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?

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 omogući 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 dizajner 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 asocirane s objektom tipa Student.

Da bismo održali jednostavnost primjera uključit ćemo ograničeni skup operacija:

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

2. Bit će potrebno omogućiti promjenu programa studija (stupnja)

3. 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.

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

5. 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

- **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 asociranih 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:

```
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.
- `()` 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()

{  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)

S ovim smo kompletirali definiciju klase Student. Još ostaje dodati zaglavlje cijele klase. U klasi su prvo stavljena polja iako redoslijed 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++;
    }

    /* Vrati 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
       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();

      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. Intepreter 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 intepreter s naredbom:

java klasa

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

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

(Naziv parametra može biti bilo koji naziv.) Ako intepreter nađe takvu metodu počet će izvršavati njene naredbe. Da bi uspješno upravljao s potrebnim objektima i metodama, intepreter 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

```
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);
```

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`.

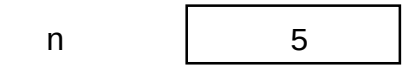
---

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;
```

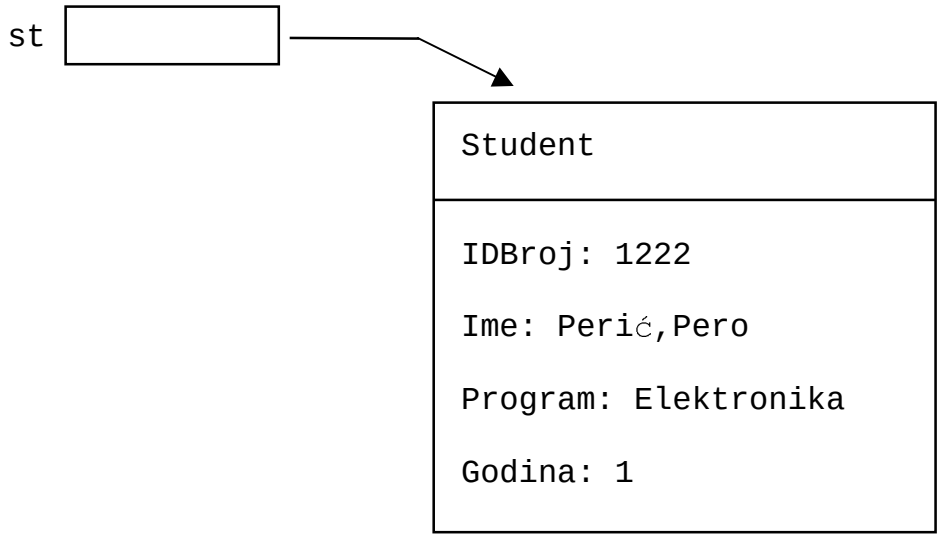
Nakon ovoga izraza 32 bita memorije su rezervirana za novu varijablu `n`, i vrijednost 5 je spremljena na to mjesto u memoriji.
Ovaj postupak možemo vizualizirati na slijedeći način: (Pravokutnik pokazuje rezervirano mjesto u memoriji)



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 proslijeđene kao parametri.

Nikad nećemo moći vidjeti stvarnu vrijednost vrijednosti 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:

```
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.

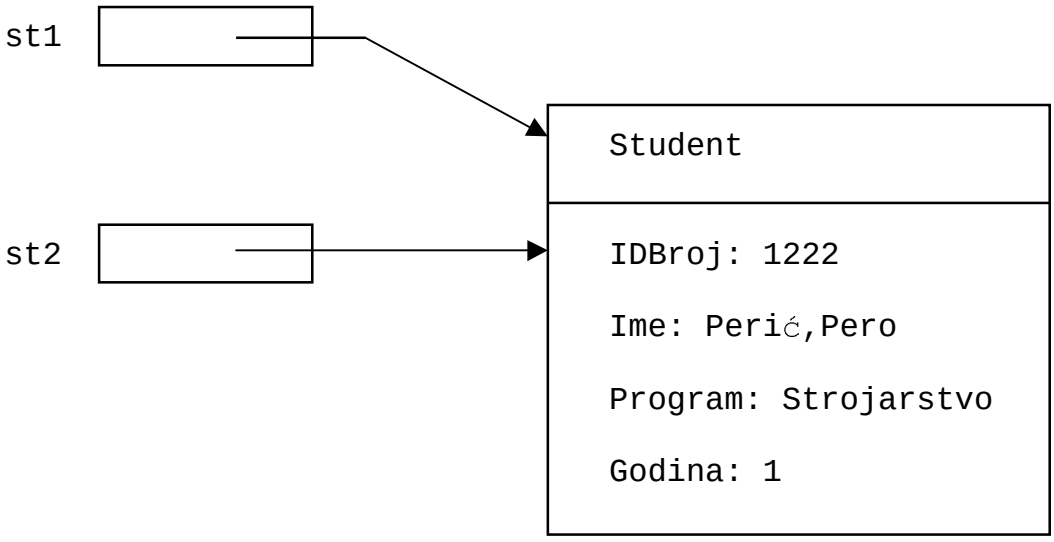
2. `Student st2 = st1;`

   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.** Obje 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.

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

1. 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 položeni 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.