

```
in.readDouble(); // vraća double vrijednost.
```

## 2. Brojevi i znakovni nizovi

```
in.readLine(); // vraća string. Sastoje se od znakova koje korisnik
ukuca do kraja linije.
```

Ovo je nova verzija programa za preračun maraka u eure.

Započinje se kreiranjem `ConsoleReader` objekta, koji je označen sa `in`. Tada traži od korisnika da utipka vrijednosti maraka i pfeniga. Naredbe koje čitaju vrijednosti naznačene su podebljano (ne i u stvarnom kodu programa).

### PRIMJER 5

```
public class Mjenjacnica2
{ /* Ucitaj iznos u markama i preracunaj u eure */
public static void main(String[] args)
{

    ConsoleReader in = new ConsoleReader(System.in);
    System.out.println("Unesi iznose eura i centa");
        System.out.print("Iznos u eurima =");
    int eura = in.readInt();
    System.out.print("Iznos u centima =");
    int centa = in.readInt();

        centa = 100*eurima + centa;
        double kuna = centa * 0.0751;
        System.out.print("Kuna: ");
        System.out.print(kuna);

    }
}
```

Na ekranu bi trebalo nakon svega pisati (ono što korisnik ukuca je ovdje naznačeno podebljano)

```
Unesi iznose eura i centa
Iznos u eurima = 110
Iznos u centima = 55
Kuna: 830.2305
```

`ConsoleReader` objekt nije robusan. Npr. ako unesete dva broja u jednu liniju ili ako unesete slova umjesto broja program će javiti run-time grešku.

Prije pokretanja bilo kojeg programa koji koristi klasu `ConsoleReader` potrebno je da istu imate u istome direktoriju kao i program u kojem je koristite.

Pretpostavimo da želite koristiti primjer 5. Prvo ćete primjer ukucati u datoteku s istim nazivom kao i klasa dakle `Mjenjacnica2.java`. U isti direktorij kopirajte i datoteku `ConsoleReader.java`. Tada prevedite i pokrenite `Mjenjacnica2` na uobičajen način. Kada Java prevodilac vidi da `java` koristi `ConsoleReader` klasu, automatski će je prevest.

## 6. Znakovni niz - string.

Nadalje ćemo se koristiti izrazom **string**. **String** je jedan od najčešće korištenih tipova podataka u Javi.

Niz znakova (**literal** string) možemo napisati kao niz znakova zatvorenih u dvostruke navodnike poput slijedećeg:

```
"Jedan dan"
```

Literal stringovi mogu biti proizvoljne duljine, te uključivati bilo koji znak. Čak možete imati i string bez karaktera. Takav string nazivamo **prazan** string i zadaje se kao "".

## 2. Brojevi i znakovni nizovi

String može sadržavati **kontrolne znakove** (control characters). Ti znakovi znače stvari poput novog reda, tabulatora, ... Ako ispisujete string korištenjem `System.out.println` metode i ako ta metoda naiđe na kontrolni karakter npr. na newline kontrolni karakter ona će tada pomaknuti kursor na novi red.

Ako želite uključiti kontrolni karakter u literalni string, potrebno je korištenje `\` karaktera nakon kojega slijedi slovo. `\` se naziva **escape** karakter. Koristi se i za uvođenje određenih karaktera u string poput `"` koji bi inače značio kraj stringa. Slijedi lista najznačajnijih kombinacija karaktera:

<code>\n</code>	novi red (newline)
<code>\t</code>	tabulator – pomak (tab)
<code>\b</code>	nazad (backspace)
<code>\r</code>	return – pomak na početak slijedeće linije
<code>\f</code>	line feed – pomak na slijedeću liniju bez odlaska na početak
<code>\\</code>	<code>\</code> znak
<code>\'</code>	' znak
<code>\"</code>	" znak

Npr. izraz

```
System.out.print("Jedan\ndan");
```

napisat će 'Jedan' u jednu liniju te 'dan' u slijedeću liniju.

Ako želite pohraniti string da bismo ga koristili kasnije u programu koristit ćemo varijablu tipa **String**. Slijedi primjer izraza koji kreira novu varijablu tipa `String`, nazvanu `automobil` kojoj dodjeljuje vrijednost "BMW".

```
String automobil = "BMW";
```

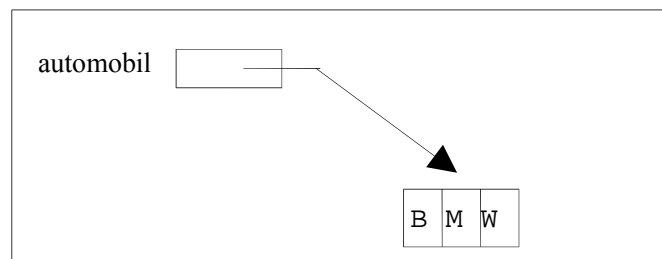
Nije ispravno da će ovaj izraz pohraniti string "BMW" na memorijsku lokaciju varijable `automobil`. Što se stvarno događa uključuje jednu vrlo bitnu osobinu rada s memorijom. Svaka lokacija u memoriji ima svoju adresu (**reference value** or **address**).

To je slično kao što svaka kuća u gradu ima svoju poštansku adresu

Kad se izvrši gornji izraz izvrše se slijedeći koraci:

- (1) Kreira se varijabla nazvana `automobil`
- (2) String "BMW" pohrani se u **drugi dio memorije !**
- (3) Referenca (adresa) gdje je string pohranjen dodijeli se varijabli `automobil`.

Rezultirajuće stanje u memoriji može se prikazati dijagramom na slici 1.



Slika 2.1. Strelica od varijable prema stringu reprezentira činjenicu da varijabla sadrži referencu na lokaciju gdje je string pohranjen.

Ovaj način tretiranja String varijable znači da je sama varijabla uvijek iste veličine – dovoljno velika da sadrži vrijednost reference (adrese)

Npr. ako kasnije izvršimo naredbu:

## 2. Brojevi i znakovni nizovi

---

```
automobil = "Alfa Romeo";
```

varijabla `automobil` će nakon toga opet sadržavati vrijednost reference, dakle istu količinu memorije. Razlika je u tome što će to biti referenca na neki drugi dio memorije koji sada sadrži drugi (duži) string "Alfa Romeo".

Postoji jedan vrlo koristan operator koji se može primijeniti na stringove. Naziva se operator spajanja (concatenation) i označava se znakom '+'.

Ako su `st1` i `st2` stringovi, onda `st1+st2` znači string koji se sastoji od svih karaktera `st1` nakon kojih slijede karakteri iz stringa `st2`.

Npr. ako je varijabli `automobil` dodijeljena referenca na string "BMW" onda će izraz

```
"Novi " + automobil + " je skup."
```

znači string:

```
"Novi BMW je skup."
```

Izrazi poput ovoga mogu se koristiti bilo gdje u programu gdje je potreban neki složeni izraz. Npr. slijedeći izraz :

```
System.out.println("Novi " + automobil + " je skup.");
```

će ispisati na ekranu "Novi BMW je skup."

Kako često trebamo ispisivati poruke koje sadrže i brojeve Java dopušta da se kombiniraju brojevi i stringovi korištenjem + operatora.

Npr. `cijena` je cjelobrojna vrijednost koja sadrži vrijednost 30000. Tada će izraz:

```
"Cijena novog " + automobil + " je " + cijena + " eura".
```

značiti string:

```
"Cijena novog BMW je 30000 eura".
```

Primijetite da je Java uzela vrijednost iz varijable `cijena` i pretvorila je u string sa znamenkama pogodnim prikazu cijelih brojeva.

I u primjerima 2 i 3 mogli smo koristiti sličan način :

Izraze:

```
System.out.print("Racun je: ");  
System.out.print(total/100);  
System.out.print(" kn i ");  
System.out.print(total%100);  
System.out.println(" lipa.");
```

Mogli smo zamijeniti s jednim izrazom:

```
System.out.println  
("Racun je: "+ (total/100) + " kn i " +  
(total%100) + " lipa. ");
```

Točnije, zagrade oko izraza `total/100` i `total%100` nisu potrebne jer Java primjeni pravilo da operatori `/` i `%` imaju veći prioritet od operatora `+`.

## 2. Brojevi i znakovni nizovi

Slijedi program koji obavlja luckastu konverzaciju s korisnikom. Program čita korisnikovo ime i dodjeljuje ga varijabli `ime`. Također program starost korisnika i dodjeljuje ga varijabli `godina`. Nakon toga koristi te dvije vrijednosti da bi konstruirao par rečenica koje će ispisati na ekran.

Program započinje konstrukcijom `ConsoleReader` objekta da bi mogao čitati što korisnik utipka. U ovom slučaju taj objekt je označen nazivom `korisnik`. Metoda `korisnik.readLine()` koristi se za čitanje imena, a metoda `user.readInt()` za čitanje broja godina.

### EXAMPLE 6

```
public class Prica
{
    /* Pricaj s korisnikom. */

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

        System.out.println
            ("Cao. kako se zoveš?");
        String ime = korisnik.readLine();
        System.out.println
            ("Koliko imaš godina " + ime + "?");
        int godina = korisnik.readInt();
        System.out.print(godina + " su lijepe godine, ");
        System.out.println
            ("ali " + (godina+1) + " je bolje.");
        System.out.println
            ("Vidimo se kasnije " + ime + "!");
    }
}
```

Ovo bi se trebalo pojaviti na ekranu kad se program pokrene. (Korisnikov unos je podebljan.)

```
Cao. kako se zoveš?
Ivica.
koliko imaš godina Ivica?
20
20 su lijepe godine , ali 21 je bolje.
Vidimo se kasnije Ivica!
```

Stringovi imaju mnogo korisnih metode koje su vezane za njih.

Jedna od vrlo korisnih se naziva `substring`. Java koristi konvenciju da su karakteri unutar stringa numerirani s 0, 1, 2, ... s lijeva nadesno. Pretpostavimo da `st` označava string, a da su `m` i `n` cjelobrojne vrijednosti. Tada izraz :

```
st.substring(m,n)
```

daje novi string koji se sastoji od karaktera stringa `st`, počevši od pozicije `m` i završavajući s pozicijom `n-1`. (`n` je prva pozicija koja nije uključena !). Npr. ako varijabla `automobil` označava string "Mercedes", onda izraz

```
automobil.substring(5,8)
```

će dati string "des", kako je to dolje pokazano.

M e	r	c	e	<b>d</b>	<b>e</b>	<b>s</b>			
	0	1	2	3	4	5	6	7	8

## 2. Brojevi i znakovni nizovi

---

Navest ćemo još neke standardne metode koje možemo koristiti u radu sa stringovima. U svakom primjeru st označava string.

<code>st.length()</code>	Daje broj znakova u stringu ("duljina" stringa st) Npr. <code>"Mercedes".length()</code> vraća vrijednost 8.
<code>st.toLowerCase()</code>	Daje string sa svim velikim slovima pretvorenim u mala slova. Npr. <code>"Mercedes".toLowerCase()</code> vraća string <code>"mercedes"</code>
<code>st.toUpperCase()</code>	Daje string sa svim malim slovima pretvorenim u velika slova. Npr. <code>"Mercedes".toUpperCase()</code> vraća string <code>"MERCEDES"</code>

Ako ste izračunali broj i želite ga pretvoriti u odgovarajući string sve što je potrebno je kombinirati prazni string i broj koristeći `+` operator. Npr. ako varijabla `rezultat` sadrži vrijednost 142, onda izraz `"" + rezultat` će proizvesti string `"142"`.

To radi jer Java koristi pravilo koje kaže da ako kombinirate string i broj pomoću operatora `+`, broj će biti konvertiran u string, a zatim će oba stringa biti spojena. U ovome slučaju string nastao iz zbroja je spojen s praznim stringom.

U drugom smjeru nije tako jednostavno. Potrebno je koristiti neki od slijedećih metoda.

`Integer.parseInt(st)` Ako string `st` predstavlja cjelobrojnu vrijednost ovo će dati odgovarajuću vrijednost

`Double.parseDouble(st)` Ako string `st` predstavlja broj u pokretnom zarezu ovo će dati odgovarajuću vrijednost.

Na kraju ovog dijela još pokoje pravilo o kombiniranju brojeva i stringova korištenjem `+` operatora. Razmotrimo slijedeće izraze:

```
"Rezultat je " + 3 + 7
```

```
3 + 7 + " je rezultat"
```

na prvi pogled izgleda kao da želimo reći istu stvar. Međutim ako ih ispišemo koristeći metodu `System.out.println` vidjet ćete da ćete dobiti sasvim nešto drugo.

Da biste predvidjeli što će se u ovakvim slučajevima potrebno je koristiti slijedeće pravila:

Ako Java treba odrediti vrijednost izraza `x+y` gdje su `x` i `y` bilo brojevi ili stringovi, tada će se dogoditi slijedeće:

(1) Ako su `x` i `y` oboje stringovi onda će ih spojiti.

Ako je jedan string, a drugi broj onda će pretvoriti broj u string i spojiti ga s drugim stringom

Ako su `x` i `y` oboje brojevi bit će zbrojeni.

Ako se računa izraz koji sadrži više `+` operatora izraz se računa slijeva nadesno !

Ponovo razmotrimo izraz:

```
"Rezultat je " + 3 + 7
```



## 2. Brojevi i znakovni nizovi

Prvo što će Java obraditi je "rezultat je " + 3. Pretvorit će broj 3 u string "3", spojiti ga s prvim stringom i dati "Rezultat je 3". Zatim će obraditi "Rezultat je 3" + 7, i konačno dati string:

```
"Rezultat je 37"
```

Sad razmotri:

```
3 + 7 " je rezultat"
```

Ovaj put će Java prvo zbrojiti 3 i 7 i dati broj 10. Zatim će obraditi 10 + " je rezultat " i konačno dati:

```
"10 je rezultat "
```

## 7. Zadaci za poglavlje 2

Za ove zadatke kreirajte direktorij P2 unutar direktorija JProg. Potrebno je unijeti i prevesti i izvršiti 4 zadatka.

1. Zadatak (Koristi naziv klase i datoteke Aritmetika)  
Napiši program koji traži od korisnika da ukuca dva cijela broja i zatim ispiši:

```
sumu  
razliku  
umnožak  
prosjek (double !)  
udaljenost (apsolutna vrijednost razlike)
```

\*1Koristi metodu `Math.abs` za račun apsolutne vrijednosti

2. Zadatak (Koristi naziv klase i datoteke Mjenjacnica)

Napiši program koji učitava iznos u kunama i preračunava ih u eure (kurs npr. 1 euro = 7.50 kn)

Zadatak. (Koristi naziv klase i datoteke Upoznavanje)

Napiši program koji će ispisati prva dva reda kao što je prikazano, zatim učitati ime i nakon toga ispisati pozdrav. Podebljan je upis od strane korisnika . Ime može biti bilo koje.

```
Bok. Moje ime je Java.  
Kako se ti zoveš ?  
Ivica  
Ivica ! Drago mi je što smo se upoznali.
```

4. Zadatak. (Koristi naziv klase i datoteke Tablica)  
Napiši program koji će nacrtati tablicu na ekranu..

```
+---+---+---+  
|   |   |   |  
+---+---+---+  
|   |   |   |  
+---+---+---+  
|   |   |   |  
+---+---+---+
```

Učini to na način da kreiraš dvije string varijable za oba uzorka pa onda te varijable naizmjenično ispiši.



## 3. Objekti i klase

U prva poglavlja koristili smo nekoliko objekata: objekt koji se zove `System.out` za slanje informacija na izlaz, `ConsoleReader` za čitanje informacija s tipkovnice, i `String` objekt.

Ovo poglavlje bavi se konstruiranjem novih objekata. Većinom ćemo se baviti konstruiranjem objekta koji će sadržavati informacije o studentu.

Sadržaj:

1. Što je to objekt ?
2. Klasa student
3. Više o klasama
4. Reference na objekte (Objects References).
5. `null` vrijednost
6. Identifikatori
7. Zadaci

### 1. Što je to objekt?

U ranim danima programiranja prvi programski jezici poput Fortrana dozvoljavali su programerima da rukuju s vrlo ograničenim skupom tipova podataka npr. cijeli brojevi, brojevi s pokretnim zarezom i nizovi brojeva.

Nekih 10 godina kasnije programski jezik Simula 67 omogućio je programerima definiranje vlastitih tipova podataka nazvanih *objekti*. Objekti su se dokazali kao jedna od najznačajnijih inovacija u programiranju. Danas predstavljaju osnovu programskih jezika poput C++, Java, C# pa čak i Visual Basica.

Što je to objekt ? Postoje dva gledišta na objekt. Jedno sa strane programera koji koristi objekt, a drugo sa strane programera koji dizajnira objekt. Prvog programera nazvat ćemo "korisnik", a drugog programera "dizajner" (možda je bolji naziv "konstruktor"). Često se događa da isti programer prvo dizajnira objekt, a zatim ga koristi. Međutim vidjeli smo na primjeru objekata tipa `ConsoleReader` da je taj objekt dizajnirao neki drugi programer, a mi smo ga vrlo lako koristili.

Razlika u tome što dizajneru predstavlja objekt, a što predstavlja korisniku nije samo ograničena na Javu. Pogledajte na vaš sat(ako ga imate). Ako je dobro dizajniran omogućavat će očitavanje vremena, promjenu vremena, start-stop funkciju štoperice, itd. vama kao korisniku nije potrebno poznavati što se događa unutar sata, koji su to mehanizmi koji stoje iza svih njegovih funkcija. Vi ste "korisnik" sata. S druge strane osoba koja je dizajnirala sat znat će sve što se događa unutra. Dizajneri će potrošiti dosta vremena pokušavajući konstruirati sat da bi vama omogućili korištenje uobičajenih funkcija sata.

Isto je točno za bilo koji uređaj npr. automobil, mobitel, ...

Iako ih ne možete opipati Java objekti su isto tako realni. Programer korisnik je upućen koje su funkcije pridružene pojedinoj vrsti objekata. te funkcije nazivamo **sučelje (interface)**. Programer koji je programirao objekt zna koje će se operacije izvršavati kad se te funkcije pozovu. Taj dio nazivamo **implementacija** objekta.

Korisnik će na pitanje "Što je to objekt" odgovoriti s "Ne znam od čega se sastoji već mi je poznato njegovo sučelje i što mogu s njim učiniti". Dizajner će odgovoriti s: "Sastavljen je od varijabli i metoda koje međusobno djeluju da bi realizirale sučelje objekta.

U ovom poglavlju pokušat ćemo objasniti kako objekti izgledaju sa stanovišta dizajnera objekata. U stvari dizajnirat ćemo pojedine objekte.

Svi objekti koji se grade na osnovu istog dizajna pripadaju istoj **klasi**. Dizajn klase je dio programa koji nazivamo **definicija klase**.

## 2. Klasa Student

Zamislite da ćemo u Javi napisati program koji će u bazi podataka održavati podatke o svim studentima na ovome fakultetu. Program poput toga vjerojatno bi sadržavao mnogo različitih tipova objekata. Jedan o tipova bi sigurno bio onaj koji bi sadržavao podatke o pojedinom studentu. Za očekivat je i postojanje objekta koji bi predstavljao listu svih studenata u bazi podataka.

U ovome poglavlju nećemo pokušavati napisati kompletan program već ćemo se koncentrirati na dizajn objekata i to na objekt koji će sadržavati podatke za pojedinog studenta. klasu koja predstavlja dizajn te vrste objekta nazvat ćemo **Student**.

Ako bismo dizajnirali sat, počeli bismo od ideje kako konstruirati sučelje tj. skup operacija koje sat treba izvoditi. Nakon toga bismo radili na implementaciji sučelja, odnosno kako složiti stavi unutar objekta da se osigura tražena funkcionalnost.

Slično, prvo u definiranju klase Student je odluka koje operacije će biti asociirane s objektom tipa Student.

Da bismo održali jednostavnost primjera uključit ćemo ograničeni skup operacija:

Bit će potrebno omogućiti mijenjanje godine studenta s 1 na 2 ili 2 na 3. Ova akcija će se normalno odvijati na kraju godine.

Bit će potrebno omogućiti promjenu programa studija (stupnja)

Bit će potrebno omogućiti način da objekt vrati ime studenta kako bi program mogao ispitati kojemu studentu pripada odgovarajući objekt. Dobivanje ostalih informacija zasad nećemo realizirati.

Bit će potrebno omogućiti ispisivanje podataka o studentu na ekran. Osigurat ćemo metodu  `prikaz`  koja će to činiti.

Bit će potrebno kreirati nove objekte tipa Student. Svaki put kad bude kreiran bit će potrebno dati informacije o studentu, npr. ime, program studija (stupanj) , ...

Sad je vrijeme za definiciju klase student. U terminologiji ručnog sata, potrebno je odrediti kako će pojedine komponente biti raspoređene i povezane unutar kućišta da bi proizvele sat koji radi.

Svaki objekt u Javi sastavljen je od nekog broja odvojenih dijelova nazvanih **članovi (members)**. Postoje tri vrste članova.

- **Varijable.** U ove članovi spremaju se informacije koje objekt sadrži. Npr. Student objekt može sadržavati varijablu koje sadrži studentovo ime i drugu koja sadrži naziv programa studija. Skup vrijednosti koje sadrže varijable pojedinog objekta nazivaju se **stanje (state)** objekta. Da bismo razlikovali te varijable od varijabli koje smo definirali unutar metoda (u prošlom poglavlju), varijable koje pripadaju objektu nazivamo **polja (fields)** ili varijable instance (**instance variables**).

**Metode.** Ovi članovi izvršavaju operacije na objektu (ili s objektom). Npr. vidjeli smo da ConsoleReader objekt ima metodu koja se naziva `readInt` koja koristi taj objekt da bi se dobila vrijednost koju je korisnik utipkao. Kada napišemo definiciju klase Student imat ćemo metodu `uNovuGodinu` koja će dodavati jedan na tekuću studentovu godinu.

Ova vrsta metoda nazivaju se **metode instance (instance methods)** kako bi ih razlikovali od statičkih metoda koje ne pripadaju nijednom objektu (više o statičkim metodama kasnije

### 3. Objekti i klase

**Konstruktori.** Ovi članovi se koriste da bismo konstruirali objekte određene klase. U klasu student uključit ćemo jedan konstruktor jer inače ne bi bilo moguće napisati program koji bi koristio objekte te klase. Konstruktori su vrlo slični metodama na mnogo načina.

Sad ćemo specificirati koje će članove posjedovati objekti tipa Student. Prvo ćemo definirati polja (fields). Te varijable sadržavat će informacije koje su pohranjene u svakom objektu tipa Student. Pretpostavimo za sada 4 informacije (slaže se sa zahtjevima operacija navedenih prije.)

1. Studentov ID.
2. Ime.
3. Naziv programa.
4. Godina studija.

Potrebno je odabrati nazive za ta polja i odlučiti koji će se tip podataka koristiti.

Npr. ID broj može biti cijeli broj ili string. ID broj je u stvarnosti samo niz karaktera. Neće biti nikakve razlike ako se sastoji od niza znakova umjesto od znamenaka. Zbog toga ćemo odabrati tip podatka `String`. Ime studenta i naziv programa studija također će biti tipa `String`. Godina studija bit će cjelobrojna vrijednost tipa `int`.

Slijede definicije polja. Linije koje počinju s `//` su komentari koji objašnjavaju sadržaj svakog polja. Java ignorira sve od `//` do kraja reda.

```
private String idBroj;
    // ID broj Studenta.

private String ime;
    // studentovo ime.

private String programStudija;
    //Program studija.

private int godina;
    // Godina studija
    // (1, 2 ili 3).
```

Riječ `private` određuje da se polju ne može direktno pristupiti izvan definicije klase Student. Ako programer "korisnik" objekta napiše kod koji zove takvu varijablu prevodioc će javiti grešku prevodioca. Polja u klasi su obično označena kao `private`. (Kad dođemo do metoda asociiranih sa objektom Student, vidjet ćemo da počinju s riječju `private` koja znači da to metodu može pozvati korisnik objekta.

Nakon riječi `private` dolazi tip vrijednosti koje spremamo u polje te naziv polja. Cijela linija predstavlja deklaraciju varijable. Moguće je uključiti i inicijalnu vrijednost.

Kad smo odabrali polja koja će objekt sadržavati, sad trebamo napisati metode koje će izvršavati osnovne operacije pridružene objektu.

Prvo ćemo konstruirati metodu koja će se koristiti za promjenu studentovog programa studija. Nazvat ćemo je **setProgramStudija**. Kad je pozovemo bit će potrebno znati koji je novi naziv novog programa studija. Ta dodatna informacija bit će **parametar** metode. Općenito metoda može imati bilo koji broj parametara. Svaki put kad pozovemo metodu svakome od tih parametara bit će dodijeljena vrijednost koju će moći koristiti tijekom izvršavanja metode.

`setProgramStudija` ima samo jedan parametar – naziv studentovog novog programa. U definiciji metoda taj parametar nazvat ćemo jednostavno `p`.

Tijelo svakog parametra sastojat će se od niza izraza koji će izvršavati potrebne akcije. U ovom slučaju potrebna je samo jedna akcija: novi naziv programa studija (dakle parametar `p`) potrebno je pohraniti u polje `programStudija`. Slijedeće dodjeljivanje će obaviti taj zadatak:

### 3. Objekti i klase

---

```
programStudija = p;
```

Slijedi kompletna definicija metoda `setProgramStudija`:

```
/* Promijeni programStudija u p. */

public void setProgramStudija(String p)
{
    programStudija = p;
}
```

Kao obično počinjemo s komentarom što program radi. Slijedeća linija je zaglavlje metoda . Svaka od riječi ima značenje:

`public` kaže da se metoda može koristiti bilo gdje u programu, dakle i unutar klase i van definicije klase

`void` kaže da metoda ne vraća nikakvu vrijednost.

`setProgramStudija` je naziv metode.

`(String p)` je lista svih parametara metode. U ovome slučaju radi se o samo jednom parametru koji je tipa `String`, a naziva se `d`.

Na kraju imamo tijelo metoda zatvoreno u vitičaste zagrade `{...}`. U ovome slučaju tijelo metoda sastoji se samo od jedne naredbe.

Pretpostavimo da programer korisnik piše nekakav program u nekoj drugoj klasi odvojeno od klase `student`. Pretpostavimo da je kreirao određeni objekt studenta i pridodijelit ga varijabli `stud1`. Tada ako programer želi postaviti za tog student program studija na npr. na strojarstvo može pisati naredbu:

```
stud1.setProgramStudija("strojarstvo");
```

Ovaj naredba kaže: izvrši metodu `setProgramStudija` koja je asocirana s objektom `stud1` (tipa `Student`) koristeći vrijednost parametra "strojarstvo".

Rezultat će biti će polje objekta `programStudija` biti postavljeno na vrijednost "strojarstvo".

Slijedeće ćemo definirati metodu koja dodaje jedan na studentovu godinu studija. Ovo je još jednostavnije jer metoda nema nikakvih parametara. Cijelo tijelo je izraz koji dodaje jedinicu na polje godina .

```
godina++;
```

Slijedi kompletna metoda.

```
/* Povećaj godinu studija za jedan */

public void povecajGodinu()
{
    godina++;
}
```

Što kaže zaglavlje:

`public` znači da se metoda može koristiti bilo gdje u programu.

`void` znači da metoda ne vraća nikakvu vrijednost.

`povecajGodinu` je naziv metode.

### 3. Objekti i klase

---

`()` pokazuje da nema parametara. To znači da se ne prosljeđuje nikakva informacija metodu koji se poziva.

Van klase student slijedeći izraz može biti korišten za dodavanje jedinice na broj godine studenta. Koristimo opet objekt `stud1` (klase `Student`)

```
stud1.povecajGodinu();
```

Slijedeća metoda vraća ime koje je pohranjeno u objektu tipa `Student`. Podsjetite se da se toj varijabli ne može pristupiti direktno izvan klase jer je polje `ime` označeno kao `private`. Metodu smo nazvali `getIme`. Za razliku od dva prethodna metoda ovaj "vraća vrijednost". Vrijednost se vraća na mjesto poziva metoda. Pretpostavimo da imamo slijedeću naredbu:

```
String st = stud1.getName();
```

Java će kreirati varijablu `st` te nakon toga pozvati metodu

```
stud1.getName()
```

Metoda će biti pozvana za objekt `stud1`. (metoda nema parametara) Metoda će vratiti ime studenta, i Java će ga pohraniti u varijablu `st`. Primijetite da je `stud1.getName()` korišten na isti način kao što je korišteno npr. `Math.sqrt(1.5)`

```
double x = 1 - Math.sqrt(1.5);
```

Ovdje je kompletna metoda:

```
/* Vrati studentovo ime. */  
  
public String getIme()  
{  
    return ime;  
}
```

Zaglavlje znači:

`public` kaže da se metoda može koristiti bilo gdje.

`String` kaže da metoda vraća vrijednost tipa `String`. Ovo se naziva **vraćeni tip podatka (return type)**.

`getIme` je naziv metoda.

`()` pokazuje da metoda nema parametara.

Tijelo metoda je jedna naredba:

```
return ime;
```

Ona kaže: vrati vrijednost polja `ime`. To je tip naredbe koji nismo dosada sreli. Naziva se **naredba vraćanja vrijednosti(return statement)**.

Zadnja metoda ispisuje na ekran sve informacije o studentu. Nazvat ćemo je `prikaz`

```
/* Prikaži informacije o studentu na ekran. */  
  
public void prikaz()
```

### 3. Objekti i klase

---

```
{ System.out.println
    ("Student ID: " + idBroj);
  System.out.println
    ("Ime: " + ime);
  System.out.println
    ("Program studija: " + programStudija);
  System.out.println
    ("Godina: " + godina);
}
```

Zaglavlje pokazuje da metoda ne vraća nikakvu vrijednost i nema parametara pa će poziv metoda izgledati ovako:

```
stud1.prikaz();
```

Naposljetku treba napisati konstruktor koji će programer korisnik moći upotrijebiti za stvaranje novih objekata tipa `Student`. Konstruktor uvijek ima naziv identičan nazivu klase. U ovom slučaju konstruktor se zove `Student`. Konstruktor nema naznaku što vraća jer je očito što bi trebao vratiti, a to je objekt koji se njime kreira. Konstruktor može imati bilo koji broj parametara. Ovaj ovdje ima tri: studentov ID broj, ime i naziv programa studija. Ovdje je pretpostavljeno da će godina studija biti 1 kada je objekt kreiran.

Slijedi konstruktor:

```
/* Kreiraj novog studenta sa zadanim
   ID brojem, imenom i programom studija
   Polje godina bit će postavljeno na 1.
*/

public Student(String id, String im, String p)
{ idBroj = id;
  ime = im;
  programStudija = p;
  godina = 1;
}
```

Sa dana tri parametra konstruktor će:

1. stvoriti novi objekt,
2. pohraniti string `id` u polje `idBroj` novostvorenog objekta,
3. pohraniti string `im` u polje `ime`,
4. pohraniti string `p` u polje `programStudija`,
5. pohraniti `1` u polje `godina`.

Primijetite da ne postoji ništa u tijelu konstruktora što indicira prvi korak stvaranja objekta. Java to radi automatski kada izvršava konstruktor.

Slijedi navedeni konstruktor iskorišten u stvaranju objekta tipa `Student`:

```
Student stud1 =
    new Student("9912345", "Ivo Petrić", "strojarstvo");
```

Ovo znači: prvo kreiraj varijablu nazvanu `st` koja će se odnositi na `Student` objekt. Zatim kreiraj novi objekt tipa `student` s podacima naznačenim u listi parametara.

Primijetite upotrebu ključne riječi `new`. Potrebno ju je upotrijebiti svaki put kada s konstruktorom stvaramo novi objekt. Svaka varijabla koja će označavati objekt tipa `Student` mora biti deklarirana kao varijabla tipa `Student`. (vrijedi za sve tipove objekata)



### 3. Objekti i klase

S ovim smo kompletirali definiciju klase Student. Još ostaje dodati zaglavlje cijele klase. U klasi su prvo stavljena polja iako redosljed članova nije bitan.

#### PRIMJER1

```
/* Student objekt za studenta FESB-a
*/

{ public class Student

    private String idBroj;
        // ID broj Studenta.

    private String ime;
        // studentovo ime.

    private String programStudija;
        //Program studija.

    private int godina;
        // Godina studija
        // (1, 2 ili 3).

    /* Promijeni programStudija u p. */

    public void setProgramStudija(String p)
    { programStudija = p;
    }

    /* Povećaj godinu studija za jedan */

    public void povecajGodinu()
    { godina++;
    }

    /* Vрати studentovo ime. */

    public String getIme()
    { return ime;
    }

    /* Prikaži informacije o studentu na ekran. */

    public void prikaz()

    { System.out.println
        ("Student ID: " + idBroj);
      System.out.println
        ("Ime: " + ime);
      System.out.println
        ("Program studija: " + programStudija);
      System.out.println
        ("Godina: " + godina);
    }

    /* Kreiraj novog studenta sa zadanim
      ID brojem, imenom i programom studija
```

### 3. Objekti i klase

---

```
        Polje godina bit će postavljeno na 1.
    */

    public Student(String id, String im, String p)
    {
        idBroj = id;
        ime = im;
        programStudija = p;
        godina = 1;
    }
}
```

Konačno jedan veoma jednostavan program koji će koristiti objekt tipa Student. Poput programa iz prošlog poglavlja sastoji se od samo jedne statičke metode `main`. On je sam stavljen u klasu koju ćemo nazvati `TestStudent`. Metoda prvo čita informacije o studentu unesene preko tipkovnice. Nakon toga kreira objekt tipa `Student` koristeći konstruktor i varijablu `st`. Nakon toga povećava godinu studija za jedan. koristeći izraz:

```
st.povecajGodinu();
```

Kako je konstruktor već postavio tu vrijednost na 1 sad bi trebala biti 2.

Zatim se program studija promijeni s izrazom:

```
st.setProgramStudija("elektronika");
```

Na kraju se podaci o studentu ispišu na ekran.

```
st.prikaz();
```

Slijedi `TestStudent` klasa:

#### PRIMJER 2

```
public class TestStudent
{
    /* Učitaj podatke o studentu.
       Kreiraj objekt tipa student.
       Prikaži ne ekranu podatke objekta.
    */
    public static void main(String[] args)
    {
        /* Kreiraj ConsoleReader za učitavanje
           što korisnik tipka */
        ConsoleReader in =
            new ConsoleReader(System.in);

        /* Učitaj podatke o studentu i kreiraj
           objekt za njih. */

        System.out.println
            ("Koji je ID broj studenta?");

        String i = in.readLine();

        System.out.println("Koje je ime studenta?");

        String n = in.readLine();
    }
}
```

### 3. Objekti i klase

---

```
System.out.println("Koji je program studija?");

String d = in.readLine();

Student st = new Student(i,n,d);

/* Povećaj broj godine studija
   i prikaži informacije na ekranu. */
st.povecajGodinu();
st.setProgramStudija("elektronika");
System.out.println();
st.prikaz();
    }
}
```

Ovo se može pojaviti na ekranu tijekom izvođenja programa:

```
Koji je ID broj studenta?");
9912345
Koje je ime studenta?
Ivo Petric
Koji je program studija?
Strojarstvo

Student ID: 9912345
Name: Ivo Petric
Program Studija: elektronika
Godina: 2
```

### 3. Više o klasama

U tijeku izvršavanja vaših Java programa, bit će kreirani različiti objekti i bit će pozivane neke od njihovih metoda. Konstruktori će se koristiti za kreiranje objekata, ali neće se pozvati niti jedan izraz za njihovo uklanjanje. Objekte će ukloniti iz memorije interpreter u trenutku kad mu zatreba više memorije. Interpreter provjerava sve postojeće objekte i briše one objekte na koje ne upućuje niti jedna varijabla ili drugi objekt. Taj proces naziva se "odvoz smeća" (garbage collection).

Proces kreiranja objekata i izvršavanja metoda počinje unutar `main` metode. Kada pozovete interpreter s naredbom:

```
java klasa
```

gdje je klasa prevedena klasa (bytecode), interpreter će poći do definicije klase i tražiti metodu sa zaglavljem

```
public static void main(String[] a)
```

(Naziv parametra može biti bilo koji naziv.) Ako interpreter nađe takvu metodu početak će izvršavati njene naredbe. Da bi uspješno upravljao s potrebnim objektima i metodama, interpreter mora imati pristup definicijama svih objekata koje treba koristiti. Neke od tih klasa ćete i sami napisati. Neke će biti napisane u Java biblioteci, poput `PrintStream` klase koja sadrži `System.out.println` metodu. Neke klase će napisati drugi programeri, poput `ConsoleReader` klase.

Npr. za uspješno pokretanje programa iz primjera 2 osim klasa iz biblioteka potrebne su još tri klase: `TestStudent`, `Student` i `ConsoleReader`. One će se obično nalaziti u tri datoteke nazvane: `TestStudent.java`, `Student.java` i `ConsoleReader.java`.

U procesu prevođenja bit će dovoljno izvršiti naredbu

### 3. Objekti i klase

---

```
javac TestStudent.java
```

Prevodioc će naći u klasi `TestStudent` reference ne dvije ostale klase i automatski ih prevesti. Napišite slijedeću naredbu:

```
java TestStudent
```

intepreter će doći do `TestStudent` klase i potražiti `public, static` metodu s nazivom `main`. Ako je uspije pronaći pokušat će je izvršiti. Ako je ne nađe javit će odgovarajuću pogrešku i stati s interpretiranjem.

Da bismo napravili program kompaktnijim moguće je staviti `main` metodu u klasu `Student` i izbjeći stvaranje `TestStudent` klase. Tada ćete ukucati naredbu :

```
java Student
```

To će natjerati intepreter da u klasi `Student` traži `main` metodu.

Kada pogledate na programski kod koji opisuje različite članove klase vidjet ćete da definicija svakog člana počinje s ključnom riječi `public` ili `private`.

Ako je član neke klase označen kao `public` tada mu se pristupiti iz bilo koje metode (ili konstruktora) bilo gdje u programu. Ako je označen kao `private` onda mu se može pristupiti samo iz metoda koje su članovi iste klase. To znači da kada pišete definiciju klase A onda u nju ne možete uključiti izraze koji pozivaju članove klase B koji su označeni kao `private`. Možete pristupiti samo `public` članovima objekta B. Članovi označeni s `public` sačinjavaju sučelje objekta (interface).

Postoji niz razloga zbog kojih bi programer koji dizajnira klasu ograničio pristup drugome programeru svim poljima i metodama unutar klase. Razlog je da se korisniku klase koji ne poznaje u potpunosti rad klase ograniči mogućnost krive upotrebe klase. S druge strane kad je implementirano sučelje klase ista se može isporučiti korisniku, a za kasnije ostaviti poboljšanje unutarnje funkcionalnosti klase poput ubrzanja i slično. Naravno treba paziti da se ne promijeni sučelje.

Java omogućava da se izostavi riječ `public` ili `private`. Tada će polje ili metoda biti tretirana kao da je `public`.

Dosad smo diskusiju o klasama u ovom poglavlju zasnivali na ideji da je definicija klase opis određenog objekta. To je u određenoj mjeri i točno, ali klase su ipak malo kompliciraniji pojam. Svaka klasa ima niz varijabli i metoda koje egzistiraju neovisno u bilo kojem određenom objektu. Takvi članovi klase poznati su kao **statička** polja i metode. Drugi naziv je s engleskog neprevodiv – class fields and methods. U njihovim definicijama pojavljuje se ključna riječ `static`.

Počeli smo poglavlje 1 s programom koji se sastojao od samo jedne klase (`Hello`), i sadržavao je samo jednu metodu `main`. Zaglavlje metode je bilo:

```
public static void main(String[] a)
```

Upotreba riječi `static` ovdje indicira da ova metoda nije asocirana s niti jednim objektom. U stvari kada se program pokrene ni jedan objekt tipa klase `Hello` neće biti kreiran !

Uloga ove klase je samo da omogući prikladno mjesto za `main` metodu.

Druga klasa od koje se ne proizvode objekti, ali je sačinjena od statičkih polja i metoda je `Math` klasa iz Java biblioteke funkcija. Ona sadrži metode koje računaju uobičajene matematičke funkcije poput sinusa i kosinusa. Slijedi primjer izraza koji koristi navedenu klasu:

```
double x = Math.sin(a);
```

### 3. Objekti i klase

Dosad smo vidjeli da kada pozivate metodu koja pripada instanci neke klase da se prvo piše naziv objekta koji pokazuje na objekt neke klase. U slučaju pozivanja statičkog metoda (koji nije asociran s niti jednim objektom) **piše se ispred naziva metoda naziv klase**, dakle u ovom slučaju `Math`.

### 4. Reference na objekte (Object References)

U prethodnom poglavlju spomenuli smo da se vrijednosti poput cijelih brojeva i brojeva u pokretnom zarezu tretiraju na drukčiji način nego objekti. Jedna od različitosti je način na koji se pohranjuju u memoriji. Vrijednost varijable primitivnog tipa spremljena je u samoj varijabli. Pretpostavimo da je izvršen slijedeći izraz:

```
int n = 5;
```

Nakon ovoga izraza 32 bita memorije su rezervirana za novu varijablu `n`, i vrijednost 5 je spremljena na to mjesto u memoriji.

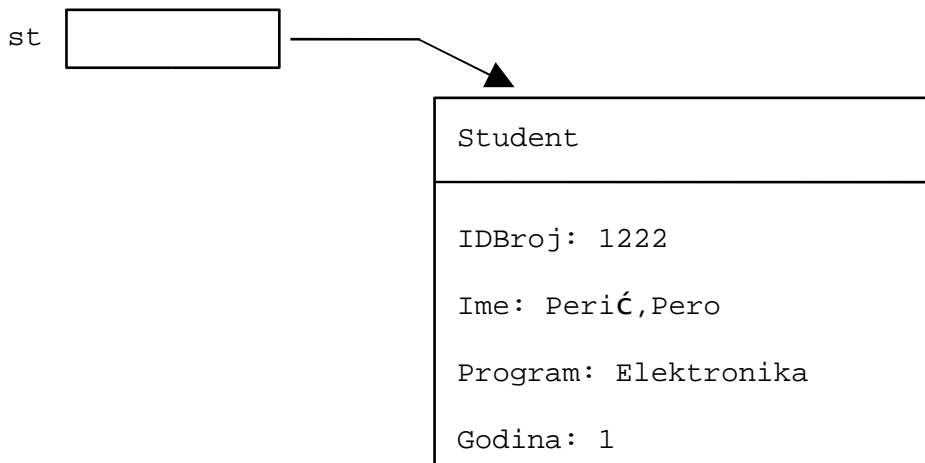
Ovaj postupak možemo vizualizirati na slijedeći način: (Pravokutnik pokazuje rezervirano mjesto u memoriji)



Kako smo vidjeli u odjeljku o String klasi, objekt se ne sprema u varijablu. Pretpostavimo da je izvršena slijedeća naredba:

```
Student st =  
    new Student("1222", "Perić,Pero", "Elektronika ");
```

Kao i prije određeno mjesto u memoriji bit će alocirano za novu varijablu. U ovom slučaju to je varijabla `st`. Međutim **objekt** koji se kreira u desnom dijelu izraza **neće biti spremljen na mjesto varijable**. Za njega će se rezervirati posebno mjesto u memoriji. Vrijednost koja je pohranjena u `st` je **vrijednost reference**. **Referenca** je vrijednost koja jednoznačno određuje gdje je u memoriji lociran objekt (u ovom slučaju objekt tipa `Student`). U slijedećem dijagramu strelica od varijable `st` do objekta tipa `Student` znači da varijabla sadrži referencu na objekt.



Kada pokrenemo Java program svaka vrijednost bit će ili primitivna vrijednost ili vrijednost reference. To vrijedi za vrijednosti pohranjene u varijablama, vrijednosti vraćene pomoću metoda i vrijednosti prosljeđene kao parametri.

Nikad nećemo moći vidjeti stvarnu vrijednost reference. Nemojte misliti da su to cijeli brojevi. Sve što znamo je da samo varijabla koja je npr. tipa `Student` može sadržavati referencu na objekt tipa `Student`.

Postavlja se pitanje da li ima razlike u tome da varijabla sadrži referencu na objekt, a ne sam objekt? Ima. Razmotrimo slijedeći kod:

### 3. Objekti i klase

```
Student st1 =  
    new Student("1222", "Perić,Pero", "Elektronika");  
  
Student st2 = st1;  
  
st1.setProgramStudija("Strojarstvo");  
  
st2.prikaz();
```

Što će se prikazati na ekranu. Što će biti ispisano za programStudija. Odgovor je da će to biti "Strojarstvo". Da bi to znali moramo razmotriti liniju po liniju:

1. `Student st1 =  
 new Student("1222", "Perić,Pero", "Elektronika");`

Ovo će kreirati varijablu `st1`, zatim objekt tipa `Student` koji će sadržavati zadane podatke te naposljetku u `st1` bit će spremljena referenca na kreirani objekt.

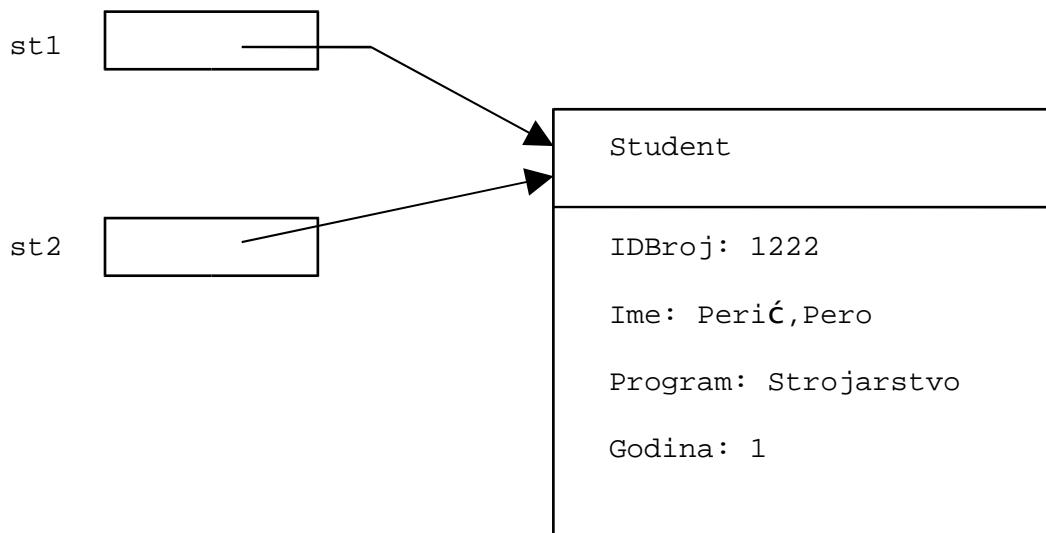
2. `Student st2 = st1;`

Ova linija će kreirati varijablu `st2` i dodijeliti joj vrijednost koja se nalazi u varijabli `st1`, u ovom slučaju referencu na objekt `student` koji je kreiran u prvom koraku. **Primijetite da iako postoje dvije varijable samo je jedan Student objekt.** Objke varijable su reference na isti objekt.

3. `st1.setProgramStudija("Strojarstvo");`  
Polje `programStudija` objekta na koji referencira varijable `st1` mijenja se u string "CS".

4. `st2.prikaz();`

Prikazuju se detalji objekta na kojeg referencira `st2`.



Da se objekti spremaju u varijable (**a nisu !**) prva naredba bi spremila objekt `Student` u `st1`. Druga bi naredba napravila kopiju istog objekta u `st2`. Tada bi postojala dva objekta tipa `student`. Treća bi promijenila polje `programStudija` u objektu `st1` i ne bi promijenila objekt u `st2`. Četvrta naredba bi na kraju ispisala detalje objekta u `st2`, uključujući originalni naziv studija "Elektronika".

Primjeri poput ovoga stalno se događaju u Java programiranju. Stoga je vrijeme da se počne misliti u terminima referenci.

#### 5. null vrijednost

Ranije smo rekli da varijabli tipa `student` može biti pridružena samo referenca na objekt tipa `Student`. To nije sasvim točno. Postoji jedna vrijednost koja može biti pridružena svakoj varijabli koja inače sadržava vrijednost reference. Ta vrijednost se označava sa `null`. Slijedi deklaracija koja kreira varijablu tipa `Student` i sprema u nju vrijednost `null`:

```
Student st = null;
```

Za primjer gdje se vrijednost `null` može koristiti uzmimo pretraživanje baze gdje imamo metodu koja pretražuje cijelu bazu podataka o studentima i traži objekt s određenim ID brojem. Ako se objekt s takvim ID brojem nađe vratit će se referenca na taj objekt. Međutim što vratiti ako ne postoji objekt s tim ID brojem. U tom slučaju bit će uobičajeno da ta metoda vrati vrijednost `null`.

Ako varijabla poput prijašnjeg primjera varijable `st` sadrži vrijednost `null`, potrebno je osigurati da se ne pokuša pristupati metodama koje pripadaju objektu na kojeg se odnosi varijabla `st`. (varijabla `st` ne odnosi se u ovom slučaju na nikakav objekt)

Npr. slijedeća naredba neće biti izvršena:

```
st.setProgramStudija("Strojarstvo");
```

`st` ne referencira na nikakav kreirani objekt tipa `Student` te će Java intepreter javiti run-time pogrešku. Greška će biti indicirana kao `NullPointerException`. Naravno, ako varijabli `st` pridružimo objekt ta će se ista naredba bez problema izvršiti.

Još jedno mjesto gdje Java može uvesti vrijednost `null` je kod kreiranja objekta konstruktorom. U primjerima smo preko argumenata konstruktora prenosili argumente koji su nam poslužili kao inicijalne vrijednosti za polja objekta.

Što se događa kada ne specificirate inicijalnu vrijednost polja u objektu. Java će izabrati sama početnu vrijednost. Ako je polje neki broj onda će biti inicijaliziran na nulu. Ako polje je referenca na objekt početna vrijednost će biti `null`.

Npr. tri polja objekta `Student` su tipa `String`. **Stringovi su objekti**. Tako ta polja sadrže reference na mjesto gdje su stvarno u memoriji pohranjeni ti karakteri. Ako ne specificiramo inicijalne vrijednosti za njih one će sadržavati `null` vrijednost. To ne znači da se radi o praznom stringu ! Varijabla može referencirati na prazan string, ali to je regularan string kao i svi drugi.

Dakle **poljima** koja nisu inicijalizirana Java pridružuje pretpostavljene početne vrijednosti (default values). To je različito od onoga što se javlja s deklaracijama varijabli unutar tijela metoda. Te varijable se ne inicijaliziraju automatski ! U njima je nepredvidljiva vrijednost ako ih sami nismo inicijalizirali.

#### 6. Identifikatori

Nazivi koje koristimo u Java programima za varijable, metode ili klase nazivamo **identifikatori**. Pravila za pisanje identifikatora u Javi su sljedeća:

mora početi s slovom ili oznakom valute (\$,£,...), ili povlakom ( \_ )

ostali znakovi mogu biti slova ili brojevi

dužina nije ograničena

ne mogu se koristiti ključne riječi kao identifikatori (poput `class`, `int`, `private`, `return`, ...)

U Javi je pojam brojki i slova dosta širok. Slovo može biti korejsko, grčko, japansko. To je zbog toga što Java znakove čuva kao 16-bitne *unicode* vrijednosti. To je ogroman skup znakova koji uključuje znakove većine svjetskih jezika. Naše tastature omogućavaju unos limitiranog skupa znakova.

### 3. Objekti i klase

---

Poput čvrstih pravila za identifikatore, postoje i konvencije. Jedna je da nazivi varijabli, parametara i metoda počinju s malim slovima, a nazivi klasa počinju s velikim slovima. Te konvencije su tako širom prihvaćene da će vas ostali programeri čudno gledati ako ih se ne držite.

Druga konvencija je da ako koristite nazive sastavljene od više riječi da svaku riječ poslije prve počinjete s velikim slovom npr. `ConsoleReader`, `readDouble`. Java programeri vole duge nazive pa Java biblioteka sadrži nazive poput `GridBagLayoutManager` i `NoSuchElementException`. Korisno je koristiti duge nazive jer se onda može iz toga vidjeti funkcionalnost varijable ili metode. Međutim kod u cjelini postaje nečitljiviji.



#### 7. Zadaci za poglavlje 3

Proširite klasu Student opisanu u odjeljku 3 tako da povećate količinu informacije koja je zapisana u svakom objektu tipa Student. Dodatni stavci bi trebali biti: ime studentovog mentora i broj bodova koje je stekao na osnovu polaganja ispita. Za svaki polozeni ispit student dobiva 0.5 bodova. Proširena klasa treba osigurati i slijedeće operacije koje ćemo pozvati iz test programa:

Metoda nazvana `promijeniMentora (m)` koja će promijeniti ime mentora u ime `m`.

Metoda `dodajBod (n)` koja će dodati broj bodova na ukupan broj bodova koje student ima .

Na kraju trebat će modificirati konstruktor i ubaciti naziv mentora kao parametar. Inicijalna vrijednost broja bodova treba biti 0.

### 4. Grafika

Rane verzije Jave omogućavale su veoma limitirane grafičke sposobnosti. Java 2 je omogućila mnogo bogatiji skup grafičkih operacija. Od tog skupa koji obuhvaća crtanja počevši od najjednostavnijih linija, preko drugih geometrijskih oblika pa sve do manipulacije s fotografijama u ovome poglavlju preći ćemo maji dio.

Sadržaj:

1. Korištenje grafike.
2. Applet koji crta linije.
3. Drugi oblici.
4. Boje.
5. Bojanje likova.
6. Pisanje teksta.
7. Smajli.
8. Čitanje ulaza.
9. Daljnja čitanja.
10. Zadaci

#### 1. Korištenje grafike

Grafika je tema koju je zgodno uključiti rano unutar predavanja o programiranju jer programeri vole raditi s grafikom, a Java klase koje se odnose na grafiku daju dobar primjer mnogih osobina Jave koje ćemo i dalje susretati na ovim predavanjima. Cijena koju treba platiti je da je potrebno uvest drugačiji način pokretanja programa nego što je to za programe koji nisu koristili grafiku.

Postoje dva načina na koji se mogu pokrenuti Java programi: **aplikacije** i **appleti**. Svi programi koje smo dosada analizirali bili su aplikacije. To znači da su bili pokretani od strane Java SDK interpretera (java) i izvršavanje je započinjalo unutar `main` metode.

Sve aplikacije koje pišemo unutar ovih predavanja ispisivale su u prozor konzole (DOS prozor) koji je prikladan samo za ispisivanje karaktera. Ako bi aplikacija trebala proizvesti neku grafiku bilo bi potrebno da kreira grafički prozor. To je u Javi moguće, ali nećemo se time baviti u ovim predavanjima.

**Applet** je Java program koji se izvršava unutar Web pretraživača, poput Internet Explorera, ili Netscape pretraživača. Kada Web pretraživač stavlja sadržaj stranice na ekran i ako ta stranica sadrži link na applet, pretraživač će odvojiti dio stranice odnosno područje prikaza (display area) gdje će applet moći prikazati bilo kakvu informaciju, uključujući i grafiku.

Nakon toga će pretraživač započeti s izvršavanjem appleta. To će moći jer sadržava vlastiti Java interpreter.

Kraće rečeno: napišite applet koji može iscrtavati grafiku, napišite Web stranicu koja sadrži link na istu i zatražite od pretraživača da prikaže Web stranicu.

Postoji određeni problem s ovim pristupom. Pojedini pretraživači ne izvršavaju Java 2 programe. Bilo da slabo prate napredovanje Jave ili namjerno ne podržavaju Javu. Dva su načina da riješimo taj problem:

Rješenje 1. Nabavite *plug-in* za pretraživač koji će omogućiti pokretanja Java 2 programa.

Rješenje 2. Koristite **appletviewer**. Ovaj program je dio SDK. Pokrenut će bilo koji applet.

Appletviewer ima prvenstvenu namjenu testiranja funkcionalnosti appleta, ali predstavlja najjednostavniji način da vidimo što naš grafički program iscrtava.

Dakle, kada želimo kreirati neki grafički ispis, napisat ćemo applet koji crta grafiku u područje prikaza, zatim napisati malu Web stranicu koja samo sadržava link na applet. Zatim ćemo iskoristiti appletviewer za procesiranje Web stranice.

Jedini nedostatak ovoga pristupa je da appletviewer ne prikazuje ništa osim onoga što proizvede applet. dakle nećemo se baviti kreacijom Web stranica.

Sve što je potrebno znati o HTML za potrebe predavanja je slijedeće. Web stranica sastoji se od tekst datoteke koja koristi HTML notaciju. U toj datoteci bit će samo link na datoteku `Prog.class` u kojoj će biti prevedeni Java applet kojega će izvršiti.

## 4. Grafika

```
<APPLET CODE="Prog.class" WIDTH=300 HEIGHT=300>
</APPLET>
```

Osim linka navedena je i veličina područja prikaza koje će biti dodijeljeno appletu na korištenje:

```
WIDTH=300 HEIGHT=300
```

### 2. Applet koji crta linije

Applet je također jedna Java klasa. U mnogim pogledima je nalik na klase koje smo koristili dosada. Sastavljen je od polja, metoda i konstruktora. Međutim razlikuje se od klase koje smo dosada koristili u jednoj bitnoj stvari. Posjeduje najmanje jedna metoda koja je namijenjena da je koristi *windows manager*. Windows manager je objekt koji održava prozore na ekranu, omogućavajući njihovo otvaranje, skrivanje (minimiziranje), zatvaranje, itd..

Jedna od metoda koje ćemo priskrbiti za windows manager je metoda *paint*. Windows manager će prvi put pozvati tu metodu kad se područje prikaza pojavi na ekranu. Također kad god bude bilo potrebno osvježiti sadržaj ekrana bit će pozvana metoda *paint*. Osvježavanje prikaza bit će potrebno npr. kad pomicanjem (skroliranjem) Web stranice područje prikaza izađe izvan vidljivosti pa se novim pomicanjem opet vrati na ekran. U tom slučaju sadržaj područja prikaza mora se ponovo iscrtati odnosno osvježiti. Drugi primjer kad je to potrebno je kad preko područja prikaza npr. otvorimo prozor neke druge aplikacije. Kada zatvorimo taj prozor opet će biti potrebno ponovo osvježiti sadržaj koji je dotada bio pokriven prozorom.

Zadatak *paint* metode je da isporuči (render) sadržaj u područje prikaza (nacrtat linije, oboji područja, napiše tekst, ...). Da bi to napravili moramo koristiti jedan objekt koji nazivamo **Graphics2D** objekt. To je nešto slično kao *System.out* objekt koji smo koristili za ispis karaktera u prozoru konzole. Međutim *Graphics2D* objekt spojen je na područje prikaza appleta (visoka rezolucija), a ne primitivni DOS prozor.

*Graphics2D* objekt posjeduje niz metoda kao npr. *drawString* kojim možemo ispisivati tekst, metodu *draw* koja može iscrtavati oblike poput kruga, kvadrata, metodu *drawImage* koju možemo koristiti za ispis kompletnih slika. Također *Graphics2D* objekt posjeduje metode kojima se može zadati određena boja ili uzorak kojom će se crtati neki lik ili ispisivati tekst. Posjeduje i metode kojima se može zadati neka grafička transformacija lika npr. rotacija.

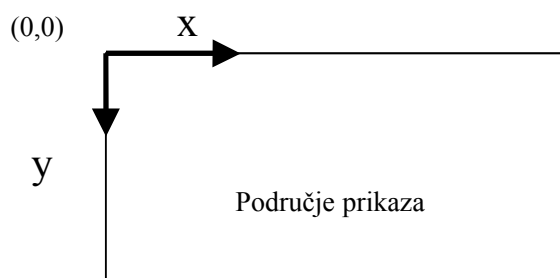
*Graphics2D* objekt ne moraju biti spojeni samo na područje prikaza na ekranu. Možemo ih spojiti npr. na printer ili na neko područje u memoriji (slika će biti pohranjena u memoriju).

Pretpostavimo da je *g2* objekt tipa *Graphics2D* spojen na područje prikaza appleta. Pretpostavimo da je *line1* linija između dvije točke. Slijedeća naredba će prikazati liniju na ekranu:

```
g2.draw(line1);
```

Postoje različiti geometrijski oblici koji mogu biti iscrtani ili obojani korištenjem *Graphics2D* objekta. Koristit ćemo 4 tipa: linije, pravokutnike (rectangles), elipse (uključujući i krugove) te dijelove elipsi tj. lukove (arcs). Svaki od ovih geometrijskih oblika u Javi je reprezentiran kao klasa. Npr. linija je reprezentirana objektom tipa *Line2D.Double*. Riječ *Double* pokazuje da se podaci o liniji pohranjeni ka 64-bitni broj s pokretnim zarezom. (Ako želite uštediti memoriju na raspolaganju je i *Line2D.Float*).

Da bismo kreirali liniju potrebno je specificirati koordinate njenih krajnjih točaka. Pozicije su dane ka *x* i *y* koordinate i njihova orijentacija je slijedeća:



Koordinate se mjere u pikselima (pixel = picture element). Npr. područje koje ima 300×300 piksela protezat će se otprilike na trećini ekrana.

da bismo kreirali objekt linije koristit ćemo `Line2D.Double` konstruktor koji ima koordinate krajnjih točaka linije kao parametre:

```
Line2D.Double(x0, y0, x1, y1)
```

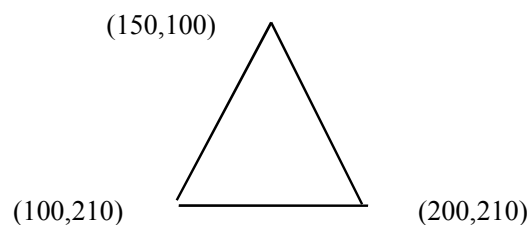
Kreira objekt koji predstavlja liniju od (x0,y0) do (x1,y1).

(Vrijednosti x0, y0, x1 i y1 su `double`. Ako napišete cjelobrojne koordinate Java će ih konvertirati u `double`.)

Primijetite da kreiranje `Line2D.Double` objekta **nije** isto što i njegovo prikazivanje na ekranu.

Taj objekt bit će prikazan na ekranu tek onda kad ga prosljedimo `draw` metodi `Graphics2D` objekta koja će ga iscrtati u području prikaza.

Npr. želimo nacrtati slijedeći trokut na ekranu:



Bit će potrebno koristiti slijedeće izraze. Prvo kreiramo objekt koji predstavlja lijevu stranu trokuta tj. objekt tipa `Line.Double` te ga dodijelimo varijabli `line1`. Zatim ga iscrtamo korištenjem `g2.draw` metode. proces ponovimo za ostale linije.

```
Line2D.Double line1 =  
    new Line2D.Double(150,100,200,210);  
g2.draw(line1);
```

```
Line2D.Double line2 =  
    new Line2D.Double(200,210,100,210);  
g2.draw(line2);
```

```
Line2D.Double line3 =  
    new Line2D.Double(100,210,150,100);  
g2.draw(line3);
```

Pretpostavimo da pišemo applet koji će samo crtati taj trokut. Napisat ćemo `paint` metodu koja koristi prije navedene naredbe. Postavlja se pitanje kako ćemo kreirati `Graphics2D` objekt tj. označen s `g2`? Odgovor je da ga nećemo mi kreirati. `Windows manager` će kreirati taj objekt i prosljediti ga `paint` metodi kao parametar.

Postoji jedan mali tehnički problem na koji u svemu moramo paziti. Appleti koriste `paint` metode još iz najranijih dana Jave, davno prije nego što je Java 2 uvela `Graphics2D` objekte.

Prije je parametar `paint` metoda imala parametar koji je bio objekt tipa `Graphics` (bez 2D!). Taj objekt je bio sličan `Graphics2D` objektu, ali daleko manje funkcionalnosti.

Problem je u tome što Java 2 treba pokretati i stare programe pisane za ranije verzije Jave. Zato je ostavljeno da Java kao parametar `paint` metode appleta očekuje `Graphics` objekt što ćemo morati napisati u zaglavlju `paint` metode:

```
public void paint(Graphics g)
```

## 4. Grafika

---

Prevodioc se neće buniti što windows će manager proslijediti Graphics2D objekt kao parametar kad bude pozivao paint metodu ! To je zato što je Graphics2D objekt specijalni slučaj Graphics objekta.

Međutim prevodioc će se pobuniti ako počnemo objekt označen s g tretirati kao Graphics2D objekt. Morat ćemo napisati slijedeći izraz:

```
Graphics2D g2 = (Graphics2D) g;
```

Ovaj izraz kaže: kreiraj Graphics2D varijablu, g2, i dodijeli joj objekt g za kojeg obećavamo da je Graphics2D objekt. Kastiranje (Graphics2D) je naše obećanje prevodiocu da je parametar stvarno Graphics2D objekt. U ostatku metoda g2 tretiramo kao Graphics2D objekt.

Slijedi kompletan program koji će iscrtati trokut. Potrebna je samo jedna klasa. Zaglavlje klase sadrži riječi extends applet. Tim pokazujemo prevodiocu da ta klasa definira applet. Prve tri linije pokazuju Java prevodiocu koje tri linije Java biblioteke će biti potrebne.

### PRIMJER 1

```
import java.applet.*;
import java.awt.*;
import java.awt.geom.*;

/* Applet koji prikazuje trokut . */

public class Triangle extends Applet

{   public void paint(Graphics g)
    {   Graphics2D g2 = (Graphics2D) g;

        Line2D.Double line1 =
            new Line2D.Double(150,100,200,210);
        g2.draw(line1);

        Line2D.Double line2 =
            new Line2D.Double(200,210,100,210);
        g2.draw(line2);

        Line2D.Double line3 =
            new Line2D.Double(100,210,150,100);
        g2.draw(line3);
    }
}
```

Primijetite da bi se neke naredbe mogle sažetije pisati ako bismo izbjegli uvođenje pomoćnih varijabli line1, line2 , line3.

Umjesto pisanja:

```
Line2D.Double line1 =
    new Line2D.Double(150,100,200,210);
g2.draw(line1);
```

Mogli smo pisati

```
g2.draw(new Line2D.Double(150,100,200,210));
```

Obje verzije su ispravne.

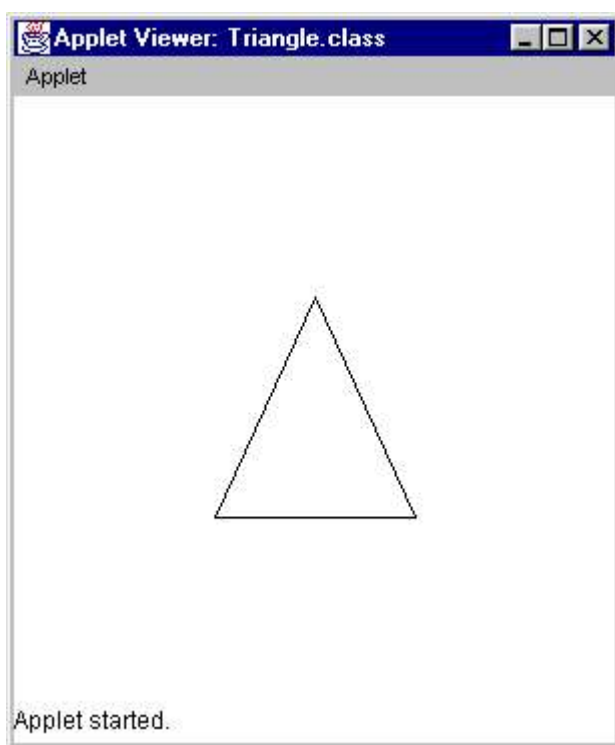
Da bismo pokrenuli ovaj program može biti korištena slijedeća tekstualna datoteka. Ona sadrži ključnu vezu s prevedenom Triangle klasom.

```
<APPLET CODE="Triangle.class"  
        WIDTH=300  
        HEIGHT=300>  
</APPLET>
```

(Smještaj linija nije bitan. Možete otkucati sve u jednoj liniji ako tako želite). Pretpostavimo da je ta datoteka nazvana `Triangle.txt`. Nakon što smo preveli `Triangle.java` utipkajmo:

```
appletviewer Triangle.txt
```

i slijedeći prozor će se pojaviti na ekranu.



Upamtite. Trokut se iscrtao zato što windows manager poziva `paint` metodu. Pretpostavim da pokušate pokvariti sadržaj prozora tako da npr. pokrenete neki drugi program koji će prekriti applet prozor. Čim učinite nešto što će vratiti vidljivost appleta windows manager će odmah ponovo pozvati `paint` metodu i ponovo iscrtati sadržaj prozora.

Java biblioteka je podijeljena u veći broj dijelova koji se nazivaju **paketi** ili **packages**. Prva linija primjera:

```
import java.applet.*;
```

kaže prevodiocu da će se koristiti paket koji se naziva `java.applet`. Oznaka `*` pokazuje prevodiocu da očekuje upotrebu *bilo koje* klase iz navedenog paketa. U primjeru se koristi samo jedna klasa iz `java.applet` paketa. Da nismo *importirali* `java.applet` klasu bili bismo prisiljeni koristiti puno ime `java.applet.Applet` umjesto samo `Applet`. To je jedini razlog korištenja `import` naredbe.

## 4. Grafika

U `java.awt` paketu sadržane su mnoge grafičke klase. U prethodnom primjeru koristili smo `Graphics` i `Graphics2D` klase. Npr. paket `java.awt.geom` omogućava upotrebu geometrijskih oblika poput `Line2D.Double` objekta.

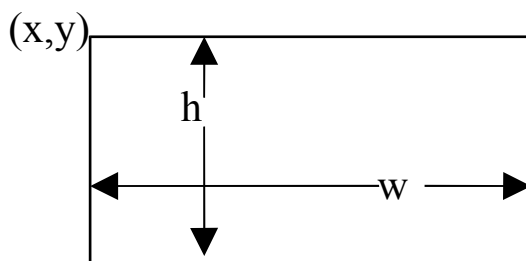
Nije potrebno importirati sve pakete Java biblioteke. Paket `java.lang` uključen je automatski. On uključuje veliki broj često korištenih klasa, poput klase `String`, `Math` i `System`.)

### 3. Drugi oblici

nova grafička biblioteka u javi 2 uključuje različite korisne grafičke oblike. Jedan od njih je i pravokutnik, sadržan u klasi `Rectangle2D.Double`. Pravokutnik možemo konstruirati pomoću slijedećeg konstruktora :

```
Rectangle2D.Double(x, y, w, h)
```

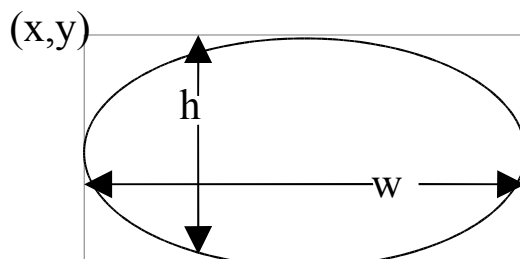
Gornji lijevi kut pravokutnika ima koordinate  $(x,y)$ . Njegova širina je  $w$ , a visina je  $h$ .



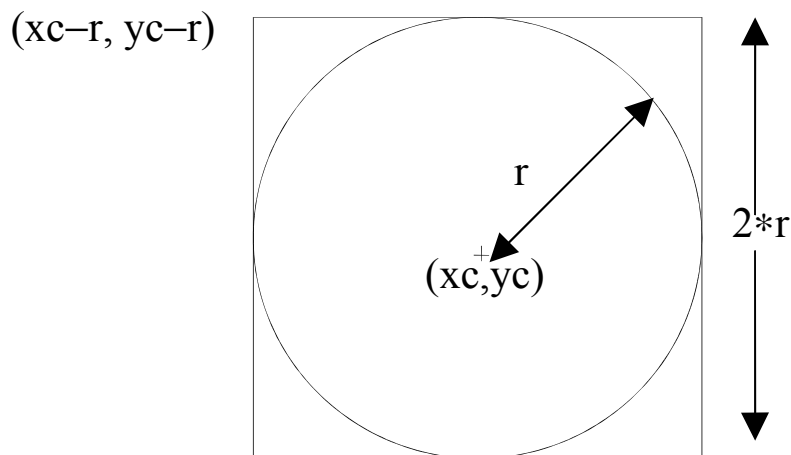
Slično vrijedi i za elipsu koja je definirana klasom `Ellipse2D.Double`. Može biti kreirana slijedećim konstruktorom

```
Ellipse2D.Double(x, y, w, h)
```

Pozicija i oblik elipse definiran je parametrima  $x$ ,  $y$ ,  $w$  i  $h$ . Ovi parametri određuju pravokutnik koji sadržava elipsu i zadaju se isto kao i za pravokutnik.



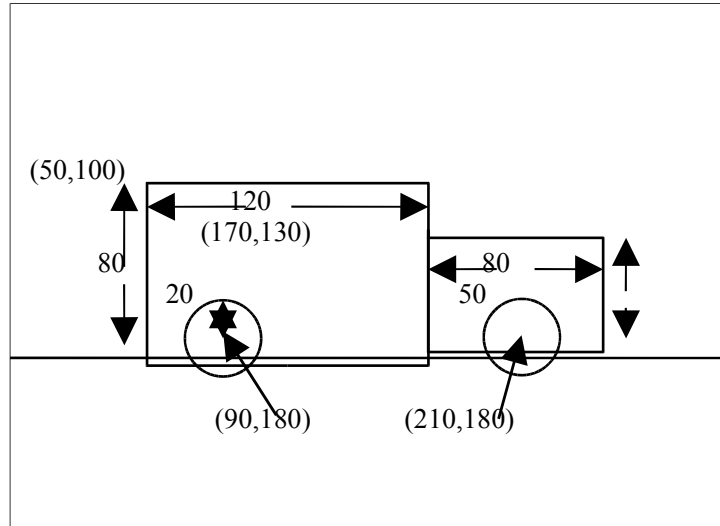
Ne postoji objekt koji bi bio rezerviran za definiciju kruga, međutim krug možemo nacrtati kao poseban slučaj elipse čija je širina jednaka visini. Ako je centar kruga u  $(x_c, y_c)$ , a njegov radijus je  $r$ , tada je to ekvivalentno elipsi koju definiramo preko pravokutnika sa gornjim lijevim kutom u  $(x_c - r, y_c - r)$  te širinom  $2*r$  i visinom  $2*r$ .





U sljedećem primjeri koristit ćemo linije, pravokutnike i elipse te nacrtati jedan mali kamion. Prije nego počnemo pisati kod potrebno je pažljivo definirati položaj pojedinih likova. Slijedi skica kamiona u području prikaza od 300×300.

(0,0)



Slijedi applet koji će nacrtati ovu sliku. Za njega je kamion predstavljen kao dva pravokutnika plus dva kruga. Tu je još i linija ceste.

#### PRIMJER 2

```
import java.applet.*;
import java.awt.*;
import java.awt.geom.*;

public class DrawVan extends Applet
{   public void paint(Graphics g)
    {   Graphics2D g2 = (Graphics2D) g;

        Rectangle2D.Double back =
            new Rectangle2D.Double(50,100,120,80);
        g2.draw(back);

        Rectangle2D.Double back =
            Rectangle2D.Double(170,130,80,50);
        g2.draw(back);

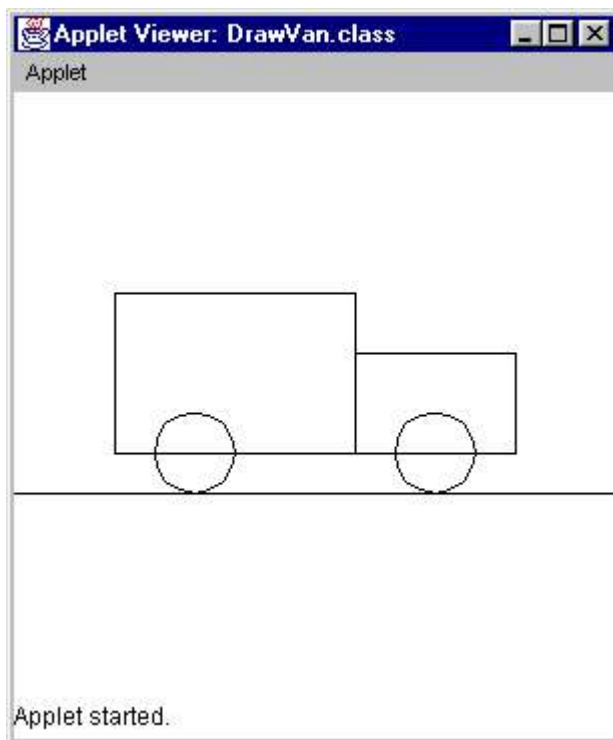
        Ellipse2D.Double backWheel =
            new Ellipse2D.Double(70,160,40,40);
        g2.draw(backWheel);

        Ellipse2D.Double frontWheel =
            new Ellipse2D.Double(190,160,40,40));
        g2.draw(frontWheel);

        Line2D.Double baseLine =
            new Line2D.Double(0,200,300,200);
        g2.draw(baseLine);
    }
}
```

Slijedi prikaz kako bi se trebao vidjeti na ekranu:





#### 4. Boja

Ako drugačije ne navedemo Graphics2D objekt će crtati crnom bojom. Međutim moguće je da promijenite boju kad god to zaželite. Klasa Graphics2D posjeduje `setColor` metodu koja će promijeniti boju u onu koju joj prosljedite kao parametar.

Boje su u Javi predstavljene kao objekti tipa **Color**. Klasa Color sadrži trinaest konstanti koje predstavljaju standardne boje. (konstanta je varijabla s fiksnom vrijednošću). Npr. među njima je konstanta `Color.red` koja predstavlja crvenu boju. Kompletna lista boja navedena je kasnije.

Stoga, ako želite da objekt `g2` (klase Graphics2D) počne pisati crvenom bojom, onda upotrijebite slijedeći izraz prije korištenja `draw` metode:

```
g2.setColor(Color.red);
```

Od momenta kad je izvršena pa sve dok je ponovo ne izvršimo s drugom bojom kao argumentom, sve što `g2` bude prikazivao bit će obojeno crveno.

Objekt Color može predstavljati mnogo više boja nego standardnih 13. Možete definirati svoje boje specificiranjem količine tri osnovne boje : crvene, zelene i plave (red-green-blue = RGB).

Koristi se *aditivni* sustav boja. njegovo osnovno svojstvo je da ako kombiniramo maksimalnu količinu crvene, zelene i plave boje dobivamo bijelu boju. Kombiniranjem nultih vrijednosti crvene, zelene i plave boje dobivamo crnu boju (Ovo je suprotno korištenju boja u tiskanju ili slikarstvu. tu se koriste *substraktivni* sustavi. Npr. miješanjem različitih uljanih boja dobivate sve tamnije boje slične crnoj.

Klasa Color posjeduje konstruktor za kreiranje RGB boje

```
Color(r, g, b)
```

Kreiraj boju kombinirajući `r` količinu crvene, `g` količinu zelene i `b` količinu plave. Ove vrijednosti su brojevi u pokretnom zarezu s vrijednostima od 0 do 1 tipa `float`.

Moramo pažljivo pisati te parametre. Ako npr. napišete 0.5 prevodioc će to shvatiti kao `double` vrijednost i javiti pogrešku. Potrebno je specificirati da se radi o tipu `float`, tj. napisati `0.5f`. Slijedi primjer definiranja žute boje:

## 4. Grafika

---

```
Color c = new Color(1.0f, 1.0f, 0.0f);
```

(Ne pišite `Color(1, 1, 0)`. Normalno možete koristiti cijele brojeve na mjestu gdje se očekuje float vrijednost, ali ne i ovdje ! Razlog je što `Color` klasa ima još jedan konstruktor s drukčijim parametrima koji su definirani kao cjelobrojni iznosi crvene, plave i zelene kao *cijeli brojevi u rasponu od 0 do 255* ! Ako dakle napišete `Color(1, 1, 0)`, Java će pretpostaviti da koristite sasvim drugi konstruktor gdje ste specificirali neznatan iznos crvene i zelene i ništa plave te će rezultat biti skoro crna boja.

Slijedi lista 13 standardnih boja i količine crvene, zelene i plave od kojih se sastoje:

<code>Color.black</code>	<code>0.0f</code>	<code>0.0f</code>	<code>0.0f</code>
<code>Color.blue</code>	<code>0.0f</code>	<code>0.0f</code>	<code>1.0f</code>
<code>Color.cyan</code>	<code>0.0f</code>	<code>1.0f</code>	<code>1.0f</code>
<code>Color.gray</code>	<code>0.5f</code>	<code>0.5f</code>	<code>0.5f</code>
<code>Color.darkGray</code>	<code>0.25f</code>	<code>0.25f</code>	<code>0.25f</code>
<code>Color.lightGray</code>	<code>0.75f</code>	<code>0.75f</code>	<code>0.75f</code>
<code>Color.green</code>	<code>0.0f</code>	<code>1.0f</code>	<code>0.0f</code>
<code>Color.magenta</code>	<code>1.0f</code>	<code>0.0f</code>	<code>1.0f</code>
<code>Color.orange</code>	<code>1.0f</code>	<code>0.8f</code>	<code>0.0f</code>
<code>Color.pink</code>	<code>1.0f</code>	<code>0.7f</code>	<code>0.7f</code>
<code>Color.yellow</code>	<code>1.0f</code>	<code>1.0f</code>	<code>0.0f</code>
<code>Color.white</code>	<code>1.0f</code>	<code>1.0f</code>	<code>1.0f</code>

## 5. Bojanje likova

Objekt `Graphics2D` može se koristiti za bojanje unutrašnjosti likova poput pravokutnika ili elipse. Bit će upotrijebljena boja koja je zadnja postavljena s metodom `setColor`. Ako nijedna boja nije postavljena bit će upotrijebljena crna boja.

Metoda koju ćemo koristiti naziva se `fill`. Pozivamo je na slijedeći način:

```
g2.fill(s)
```

Koristi `Graphics2D` objekt referenciran s `g2` za prikaz oblika `s` ispunjenog s trenutnom bojom pridruženoj `g2`.

Npr. slijedeće naredbe kreirat će mali krug i prikazat ga ispunjenog žutom bojom.

```
Ellipse2D.Double sun =  
    new Ellipse2D.Double(140, 50, 20, 20);  
g2.setColor(Color.yellow);  
g2.fill(sun);
```

Slijedeći primjer koristi istu tehniku za crtanje kamiona sličnog kao u primjeru 1, ali obojenog u crvenu boju. Točkovi su bijeli, a gume su tamno sive boje. Linija ceste zamijenjena blijedo-plavom bojom. Primijetite da je potrebno iscrtati pozadinu prije kamiona.

### PRIMJER 3

```
import java.applet.*;  
import java.awt.*;  
import java.awt.geom.*;  
  
public class PaintVan extends Applet  
{  
    public void paint(Graphics g)
```

```
{ Graphics2D g2 = (Graphics2D) g;

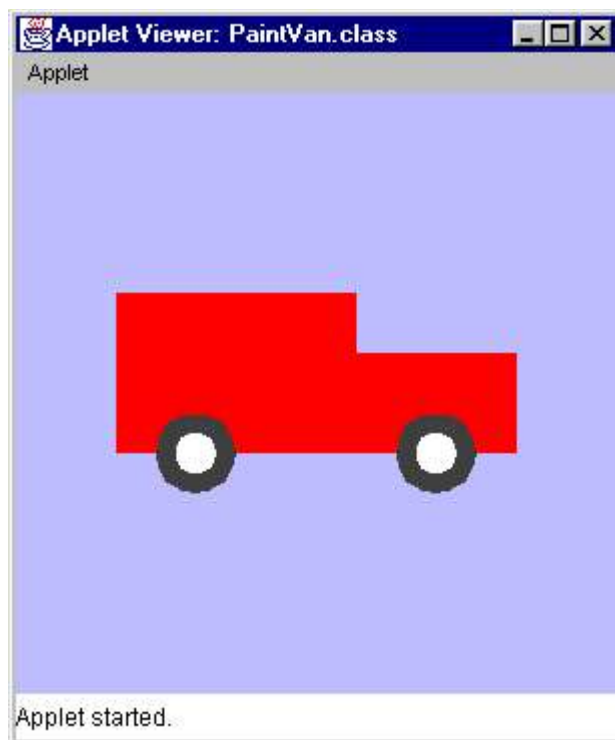
    /* Paint the background. */
    Color paleBlue =
        new Color(0.75f, 0.750f, 1.0f);
    g2.setColor(paleBlue);
    g2.fill(new Rectangle2D.Double(0,0,300,300));

    /* Paint the body of the van. */
    g2.setColor(Color.red);
    g2.fill
        (new Rectangle2D.Double(50,100,120,80));
    g2.fill
        (new Rectangle2D.Double(170,130,80,50));

    /* Paint the back wheel. */
    g2.setColor(Color.darkGray);
    g2.fill(new Ellipse2D.Double(70,160,40,40));
    g2.setColor(Color.white);
    g2.fill(new Ellipse2D.Double(80,170,20,20));

    /* Paint the front wheel. */
    g2.setColor(Color.darkGray);
    g2.fill(new Ellipse2D.Double(190,160,40,40));
    g2.setColor(Color.white);
    g2.fill(new Ellipse2D.Double(200,170,20,20));
}
```

Slijedi obojani kamion:



### 6. Pisanje teksta

Osim za crtanje i bojanje oblika, Graphics2D objekt može se koristiti za ispis teksta na ekranu. Prije nego što to pokušamo učiniti, potrebno je Graphics2D objektu naznačiti koji će *font* koristiti. Font je reprezentiran s objektom tipa `Font`. Koristimo slijedeći konstruktor za kreiranje Font objekta.

```
Font(familija, stil, veličina) (family, style, size)
```

Parametar `familija` je string koji predstavlja naziv skupa srodnih fontova. Primjeri srodnih familija su Times Roman koji se koristi kao glavni font ovog teksta, i Courier koji se koristi za tekst programa. Ova rečenica je tiskana u Arial fontu. Slijedeća logički nazivi mogu se koristiti za familije fontova:

```
Serif  
SansSerif  
Monospaced  
Dialog  
DialogInput
```

Parametar `stil` ima jednu od slijedećih vrijednosti koje naznačuju da li se koristi *italic (zakošeno)* ili **bold (podebljano)** pisanje:

```
Font.PLAIN  
Font.ITALIC  
Font.BOLD  
Font.ITALIC+Font.BOLD.
```

Parametar `veličina` je *veličina fonta u točkama*. To je visina karaktera mjerena u jedinci od 1/72 inča. Veličina točke od 1 približno odgovara jednom pikselu. Ova predavanja napisana su fontom veličine 12 i karakteri bi trebali biti visoki oko 1/6 inča.

Jednom kada kreirate Font objekt, potrebno je Graphics2D objektu narediti da ga koristi. To činimo upotrebom slijedeće metode:

```
g2.setFont(f)  
Postavi Graphics2D objekt referenciran s g2 tako da koristi f za pisanje teksta.
```

Sve što preostaje je da se ispiše string na određeno mjesto na ekranu:

```
g2.drawString(text, x, y)
```

Koristi Graphics2D objekt `g2` za ispis stringa `text`, počevši od pozicije (x,y) u području prikaza. `x` i `y` su of tipa `int`. (`x` je pozicija lijeve strane teksta, a `y` je visina na kojoj se nalazi bazna linija teksta.

Slijedeći primjer ilustrira upotrebu logičkih familija fontova. U svakom koraku Graphics2D objektu pridružujemo jedan logički font te pomoću tog fonta ispisujemo naziv familije. U svakom od slučajeva koristimo stil `FONT.PLAIN` te je veličina fonta jednaka 30. 0.

#### PRIMJER 4

```
import java.applet.*;  
import java.awt.*;  
  
public class Fonts extends Applet  
{ final int SIZE = 30;  
  
    public void paint(Graphics g)  
    { Graphics2D g2 = (Graphics2D) g;
```

```
Font font1 =
    new Font("Serif", Font.PLAIN, SIZE);
g2.setFont(font1);
g2.drawString("Serif", 50, 50);

Font font2 =
    new Font("SansSerif", Font.PLAIN, SIZE);
g2.setFont(font2);
g2.drawString("SansSerif", 50, 100);

Font font3 =
    new Font("Monospaced", Font.PLAIN, SIZE);
g2.setFont(font3);
g2.drawString("Monospaced", 50, 150);

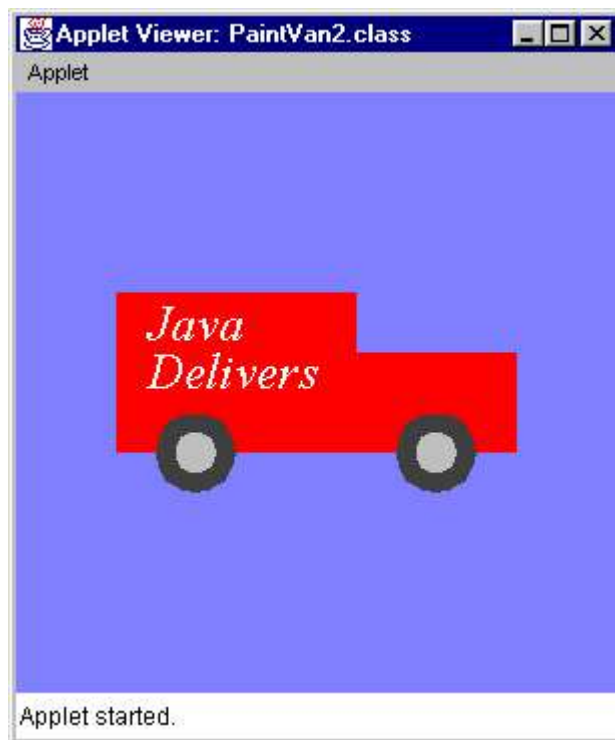
Font font4 =
    new Font("Dialog", Font.PLAIN, SIZE);
g2.setFont(font4);
g2.drawString("Dialog", 50, 200);

Font font5 =
    new Font("DialogInput", Font.PLAIN, SIZE);
g2.setFont(font5);
g2.drawString("DialogInput", 50, 250);
}
}
```



Za sljedeći primjer korištenja ispisa teksta u primjeru 3 (koji crta obojani kamion) dodajte sljedećih pet naredbi na kraj paint metode.

```
/* nacrtaj logo na stranici kamiona */
Font f = new Font("Serif", Font.ITALIC, 25);
g2.setFont(f);
g2.setColor(Color.white);
g2.drawString("Java", 66, 124);
g2.drawString("Delivers", 66, 148);
```



### 7. Smajli

Grafički objekti koje smo koristili u ovom poglavlju, poput `Line2D.Double` objekta na neki način posjeduju pridruženi prikaz. Ovu ideju možemo dalje širiti. Ako bismo trebali napraviti grafički program za očekivat bi bilo da imamo niz Java objekata za koje bi trebalo realizirati prikaz na ekranu. U slijedećem primjeru ćemo napraviti jedan takav objekt.

Definirat ćemo tip objekta koji predstavlja osobu. Nazvat ćemo je `Person` (osoba) klasa. Postoje mnogi atributi koje bismo mogli definirati za objekt tipa `Person`. U ovom slučaju upotrijebit ćemo samo jedan atribut koji će definirati raspoloženje osobe.

To je vrijednost između 0 i 1 i nazvat ćemo je `raspoloženje`. Osoba s vrijednošću `raspoloženje` ode 1 je veoma raspoložena osoba. 0 znači da je u pitanju potpuno neraspložena osoba.

Jedna od operacija koje ćemo asociirati s objektom `Person` je crtanje lika osobe. To će raditi slijedeća metoda:

```
p.crtajLice(double x, double y, Graphics2D g2)
```

Nacrtaj sliku lica osobe reprezentirane s `Person` objektom referenciranog s varijablom `p`. Centar lica su koordinate `x` i `y`. Slika će se iscrtati pomoću `Graphics2D` objekta `g2`.

Lice će biti jednostavno sa osmjehom proporcionalnim raspoloženju.

Imat ćemo dva metoda koja će uticati na raspoloženje:

```
p.oraspoloži() // Povećaj raspoloženje osobe za konstantan faktor.
```

```
p.oneraspoloži() // Smanji raspoloženje osobe za konstantan faktor.
```

Konstruktor za kreiranje `Person` objekta:

```
Person(double h) // Kreiraj Person objekt s 'raspoloženje' postavljenim na h.
```

Prije same implementacije `Person` klase, slijedi `paint` metoda koji kreira `Person` objekt i mijenja mu raspoloženje nekoliko puta. Ova metoda crta lice osobe na tri različite lokacije u području prikaza appleta.

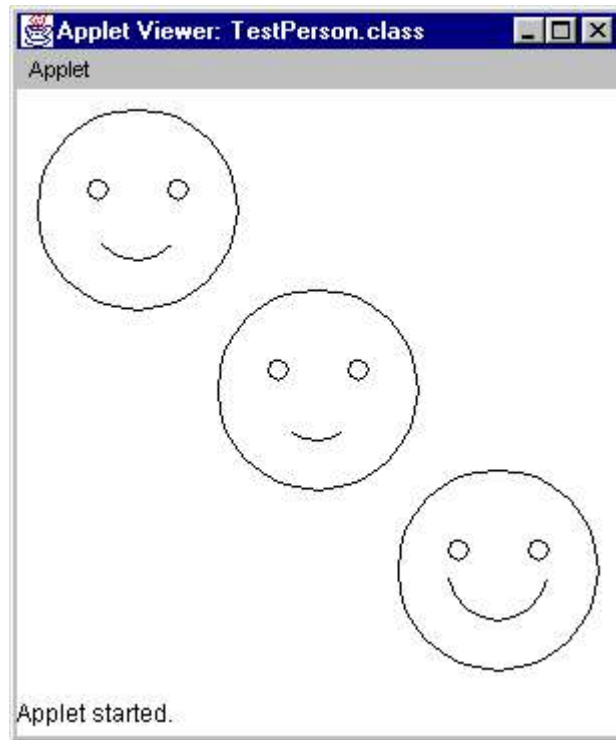
```
PRIMJER 5 public void paint(Graphics g)
```

```

{ Graphics2D g2 = (Graphics2D) g;

  Person kim = new Person(0.5);
  kim.crtajLice(60,60,g2);
  kim.oneraspoloži();
  kim.crtajLice(150,150,g2);
  kim.oraspoloži();
  kim.oraspoloži();
  kim.oraspoloži();
  kim.oraspoloži();
  kim.crtajLice(240,240,g2);
}

```



Sada slijedi implementacija. Kako pretpostavljate, klasa `Person` sadržava jedno polje nazvano `raspoloženje`. Polje je tipa `double`, i sadrži vrijednost u opsegu od 0 do 1. Konstruktor postavlja to polje na vrijednost parametra.

Metoda `oneraspoloži` množi vrijednost `raspoloženja` s konstantnim faktorom. Taj faktor se drži u varijabli nazvanoj `factor`. Tijelo metode sadrži samo jednu naredbu:

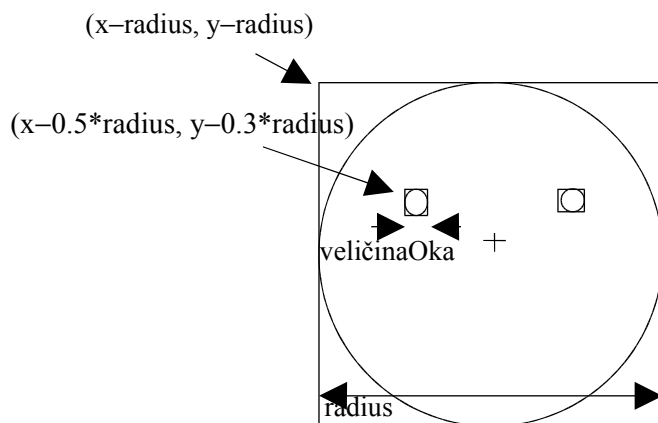
```
raspoloženje = factor*raspoloženje;
```

`factor` je postavljen unaprijed na 0.65. Bilo je jednostavnije pisati `raspoloženje = 0.65*raspoloženje`. Umjesto toga smo ipak smo definirali konstantu i dali joj naziv 'factor'. Poslije ćemo lakše promijeniti njenu vrijednost na jednom mjestu nego na svim mjestima gdje je budemo koristili.

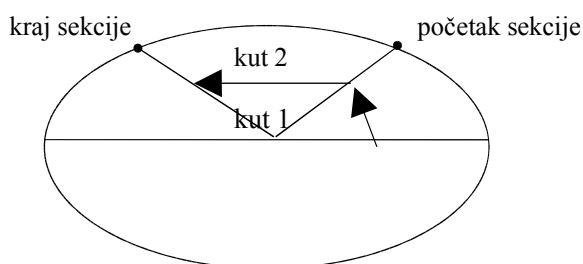
Metoda `oraspoloži` izvodi slijedeći izraz:

```
raspoloženje = 1 - factor*(1 - raspoloženje);
```

Jedini 'teži' dio implementacije klase `Person` je implementacija metoda `crtajLice`. Svako lice sastojat će se od konture, lijevog oka, desnog oka i osmjeha. Prva tri su `Ellipse2D.Double` objekti. Parametri korišteni u konstrukciji lica su izračunati koristeći vrijednosti `x` i `y` koji predstavljaju centar lica., `radius` koji je radijus lica, `veličinaOka` koji je dijametar svakog oka.



Konstrukcija osmjeha je kompliciranija jer moramo koristiti novi objekt iz java biblioteke tj. `Arc2D.Double` objekt. Ovaj objekt predstavlja dio elipse – sekciju. Potrebno je specificirati kut koji predstavlja početak sekcije te kut koji predstavlja kut sekcije.



Postoji više konstruktora koji su mogući za `Arc2D.Double` objekt. Slijedi jedan od njih:

```
Arc2D.Double(double x, double y,
             double w, double h,
             double start, double extent,
             int type)
```

Kreiraj sekciju elipse s gornjim lijevim kutom na  $(x,y)$ , širine  $w$  i visine  $h$ . Sekcija je definiran početnim kutom `start` i drugim kutom `extent`. Vrsta sekcije definirana je parametrom `type`.

Za kreirati samo dio sekcije koji se sastoji od krivulje (bez spojenih krajeva) parametar `type` treba postaviti na konstantu `Arc2D.Double.OPEN`.

Slijedi naredba za konstrukciju osmjeha:

```
Shape osmjeh =
    new Arc2D.Double
        (x-0.5*radius, y-0.5*radius, radius, radius,
         180+90*(1-raspoloženje), 180*raspoloženje,
         Arc2D.Double.OPEN);
```

Ako pažljivije pogledate vidjet ćete da je osmjeh dio kruga s centrom u centru lica s duplo manjim radijusom od radijusa lica. Ako je `raspoloženje` jednako 1, sekcija je polukrug. Ako je `raspoloženje` 0 onda sekcija u potpunosti nestaje.

Varijabla `osmjeh` je deklarirana kao klasa tipa `Shape`, ne kao `Arc2D.Double`. U stvari svaki geometrijski oblik koji se može nacrtati pomoću objekta `Graphics` može biti tipa `Shape`. 'Shape' je primjer Java sučelja (**interface**). Interface `Shape` definira koje metode neki objekt treba posjedovati da bi se mogao referirati varijablom tog tipa.

Slijedi kompletna definicija klase `Person`



##### PRIMJER 5b

```
import java.awt.*;
import java.awt.geom.*;

/* Person objekt predstavlja osobu
   s različitim nivoom raspoloženja raspoloženje.
*/

public class Person
{
    private double raspoloženje;

    private double factor = 0.65;
        // Faktor korišten za povećanje ili smanjenje
        // raspoloženja osobe

    /* Oraspoloži osobu
    */
    public void oraspoloži()
    { raspoloženje = 1 - factor*(1 - raspoloženje);
    }

    /* Oneraspoloži osobu
    */

    public void oneraspoloži()
    { raspoloženje = factor*raspoloženje;
    }

    /* Nacrtaj lice osobe korištenjem Graphics2D objekta g2.
       Centar lica je (x,y), radius 50.
    */
    public void
        crtajLice(double x, double y, Graphics2D g2)
    { double radius = 50;
      double veličinaOka = 10;

      Shape outline =
          new Ellipse2D.Double
              (x-radius, y-radius, 2*radius, 2*radius);
      Shape leftEye =
          new Ellipse2D.Double
              (x-0.5*radius, y-0.3*radius,
              veličinaOka, veličinaOka);
      Shape rightEye =
          new Ellipse2D.Double
              (x+0.5*radius-veličinaOka, y-0.3*radius,
              veličinaOka, veličinaOka);
      Shape osmjeh =
          new Arc2D.Double
              (x-0.5*radius, y-0.5*radius,
              radius, radius,
              180+90*(1-raspoloženje), 180*raspoloženje,
```

```
Arc2D.Double.OPEN);

    g2.draw(outline);
    g2.draw(leftEye);
    g2.draw(rightEye);
    g2.draw(osmjeh);
}

/* Kreiraj osobu tj. objekt tipa Person s raspoloženjem
   postavljanim na h      */
public Person(double h)
{   raspoloženje = h;
}
}
```

## 8. Čitanje ulaza

U appletima moramo unošenju podataka s tipkovnice pristupiti na različit način nego što smo to radili u dosadašnjim aplikacijama.

Koristit ćemo se sekundarnim prozorima koje ćemo nazvati *dialog*.

Kreirat ćemo ih korištenjem **JOptionPane** objekta. Ova klasa realizira dialog koji sadrži poruku od korisnika te prostor za upis podataka.

Slijedeća rečenica će uzrokovati pojavu dijaloga s porukom ‘Please enter your name.’ Dijalog će čekati da korisnik upiše niz znakova i pritisne OK (ili pritisne Enter). Na kraju će uneseni string biti pridružen varijabli `reply`.

```
String reply =
    JOptionPane.showInputDialog
        ("Please enter your name.");
```

To će izgledati ovako:



Istu tehniku možete koristiti za unos brojeva. Prvo učitajte korisnikov upis kao string, `st`, i onda ga konvertirajte korištenjem slijedećih metoda:

```
Integer.parseInt(st) ili Double.parseDouble(st).
```

U primjeru 6 program opet crta smajlija. Ovaj put program preko dijaloga pita korisnika da unese broj koji odgovara raspoloženju osobe. Na osnovu unesenog broja konstruira se `Person` objekt s unesenim raspoloženjem.

Gdje ubaciti izraz za prikaz dijaloga za unos početnog raspoloženja. Nije dobro mjesto `Paint` metoda jer se ona izvršava svaki put kada windows manager osvježava prikaz na ekranu.

Potrebno je da se dialog pojavi samo jednom prilikom pokretanja appleta. Pravo mjesto je metoda

```
public void init()
```

## 4. Grafika

---

Osim `paint` metode windows manager poziva i metodu `init`, ali samo jednom na početku izvršavanja appleta. Stoga se u tu metodu mogu smjestiti naredbe koje su vezane za inicijalizaciju parametara appleta.

Primijetite da je objekt tipa `Person` dodijeljen polju `fred` kojemu se kasnije pristupa u metodi `paint`. Nismo mogli deklarirati varijablu tipa `Person` unutar metode `Init`!

### PRIMJER 6

```
import java.applet.*;
import java.awt.*;
import javax.swing.*;
    // (Potrebno za JOptionPane klasu.)

/* Čitaj 'raspoloženje' vrijednost i
   zatim nacrtaj lice.
*/

public class DrawSmiley extends Applet

{   Person fred;

    public void init()
    {   String input =
        JOptionPane.showInputDialog
            ("Koliko je osoba sretna?");
        double h = Double.parseDouble(input);
        fred = new Person(h);
    }

    public void paint(Graphics g)
    {   Graphics2D g2 = (Graphics2D) g;
        fred.crtajLice(150, 150, g2);
    }
}
```

Ovaj applet je napravljen za dimenzije područja prikaza od 300×300.

Možete applet učiniti pametnijim tako da prvo provjeri dimenzije trenutne dimenzije područja prikaza. Metoda `getWidth()` vratit će trenutnu širinu područja prikaza, dok će metoda `getHeight()` vratiti trenutnu visinu.

Dakle možete pisati slijedeće:

```
double x = getWidth()/2.0;
double y = getHeight()/2.0;
fred.crtajLice(x, y, g2);
```

Probajte rastezati granice appleta i vidjet ćete da će ispis uvijek biti centriran.

Moguće je u `init` metodi postaviti trajnu boju pozadine s:

```
setBackground(Color.blue);
```

#### 4. Grafika

---

Ako želite vidjeti što applet čini u određenom trenutku možete ubaciti pozive metode `System.out.println`. To će ispisati poruku u DOS prozoru na uobičajen način.

Stavite npr.:

```
System.out.println("paint called");
```

na početak `paint` metode i onda ćete u DOS prozoru vidjeti koliko često windows manager poziva `paint` metodu.

### 10. Zadaci

Unesite i testirajte primjer 5.

Napišite grafički program koji će ispisati vaše ime u centru ekrana unutar plavog pravokutnika . Applet nazovite `ImeApplet.java`, a Html datoteku `ImeApplet.txt`

Napišite grafički program koji će zatražiti unos radijusa i zatim nacrtati krug s tim radijusom. Applet nazovite `KrugApplet.java`, a Html datoteku `KrugApplet.txt`

## 5. ISPITIVANJE UVJETA

Peto poglavlje sastoji se od brojnih primjera korištenja `if` uvjetnog izraza kojima biramo između dvije alternativne akcije.

### SADRŽAJ

1. Izrazi za donošenje odluka.
2. `if` naredba s jednom granom.
3. Izbor između različitih alternativa.
4. Tip podataka `boolean`.
5. Usporedba stringova.
6. Blokovi i doseg varijabli
7. Zadaci

### 1. Izrazi za donošenje odluka

U dosadašnjim programima svaki put se prilikom programa izvršavao isti niz izraza u redoslijedu kako su bili napisani. Java poput svih programskih jezika omogućava pisanje programa koji samostalno odlučuju koji će se naredbe izvršiti. Ta vrsta odluke se u Javi najčešće donosi pomoću **if uvjetne naredbe**. Slijedi primjer `if` uvjetne naredbe:

```
if (odgovor == 6048)
    System.out.println("Točno!");
else
    System.out.println("Netočno!");
```

Ova naredba znači:

Ako je vrijednost varijable `odgovor` jednaka 6048, prikaži poruku "Točno!".  
inače prikaži poruku "Netočno."

Primijetite da je izraz

```
odgovor == 6048
```

upotrijebljen za ispitivanje uvjeta 'odgovor je jednak 6048'. Simbol `==` koristi se za ispitivanje jednakosti pošto se `=` koristi za dodjeljivanja.

Slijedi program koji koristi `if` naredbu. Program pita koliki je rezultat računske operacije 72 puta 84? Zatim čita korisnikov odgovor i sprema ga u varijablu `odgovor`. Nakon toga slijedi `if` naredba koja provjerava da li korisnik pravilno odgovorio na postavljano pitanje. Ovaj program je poput većine primjera u ovom poglavlju aplikacija, a ne applet.

### PRIMJER 1

```
public class Multi1
{
    /* Traži od korisnika da unese odgovor na pitanje
       koliko je 72 puta 84, i onda provjeri odgovor. */

    public static void main(String[] args)
    {
        ConsoleReader user =
            new ConsoleReader(System.in);
    }
}
```

## 5. Ispitivanje uvjeta

---

```
System.out.println("Koliko je 72 puta 84?");
int odgovor = user.readInt();
if (odgovor == 6048)
    System.out.println("Točno!");
else
    System.out.println("Netočno!");
}
}
```

Opći oblik `if` naredbe dan je uokviren dolje. Naziv **Logički izraz** zamjenjuje bilo koji izraz čiji je rezultat “točno” ili “netočno” (true or false).

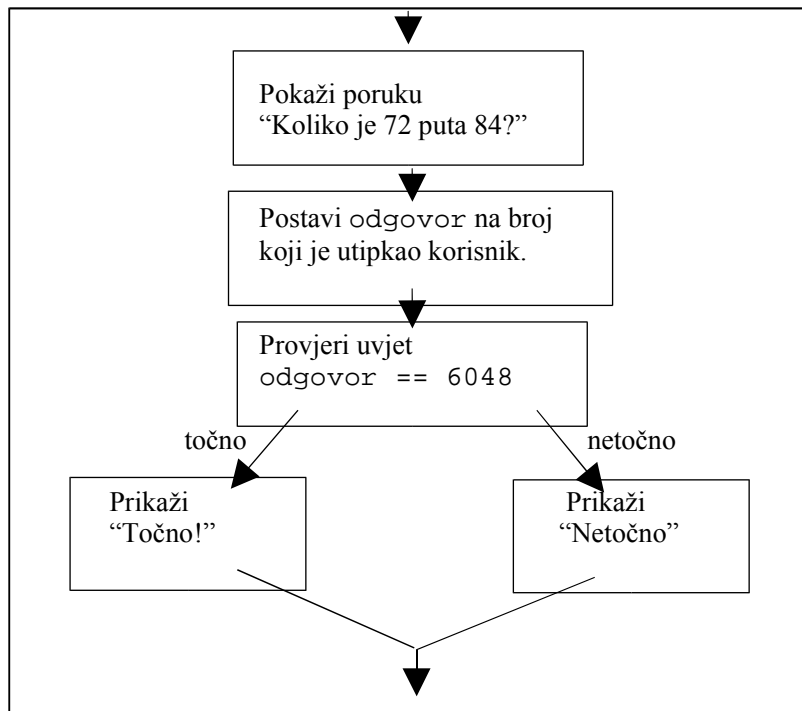
if naredba (s dvije grane)

```
if (LOGIČKI-IZRAZ)
    IZRAZ-1
else
    IZRAZ-2
```

Koda Java izvršava ovu naredbu prvo provjerava da li je rezultat *LOGIČKI-IZRAZ* *točno ili netočno*. Ako je točno izvršit će *IZRAZ-1*. Ako je netočno izvršit će *IZRAZ-2*.

*IZRAZ-1* se ponekad naziva **prva grana**. *IZRAZ-2*, je **druga grana**. Bilo koji izraz može se upotrijebiti umjesto *IZRAZ-1* i *IZRAZ-2*, čak i druga `if` naredba.

Za vizualizaciju događanja u programu koristit ćemo **dijagram toka**. Dijagram toka prethodnog programa izgleda ovako:



Ponekad je potrebno izvršiti nekoliko naredbi kad je uvjet zadovoljen, a nekoliko kad nije.

U tom slučaju možete zamijeniti *UVJET-1* ili *UVJET-2* (ili oboje) s **blokom naredbi**. **Blok** (naredbi) je niz naredbi koje su zatvorene unutar vitičastih zagrada. Čak možete imati i blok bez i jedne naredbe poput `{ }`. Blok se može staviti na bilo koje mjesto u programu na kojem inače dolazi naredba.

Slijedi primjer programa s blokom umjesto grane `if` naredbe (blok je prikazan podebljano).

#### PRIMJER 2

```

public class Multi2

{ /* Traži od korisnika da unese odgovor na pitanje
   koliko je 72 puta 84, i onda provjeri odgovor. */

public static void main(String[] args)

{ ConsoleReader user =
  new ConsoleReader(System.in);

System.out.println("Koliko je 72 puta 84?");
int ans = user.readInt();
if (ans == 6048)
  System.out.println("Točno!");
else
  { System.out.println
    ("Netočno. Još jednom pokušaj !");
  ans = user.readInt();
  if (ans == 6048)
    System.out.println
    ("Napokon točno");
  else
    System.out.println
    ("Opet netočno. Točan odgovor je 6048");
  }
}
}

```

```
    }  
  }  
}
```

### 2. if naredba s jednom granom

Postoji kraća forma if naredbe u kojoj se izostavlja riječ else i druga grana. Slijedi primjer:

```
if (a < b) then  
    System.out.println("Manja vrijednost je " + a);
```

Ovo znači: ako je a manje od b tada prikaži poruku koja kaže da je manja vrijednost a inače čini ništa. Ovakva naredba ima isti efekt kao slijedeća if naredba.

```
if (a < b) then  
    System.out.println("Manja vrijednost je " + a);  
else  
    {}
```

Opći oblik if naredbe s jednom granom je:

if naredba (s jednom granom)

```
if (LOGIČKI-IZRAZ)  
    IZRAZ
```

Ako je *LOGIČKI-IZRAZ* točan tada će se izvršiti *IZRAZ*. Inače je if naredba bez ikakvog efekta.

Slijedi još jedan primjer u kojemu tražimo najmanji od četiri broja a,b,c,d i rezultat pohranjujemo u varijablu s.

```
s = a;  
if (b < s) s = b;  
if (c < s) s = c;  
if (d < s) s = d;
```

Za misaonu vježbu pretpostavite slijedeće vrijednosti i odredite tijek izvršavanja:

a = 5,      b = 10,      c = 3,      d = 8.

Slijedi primjer programa s if naredbom s jednom granom. Program računa kvadratni korijen unesenog broja. U slučaju da korisnik unese negativan broj program će ispisati poruku o grešci te izvršiti return naredbu.

Ključna riječ return znači: završi s izvođenjem metode, u ovom slučaju programa.

#### PRIMJER 3

```
public class Sqrt  
{  
    /* Pitaj korisnika da unese broj a zatim ispiši  
    kvadratni korijen unesenog broja. */  
  
    public static void main(String[] args)  
    {  
        ConsoleReader in =  
            new ConsoleReader(System.in);  
  
        System.out.print("Unesi broj: ");  
        double x = in.readDouble();
```



## 5. Ispitivanje uvjeta

---

```
    if (x < 0)
    {   System.out.println
        ("Broj treba biti >= 0.");
        return;
    }
    System.out.println
        ("Kvadratni korijen je " + Math.sqrt(x));
}
```

Primjeri korištenja programa:

```
Unesi broj: 9.2
Kvadratni korijen je 3.03315017762062
```

```
Unesi broj: -3
Broj treba biti >=0.
```

Često se s greškama postupa na ovakav način tj. ispitivanjem s `if` naredbom te vraćanjem s naredbom `return`.

Return naredba ima dva osnovna oblika:

return naredba  
(vraća vrijednost)

```
return IZRAZ;
```

return naredba  
(ne vraća vrijednost)

```
return;
```

Prva vrsta `return` naredbe koristi se unutar metoda koji vraćaju vrijednost. Znači: završi s izvršavanjem metode i vrati vrijednost danu s `IZRAZ`. Druga se koristi unutar metoda koje ne vraćaju nikakvu vrijednost i znači samo kraj izvršavanja metode

### 3. Izbor između različitih alternativa

Ponekad je potrebno napisati kod koji bira između 3 ili više alternativa. To se može učiniti kombinacijom nekoliko `if` naredbi. Npr. student ostvari određen broj bodova na ispitu od maksimalnih 100. Na osnovu toga treba izračunati ocjenu.

broj bodova 70 - 100	ocjena = 5
broj bodova 60 - 69	ocjena = 4
broj bodova 50 - 59	ocjena = 3
broj bodova 40 - 49	ocjena = 2
broj bodova 0 - 39	ocjena = 1

Pretpostavimo da imamo varijablu `bodova`, koja sadrži broj bodova, a ocjenu želimo pohraniti u varijablu `ocjena`.

**Pseudo kod** bi izgledao ovako:

```
if (bodova >= 70) ocjena = "5";
else Postupaj s bodovima u području od 0 do 69.
```

## 5. Ispitivanje uvjeta

---

Da bismo postupali s bodovima u području od 0 do 69, možemo koristiti drugu if naredbu, poput sljedećeg:

```
if (bodova >= 70) ocjena = "5";
else if (bodova >= 60) ocjena = "4";
else Postupaj s bodovima u području od 0 do 59.
```

Dalje postupamo na sličan način:

```
if (bodova >= 70) ocjena = "5";
else if (bodova >= 60) ocjena = "4";
else if (bodova >= 50) ocjena = "3";
else Postupaj s bodovima u području od 0 do 49
```

Itd. Na kraju slijedi kompletan izraz:

```
if (bodova >= 70) ocjena = "5";
else if (bodova >= 60) ocjena = "4";
else if (bodova >= 50) ocjena = "3";
else if (bodova >= 40) ocjena = "2";
else ocjena = "1";
```

Primijetite da je ovaj komad koda zapravo if naredba !

Slijedi kompletan program za proračun ocjene iz broja bodova:

### PRIMJER 4

```
public class Ocjena
{ /* Učitaj broj bodova u području od 0 do 100,
   i ispiši odgovarajuću ocjenu.
   */
  public static void main(String[] args)
  { ConsoleReader in =
      new ConsoleReader(System.in);

    System.out.print("Unesi broj bodova ");
    int bodova = in.readInt();
    String ocjena;
    if (bodova >= 70) ocjena = "5";
    else if (bodova >= 60) ocjena = "4";
    else if (bodova >= 50) ocjena = "3";
    else if (bodova >= 40) ocjena = "2";
    else ocjena = "1";
    System.out.println("Ocjena = " + ocjena);
  }
}
```

Ovom tehnikom ispitujemo niz uvjeta i čim je jedan zadovoljen izvršavamo odgovarajuću akciju.

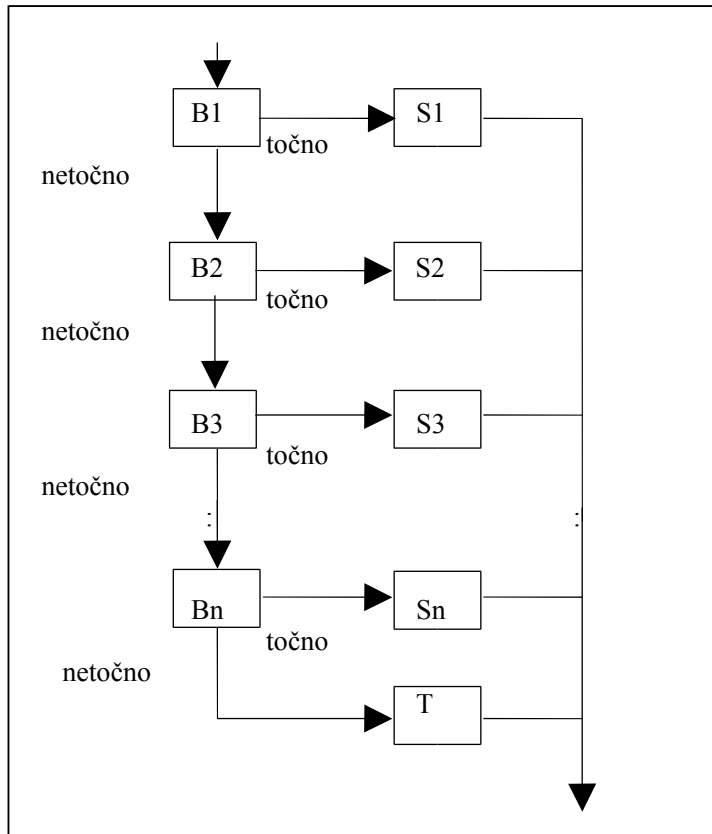
```
if (B1) S1
else if (B2) S2
else if (B3) S3
:
:
else if (Bn) Sn
```

## 5. Ispitivanje uvjeta

---

else T

Blok dijagram ovakve if naredbe izgleda ovako:



#### 4. Tip podataka `boolean`

Logički (boolean) izraz poput `odgovor == 6048` izračunava se u jednu od slijedećih vrijednosti `true` (točno) and `false` (netočno). Ove dvije vrijednosti nazivaju se **logičke vrijednosti**. Zajedno čine primitivni tip podataka koji u Javi nazivamo `boolean`.

U mnogim pogledima ove vrijednosti se ponašaju poput drugih tipova vrijednosti kao npr. cjelobrojnih vrijednosti. Npr. metode mogu poput vraćanja cjelobrojnih vrijednosti vraćati logičke vrijednosti. Možete kreirati varijablu tipa `boolean` i dodjeljivati im vrijednosti.

Npr.:

```
boolean ok = (x > 0);
```

kao primjer korištenja logički vrijednosti, proširit ćemo `Student` klasu iz trećeg poglavlja. Dodat ćemo logički polje u koje ćemo bilježiti je li student upisao predmete tekuće godine.

Polje ćemo nazvati `jeUpisan`. Bit će postavljeno na `true` ako je student upisao predmete, a na `false` ako nije. Polje je deklarirano na slijedeći način:

```
private boolean jeUpisan;
```

bit će potrebno učiniti još neke promjene u klasi `student`. Dodat ćemo slijedeće metode:

```
s.upiši()
Zapiši da je student s upisan.
```

```
s.ispiši()
Zapiši da Student s nije upisan.
```

## 5. Ispitivanje uvjeta

---

s.jeliUpisan()  
vrati true ako je s upisan, inače vrati false.

metode su vrlo jednostavne. Prva postavlja polje jeUpisan na true, druga postavlja polje jeUpisan na false i treća vraća vrijednost polja jeUpisan.

napravljene su još dvije promjene u klasi Student. U metodi prikaz dodano je ispitivanje vrijednosti polja jeUpisan na osnovu koje se ispisuje 'Upisan je' ako je njegova vrijednost postavljena na true, ili 'Nije upisan' ako je false. Konstruktor postavlja vrijednost polja jeUpisan na false.

Slijedi primjer promijenjene Student klase. Metodi koji nisu mijenjani napisani su u skraćenom obliku.

### PRIMJER 5

```
public class Student

{   private String idBroj;
    private String ime;
    private String programStudija;
    private int godina;
    private boolean jeUpisan;

    /* Zabilježi da je student upisan.
    */
    public void upiši()
    {   jeUpisan = true;
    }

    /* Zabilježi da student nije upisan
    */
    public void ispiši()
    {   jeUpisan = false;
    }

    /* vrati true ako je student upisan
    inače vrati false
    */
    public boolean jeliUpisan()
    {   return jeUpisan;
    }

    /* Prikaži detalje o studentu
    */
    public void prikaz()
    {   System.out.println
        ("Student ID: " + idBroj);
        System.out.println("Ime: " + ime);
        System.out.println
        ("Program studija: " + programStudija);
        System.out.println("Godina: " + godina);
        if (jeUpisan)
            System.out.println("Nije upisan.");
        else
            System.out.println("Upisan je.");
    }
}
```

## 5. Ispitivanje uvjeta

---

Definicije klase `setProgramStudija`, `povecajGodinu` i `getIme` su nepromijenjene

```
/* Kreiraj novog studenta sa zadanim
   ID brojem, imenom i programom studija
   Polje godina bit će postavljeno na 1.
   Student početno nije upisan.
*/

public Student(String id, String nm, String prog)
{
    idBroj = id;
    ime = nm;
    programStudija = prog;
    godina = 1;
    jeUpisan = false;
}
}
```

Kada pišemo logičke izraze možemo koristiti slijedeće relacijske operatore:

<code>==</code>	jednako
<code>!=</code>	nije jednako
<code>&lt;</code>	manje od
<code>&lt;=</code>	manje ili jednako
<code>&gt;</code>	veće
<code>&gt;=</code>	veće ili jednako

Ovi operatori se koriste između bilo koja dva izraza koji označavaju cjelobrojnu ili vrijednost u pokretnom zarezu. Kompleksnije logičke izraze formiramo korištenjem logičkih operatora. (B1 i B2 su logički izrazi)

<i>operator</i>	<i>značenje</i>
<code>B1 &amp;&amp; B2</code>	B1 I B2
<code>B1    B2</code>	B1 ILI B2
<code>! B1</code>	NEGACIJA B1

Izraz `B1 && B2` daje rezultat `true` ako su i B1 i B2 `true`, inače daje `false`. Kada Java izračunava izraz `B1 && B2`, prvo računa izraz B1. Ako je B1 `false` izraz B2 se ne izračunava, jer neovisno o njemu cijeli izraz je `false`.

Izraz `B1 || B2` bit će `true` ako najmanje jedan od B1 i B2 je `true`. Ako su i B1 i B2 vrijednosti `false`, rezultat je `false`. Java također izračunava `B1 || B2` na 'lijeni način'. Ako je B1 `true` tada B2 se ne računa! (Zašto?)

Operatori `&&` i `||` su manjeg prioriteta od drugih operatora koje smo dosada sreli. Java će izraz:

```
n >= 0 && n < 10 || n >= 20 && n < 30
```

interpretirati kao:

```
((n >= 0) && (n < 20)) || ((n >= 20) && (n < 30))
```

Operator 'not' tj., `!`, je većeg prioriteta nego ostali operatori. Operator `=` je najmanjeg prioriteta od dosad spomenutih. Stoga možemo izostaviti zagrade u već spomenutom primjeru:

```
boolean ok = (x > 0);
```

U sljedećem dijelu su primjeri upotrebe logičkih operatora.

### 5. Usporedba stringova

Jedan od najčešće upotrebljivanih metoda koja vraća `boolean` vrijednost je metoda `equals`. Svaki `string` (string je klasa) ima `equals` metodu koja se koristi da li string za kojeg smo pozvali metodu ima istu sekvencu karaktera kao i string koji je proslijeđen kao parametar.

Slijedi primjer u kojemu testiramo da li se string referenciran s `st` odnosi na isti niz znakova koji su proslijeđeni kao parametar:

```
st.equals("cat")
```

Slijedi primjer jednostavnog programa koji testira sadržaj stringa na navedeni način. program pokušava prepoznati što je korisnik utipkao te daje odgovarajuće odgovore.

#### PRIMJER 6

```
public class KakoSi
{ /* Pitaj korisnika "Kako je" i daj odgovarajući
  komentar.

  */

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

    System.out.println
      ("Ćao korisniĆe. kako ide danas ?");
    String odgovor = in.readLine();
    if (odgovor.equals("Dobro"))
      System.out.println
        ("Lijepo je Ćuti da nekom ide dobro.");
    else if (odgovor.equals("Loše"))
      System.out.println("Źao mi je, valjda Će biti bolje");
    }
  }
```

Operator `==` se normalno ne koristi da bi se ispitalo da li su dva stringa ista. Razlog je što Java izraz:

```
S1 == S2
```

gdje `S1` i `S2` referenciraju stringove interpretira na naĉin da uspoređuje reference, a ne sadržaj stringova. Znaĉi, testira se da li obje reference pokazuju na isti string u memoriji.

Promotrite sljedeći kod:

```
String S1 = "Isti smo";
String S2 = "Isti smo";

if (S1==S2) {
  System.out.println("Isti")
}
```

## 5. Ispitivanje uvjeta

---

```
else{
    System.out.println("Različiti")
}
```

Rezultat ispisa bit će "Različiti" jer varijable S1 i S2 referenciraju na **različite stringove** s istim sadržajem.

Korisna alternativa equals metodi je slijedeća metoda.

```
st1.equalsIgnoreCase(st2)
```

Ova metoda ne razlikuje velika i mala slova pa ako npr. st1 je "the dog", a st2 je "The Dog", onda će metoda vratiti true.

Postoji još mnogo korisnih metoda vezanih za stringove koje možete pogledati u Java dokumentaciji.

Slijedi primjer appleta koji na osnovu korisnikova unosa željenog oblika i boje iscrtava lik na ekranu. Prvo se traži naziv boje te se na osnovu unosa kreira odgovarajući objekt.

Slijedi dio za unos i kreiranje boje:

```
String odgovor1 =
    JOptionPane.showInputDialog
        ("Naziv željene boje?");
if (odgovor1.equalsIgnoreCase("crvena"))
    color = Color.red;
else if (odgovor1.equalsIgnoreCase("žuta"))
    color = Color.yellow;
else if (odgovor1.equalsIgnoreCase("plava"))
    color = Color.blue;
else
    color = null;
```

Program prepoznaje samo nazive 'crvena', 'žuta' i 'plava'. Ako korisnik utipka bilo što drugo npr. 'zelena' ili '%7df&#yx!', program će spremiti null u varijablu color.

Slijedeći kod pita korisnika za oblik i rezultat sprema u varijablu shape. Program prepoznaje samo dva oblika.

```
String odgovor2 =
    JOptionPane.showInputDialog
        ("Koji oblik želite?");
if (odgovor2.equalsIgnoreCase("kvadrat"))
    shape = new Rectangle2D.Double(50, 50, 200, 200);
else if (odgovor2.equalsIgnoreCase("krug"))
    shape = new Ellipse2D.Double(50, 50, 200, 200);
else
    shape = null;
```

Kad smo postavili vrijednosti u varijable color i shape izvršit ćemo dvije slijedeće naredbe:

```
if (color != null)
    g2.setColor(color);
if (shape != null)
    g2.fill(shape);
```



## 5. Ispitivanje uvjeta

---

Ako smo unijeli prepoznatljivu boju Graphics2D objekt bit će postavljen na zadanu boju, a ako nismo, bit će ostavljena njegova prethodna boja.

Ako je program prepoznao lik onda će ga nacrtati, inače ne.

Slijedi kompletan program:

### PRIMJER 7

```
// Applet koji vas pita za boju i oblik
// te prikazuje unijeti oblik ispunjen sa zadatom bojom

import java.applet.*;
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

public class Oblici extends Applet

{   private Color boja;
    private Shape oblik;

    public void init()
    {
        String odgovor1 =
            JOptionPane.showInputDialog
                ("Naziv željene boje?");
        if (odgovor1.equalsIgnoreCase("crvena"))
            boja = Color.red;
        else if (odgovor1.equalsIgnoreCase("žuta"))
            boja = Color.yellow;
        else if (odgovor1.equalsIgnoreCase("plava"))
            boja = Color.blue;
        else
            boja = null;

        String odgovor2 =
            JOptionPane.showInputDialog
                ("Koji oblik želite?");
        if (odgovor2.equalsIgnoreCase("kvadrat"))
            oblik = new Rectangle2D.Double(50,50,200,200);
        else if (odgovor2.equalsIgnoreCase("krug"))
            oblik = new Ellipse2D.Double(50,50,200,200);
        else
            oblik = null;
    }

    public void paint(Graphics g)
    {   Graphics2D g2 = (Graphics2D) g;

        /* Prikaži obojani lik. */

        if (boja != null)
            g2.setColor(boja);

        if (oblik != null)
            g2.fill(oblik);
    }
}
```

Analizirajmo formu appleta. Postoje dva osnovna zadatka:

## 5. Ispitivanje uvjeta

---

Korak 1. Unesi željenu boju i oblik.

Korak 2. Naslikaj željeni lik u željenoj boji.

Drugi korak će se izvršavati svaki put kad bude potrebno osvježiti sadržaj područja prikaza.

Stoga je uključen u applet `paint` metodu.

S druge strane prvi korak treba biti izvršen samo jednom pa je uključen u `init` metodu.

Za zadnji primjer u ovom poglavlju slijedi dodavanje još jedne metode u klasu `Student`. Nazvat ćemo je `edit`. Koristit će `equals` metodu za usporedbu stringova i logičke operatore.

```
s.edit(in)
```

Prikazuje podatke o objektu `s` tipa `Student`, pita korisnika da li želi napraviti kakve promjene. Ako želi `s` se odgovarajuće ažurira. Korisnikov unos učitavamo pomoću `ConsoleReader` objekta.

Metoda `edit` prikazuje redom vrijednost svakog polja, i čeka korisnikov odgovor. Ako je korisnik zadovoljan s trenutnom vrijednošću onda utipka samo `Return` (`Enter`). Ako unese neki string metoda će pretpostaviti da se radi o novoj vrijednosti i spremi je u odgovarajuće polje.

Primjer mogućeg prikaza na ekranu prilikom izvršavanja metode `edit`:

```
ID (9912345)
Ime (Jure Antić)
Program studija (Računarstvo) CS
Godina (1) 2
Upisan? (Da) no
```

Svaka se linija sastoji od naziva informacije i njene trenutne vrijednosti te podebljano ispisanog korisnikovog unosa. U slučaju prva dva retka korisnik je samo utipkao `Enter` i odgovarajuće vrijednosti su ostale iste. Slijedeća tri retka korisnik je unio nove podatke i potrebno ih je pohraniti u odgovarajuća polja.

Slijedi kod koji radi s ID brojem.

```
System.out.print("ID (" + idBroj + ") " );
String odgovor = input.readLine();
if (!odgovor.equals(""))
    idBroj = odgovor;
```

Za studentovo ime postupa se skoro na isti način.

Za program studija se postupa na malo drukčiji način. Ispitivanje sadržaja stringa obavlja se na slijedeći način:

```
if (!odgovor.equals(""))
    if (odgovor.equals("Elektronika") ||
        odgovor.equals("Računarstvo") ||
        odgovor.equals("Strojarstvo") ||
        odgovor.equals("Brodogradnja") )
        programStudija = odgovor;
    else
        System.out.println("Nepoznat program studija.");
```

Metoda `edit` postupa s poljem `jeUpisan` opet na različit način. Umjesto ispisa stvarnog sadržaja ispisuje se 'da' ili 'ne' za indicaciju upisa studenta. Korisnik mora odgovoriti s 'da' or 'ne' ako želi napraviti promjenu u tom polju

Slijedi potpuna metoda edit:

### PRIMJER 8

```
/* Editiraj studentove podatke
*/

public void edit(ConsoleReader input)
{
    System.out.print("ID (" + idBroj + ") ");
    String odgovor = input.readLine();
    if (! odgovor.equals(""))
        idBroj = odgovor;

    System.out.print("Ime (" + ime + ") ");
    odgovor = input.readLine();
    if (! odgovor.equals(""))
        ime = odgovor;

    System.out.print
        ("Program studija (" + programStudija + ") ");
    odgovor = input.readLine();
    if (! odgovor.equals(""))
if (odgovor.equals("Elektronika") ||
    odgovor.equals("Računarstvo") ||
        odgovor.equals("Strojarstvo") ||
        odgovor.equals("Brodogradnja") )
        programStudija = odgovor;
    else
        System.out.println("Nepoznat program studija.");

    System.out.print("Godina (" + godina + ") ");
    odgovor = input.readLine();
    if (! odgovor.equals(""))
        godina = Integer.parseInt(odgovor);

    if (jeUpisan)
        System.out.print("Upisan? (da) ");
    else
        System.out.print("Upisan? (ne) ");
    odgovor = input.readLine();
    if (! odgovor.equals(""))
        if (odgovor.equals("da"))
            jeUpisan = true;
        else if (odgovor.equals("ne"))
            jeUpisan = false;
        else
            System.out.println("Odgovor nije razumljiv.");
}

public class TestStudent
{
    /* Čita podatke o studentu i kreira odgovarajući
    Student objekt.
```

## 5. Ispitivanje uvjeta

---

```
Zatim editira objekt Student
    te na kraju prikazuje sadržaj.
*/

public static void main(String[] args)

{ /* Kreiraj objekt tipa ConsoleReader za učitavanje
    podataka s tastature */

    ConsoleReader in =
        new ConsoleReader(System.in);

    /* Kreiraj objekt Student. */

    System.out.println
        ("Koji je ID broj studenta?");
    String i = in.readLine();
    System.out.println("Koje je ime studenta?");
    String n = in.readLine();
    System.out.println("Koji je program studija?");
    String d = in.readLine();
    Student st = new Student(i,n,d);

    /* Editiraj pa prikaži podatke studenta. */

    System.out.println();
    System.out.println("Editiraj podatke studenta.");
    st.edit(in);
    System.out.println();
    System.out.println("Student podaci:");
    st.prikaz();
}
}
```

Slijedi što bi program mogao ispisivati (unos korisnika podebljano):

```
Koji je ID broj studenta?
9912345
Koje je ime studenta?
Jure Antić
Koji je program studija?
Računarstvo

Editiraj podatke studenta.
ID (9912345) 9912346
Ime (Jure Antić)
Program studija (Računarstvo) Elektronika
Godina (1) 2
Upisan (ne) da

Student podaci:
Student ID: 9912346
```

## 5. Ispitivanje uvjeta

---

Ime: Jure Antić  
Program studija: Elektronika  
Godina: 2  
Upisan.

U kodu programa postoji izraz slijedećeg oblika (ispitivanje polja programStudija):

```
if (B1) if (B2) S1 else S2
```

Kojemu if pripada else.  
Ovo je primjer tzv. visećeg else.

Java će to interpretirati na slijedeći način:

```
1.  if (B1) { if (B2) S1 else S2 }
```

a ne ovako:

```
2.  if (B1) { if (B2) S1 } else S2
```

Pravilo je da else tvori cjelinu s najbližim if-om.

U ovim slučajevima preporučljivo je koristiti vitičaste zagrade !

Kod sa vitičastim zagradama:

```
if (! odgovor.equals(""))
{
    if (odgovor.equals("da"))
        jeUpisan = true;
    else if (odgovor.equals("ne"))
        jeUpisan = false;
    else
        System.out.println("Odgovor nije razumljiv.");
}
```

## 6. Blokovi i doseg varijabli

Zapamtite da je blok sekvenca naredbi zatvorena u vitičaste zagrade. Jedna je bitna stvar u radu s blokovima u Javi. Varijabla koju deklaracijom unutar bloka vrijedi samo unutar bloka. čim Java napusti blok varijable iščezava tj. oslobađa se memorija koja je za nju bila korištena. promotrite slijedeći primjer:

```
if (x == 0)
{
    System.out.println("x is nula.");
    int y = 1;
} else {
    System.out.println("x nije nula.");
    int y = 2;
}
System.out.println("y je " + y);
```

## 5. Ispitivanje uvjeta

---

Ovaj program će javiti pogrešku u toku prevođenja. varijabla `y` ne vrijedi izvan bloka u kojem je deklarirana. Što više programer je kreirao dvije posebne varijable `y` unutar posebnih blokova.

Dio programa u kojemu možemo koristiti određenu varijablu naziva se **doseg** (scope) te varijable.

prethodni program ima više smisla (kompajlirat će se) ako ga preuredimo na slijedeći način:

```
int y;
if (x == 0)
{ System.out.println("x je nula.");
  y = 1;
} else {
  System.out.println("x nije nula.");
  y = 2;
}
System.out.println("y je " + y);
```

varijabla deklarirana unutar bloka naziva se **lokalna varijabla**. Tijelo metoda je također blok pa tako svaka varijabla koja je deklarirana unutar metode je lokalna varijabla.

To znači da bi svaka varijabla deklarirana unutar metode trebala prestati s postojanjem nakon izlaska iz metode.

S druge strane, polje objekta postoji sve dok postoji objekt kome pripada. Objekt traje sve dok Java ne zaključi da se više ne upotrebljava. Taj proces nije nimalo jednostavan kako to pokazuje naziv **garbage collection** (odvoz smeća).

Činjenica da lokalne varijable vrijede samo dok se metod izvršava objašnjava nam zašto smo u primjeru 7 upotrijebili dva nova polja `Color` i `Shape`. Pošto smo tim referencama u `Init` metodi dodijelili objekte boje i oblika, reference su trebale ostati sačuvane da bi objektima mogla pristupiti `paint` metoda.

## 7. Zadaci

Napiši program (`TriBroja.java`) koji čita tri broja u pokretnom zarezu i ispisuje najveći od njih:

Npr:

```
Unesi tri broja:
4 9 2.5
Najveći je broj 9.
```

- Napiši applet (`KrugTest.java`, `Krugtest.txt` (HTML) )koji crta krug radijusa 100 s centrom u (110,120). Zatim traži od korisnika da unese koordinate neke točke. Ako točka leži unutar Kruga onda ispisuje poruku "Unutar kruga", a ako leži vani poruku "Van kruga".
- Godina s 366 dana naziva se prestupna godina. Godina je prestupna ako je djeljiva s 4 (npr. 1980), osim ako nije djeljiva s 100 (npr. 1900). Izuzetak su godine djeljive s 400 (npr. 2000) koje su prestupne u svakom slučaju. Nije bilo prestupnih godina prije uvođenja Gregorijanskog kalendara na dan 15.10.1582 (npr., 1500. ipak nije bila prestupna). napiši program koji će korisnika pitati za godinu , a onda ispisati je li ili nije prestupna.

Provjera 1996 i 2000 su prestupne. 1900 i 1999 nisu.

