

Klase, objekti i njihovi elementi

Klase

Klasa predstavlja neki predmet ili pojavu iz realnog sveta izraženu u obliku pogodnom za neki (objektno-orijentisani) programski jezik. Klasa je uprošćena slika realnih predmeta i pojava i obuhvata njihove:

- karakteristike (atribute)
- ponašanja (metode)
- odnose sa drugim klasama (relacije)

Atributi, metode i relacije se nazivaju **elementi (članice) klase**.

Primer 1

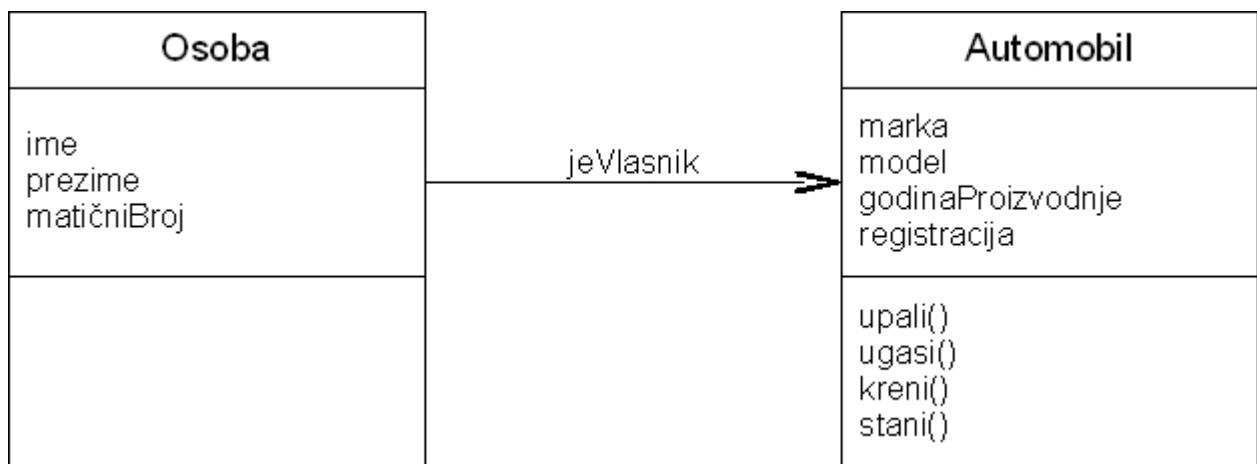
Primer klase je Automobil (Slika 1). Automobil ima sledeće karakteristike: marku, model, godinu proizvodnje i registracioni broj. Neka od ponašanja automobila su: upali, ugasi, kreni, stani. Na slici se može videti da su ove karakteristike predstavljene preko atributa, dok su ponašanja predstavljena korišćenjem metoda. Bitno je primetiti da je ovo uprošćena slika realnog automobila jer automobil može imati i niz drugih karakteristika (serijski broj, boja...) i ponašanja (skreni, otvori vrata...).

| Automobil |
|---|
| marka model godinaProizvodnje registracija |
| upali() ugasi() kreni() stani() |

*Slika 1: Primer klase - Klasa
Automobil*

Primer 2

Drugi primer klase može da bude Osoba. Recimo da osoba ima ime, prezime i matični broj, a da, u ovom primeru, nema definisano nikakvo ponašanje. Ako se uzme u obzir klasa Automobil iz prethodnog primera i klasa Osoba, između njih postoji određen odnos. Osoba može da bude vlasnik auta. Ovaj odnos se predstavlja putem relacije između klasa (Slika 2).



Slika 2: Primer relacije između dve klase

Detalji u vezi sa atributima, metodama i relacijama će biti objašnjeni u okviru narednih poglavlja. Sada je bitno videti kako se vrši deklaracija klase u Javi.

Opšti oblik deklaracije klase u Javi je sledeći:

```

class NazivKlase {

    //deklaracije atributa...

    //deklaracije metoda...

}
  
```

Deklaracija počinje ključnom reči “class”, posle koje ide naziv klase. Ovaj prvi red je **zaglavlje klase**. Sledeći element je otvorena vitičasta zagrada. Posle vitičaste zagrade počinje **telo klase** koje sadrži deklaracije atributa i metoda. Deklaracija klase se završava zatvorenom vitičastom zagradom.

Java je programski jezik u kome se **pravi razlika između velikih i malih slova** (case sensitive). To znači da se nazivi klase, atributa, metoda mogu razlikovati u zavisnosti od toga da li se napišu velikim ili malim slovima. Na primer, “KonvertorValute”, “konvertorvalute” i “KONVERTORVALUTE” će da se tretiraju kao potpuno različiti nazivi.

Trenutno, u okviru tela klase koja je data kao primer deklaracije se nalaze samo dva komentara. Uopšteno gledano, **komentari** služe tome da se napiše neko objašnjenje ili napomena u okviru samog koda, a da se to objašnjenje ne tumači kao programski kod od strane Jave.

Komentari se u Javi mogu pisati na dva načina. Jednu varijantu predstavljaju kratki, **jednolinijski komentari**. Početak ovakvog komentara se označava duplom kosom crtom, a ceo tekst mora da stane u jedan red. Primer jednolinijskog komentara je:

```

//deklaracije atributa...
  
```

Druga vrsta komentara su **višelinijski komentari**. Oni se pišu u slučaju da je potrebno dati neko detaljnije ili duže objašnjenje. Naravno, u tom slučaju je moguće napisati više jednolinijskih komentara, ali je mnogo jednostavnije ceo tekst komentara uokviriti kao jedan višelinijski komentar. Višelinijski komentar počinje kosom crtom i zvezdom (“/*”) a završava se zvezdom i

kosom crtom ("*/"):

```
/* Ovo je  
višelinijfski  
komentar */
```

Primer 3

Deklarisati Klasu Automobil. Ova klasa ne bi trebalo da ima nijedan atribut niti metodu.

```
class Automobil {  
  
}
```

Primer 4

Deklarisati Klasu Osoba. Ova klasa ne bi trebalo da ima nijedan atribut niti metodu.

```
class Osoba {  
  
}
```

Potrebno je napomenuti da se u objektno-orijentisanim programskim jezicima, u principu, poštuje **nepisano pravilo da se nazivi klase pišu sa prvim velikim slovom** ("Automobil" a ne "automobil"). Ako se naziv klase sastoji iz više reči, onda se sve reči pišu spojeno a prvo slovo svake reči je veliko ("KonvertorValute", "PoslovniCentar", "MetričkoImperijalniKonvertor"). **Naziv ne sme da ima nijedan blanko znak** (ovo važi i za nazive svih drugih elemenata - atributa, metoda...).

Takođe, običaj je da se **izvorni kod svake klase čuva u posebnom fajlu**. Tako će izvorni kod klase "Automobil" da bude sačuvan u fajlu "Automobil.java", a klase "Osoba" u fajlu "Osoba.java". Kada se kompajlira, Java izvršni kod će da bude sačuvan u fajlu "Automobil.class" tj. "Osoba.class".

Atributi

Kao što je već rečeno, **atributi** predstavljaju neke karakteristike klase. Za klasu Osoba, to može biti ime, prezime, pol, starost itd. U pitanju su, najčešće, neke jednostavne karakteristike koje se mogu izraziti putem nekog broja, slova ili niza slova.

Deklaracija atributa je relativno jednostavna i izgleda ovako:

```
tip_podatka nazivAtributa;
```

Tip podatka predstavlja tip podatka koji taj atribut sadrži - ceo broj, realan, broj, slovo, niz slova ili nešto drugo. Prema nepisanom pravilu, **nazivi atributa bi trebalo da počinju malim slovom** ("starost", "marka", "model" itd). Ako se naziv atributa sastoji od više reči, sve reči se pišu spojeno, prva počinje malim slovom, a sve ostale velikim ("matičniBroj", "brojMotora" itd). Deklaracija se završava tačka-zarezom.

Jedno jako važno pravilo za pisanje komandi u Javi je da se **sve komande moraju završiti znakom tačka-zarez.**

Neki **najčešće korišćeni tipovi podataka** u Javi su:

| <i>Naziv tipa podatka</i> | <i>Opis</i> | <i>Primeri</i> |
|---------------------------|---|---|
| int | celi brojevi | 1, -55, 0, 100000 |
| double | realni brojevi | 12.55, -234.77, 0.21 |
| char | znak (slovo, cifra ili neki drugi znak) | 'a', 'A', 'e', '!', ';', ' '(blanko znak), '4', '9' |
| boolean | logička promenljiva | true, false (samo ove dve vrednosti) |
| String | niz znakova | “Reka”, “Pera”, “123”, “zx!” |
| Calendar | datum i vreme | 2008-12-31 10:50 |

Int, double, char i boolean su **prosti tipovi** podataka, dok su String i Calendar **složeni** pa se, u stvari, predstavljaju korišćenjem odgovarajućih predefinisanih klasa. U Javi se sve klase smatraju složenim tipovima podataka, ali će o tome biti više reči u narednim poglavljima. Potrebno je napomenuti da postoje i još neki prosti tipovi podataka ali da se ne koriste toliko često: long (celi brojevi, ali je raspon veći nego kod int), short (celi brojevi, ali je raspon manji nego kod int) i float (realni brojevi, ali je preciznost manja nego kod double).

Takođe, u Javi se **char vrednosti (znakovi) moraju pisati pod jednostrukim znacima navoda**, dok se **String vrednosti obavezno pišu pod dvostrukim znacima navoda**.

Primer 5

Deklarisati Klasu AutomatNovca. Ova klasa bi trebalo da ima samo atribut “stanje” koji predstavlja iznos novca koji se trenutno nalazi u automatu (realan broj).

```
class AutomatNovca {  
  
    double stanje;  
  
}
```

Primer 6

Deklarisati Klasu Računar. Ova klasa bi trebalo da ima sledeće attribute: takt procesora (realan broj, npr. 4.0 GhZ), radna memorija (realan broj, npr. 2.0 Gb), hard disk (ceo broj, npr. 120 Gb).

```
class Racunar {  
  
    double taktProcesora;  
    double radnaMemorija;  
    int hardDisk;
```

```
}
```

Redosled u kojem se deklariraju atributi u okviru klase nije bitan. Atributima je moguće dodeliti i neku podrazumevanu, početnu vrednost odmah pri deklaraciji klase. To se radi na sledeći način:

```
tip_podatka nazivAtributa = vrednost;
```

Primer 7

Prepraviti klasu **Racunar** iz prethodnog primera tako da početna vrednost za takt procesora bude 4.0 (GhZ), radna memorija iznosi 2.0 (Gb) a hard disk ima 120 (Gb).

```
class Racunar {  
  
    double taktProcesora = 4.0;  
    double radnaMemorija = 2.0;  
    int hardDisk = 120;  
  
}
```

Zadaci

Zadatak 1

Napraviti klasu **Televizor**. Ova klasa bi trebalo da ima:

- Atribut **jacinaTona** koji je ceo broj i označava trenutnu jačinu tona na televizoru. Početna vrednost ovog atributa je 0 (ton je utišan do kraja).
- Atribut **trenutniProgram** koji označava broj programa koji je trenutno na televizoru (npr. uključen je program 5). Početna vrednost ovog atributa je 1.
- Atribut **uključen** koji označava da li je televizor uključen ili nije (ako je uključen ima vrednost TRUE, a inače FALSE). Smatra se da je na početku televizor isključen.

Rešenje:

```
class Televizor {  
  
    int jacinaTona = 0;  
    int trenutniProgram = 1;  
    boolean ukljucen = false;  
  
}
```

Zadatak 2

Napraviti klasu **Radio**. Ova klasa bi trebalo da ima:

- Atribut **fmFrekvencija** koji označava trenutnu FM radio frekvenciju koju radio pušta (npr. 102.2 ili 87.5). Početna vrednost ovog polja je 87.5.
- Atribut **amFrekvencija** koji označava trenutnu AM radio frekvenciju koju radio pušta (npr. 567 ili 1500). Početna vrednost ovog parametra je 567.
- Atribut **band** koji ima vrednost 'A' (slovo A) ako radio pušta AM frekvenciju ili 'F' (slovo F) ako radio pušta FM frekvenciju. Početna vrednost ovog parametra je 'F'.

Rešenje:

```
class Radio {
```

```

    double fmFrekvencija = 87.5;
    int amFrekvencija = 567;
    char band = 'F';
}

```

Zadatak 3

Napraviti klasu **Student**. Ova klasa bi trebalo da ima:

- Atribut **ime**. Početna vrednost ovog atributa je “nepoznato”
- Atribut **prezime**. Početna vrednost ovog atributa je “nepoznato”
- Atribut **pol** koji može imati vrednosti 'M' ili 'Z'.
- Atribut **brojIndeksa** (niz slova).
- Atribut **prosecnaOcena**.

Rešenje:

```

class Student {

    String ime = "nepoznato";
    String prezime = "nepoznato";
    char pol;
    String brojIndeksa;
    double prosecnaOcena;

}

```

Zadatak 4

Napraviti klasu **FudbalskaUtakmica**. Ova klasa bi trebalo da ima:

- Atribut **gostujuciTim** koji predstavlja naziv fudbalskog tima koji gostuje.
- Atribut **domaciTim** koji predstavlja naziv fudbalskog tima koji je domacin utakmice.
- Atribut **lokacija** koji predstavlja naziv grada u kome se igra. Početna vrednost za ovaj atribut je “nepoznata”.
- Atribut **brojGolovaGostiju**.
- Atribut **BrojGolovaDomacih**.

Rešenje:

```

class FudbalskaUtakmica {

    String gostujuciTim;
    String domaciTim;
    String lokacija = "nepoznata";
    int brojGolovaGostiju = 0;
    int brojGolovaDomacih = 0;

}

```

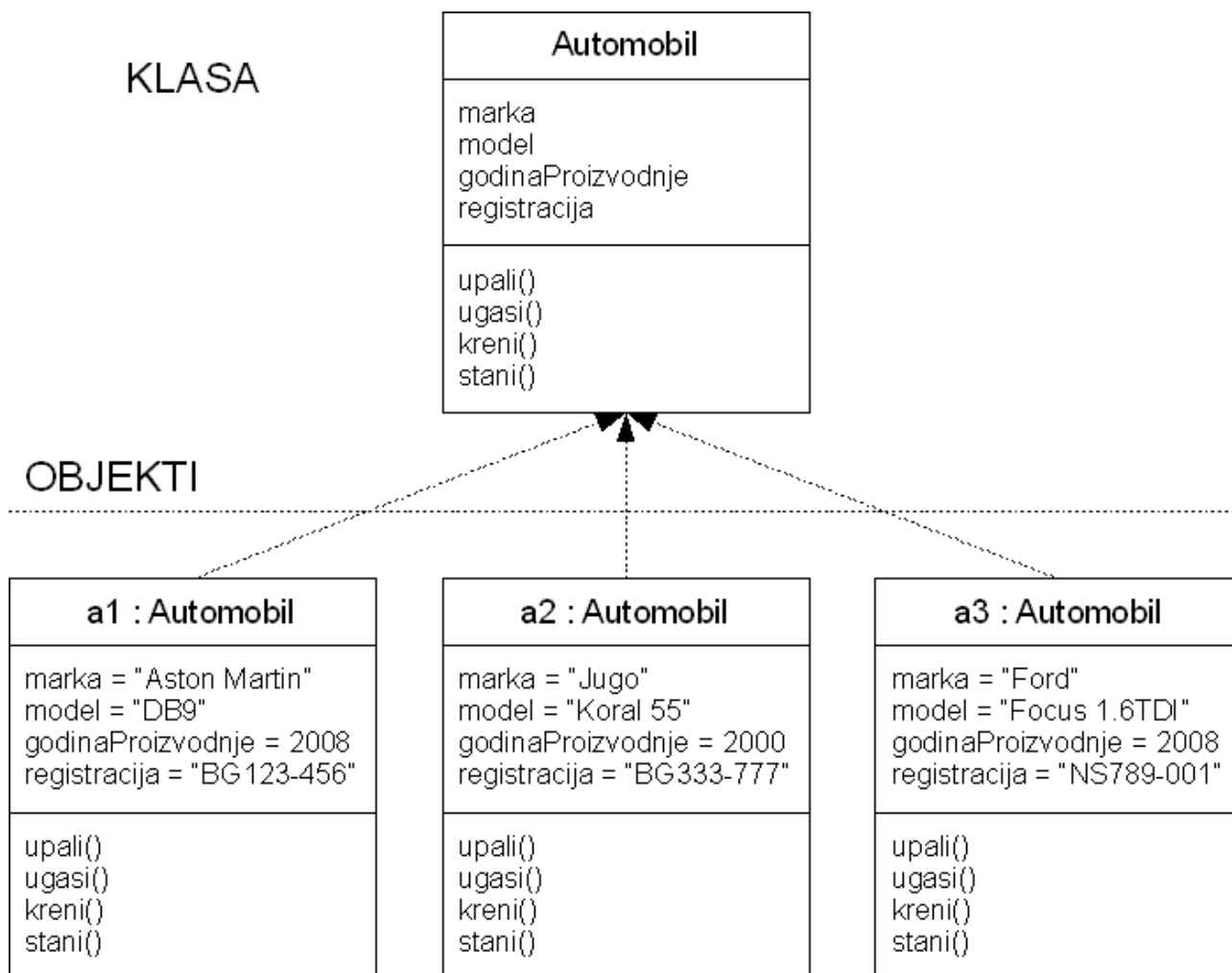
Objekti

Objekat predstavlja jedan konkretan primerak (instancu) neke klase. Prema tome, klasa može da se definiše kao skup objekata koji imaju iste osobine. Odnos klase i objekta je prikazan u sledećem primeru.

Primer 8

Automobil predstavlja klasu jer je to opšti nacrt nekih karakteristika i ponašanja koje svaki

automobil ima. Međutim, “Aston Martin DB9” sa godinom proizvodnje 2008 i registracijom “BG123-456” predstavlja jedan konkretan primerak automobila tj. objekat klase automobil. Odnos klase i objekta je prikazan na sledećoj slici (Slika 3).



Slika 3: Odnos klase i objekta

U Javi, objekti se deklarišu na sličan način kao i atributi klase. Prvo se navede naziv klase pa onda naziv konkretnog objekta. Na ovaj način se stvara promenljiva koja će da referencira konkretan objekat:

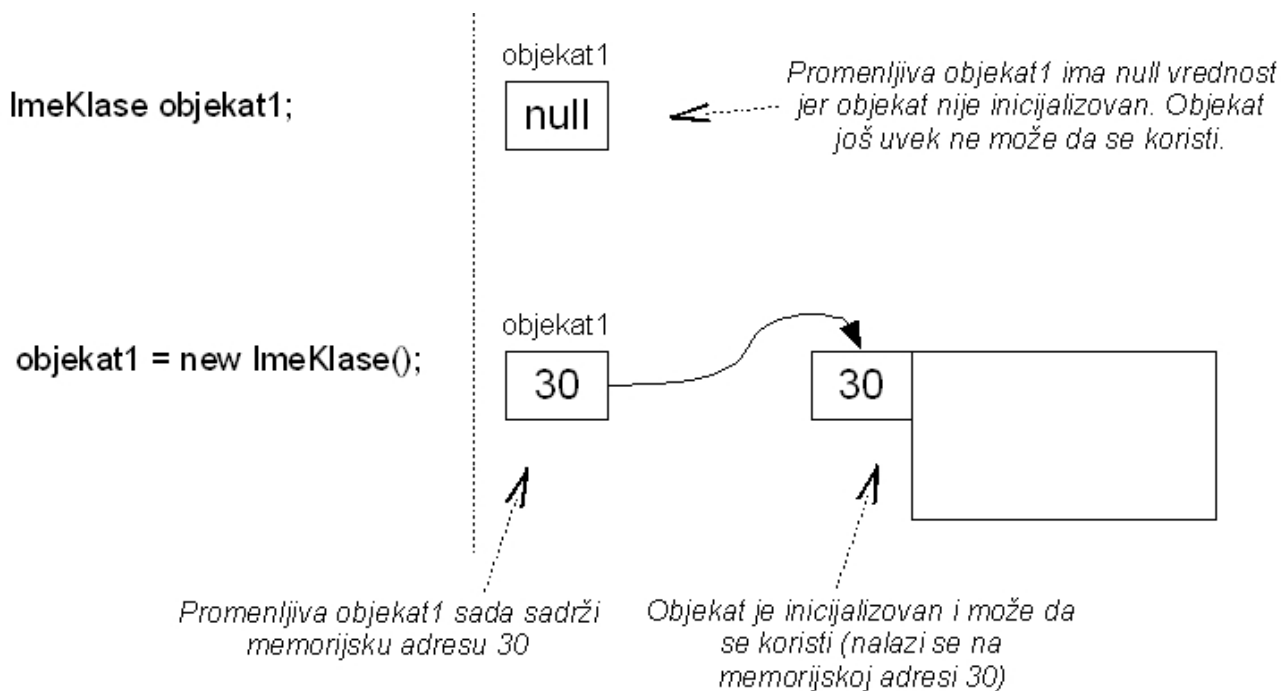
```
NazivKlase nazivobjekta;
```

Međutim, **da bi objekat mogao da se koristi (da se pozivaju njegove metode, menjaju vrednosti atributa itd.), potrebno ga je inicijalizovati**. Ako se objektu pokuša pristupiti bez inicijalizacije, Java javlja grešku. Inicijalizacija se vrši korišćenjem naredbe “new” na sledeći način:

```
nazivobjekta = new NazivKlase();
```

Šta se zapravo dešava kada se inicijalizuje objekat? Naime, kada se deklariše objekat (promenljiva), samo se stvori pokazivač koji će da referencira objekat. Ali, kada se izvrši inicijalizacija, tek se onda alokira deo memorije računara (deo RAM memorije) u kome će da bude objekat i poveže se sa pokazivačem - pokazivač tada sadrži adresu objekta u memoriji (Slika 4). Tek sada objekat može da

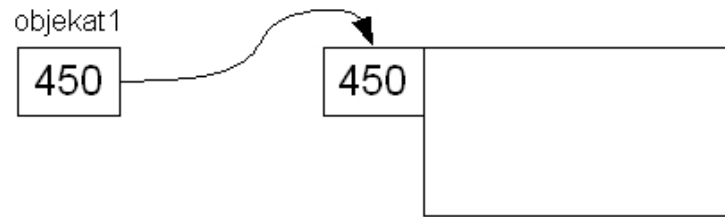
se koristi.



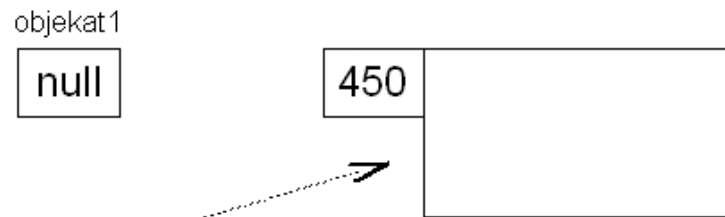
Slika 4: Inicijalizacija objekta

Bilo bi zanimljivo objasniti i kako se brišu objekti u Javi. Potpuno je normalna situacija da se u toku izvršavanja nekog ozbiljnijeg programa za kratko vreme stvori (inicijalizuje) na stotine ili hiljade objekata. Svaki od tih objekata zauzima neki deo memorije računara, a često se dešava da mnogi od njih uopšte više nisu potrebni, pa bi ih trebalo obrisati tj. izbaciti iz memorije. Ako se ovo ne uradi, može doći do preopterećenja memorije. Java ima mehanizam koji se zove **“garbage collection” (sakupljanje smeća)** koji vrši automatsko brisanje nepotrebnih objekata. Objekat je nepotreban ako na njega ne pokazuje ni jedna promenljiva niti pokazivač. Drugim rečima, dovoljno je promenljivoj (pokazivaču) dodeliti null vrednost ili inicijalizovati novi objekat preko iste promenljive pa da stari objekat bude automatski izbrisan posle nekog vremena (Slika 5).

`objekat1 = new ImeKlase();`

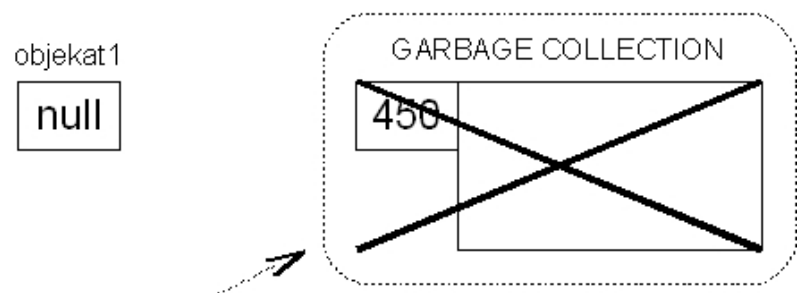


`objekat1 = null;`



Objekat je sada nepotreban jer pokazivač više ne pokazuje na njega

`//Posle nekog vremena...`



Posle nekog vremena, aktivira se garbage collection i briše objekat

Slika 5: Brisanje objekata u Javi ("garbage collection")

Sada je potrebno dati jedno objašnjenje: nijedan Java primer niti zadatak koji je prikazan do sad ne može da se pokrene (izvrši). **Da bi neki Java program mogao da se pokrene, potrebno je da ima tzv. "main" metodu.** Deklaracija ove metode je uvek ista i piše se u okviru tela klase na sledeći način:

```
public static void main (String[] args){  
  
    //naredbe...  
  
}
```

Primer 9

Napraviti klasu Motocikl koja ima:

- *atribut markaIModel.*
- *Atribut kubikaza (ceo broj).*

Napraviti klasu Test koja sadrži main metodu i, u okviru nje, pravi dva objekta klase Motocikl.

```

class Motocikl {

    String markaIModel;
    int kubikaza;

}

class Test {

    public static void main(String[] args) {

        Motocikl m1;
        Motocikl m2;

        m1 = new Motocikl();
        m2 = new Motocikl();

    }

}

```

Prava slika o objektima se stiće tek kada se vidi kako se menjaju vrednosti atributa. Pristup atributima se vrši tako što se napiše naziv objekta, pa tačka, pa naziv atributa:

```
nazivobjekta.nazivAtributa
```

Primer 10

Iskoristiti klasu Motocikl iz prethodnog primera i prepraviti klasu Test koja sadrži main metodu tako da pravi dva objekta klase Motocikl. Prvi bi trebalo da bude “Suzuki GS” od 500 kubika (dodeliti atributima prvog objekta ove vrednosti) a drugi “Yamaha R6” od 600 kubika (dodeliti atributima drugog objekta ove vrednosti).

```
//Kod za klasu Motocikl ostaje isti
```

```

class Test {

    public static void main(String[] args) {

        Motocikl m1;
        Motocikl m2;

        m1 = new Motocikl();
        m2 = new Motocikl();

        m1.markaIModel = "Suzuki GS";
        m1.kubikaza = 500;

        m2.markaIModel = "Yamaha R6";
        m2.kubikaza = 600;

    }

}

```

```

    }

}

```

U Javi se komande izvršavaju u redosledu u kojem su napisane (“odozgo na dole”). Ako se, na primer, u okviru main metode prvo napišu komande za pristup atributima, a tek posle sledi komanda za inicijalizaciju objekta, Java će da prijavi grešku.

Prethodna dva primera sadrže main metodu pa mogu da se izvrše. Međutim, rezultat njihovog izvršavanja neće da bude ništa vidljivo - objekti će da se inicijalizuju, dodeliće se vrednosti atributima, ali ništa neće da se ispiše na ekranu. Komanda za **ispisivanje na ekranu (u Javi se to zove “standardni izlaz”)** je sledeća:

```
System.out.println( ...neki tekst i/ili vrednost... );
```

Rezultat izvršavanja ove komande je ispis (na ekranu) sadržaja koji se nalazi između zagrada i prelazak u novi red. Taj sadržaj može da bude neki tekst, vrednost nekog atributa (ili promenljive) ili kombinacija ova dva. U sledećem primeru je prikazano šta ova komanda tačno radi i kako se koristi.

Primer 11

Sledi nekoliko primera upotrebe komande “println” za ispis na ekranu i rezultata koji će biti prikazani:

```
System.out.println("Lep dan danas");
//Ispisace se na ekranu:
//Lep dan danas

int broj;
broj = 12;
System.out.println(broj);
//Ispisace se na ekranu:
//12

System.out.println("Ova recenica ide u prvi red");
System.out.println("Ova recenica ide u drugi red");
//Ispisace se na ekranu:
//Ova recenica ide u prvi red
//Ova recenica ide u drugi red

int temperatura;
temperatura = 21;
System.out.println("Temperatura je: "+temperatura);
//Uloga znaka plus (+) je da omoguci nadovezivanje vrednosti
//temperature na poruku
//Ispisace se na ekranu:
//Temperatura je: 21

```

Primer 12

Iskoristiti klasu *Motocikl* iz prethodnih primera i klasu *Test* koja sadrži *main* metodu i pravi dva objekta klase *Motocikl* - “Suzuki GS” od 500 kubika i “Yamaha R6” od 600 kubika. Prepraviti *main* metodu klase *Test* tako da ispisuje vrednosti atributa oba objekta na ekranu.

```
//Kod za klasu Motocikl ostaje isti

class Test {

    public static void main(String[] args) {

        Motocikl m1;
        Motocikl m2;

        m1 = new Motocikl();
        m2 = new Motocikl();

        m1.markaIModel = "Suzuki GS";
        m1.kubikaza = 500;

        m2.markaIModel = "Yamaha R6";
        m2.kubikaza = 600;

        System.out.println(m1.markaIModel);
        System.out.println(m1.kubikaza);

        System.out.println(m2.markaIModel);
        System.out.println(m2.kubikaza);

        //Ispisace se na ekranu:
        //Suzuki GS
        //500
        //Yamaha R6
        //600

    }

}
```

Jedna od varijanti naredbe za ispis na ekranu je i “print” naredba. Jedina razlika u odnosu na “println” (“print-line”) naredbu je ta što se posle ispisa teksta ne prelazi u sledeći red već se i dalje ispisuje u postojećem redu.

```
System.out.print( ...neki tekst i/ili vrednost... );
```

Primer 13

Sledi nekoliko primera upotrebe komande “print” za ispis na ekranu i rezultata koji će biti prikazani:

```
System.out.print("Rec1");
System.out.print("Rec2");
```

```
//Ispisace se na ekranu:
//Rec1Rec2

int temperatura;
temperatura = 21;
System.out.print("Temperatura je: "+temperatura+" stepen ");
System.out.print("u Beogradu");
//Ispisace se na ekranu:
//Temperatura je 21 stepen u Beogradu
```

Zadaci

Zadatak 1

Napraviti klasu **Grad**. Ova klasa bi trebalo da ima:

- Atribut **naziv**. Početna vrednost ovog atributa je “nepoznat”
- Atribut **brojStanovnika**. Početna vrednost ovog atributa je 0.

Napraviti klasu TestGrad koja ima main metodu i u okviru nje kreira tri objekta klase Grad: Beograd (1.800.000 stanovnika), Njujork (20.000.000 stanovnika) i Vankuver (nepoznat broj stanovnika - ne dodeljivati nikakvu vrednost). Potrebno je ispisati vrednosti atributa svih objekata na ekranu.

Rešenje:

```
class Grad {

    String naziv = "nepoznat";
    int brojStanovnika = 0;

}

class TestGrad {

    public static void main(String[] args) {

        Grad g1 = new Grad();
        Grad g2 = new Grad();
        Grad g3 = new Grad();

        g1.naziv = "Beograd";
        g1.brojStanovnika = 1600000;

        g2.naziv = "Njujork";
        g2.brojStanovnika = 20000000;

        g3.naziv = "Vankuver";

        System.out.println(g1.naziv);
        System.out.println(g1.brojStanovnika);

        System.out.println(g2.naziv);
        System.out.println(g2.brojStanovnika);

        System.out.println(g3.naziv);
        System.out.println(g3.brojStanovnika);

        //Ispisace se na ekranu:
        //Beograd
        //1600000
```

```

        //Njujork
        //20000000
        //Vankuver
        //0
    }
}

```

Zadatak 2

Napraviti klasu **Osoba**. Ova klasa bi trebalo da ima:

- Atribut **imePrezime**. Početna vrednost ovog atributa je “nepoznato”
- Atribut **tezina**. Tezina može imati vrednosti 56.5 kg, 123.1 kg itd.

Napraviti klasu TestOsoba koja ima main metodu i u okviru nje kreira dva objekta klase Osoba: Pera Perić (81.6kg) i Mika Mikic (100.0kg). Prvo, potrebno je ispisati vrednosti atributa ovih objekata na ekranu. Posle, potrebno ih je ispisati ponovo ali tako da se uz ispis vrednosti atributa daje i neki propratni tekst, npr. “Ime osobe je: ” i “Tezina ove osobe je:”. Podatke svakog atributa bi trebalo ispisati u posebnom redu. Konačno, potrebno je ispisati vrednosti atributa oba objekta ali tako da se vrednosti svih atributa jednog objekta ispisuju u istom redu u formatu “Ime: ##### Tezina: #####”.

Rešenje:

```

class Osoba {

    String imePrezime = "nepoznato";
    double tezina;

}

class TestOsoba {

    public static void main(String[] args) {

        Osoba o1 = new Osoba();
        Osoba o2 = new Osoba();

        o1.imePrezime = "Pera Peric";
        o1.tezina = 81.6;

        o2.imePrezime = "Mika Mikic";
        o2.tezina = 100.0;

        System.out.println(o1.imePrezime);
        System.out.println(o1.tezina);

        System.out.println(o2.imePrezime);
        System.out.println(o2.tezina);
        //Ispisace se na ekranu:
        //Pera Peric
        //81.6
        //Mika Mikic
        //100.0

        System.out.println("Ime osobe je: "+o1.imePrezime);
        System.out.println("Tezina ove osobe je: "+o1.tezina);

        System.out.println("Ime osobe je: "+o2.imePrezime);
        System.out.println("Tezina ove osobe je: "+o2.tezina);
        //Ispisace se na ekranu:
    }
}

```

```

        //Ime osobe je: Pera Peric
        //Tezina ove osobe je: 81.6
        //Ime osobe je: Mika Mikic
        //Tezina ove osobe je: 100.0

        System.out.println("Ime: "+o1.imePrezime+" Tezina: "+o1.tezina);

        System.out.println("Ime: "+o2.imePrezime+" Tezina: "+o2.tezina);
        //Ispisace se na ekranu:
        //Ime: Pera Peric Tezina: 81.6
        //Ime: Mika Mikic Tezina: 100.0

    }

}

```

Metode

Već je rečeno da klase mogu imati i određena ponašanja i da se ta ponašanja izražavaju u formi **metoda**. Na primer, metode klase Automobil, mogu biti: upali, ugasi, kreni, stani itd.

Deklaracija metoda u Javi se vrši u okviru tela klase i to na sledeći način:

```

tip_povratne_vrednosti nazivMetode( ...ulazni parametri... ){

    //Telo metode

}

```

Tip povratne vrednosti metode, njen naziv i ulazni parametri čine **zaglavlje metode**, dok sav kod napisan između vitičastih zagrada čini **telo metode**. Telo metode počinje otvorenom vitičastom zagradom a završava se zatvorenom vitičastom zagradom.

Metode predstavljaju određeno ponašanje koje klasa ispoljava. Pored toga što nešto radi, metoda može i da **vraća neku vrednost kao rezultat izvršavanja (povratna vrednost metode)**. Na primer, metoda “saber” klase “Digitron”, može da vrati broj koji predstavlja rezultat sabiranja. Sa druge strane, neke metode nemaju povratnu vrednost. Primer jedne takve metode bi bila metoda “ispišiIme” klase “Osoba” koja na ekranu samo ispisuje ime osobe. **Kada metoda ne vraća nikakvu povratnu vrednost, tip povratne vrednosti je “void”, a kada vraća tip povratne vrednosti može biti bilo koji prost ili složen tip podataka (int, double, char, boolean, String...).** Metoda može da ima **samo jednu povratnu vrednost**.

Prema nepisanom pravilu, **nazivi metoda se pišu na isti način kao nazivi atributa (prvo slovo malo, ako ima više reči ostala prva slova velika)**: “saber”, “oduzmi”, “ispišiIme”, “konvertujValutu”...

Konačno, metode mogu da imaju ulazne parametre. **Ulazni parametri** predstavljaju one vrednosti koje je potrebno proslediti metodi da bi ona mogla da se pravilno izvrši. Na primer, metoda “saber” klase “Digitron” može da, kao ulazne parametre, ima dva broja koja je potrebno sabrati. Ako metoda nema ulazne parametre, prostor između zagrada se ostavlja prazan (piše se samo “()”). Ulazni parametri će detaljno biti objašnjeni u daljem tekstu.

Primer 14

Napraviti klasu **Televizor**. Ova klasa bi trebalo da ima:

- Atribut **jacinaTona** koji je ceo broj i označava trenutnu jačinu tona na televizoru. Početna vrednost ovog atributa je 0 (ton je utišan do kraja).
- Atribut **trenutniProgram** koji označava broj programa koji je trenutno na televizoru (npr. uključen je program 5). Početna vrednost ovog atributa je 1.
- Atribut **uključen** koji označava da li je televizor uključen ili nije (ako je uključen ima vrednost TRUE, a inače FALSE). Smatra se da je na početku televizor isključen.
- Metodu **uključi** koja uključuje televizor (postavlja vrednost atributa ukljucen na TRUE).
- Metodu **isključi** koja isključuje televizor (postavlja vrednost atributa ukljucen na FALSE).

```
class Televizor {  
  
    int jacinaTona = 0;  
    int trenutniProgram = 1;  
    boolean ukljucen = false;  
  
    void ukljuci() {  
        ukljucen = true;  
    }  
  
    void iskljuci() {  
        ukljucen = false;  
    }  
}
```

U ovom primeru, klasa **Televizor** ima samo dve metode - **ukljuci** i **iskljuci**. Metoda **ukljuci** nema povratnu vrednost jer samo menja vrednost atributa **ukljucen**. Prema tome, tip povratne vrednosti ove metode je **void**. Ova metoda nema ni ulazne parametre jer nisu potrebni (uvek se vrednost atributa postavlja na true) pa je prostor između običnih zagrada prazan. Telo metode **ukljuci** sadrži samo jednu komandu kojom se atributu **ukljucen** dodeljuje vrednost **true**. Sve navedeno važi i za metodu **iskljuci**.

Iz prethodnog primera se može primetiti i jedno pravilo: **atributima klase se iz tela bilo koje metode te iste klase pristupa samo preko naziva**. Drugim rečima, nije potrebno navoditi naziv objekta pa tačka pa naziv atributa kao kada se pristupa iz main metode.

Primer 15

Dodati u klasu **Televizor** i sledeće metode:

- Metodu **pojacakTon** koja povećava vrednost atributa **jacinaTona** za jedan.
- Metodu **smanjiTon** koja smanjuje vrednost atributa **jacinaTona** za jedan.
- Metodu **iskljuciTon** koja potpuno utišava ton (smanjuje vrednost **jacine tona** na 0).

Posle izmene, kod klase izgleda ovako:

```
class Televizor {  
  
    int jacinaTona = 0;
```



```

int trenutniProgram = 1;
boolean ukljucen = false;

void ukljuci(){
    ukljucen = true;
}

void iskljuci(){
    ukljucen = false;
}

void pojacajTon(){
    jacinaTona = jacinaTona + 1;
}

void smanjiTon(){
    jacinaTona = jacinaTona - 1;
}

void iskljuciTon(){
    jacinaTona = 0;
}
}

```

Metode *pojacajTon*, *smanjiTon* i *iskljuciTon* nemaju povratnu vrednost (ništa ne vraćaju kao rezultat), pa je tip povratne vrednosti *void*. Ove metode nemaju ulazne parametre.

Metode koje su navedene u dosadašnjim primerima ne vraćaju nikakvu vrednost kao rezultat. Međutim, postoje situacije u kojima je to potrebno uraditi - nekada je potrebno vratiti trenutnu vrednost atributa nekog objekta ili vratiti rezultat neke računске operacije. U tim slučajevima, se kao tip povratne vrednosti umesto *void* piše tip vrednosti podatka koji će biti vraćen - ako metoda vraća ceo broj - piše se *int*, ako metoda vraća realan broj - piše se *double*, ako vraća niz slova - piše se *String* itd. Pored toga, u okviru samog tela metode **se mora napisati komanda “return”** da označi vrednost koju je potrebno vratiti. Ovde je potrebno dati i jednu napomenu: **u trenutku kada se komanda “return” izvrši, izvršavanje metode se prekida**. Drugim rečima, bilo koja komanda napisana “posle” komande “return” u okviru tela metode se nikad neće izvršiti.

Primer 16

Dodati u klasu *Televizor* i sledeće metode:

- Metodu ***promeniProgramNavise*** koja povećava vrednost atributa *trenutniProgram* za jedan.
- Metodu ***promeniProgramNanize*** koja smanjuje vrednost atributa *trenutniProgram* za jedan.
- Metodu ***vратиTrenutniProgram*** koja vraća vrednost atributa *trenutniProgram*.
- Metodu ***vратиJacinuTona*** koja vraća trenutnu vrednost atributa *jacinaTona*.
- Metodu ***daLiJeUkljucen*** koja vraća trenutnu vrednost atributa *ukljucen*.
- Metodu ***ispisiParametre*** koja ispisuje na ekranu trenutne vrednosti svih atributa televizora uz odgovarajuću poruku.

Posle svih izmena, kod klase izgleda ovako:

```

class Televizor {

    int jacinaTona = 0;
    int trenutniProgram = 1;
    boolean ukljucen = false;

    void ukljuci(){
        ukljucen = true;
    }

    void iskljuci(){
        ukljucen = false;
    }

    void pojačajTon(){
        jacinaTona = jacinaTona + 1;
    }

    void smanjiTon(){
        jacinaTona = jacinaTona - 1;
    }

    void iskljuciTon(){
        jacinaTona = 0;
    }

    void promeniProgramNavise(){
        trenutniProgram = trenutniProgram + 1;
    }

    void promeniProgramNanize(){
        trenutniProgram = trenutniProgram - 1;
    }

    int vratiTrenutniProgram(){
        return trenutniProgram;
    }

    int vratiJacinuTona(){
        return jacinaTona;
    }

    boolean daLiJeUkljucen(){
        return ukljucen;
    }

    void ispisiParametre(){
        System.out.println("Jacina tona je "+jacinaTona);
        System.out.println("Trenutni program je "+trenutniProgram);
        System.out.println("Televizor je ukljucen "+ukljucen);
    }
}

```

```
}
```

```
}
```

Metode `promeniProgramNavise` i `promeniProgramNanize` nemaju nikakve povratne vrednosti. Međutim, metode `vratiTrenutniProgram`, `vratiJacinuTona` i `daLiJeUkljucen` imaju. Metoda `vratiTrenutniProgram` bi trebalo da vrati trenutnu vrednost atributa `program`. Praktično gledano, to bi značilo da vrati broj programa koji se trenutno prikazuje na televizoru (na primer broj 5). Pošto je u pitanju ceo broj (atribut `trenutniProgram` je tipa `int`), i povratna vrednost metode će biti `int`. Telo metode `vratiTrenutniProgram` sadrži samo komandu `return` kojom se označava vrednost koju je potrebno vratiti. Primera radi, recimo da je u okviru tela ove metode napisana komanda:

```
return 2;
```

Metoda bi u ovom slučaju uvek vraćala broj 2. Metoda `vratiJacinuTona` je veoma slična prethodnoj metodi i dovoljno je reći da i ona vraća `int` vrednost. Ali, metoda `daLiJeUkljucen` vraća vrednost atributa `ukljucen`, pa je tip povratne vrednosti `boolean`.

Konačno, metoda `ispisiParametre` samo vrši ispis vrednosti svih atributa na ekranu uz odgovarajuću poruku. Tip povratne vrednosti je `void` jer ispisivanje na ekranu nije isto što i vraćanje povratnih vrednosti.

Postavlja se pitanje, a kako se metode pozivaju? Odgovor je sličan kao i za attribute. Da bi metode mogle da se pozovu, potrebno je napraviti objekat, pa tek onda pozvati neku njegovu metodu preko naziva klase i naziva metode. Razlika u odnosu na poziv atributa je u tome što se **pri pozivu metode posle naziva metode obavezno moraju napisati i zagrade** (čak i ako metoda nema ulazne parametre):

```
nazivobjekta.nazivMetode();
```

Ako metoda ima povratnu vrednost, prvo je potrebno deklarirati promenljivu koja će da primi povratnu vrednost metode, pa tek onda pozvati metodu. **Tip promenljive mora da bude isti kao tip povratne vrednosti metode:**

```
tip_promenljive naziv_promenljive;
```

```
naziv_promenljive = nazivobjekta.nazivMetode();
```

Pre nego što se pređe na konkretan primer, potrebno je razjasniti šta je promenljiva i koja je razlika između promenljive i atributa. **Promenljiva** predstavlja mesto u memoriji računara u koje se može smestiti neka vrednost. Tip promenljive određuje tip vrednosti koja se može uneti u promenljivu. Na primer, promenljiva tipa `int` predstavlja neki deo memorije računara u koji se može smestiti neka celobrojna vrednost. **Atributi su promenljive koje direktno pripadaju klasi tj. deklarišu se u okviru tela klase.** Sve ostale promenljive koje se deklarišu u okviru tela metoda uključujući i ulazne parametre se nazivaju samo promenljive.

Primer 17

Iskoristiti klasu ***Televizor*** iz prethodnog primera. Napraviti klasu ***TestTelevizor*** koja kreira jedan objekat klase *Televizor* i poziva neke od njegovih metoda. Posle svakog poziva metode, pozvati

metodu ispisiParametre i uočiti promene u vrednostima atributa.

```
class TestTelevizor {

public static void main (String[] args){

    Televizor t = new Televizor();
    boolean uklj;
    int prog;

    t.ispisiParametre();
    //Ispisace se na ekranu
    //Jacina tona je 0
    //Trenutni program je 1
    //Televizor je ukljucen false

    t.ukljuci();
    t.ispisiParametre();
    //Ispisace se na ekranu
    //Jacina tona je 0
    //Trenutni program je 1
    //Televizor je ukljucen true

    t.pojacajTon();
    t.ispisiParametre();
    //Ispisace se na ekranu
    //Jacina tona je 1
    //Trenutni program je 1
    //Televizor je ukljucen true

    t.promeniProgramNavise();
    t.ispisiParametre();
    //Ispisace se na ekranu
    //Jacina tona je 1
    //Trenutni program je 2
    //Televizor je ukljucen true

    uklj = t.daLiJeUkljucen();
    System.out.println("Televizor je ukljucen: "+uklj);
    //Ispisace se na ekranu
    //Televizor je ukljucen: true

    prog = t.vratiTrenutniProgram();
    System.out.println("Na televizoru ide program "+prog);
    //Ispisace se na ekranu
    //Na televizoru ide program 2

    t.ispisiParametre();
    //Ispisace se na ekranu
    //Jacina tona je 1
    //Trenutni program je 2
```

```

        //Televizor je ukljucen true
    }

}

```

Kada se poziva neka metoda koja nema povratnu vrednost (na primer metoda `ispisiParametre`), poziv je sličan kao za atribut objekta. Jedina razlika je u tome što se posle naziva metode pišu i zagrade u kojima se navode konkretne vrednosti za ulazne parametre. Ako metoda nema parametre, prostor između zagrada ostaje prazan ("()"):

```
t.ispisiParametre();
```

Poziv metode koja ima neku povratnu vrednost je malo drugačiji. To je slučaj sa metodama `daLiJeUkljucen` i `vратиTrenutniProgram`. U primeru se vidi da je deklarirana promenljiva "uklj" koja je tipa boolean (isto kao tip povratne vrednosti metode `daLiJeUkljucen`) i koja prima povratnu vrednost metode:

```
uklj = t.daLiJeUkljucen();
```

Kada ova promenljiva dobije povratnu vrednost metode, ta povratna vrednost se može koristiti za neka dalja izračunavanja, može se ispisati na ekranu itd.

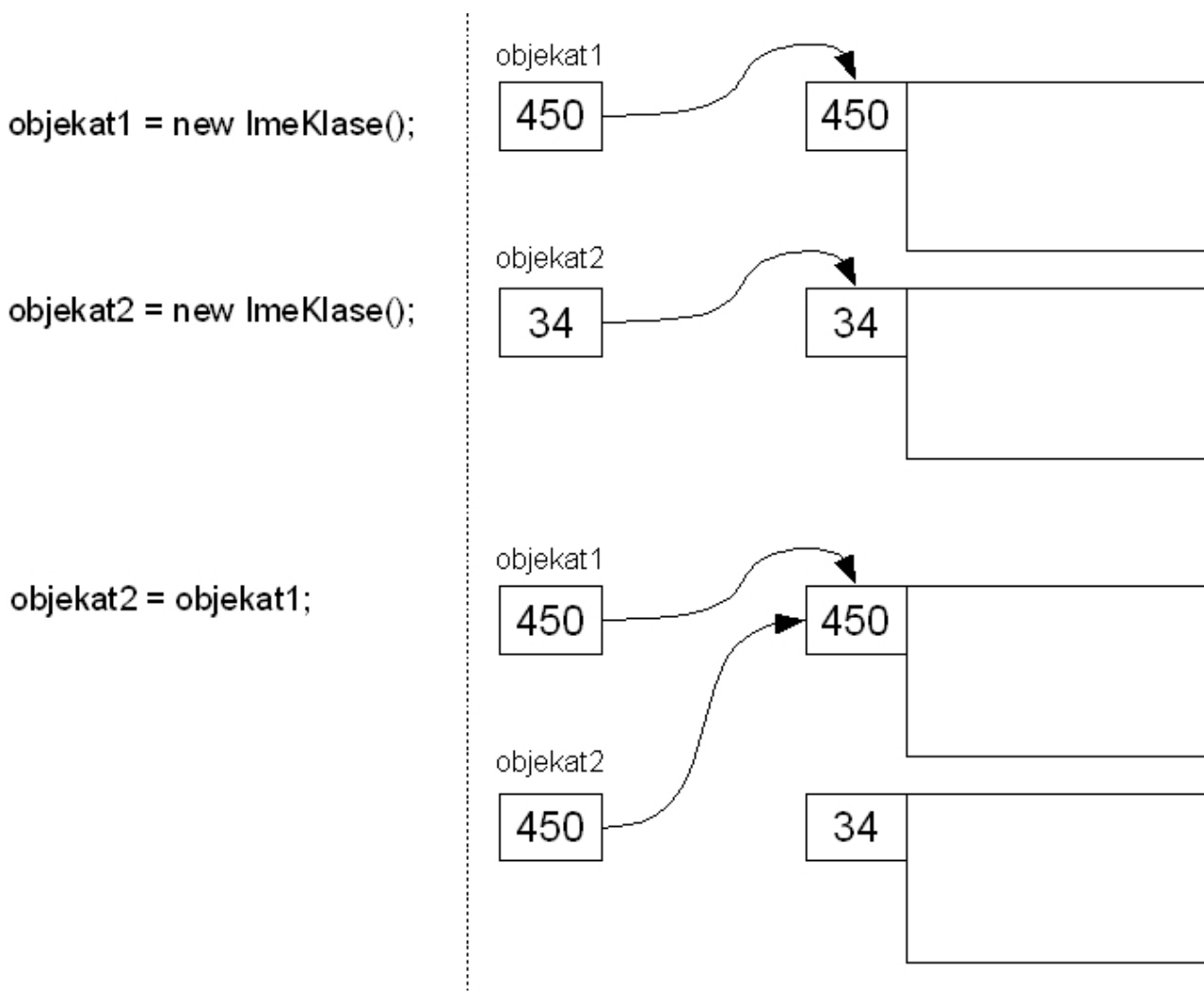
U mnogim dosadašnjim primerima i zadacima je korišćen jedan operator. U pitanju je **operator za dodelu vrednosti**. Formalni opis i par primera u vezi sa ovim operatorom su dati u sledećoj tabeli.

| Operator za dodelu vrednosti | | |
|------------------------------|------------------------------|---|
| Operator | Opis | Primeri |
| = | Operator za dodelu vrednosti | <pre> a = 2; m1.kubikaza = 750; osoba1.ime = "Pera"; logickiUslov = true; ind = obj.metoda1(); objekat1 = obekat2; </pre> |

U suštini, ovaj operator omogućava da se nekoj promenljivoj, atributu ili objektu dodeli neka vrednost. Uvek se onome što se nalazi na levoj strani znaka jednakosti dodeljuje ono što se nalazi na desnoj strani znaka jednakosti. Na desnoj strani znaka jednakosti može da bude neka konkretna vrednost, promenljiva, atribut, neki izraz, poziv metode ili objekat. Jedino je bitno da **ono što je sa leve strane znaka jednakosti bude istog tipa kao i ono sa desne strane** (npr. nije moguće dodeliti vrednost 23.44 promenljivoj int tipa).

Kada se operator za dodeljivanje vrednosti koristi u kombinaciji sa atributima, promenljivim i metodama, stvar je relativno jednostavna. Levoj strani se uvek dodeljuje vrednost sa desne strane. Međutim, ako se sa leve strane nalazi objekat i pokušava mu se dodeliti drugi objekat, izvršiće se **dodeljivanje adrese desnog objekta levom objektu**, pa će oba pokazivača (promenljive) pokazivati na istu memorijsku lokaciju (Slika 6). Memorijska lokacija na koju više ne pokazuje nijedan pokazivač će ubrzo da bude oslobođena preko "garbage collection" mehanizma.

Dodatni efekat koji se u ovom slučaju pojavljuje je taj da se sada preko oba pokazivača može pristupiti istoj memorijskoj lokaciji (u kojoj je objekat). Ovo se može videti i na jednom praktičnom primeru.



Slika 6: Dodeljivanje vrednosti objekata

Primer 18

Napisati klasu **Proizvod** koja ima samo jedan atribut - naziv i jednu metodu “ispisi” koja ispisuje vrednost ovog atributa na ekranu. Napraviti klasu **TestProizvod** koja kreira dva objekta klase **Proizvod** sa nazivima “automobil” i “traktor”. Ispisati vrednosti atributa oba proizvoda. Dodeliti drugom objektu adresu prvog objekta. Ponovo ispisati vrednosti atributa oba proizvoda. Izmeniti naziv drugog proizvoda u “kombajn” i još jednom ispisati vrednosti atributa oba proizvoda.

```
class Proizvod {  
    String naziv;  
  
    void ispisSve() {  
        System.out.println("Naziv je: "+naziv);  
    }  
}
```

```

}

class TestProizvod {

    public static void main(String[] args) {

        Proizvod p1 = new Proizvod();
        Proizvod p2 = new Proizvod();

        p1.naziv = "automobil";
        p2.naziv = "traktor";

        p1.ispisiSve();
        p2.ispisiSve();
        //Ispisace se na ekranu
        //Naziv je: automobil
        //Naziv je: traktor

        p2=p1;

        p1.ispisiSve();
        p2.ispisiSve();
        //Ispisace se na ekranu
        //Naziv je: automobil
        //Naziv je: automobil

        p2.naziv="kombajn";

        p1.ispisiSve();
        p2.ispisiSve();
        //Ispisace se na ekranu
        //Naziv je: kombajn
        //Naziv je: kombajn
    }

}

```

U trenutku kada se objektu p2 (pokazivaču) dodeli objekat p1, oba pokazivača se referenciraju na isti deo memorije. Od tada, sve promene koje se urade nad objektom p2 će važiti i za objekat p1 i obrnuto.

Ulazni parametri metode predstavljaju neke vrednosti koje je potrebno proslediti metodi da bi ona mogla pravilno da se izvrši. Metoda može da ima nula, jedan ili više ulaznih parametara, a parametri se deklarišu u okviru zaglavlja metode na sličan način kao i promenljive (prvo tip vrednosti, pa naziv parametra). Ako metoda ima više parametara, oni se odvajaju zarezom:

```

tip_povratne_vrednosti nazivMetode(tip_parametra naziv_par) {

    //Telo metode

}

```

Pozivanje metoda koje imaju ulazne parametre se vrši na taj način što se između zagrada za ulazne parametre navedu neke konkretne vrednosti ili promenljive. **U okviru poziva ovakve metode mora da se ispoštuje broj parametara i njihov redosled.** Drugim rečima, ako metoda prima dva ulazna parametra nekog tipa, svaki poziv te metode mora da bude takav da se metodi zaista i proslede tačno dva parametra tog tipa.

Primer 19

Napraviti klasu **AutomatNovca**. Ova klasa bi trebalo da ima:

- atribut **stanje** koji predstavlja trenutni iznos novca u automatu (realan broj). Početna vrednost ovog atributa je 5200.0 dinara.
- Metodu **podigniIznos** koja prima kao ulazni parametar iznos novca koji korisnik želi da podigne (realan broj npr. 550.5) i smanjuje vrednost atributa stanje za taj iznos.
- Metodu **uloziIznos** koja prima kao ulazni parametar iznos novca koji korisnik želi da uloži (realan broj) i povećava vrednost atributa stanje za taj iznos.
- Metodu **vратиStanje** koja vraća trenutni iznos novca u automatu (vrednost atributa stanje).
- Metodu **ispisiStanje** koja na ekranu ispisuje koja je trenutna količina novca u automatu (vrednost atributa stanje).

Napraviti klasu **ProveraAutomataNovca** koja kreira dva objekta klase **AutomatNovca**. U prvi automat novca je potrebno uložiti 1002.03 dinara i ispisati stanje pre i posle ulaganja. Potrebno je i podići 234.55 dinara iz drugog automata i ispisati stanje automata pre i posle ulaganja.

```
class AutomatNovca {  
  
    double stanje=5200.0;  
  
    void podigniIznos (double iznos){  
        stanje = stanje - iznos;  
    }  
  
    void uloziiIznos (double iznos){  
        stanje = stanje + iznos;  
    }  
  
    double vratiStanje (){  
        return stanje;  
    }  
  
    void ispisiiStanje (){  
        System.out.println("Trenutni iznos u automatu je: "+stanje);  
    }  
  
}  
  
class ProveraAutomataNovca {  
  
    public static void main (String[] args){
```



```

AutomatNovca a1 = new AutomatNovca();
AutomatNovca a2 = new AutomatNovca();

a1.ispisiStanje();
a1.uloziiIznos(1002.03);
a1.ispisiStanje();
//Ispisace se na ekranu
//Trenutni iznos u automatu je: 5200.0
//Trenutni iznos u automatu je: 6202.03

a2.ispisiStanje();
a2.podigniIznos(234.55);
a2.ispisiStanje();
//Ispisace se na ekranu
//Trenutni iznos u automatu je: 5200.0
//Trenutni iznos u automatu je: 4965.45
}

}

```

Metode podigniIznos i uloziiIznos imaju po jedan ulazni parametar. U oba slučaja, u pitanju su realni brojevi, pa je tip ulaznog parametra double. Naziv ulaznog parametra je uvek proizvoljan, samo je bitno obratiti pažnju na to da se taj naziv koristi u okviru tela metode kada se želi upotrebiti ulazni parametar. Ove dve metode nemaju povratnu vrednost jer samo menjaju vrednost atributa stanje.

U programskom jeziku Java je moguće pisati i razne **aritmetičke izraze**. Neki od prethodnih primera već sadrže primere korišćenja operatora za sabiranje i oduzimanje, ali se kompletna lista aritmetičkih operatora nalazi u sledećoj tabeli.

| <i>Aritmetički operatori</i> | | |
|------------------------------|--|--------------------------------------|
| Operator | Opis | Primeri |
| + | Sabiranje | a + b 12 + 5 m1.kubikaza + 300 |
| - | Oduzimanje | a - b 12 - 5 m1.kubikaza - 300 |
| * | Množenje | a * b 12 * 5 m1.kubikaza * 300 |
| / (celi brojevi) | Celobrojno deljenje (DIV) | 2 / 12 a / b |
| / (realni brojevi) | Deljenje | a / 12.0 5.0 / 123.44 |
| % | Ostatak pri celobrojnem deljenju (MOD) | 5 % 2 a % 10 |
| ++ | Uvećanje za 1 (inkrementacija) | a++ m1.kubikaza++ |
| -- | Umanjenje za 1 (dekrementacija) | a-- m1.kubikaza-- |

Operatori za sabiranje, množenje i oduzimanje se koriste na sličan način kao što bi se koristili u matematici. Ovi operatori mogu da se koriste i sa celobrojnim i realnim vrednostima i promenljivim. Java omogućava i korišćenje zagrada pa su uobičajeni i izrazi kao:

```
b * 2 * ((234.55 + 20) - (a * 3.4))
```

Operatori za inkrementaciju i dekrementaciju uvećavaju tj. smanjuju vrednost neke celobrojne promenljive za jedan. Uvedeni su radi pogodnosti, pa su sledeća dva para izraza potpuno ekvivalentni:

```
a = a + 1;
b = b - 1;
```

```
//Je isto sto i...
```

```
a++;
b--;
```

Operatori deljenja su u Javi malo složeniji nego što se na prvi pogled može učiniti. Prvo, i operator za deljenje realnih brojeva i operator za celobrojno deljenje se označavaju istim znakom - kosom crtom ("/"). Deljenje realnih brojeva je "normalno deljenje" (na primer $10.0 / 3 = 3.33333$). Celobrojno deljenje, sa druge strane, predstavlja deljenje bez ostatka ($10 / 3 = 3$). Pri ovom deljenju se odseca deo koji se nalazi iza zareza i kao rezultat se vraća ceo broj. Sa obzirom na to da se oba operatora isto pišu, postavlja se pitanje kako Java razlikuje kada se misli na jedan a kada na drugi

operator. Odgovor je veoma jednostavan - vrši se provera da li su brojevi (promenljive) koji se dele celi ili realni i primenjuje se odgovarajući operator. Ako su oba broja cela, primenjuje se celobrojno deljenje. Ako je makar jedan od brojeva realan, primenjuje se deljenje realnih brojeva.

Operator za izračunavanje ostatka pri celobrojnem deljenju se može koristiti isključivo sa celobrojnim vrednostima. Rezultat koji se dobije predstavlja ceo broj koji ostaje kao višak pri deljenju dva cela broja.

Operatore deljenja i ostatka pri deljenju je najbolje objasniti preko nekoliko jednostavnih primera.

Primer 20

Sledeći izrazi će u Javi biti protumačeni kao izrazi deljenja realnih brojeva:

```
double x = 3.5;
double y = 7;
double z;

z = y / x;
//Rezultat je 2.0
//I x i y su realni brojevi.

z = y / 7;
//Rezultat je 1.0
//Dovoljno je da je jedan od bojeva realan - y

z = 35.0 / 7;
//Rezultat je 5.0
//Dovoljno je da je jedan od bojeva realan - 35.0

z = 35 / 7.0;
//Rezultat je 5.0
//Dovoljno je da je jedan od bojeva realan - 7.0
```

Međutim, ovi izrazi će biti protumačeni kao celobrojno deljenje bez ostatka:

```
int a = 2;
int b = 12;
int c;
double r;

c = b / a;
//Rezultat je 6 (ne 6.0 nego samo 6)
//I a i b su celobrojne promenljive

c = b / 4;
//Rezultat je 3
//b je celobrojna promenljiva

c = 123 / 12;
//Rezultat je 10 (odseca se deo iza zareza)
```

```
//I 123 i 12 su celi brojevi
```

```
r = 123 / 12;
```

```
//Rezultat je 10.0 (rezultat 10 se pretvara u realan broj 10.0)
```

```
//I 123 i 12 su celi brojevi pa je pozvano celobrojno deljenje
```

Kao što se iz poslednjeg primera vidi, moguća je situacija i da se zbog nepažnje, pri deljenju dobiju neki neočekivani rezultati. Uzrok je uvek to što se, zbog pogrešno napisanog izraza, pozove operator za deljenje celih brojeva umesto operatora za deljenje realnih brojeva. Da bi prethodni izraz bio protumačen kao deljenje realnih brojeva, potrebno je napisati ga na jedan od sledećih načina:

```
r = 123.0 / 12;
```

```
r = 123.0 / 12.0;
```

```
r = 123 / 12.0;
```

```
//Rezultat je 10.25
```

```
//Bitno je da je makar jedan od brojeva realan
```

Evo i nekoliko primera korišćenja operatora za izračunavanje ostatka pri celobrojnomo deljenju:

```
c = 5 % 2;
```

```
//Rezultat je 1 (jer je 5 = 2*2 + 1)
```

```
c = 10 % 2;
```

```
//Rezultat je 0 (jer je 10 = 5*2 i nema ostatka)
```

```
c = 14 % 4;
```

```
//Rezultat je 2 (jer je 14 = 3*4 + 2)
```

```
c = 55 % 5;
```

```
//Rezultat je 0 (jer je 55 = 11*5 i nema ostatka)
```

Ovaj operator može da se koristi da se proveriti da li je neki broj paran jer će ostatak pri deljenju sa brojem 2 uvek biti nula ako jeste paran a 1 ako nije paran. Ovako se može proveriti i deljivost sa nekim drugim brojevima (ostatak je nula ako je broj deljiv a ima neku drugu vrednost ako nije).

Da bi se u potpunosti shvatio način funkcionisanja metoda, potrebno je uvesti i objasniti pojmove **vidljivosti promenljivih** i **načina prenosa ulaznih parametara**.

Svi ulazni parametri metode i promenljive koje se deklariraju u okviru tela metode su “vidljivi” (mogu se pozvati) isključivo u okviru tela te metode i nigde druge. Svi atributi klase su uvek vidljivi u okviru tela metoda te klase.

Primer 21

Neka klasa **Kalkulator** ima dve metode - *saberi* i *obimKrug*. Prva metoda dobija dva cela broja kao ulazne parametre i vraća rezultat sabiranja, dok druga prima poluprečnik kruga kao ulazni parametar i vraća obim kruga kao rezultat. Klasa ima i atribut *pi* koji predstavlja vrednost

odgovarajuće matematičke konstante.

```
class Kalkulator {  
  
    double pi = 3.14;  
  
    int saberi(int a, int b) {  
        int rezultat;  
        rezultat = a + b;  
        return rezultat;  
    }  
  
    double obimKrug(a(double poluprecnik) {  
        double rezultat;  
        rezultat = 2* poluprecnik * pi;  
        return rezultat;  
    }  
  
}
```

Pi je atribut klase Kalkulator, pa se bez problema može pozvati i koristiti u okviru tela bilo koje metode klase Kalkulator (koristi se u okviru tela metode obimKrug(a)). Promenljive a i b su, sa druge strane, ulazni parametri prve metode. To znači da se oni mogu koristiti i pozivati isključivo u okviru tela ove metode. Ako bi se pokušalo sa pozivanjem promenljivih a i b iz tela metode obimKrug(a), javila bi se greška. Isto važi i za ulazni parametar poluprecnik - “vidljiv” je samo u okviru tela metode obimKrug(a) i nigde više.

Površnim pregledanjem koda bi se moglo zaključiti da u kodu postoji konflikt jer se deklariraju dve promenljive sa nazivom rezultat - jedna u okviru tela prve a druga u okviru tela druge metode. To, međutim, nije tako. U Javi se ove dve promenljive posmatraju kao potpuno različite jer promenljiva rezultat iz metode saberi ima vidljivost samo u okviru tela metode saberi, dok promenljiva rezultat iz metode obimKrug(a) ima vidljivost samo u okviru tela metode obimKrug(a).

Konflikt u “vidljivosti” parametara i atributa se može javiti ako se u okviru tela metode koristi ulazni parametar ili promenljiva koji imaju isti naziv kao i neki atribut klase. U tom slučaju se koristi **ključna reč “this”** da izdvoji atribut u odnosu na druge promenljive. Ova ključna reč je u stvari pokazivač objekta na samog sebe.

Primer 22

Neka klasa **Osoba** ima atribut ime i metodu unesiIme koja kao ulazni parametar prima neko ime i atributu dodeljuje tu vrednost. Neka se ulazni parametar ove metode takođe zove “ime”

```
class Osoba {  
  
    String ime;  
  
    void unesiIme(String ime) {  
        ime = ime;  
    }  
  
}
```

```
}
```

Prvo što se može primetiti je da je komanda dodeljivanja vrednosti atributu *ime* veoma nejasna. Šta je atribut a šta ulazni parametar? I atribut se naziva "ime" a i ulazni parametar. Prema tome, koji je efekat izvršavanja naredbe:

```
ime = ime;
```

Odgovor je u tome da će Java da protumači obe strane kao ulazni parametar, tako da se dodeljivanje vrednosti atributu *ime* uopšte neće ni desiti. Da bi metoda radila ono što je potrebno, koristi se ključna reč "this" da označi atribut. Posle prepravki, metoda bi trebalo da izgleda ovako:

```
void unesiIme(String ime) {  
    this.ime = ime;  
}
```

Izraz sa leve strane znaka jednakosti se odnosi na atribut *ime*, dok se izraz sa desne strane znaka jednakosti odnosi na ulazni parametar *ime*.

Ulazni parametri metoda mogu biti prosti tipovi podataka (int, double, boolean, char) ali i složeni tipovi (klase). Drugim rečima, metodi se kao parametar može proslediti i neki objekat. Pored toga, metoda može i da, kao rezultat, vrati neki objekat.

Kada se metodi prosleđuju vrednosti prostog tipa podataka vrši se **prenos parametra preko vrednosti**. To praktično znači da se vrednost ulaznog parametra kopira i u okviru tela metode se koristi njegova kopija. Krajnji efekat je da se bilo kakve promene izvršene nad ulaznim parametrom u toku izvršavanja metode neće zapamtiti (jer su promene vršene nad kopijom a ne nad originalom). Kada se metoda završi, kopija se briše.

Kada se metodi prosleđuju objekti kao ulazni parametri, vrši se **prenos parametra preko adrese**. U ovom slučaju se ulazni parametar (objekat) ne kopira već se prenosi samo njegova adresa u memoriji. Posledica ovakvog pristupa je ta da se bilo kakve promene nad ovim objektom izvršene u okviru tela metode pamte i posle kraja izvršavanja metode (promene se vrše nad originalom).

Primer 23

Neka je data klasa **Brojevi** koja ima attribute *a* i *b* koji su celi brojevi. Neka je data i klasa **ZamenaBrojeva** koja im dve metode. Prva metoda prima kao ulazne parametre dva cela broja i pokušava da im zameni mesta. Druga metoda kao ulazni parametar prima objekat klase **Brojevi** i menja mesta vrednostima atributa *a* i *b*. Klasa **TestZamenaBrojeva** poziva obe metode klase **ZamenaBrojeva** i ispisuje rezultate na ekranu.

```
class Brojevi {  
    int a;  
    int b;  
}  
  
class ZamenaBrojeva {  
    void zameniMesta(int a, int b) {
```

```

        int pomocna;
        pomocna = a;
        a = b;
        b = pomocna;
    }

    void zameniMesta2(Brojevi br) {
        int pomocna;
        pomocna = br.a;
        br.a = br.b;
        br.b = pomocna;
    }
}

class TestZamenaBrojeva {

    public static void main(String[] args) {

        int a = 10;
        int b = 12;

        Brojevi br = new Brojevi();
        br.a = 44;
        br.b = 100;

        ZamenaBrojeva zb = new ZamenaBrojeva();

        System.out.println ("Pre promene a = "+a+" b = "+b);
        zb.zameniMesta(a, b);
        System.out.println ("Posle promene a = "+a+" b = "+b);
        //Ispisace se na ekranu
        //Pre promene a = 10 b = 12
        //Posle promene a = 10 b = 12

        System.out.println ("Pre promene br.a = "+br.a+" br.b = "+br.b);
        zb.zameniMesta2(br);
        System.out.println ("Posle promene br.a = "+br.a+" br.b = "+br.b);
        //Ispisace se na ekranu
        //Pre promene br.a = 44 br.b = 100
        //Posle promene br.a = 100 br.b = 44
    }
}

```

Kada se izvrši prva metoda, brojevi neće biti zamenjeni jer su uneti kao prosti parametri i preneti preko vrednosti. Međutim, pozivom metode zameniMesta2, zamena vrednosti atributa a i b će da se izvrši i ostaće upamćena jer je ceo objekat prenet kao parametar preko adrese.

Nazivi atributa u okviru jedne klase moraju da budu jedinstveni - ne mogu se pojaviti dva atributa sa istim nazivom. Sa metodama, to nije slučaj. Naime, moguće je napisati dve ili više metoda sa istim nazivom u okviru jedne klase. Ali, da bi se razlikovale, svaka od njih mora imati drugačiju listu ulaznih parametara - bilo da je u pitanju broj ulaznih parametara ili njihov tip. Pri pozivu metode, pozvaće se ona metoda čiji ulazni parametri tačno odgovaraju unetim vrednostima. Ova pojava se naziva **preklapanje metoda** ("overloading").

Primer 24

*Napraviti klasu **Kalkulator** koja ima dve metode sa istim nazivom - saberi. Prva metoda prima dva*

cela broja kao ulazne parametre i vraća njihov zbir (ceo broj). Druga metoda prima dva realna broja kao ulazne parametre i vraća njihov zbir (realan broj). Napisati i klasu *TestKalkulator* koja poziva obe metode.

```
class Kalkulator {

    int saberi (int x, int y){
        int rezultat = x+y;
        return rezultat;
    }

    double saberi (double x, double y){
        double rezultat = x+y;
        return rezultat;
    }

}

class TestKalkulator {

    public static void main(String[] args) {

        Kalkulator k = new Kalkulator();

        double r = k.saberi(12.0, 15.0);
        System.out.println(r);
        //Ispisace se na ekranu
        //27.0

        int r2 = k.saberi(12, 15);
        System.out.println(r2);
        //Ispisace se na ekranu
        //27

    }

}
```

Prvi poziv metode *saberi* će da aktivira metodu koja vrši sabiranje realnih brojeva i vraća realan broj kao rezultat. Drugi poziv metode *saberi* će da aktivira metodu koja vrši sabiranje celih brojeva i vraća ceo broj kao rezultat. U prvom slučaju su uneti realni brojevi kao ulazni parametri a u drugom celi brojevi.

Zadaci

Zadatak 1

Napraviti klasu **Radio**. Ova klasa bi trebalo da ima:

- Atribut **fmFrekvencija** koji označava trenutnu FM radio frekvenciju koju radio pušta (npr. 102.2 ili 87.5). Početna vrednost ovog polja je 87.5.
- Atribut **amFrekvencija** koji označava trenutnu AM radio frekvenciju koju radio pušta (npr. 567 ili 1500). Početna vrednost ovog parametra je 567.
- Atribut **band** koji ima vrednost 'A' (slovo A) ako radio pušta AM frekvenciju ili 'F' (slovo F) ako radio pušta FM frekvenciju. Početna vrednost ovog parametra je 'F'.
- Metodu **podesiFMFrekvenciju** koja prima kao ulazni parametar novu FM frekvenciju i podešava vrednost atributa **fmFrekvencija** na nju.
- Metodu **podesiAMFrekvenciju** koja prima kao ulazni parametar novu AM frekvenciju i podešava vrednost

- atributa amFrekvencija na nju.
- Metodu **vratiFMFrekvenciju** koja vraća trenutnu vrednost atributa fmFrekvencija.
- Metodu **vratiAMFrekvenciju** koja vraća trenutnu vrednost atributa amFrekvencija.
- Metodu **promeniBandNaAM** koja menja vrednost atributa band na 'A'.
- Metodu **promeniBandNaFM** koja menja vrednost atributa band na 'F'.
- Metodu **vratiBand** koja vraća trenutnu vrednost atributa band.
- Metodu **ispisiParametre** koja ispisuje trenutne vrednosti svih atributa uz odgovarajuće poruke.

Napraviti klasu **TestRadio** koja kreira tri objekta klase radio. Podesiti prvi radio da radi na AM frekvenciji 570. Podesiti drugi i treći radio da rade na FM frekvencijama 87.9 i 107.9 respektivno. Ispisati parametre sva ti radija.

Rešenje:

```
class Radio {

    double fmFrekvencija = 87.5;
    int amFrekvencija = 567;
    char band = 'F';

    void podesiFMFrekvenciju (double fmFrekvencija){
        this.fmFrekvencija = fmFrekvencija;
    }

    void podesiAMFrekvenciju (int amFrekvencija){
        this.amFrekvencija = amFrekvencija;
    }

    double vratiFMFrekvenciju (){
        return fmFrekvencija;
    }

    int vratiAMFrekvenciju (){
        return amFrekvencija;
    }

    void promeniBandNaAM (){
        band = 'A';
    }

    void promeniBandNaFM (){
        band = 'F';
    }

    char vratiBand (){
        return band;
    }

    void ispisiParametre (){
        System.out.println ("Trenutna FM frekvencija je "+fmFrekvencija);
        System.out.println ("Trenutna AM frekvencija je "+amFrekvencija);
        System.out.println ("Trenutno se pusta "+band+" frekvencija");
    }

}

class TestRadio {

    public static void main (String[] args){

        Radio r1 = new Radio();
        Radio r2 = new Radio();
```

```

Radio r3 = new Radio();

r1.promeniBandNaAM();
r1.podesiAMFrekvenciju(570);

r2.promeniBandNaFM();
r2.podesiFMFrekvenciju(87.9);

r3.promeniBandNaFM();
r3.podesiFMFrekvenciju(107.9);

r1.ispisiParametre();
r2.ispisiParametre();
r3.ispisiParametre();
}
}

```

Zadatak 2

Napraviti klasu **KlimaUredjaj**. Ova klasa bi trebalo da ima:

- Atribut **temperatura** koji označava trenutnu temperaturu u Celzijusima na koju je klima podešena (ceo broj – npr. 23). Početna vrednost ovog atributa je 18 stepeni.
- Atribut **rezimRada** koji označava da li klima greje, hladi ili radi automatski. Ako greje, ovaj atribut ima vrednost 'G', ako hladi 'H' a ako radi automatski onda 'A'. U početku, klima je podešena na automatski režim rada.
- Metodu **smanjiTemperaturu** koja smanjuje trenutnu temperaturu za jedan stepen.
- Metodu **povecajTemperaturu** koja povećava trenutnu temperaturu za jedan stepen.
- Metodu **podesiTemperaturu** koja kao ulazni parametar prima konkretnu vrednost temperature i postavlja vrednost atributa temperatura na nju.
- Metodu **ukljuciRezimGrejanje** koja podešava vrednost atributa rezimRada na grejanje ('G').
- Metodu **ukljuciRezimHladjenje** koja podešava vrednost atributa rezimRada na hladjenje ('H').
- Metodu **ukljuciRezimAutomatski** koja podešava vrednost atributa rezimRada na automatski ('A').
- Metodu **ispisiParametre** koja ispisuje trenutna podešavanja klima uređaja (režim i temperaturu).

Napraviti klasu **ProveraKlimaUredjaja** koja koristi dva objekta klase KlimaUredjaj. Potrebno je prvu klimu podesiti da greje na 27 C a drugu da hladi na 20 C. Ispisati parametre obe klime na ekranu.

Rešenje:

```

class KlimaUredjaj {

    int temperatura = 18;
    char rezimRada = 'A';

    void smanjiTemperaturu () {
        temperatura--; //Moglo je i ovako: temperatura=temperatura-1;
    }

    void povecajTemperaturu () {
        temperatura++; //Moglo je i ovako: temperatura=temperatura+1;
    }

    void podesiTemperaturu (int temperatura) {
        this.temperatura = temperatura;
    }

    void ukljuciRezimGrejanje () {
        rezimRada = 'G';
    }
}

```

```

    void ukljuciRezimHladjenje () {
        rezimRada = 'H';
    }

    void ukljuciRezimAutomatski () {
        rezimRada = 'A';
    }

    void ispisiParametre () {
        System.out.println ("Klima je podesena na "+temperatura+" C");
        System.out.println ("Rezim rada: "+rezimRada);
    }
}

class ProveraKlimaUredjaja {

    public static void main (String[] args) {

        KlimaUredjaj k1 = new KlimaUredjaj ();
        KlimaUredjaj k2 = new KlimaUredjaj ();

        k1.ukljuciRezimGrejanje ();
        k1.podesiTemperaturu (27);

        k2.ukljuciRezimHladjenje ();
        k2.podesiTemperaturu (20);

        k1.ispisiParametre ();
        k2.ispisiParametre ();
    }
}

```

Zadatak 3

Napraviti klasu **Kalkulator**. Ova klasa bi trebalo da ima:

- Metodu **saberi** koja sabira dva cela broja i vraća rezultat.
- Metodu **oduzmi** koja oduzima dva cela broja i vraća rezultat.
- Metodu **pomnozi** koja množi dva cela broja i vraća rezultat.
- Metodu **pomnozi** koja množi dva realna broja i vraća rezultat.
- Metodu **podeli** koja deli dva realna broja i vraća rezultat.
- Metodu **podeliDIV** koja deli dva cela broja i vraća rezultat (celobrojno DIV deljenje).
- Metodu **podeliMOD** koja deli dva cela broja i vraća ostatak pri deljenju kao ceo broj (celobrojno MOD deljenje).

Napraviti klasu **ProveraKalkulatora** koja proverava rad klase **Kalkulator** pozivanjem svih njenih metoda sa konkretnim vrednostima i ispisivanjem rezultata na ekranu.

Rešenje:

```

class Kalkulator {

    int saberi (int x, int y) {
        int rezultat = x+y;
        return rezultat;
    }

    int oduzmi (int x, int y) {

```

```

        int rezultat = x-y;
        return rezultat;
    }

    int pomnozi (int x, int y){
        int rezultat = x*y;
        return rezultat;
    }

    double pomnozi (double x, double y){
        double rezultat = x*y;
        return rezultat;
    }

    double podeli (double x, double y){
        double rezultat = x/y;
        return rezultat;
    }

    int podeliDIV (int x, int y){
        int rezultat = x/y;
        return rezultat;
    }

    int podeliMOD (int x, int y){
        int rezultat = x%y;
        return rezultat;
    }
}

class ProveraKalkulatora {

    public static void main (String[] args){

        Kalkulator k = new Kalkulator();

        //(123+23=146)
        System.out.println("Rezultat sabiranja je: "+k.saberi(123,23));

        //(10-7=3)
        System.out.println("Rezultat oduzimanja je: "+k.oduzmi(10,7));

        //(-10*8=80).
        //Poziva se prva metoda za mnozenje u kojoj se mnoze celi brojevi.
        System.out.println("Rezultat mnozenja je: "+k.pomnozi(-10,8));

        //(5.2*8=41.6).
        //Poziva se druga metoda za mnozenje u kojoj se mnoze realni brojevi.
        System.out.println("Rezultat mnozenja je: "+k.pomnozi(5.2,8));

        //(5.2/2.6=2.0).
        System.out.println("Rezultat deljenja je: "+k.podeli(5.2,2.6));

        //(5/2=2, 4/2=2,10/3=3).
        //DIV deljenje deli dva cela broja i vraca samo ceo deo rezultata.
        System.out.println("Rezultat DIV deljenja je: "+k.podeliDIV(5,2));
        System.out.println("Rezultat DIV deljenja je: "+k.podeliDIV(4,2));
        System.out.println("Rezultat DIV deljenja je: "+k.podeliDIV(10,3));

        //(5%2=1, 4%2=0,11%3=2).
    }
}

```

```

        //MOD deljenje deli dva cela broja i vraća samo ostatak pri deljenju.
        System.out.println("Rezultat MOD deljenja je: "+k.podeliMOD(5,2));
        System.out.println("Rezultat MOD deljenja je: "+k.podeliMOD(4,2));
        System.out.println("Rezultat MOD deljenja je: "+k.podeliMOD(11,3));
    }
}

```

Zadatak 4

Napraviti klasu **KonvertorMetrickihJedinica**. Ova klasa bi trebalo da ima:

- Metodu **konvertujKilometreUMilje** koja dobija kao ulazni parametar realan broj koji predstavlja broj kilometara, pretvara ga u milje i vraća (1 milja = 1.609 km).
- Metodu **konvertujMiljeUKilometre** koja dobija kao ulazni parametar realan broj koji predstavlja broj milja, pretvara ga u kilometre i vraća.
- Metodu **konvertujStopeUSantimetre** koja dobija kao ulazni parametar realan broj koji predstavlja broj stopa, pretvara ga u santimetre i vraća (1 stopa = 30.48 cm)
- Metodu **konvertujSantimetreUStope** koja dobija kao ulazni parametar realan broj koji predstavlja broj santimetara, pretvara ga u stope i vraća.

Napraviti i klasu **TestKonvertoraMetrickihJedinica** koja kreira jedan objekat klase KonvertorMetrickihJedinica i proverava njene metode.

Rešenje:

```

class KonvertorMetrickihJedinica {

    double konvertujKilometreUMilje (double duzina_km){
        double duzina_mi;
        duzina_mi = duzina_km / 1.609;
        return duzina_mi;
    }

    double konvertujMiljeUKilometre (double duzina_mi){
        double duzina_km;
        duzina_km = duzina_mi * 1.609;
        return duzina_km;
    }

    double konvertujStopeUSantimetre (double duzina_st){
        double duzina_cm;
        duzina_cm = duzina_st * 30.48;
        return duzina_cm;
    }

    double konvertujSantimetreUStope (double duzina_cm){
        double duzina_st;
        duzina_st = duzina_cm / 30.48;
        return duzina_st;
    }

}

class TestKonvertorMetrickihJedinica {

    public static void main (String[] args){

        KonvertorMetrickihJedinica k = new KonvertorMetrickihJedinica();

        System.out.println("Razdaljina od 10.4 km je "

```

```

        +k.konvertujKilometreUMilje(10.4)+" milja");
System.out.println("Razdaljina od 50 milja je "
        +k.konvertujMiljeUKilometre(50)+" km");
System.out.println("Duzina 2 stope je "
        +k.konvertujStopeUSantimetre(2)+" cm");
System.out.println("Duzina od 55 cm "
        +k.konvertujSantimetreUStope(55)+" stopa");
    }
}

```

Zadatak 5

Napraviti klasu **KonvertorTemperature**. Ova klasa bi trebalo da ima:

- Metodu **konvertujCUF** koja prima kao ulazni parametar temperaturu izraženu u Celzijusima (realan broj npr. 27.9), pretvara je u temperaturu u Farenhajtima i vraća kao rezultat. Konverzija se vrši po formuli $T(F) = (T(C)*9)/5 + 32$.
- Metodu **konvertujFUC** koja prima kao ulazni parametar temperaturu izraženu u Farenhajtima (realan broj npr. 100.4), pretvara je u temperaturu u Celzijusima i vraća kao rezultat. Konverzija se vrši po formuli $T(C) = (T(F)-32)*(5/9)$.

Napraviti klasu **ProveraKonvertoraTemperature** koja kreira jedan objekat klase KonvertorTemperature i, pomoću njega, konvertuje temperaturu 0 C u Farenhajte i temperaturu od 41 Farenhajta u Celzijuse. U oba slučaja potrebno je ispisati rezultat konvertovanja na ekranu uz odgovarajuću poruku.

Rešenje:

```

class KonvertorTemperature {

    double konvertujCUF (double tc){
        double tf;

        //Ovde mora da stoji 5.0 i 9.0 a ne 5 i 9 da se ne bi pozvalo
        //celobrojno DIV deljenje vec deljenje realnih brojeva
        tf = ((tc*9.0)/5.0)+32;
        return tf;
    }

    double konvertujFUC (double tf){
        double tc;

        //Ovde mora da stoji 5.0/9.0 a ne 5/9 da se ne bi pozvalo
        //celobrojno DIV deljenje vec deljenje realnih brojeva
        tc = (tf-32)*(5.0/9.0);
        return tc;
    }
}

class ProveraKonvertoraTemperature {

    public static void main (String[] args){

        KonvertorTemperature k = new KonvertorTemperature();

        System.out.println ("Temperatura 0 C je "+k.konvertujCUF(0)+" F");

        System.out.println ("Temperatura 41 F je "+k.konvertujFUC(41)+" C");

    }
}

```

```
}
```

Zadatak 6

Napraviti klasu **KonvertorValute**. Ova klasa bi trebalo da ima:

- Atribut **kursEvra** koji je realan broj i označava vrednost jednog evra u dinarima.
- Atribut **kursDolara** koji je realan broj i označava vrednost jednog dolara u dinarima.
- Metodu **postaviKursEvra** koja kao ulazni parametar prima novi kurs evra u dinarima i atribut kursEvra postavlja na tu vrednost.
- Metodu **postaviKursDolara** koja kao ulazni parametar prima novi kurs dolara u dinarima i atribut kursDolara postavlja na tu vrednost.
- Metodu **konvertujDinareUEvre** koja, na osnovu unetog dinarskog iznosa izračunava i vraća iznos u evrima po trenutnom kursu.
- Metodu **konvertujDinareUDolare** koja, na osnovu unetog dinarskog iznosa izračunava i vraća iznos u dolarima po trenutnom kursu.
- Metodu **konvertujEvreUDinare** koja, na osnovu unetog iznosa u evrima izračunava i vraća dinarski iznos po trenutnom kursu.
- Metodu **konvertujDolareUDinare** koja, na osnovu unetog iznosa u dolarima izračunava i vraća dinarski iznos po trenutnom kursu.
- Metodu **ispisiKursnuListu** koja ispisuje trenutne kurseve za dolar i evro.

Napraviti klasu **TestKonvertorValute** koja kreira jedan objekat klase Konvertor valute i postavlja trenutni kurs dolara na 60.34 dinara i evra na 76.89 dinara. Iskoristiti taj objekat za izračunavanje: dinarske protivvrednosti 60 dolara, dinarske protivvrednosti 45 evra, dolarske protivvrednosti 6034 dinara i evro protivvrednosti 150 dinara. Ispisati rezultate računanaj, pa zatim ispisati kursnu listu na ekranu.

Rešenje:

```
class KonvertorValute {  
  
    double kursEvra;  
    double kursDolara;  
  
    void postaviKursEvra (double kursEvra) {  
        this.kursEvra = kursEvra;  
    }  
  
    void postaviKursDolara (double kursDolara) {  
        this.kursDolara = kursDolara;  
    }  
  
    double konvertujDinareUEvre (double iznos_d) {  
        double iznos_e;  
        iznos_e = iznos_d / kursEvra;  
        return iznos_e;  
    }  
  
    double konvertujDinareUDolare (double iznos_d) {  
        double iznos_do;  
        iznos_do = iznos_d / kursDolara;  
        return iznos_do;  
    }  
  
    double konvertujEvreUDinare (double iznos_e) {  
        double iznos_d;  
        iznos_d = iznos_e * kursEvra;  
        return iznos_d;  
    }  
  
    double konvertujDolareUDinare (double iznos_do) {
```

```

        double iznos_d;
        iznos_d = iznos_do * kursDolara;
        return iznos_d;
    }

    void ispisiKursnuListu() {
        System.out.println("Kurs evra: 1 evro = "+kursEvra+" dinara");
        System.out.println("Kurs dolara: 1 dolar = "+kursDolara+" dinara");
    }
}

class TestKonvertorValute {

    public static void main (String[] args){

        KonvertorValute k = new KonvertorValute();

        k.postaviKursDolara(60.34);
        k.postaviKursEvra(76.89);

        System.out.println("60 dolara = "+k.konvertujDolareUDinare(60)+
            " dinara");
        System.out.println("45 evra = "+k.konvertujEvreUDinare(45)+
            " dinara");
        System.out.println("6034 dinara = "+k.konvertujDinareUDolare(6034)+
            " dolara");
        System.out.println("150 dinara = "+k.konvertujDinareUEvre(150)+
            " evra");

        k.ispisiKursnuListu();
    }
}

```

Konstruktori

Svaki put kada je potrebno inicijalizovati neki objekat koristi se komanda “new”. Posle ove rezervisane reči uvek sledi naziv klase pa otvorena i zatvorena zagrada. Već je objašnjeno da se pri inicijalizaciji rezerviše prostor u memoriji računara u koji će biti smešten novi objekat. Međutim, posle rezervisanja memorijskog prostora, dešava se još jedna stvar: vrši se poziv konstruktora. **Konstruktor** predstavlja posebnu metodu pomoću koje se može izvršiti dodeljivanje početnih vrednosti atributima objekta prilikom njegove inicijalizacije. Znači, konstruktor klase se **automatski poziva svaki put kada se pozove komanda “new”**. Deklaracija konstruktora je skoro ista kao i deklaracija metode, samo uz dva ograničenja:

- **Naziv konstruktora je uvek isti kao i naziv klase** (ako se nazove drugačije, Java tu metodu neće smatrati za konstruktor).
- Konstruktori **nikada nemaju povratnu vrednost**, pa se pri njihovoj deklaraciji čak ne piše ni “void”.

Pored toga, **klasa može imati i više konstruktora**, ali se onda oni moraju razlikovati po ulaznim parametrima (broj i/ili tipovi parametara). Ovo je preklapanje konstruktora i slično je preklapanju metoda.

U dosadašnjim primerima se nisu eksplicitno pisali konstruktori klase. Kada neka klasa u Javi nema

eksplicitno napisan konstruktor, Java joj automatski dodeljuje **predefinisani (“default”) konstruktor** koji, u suštini, ne menja vrednosti atributa niti vrši bilo kakve druge vidljive promene.

Ali, u nekim situacijama je pogodno napisati konstruktor jer se odmah, pri inicijalizaciji klase, mogu dodeliti neke početne vrednosti atributima, može se nešto ispisati na ekranu itd.

Konstruktori se dele na dve grupe u zavisnosti od toga da li imaju ulazne parametre ili ne. Prvu grupu čine **parametrizovani** konstruktori a drugu **neparametrizovani** konstruktori. Jedan neparametrizovan konstruktor je prikazan u sledećem primeru.

Primer 25

Napraviti klasu **PrehrambeniArtikal** koja ima dva atributa: naziv i kalorijskaVrednost. Prvom atributu dodeliti početnu vrednost “nepoznat” a drugom 0.0.

Napraviti klasu **PrehrambeniArtikal2** koja ima dva atributa: naziv i kalorijskaVrednost. Napraviti konstruktor koji prvom atributu dodeljuje početnu vrednost “nepoznat” a drugom 0.0.

```
class PrehrambeniArtikal {  
  
    String naziv = "nepoznat";  
    double kalorijskaVrednost = 0.0;  
  
}  
  
class PrehrambeniArtikal2 {  
  
    String naziv;  
    double kalorijskaVrednost;  
  
    PrehrambeniArtikal2() {  
        naziv = "nepoznat";  
        kalorijskaVrednost = 0.0;  
    }  
  
}
```

Klasa **PrehrambeniArtikal** nema konstruktor, pa se početna vrednost atributima dodeljuje odmah pri deklaraciji. Suprotno tome, klasa **PrehrambeniArtikal2** ima neparametrizovani konstruktor (nema ulazne parametre) koji služi da se atributima dodeli početna vrednost. Prvo, potrebno je primetiti da konstruktor, za razliku od obične metode, nema povratnu vrednost - ne piše se čak ni “void” nego odmah naziv. Drugo, naziv konstruktora je potpuno isti kao i naziv klase (**PrehrambeniArtikal2** - prvo slovo svake reči je veliko). Na kraju, vidi se da je efekat izvršenja konstruktora potpuno isti kao da se atributima dodelila početna vrednost odmah pri deklaraciji.

Sve ovo znači da se pri inicijalizaciji objekta *p* klase **PrehrambeniArtikal** poziva “default” konstruktor a atributima se dodeljuje vrednost preko deklaracije atributa.

```
PrehrambeniArtikal p = new PrehrambeniArtikal();
```

Pri inicijalizaciji objekta *p2* klase **PrehrambeniArtikal2** se poziva neparametrizovani konstruktor

koji dodeljuje vrednosti atributima.

```
PrehrambeniArtikal2 p2 = new PrehrambeniArtikal2();
```

Sa obzirom na prethodni primer i činjenice da se dodela početnih vrednosti atributima može izvršiti i bez upotrebe konstruktora, postavlja se pitanje: zašto uopšte koristiti konstruktor?

Konstruktor je neophodno koristiti svaki put kada je pri inicijalizaciji objekta potrebno uraditi bilo kakav složeniji posao koji podrazumeva nešto više od dodeljivanja početnih vrednosti atributima. To može podrazumevati proveru da li su unete vrednosti u okviru nekih granica, ispisivanje neke poruke na ekranu, inicijalizacije nekog niza, pristupanje nekim fajlovima isl.

Primer 26

*Napraviti klasu **PrehrambeniArtikal3** koja ima dva atributa: naziv i kalorijskaVrednost. Napraviti konstruktor koji prvom atributu dodeljuje početnu vrednost "nepoznat" a drugom 0.0 i ispisuje poruku o početnim vrednostima atributa na ekranu.*

```
class PrehrambeniArtikal3 {  
  
    String naziv;  
    double kalorijskaVrednost;  
  
    PrehrambeniArtikal3() {  
        naziv = "nepoznat";  
        kalorijskaVrednost = 0.0;  
        System.out.println("Naziv: "+naziv);  
        System.out.println("Kalorijska vrednost: "  
            +kalorijskaVrednost);  
    }  
  
}
```

*Konstruktor klase **PrehrambeniArtikal3** vrši i ispisivanje nekih poruka na ekranu. Drugim rečima, svaki put kada se neki objekat ove klase inicijalizuje na ekranu će se ispisati početne vrednosti njegovih atributa:*

```
PrehrambeniArtikal3 p3 = new PrehrambeniArtikal3();  
//Ispisace se na ekranu  
//Naziv: nepoznat  
//Kalorijska vrednost: 0.0
```

Parametrizovani konstruktori su još korisniji jer imaju ulazne parametre pa se atributima mogu dodeliti vrednosti koje su poznate tek u trenutku izvršavanja programa. Na ovaj način se preko samo jednog poziva konstruktora mogu istovremeno dodeliti vrednosti jednom ili više atributa. To se može videti u sledećem primeru.

Primer 27

*Napraviti klasu **Osoba** koja ima dva atributa: ime i prezime. Napraviti konstruktor koji ima dva ulazna parametra od kojih prvi sadrži vrednost za ime a drugi vrednost za prezime. Ovaj*

konstruktor bi trebalo da dodeljuje unete vrednosti atributima. Napraviti i klasu TestOsoba u kojoj se kreira objekat klase Osoba i odmah mu se, preko konstruktora, dodeljuje ime "Pera" i prezime "Peric".

```
class Osoba {

    String ime;
    String prezime;

    Osoba(String ime1, String prezime1){
        ime = ime1;
        prezime = prezime1;
    }

}

class TestOsoba {

    public static void main(String[] args) {

        //Ova komanda nece biti prihvacena jer
        //klasa Osoba ima jedan konstruktor a on
        //ima ulazne argumente
        //Osoba o1 = new Osoba();

        Osoba o1 = new Osoba("Pera", "Peric");

    }

}
```

Zadaci

Zadatak 1

Napraviti klasu **ZdravstvenaUstanova**. Ova klasa bi trebalo da ima:

- Atribut **naziv**.
- Atribut **adresa**.
- Konstruktor koji nema nijedan ulazni parametar i koji postavlja vrednosti oba atributa na "nepoznato".
- Konstruktor koji kao ulazni parametar prima naziv ustanove i postavlja vrednost atributa naziv.
- Konstruktor koji kao ulazne parametre prima naziv i adresu ustanove i postavlja vrednosti oba atributa.
- Metodu ispisi koja na ekranu ispisuje vrednosti oba atributa u jednom redu u formatu "Naziv: #### Adresa: ####".

Napraviti i klasu **TestZdravstvenaUstanova** koja kreira četiri objekta klase Zdravstvena ustanova. Prvi objekat je potrebno kreirati korišćenjem neparametrizovanog konstruktora i onda mu, pozivanjem atributa, dodeliti naziv "KCS" i adresu "Pasterova 2". Drugi objekat kreirati korišćenjem konstruktora koji ima jedan ulazni parametar i proslediti mu naziv "Dom zdravlja Vozdovac". Kreirati treći objekat korišćenjem parametrizovanog konstruktora koji ima dva ulazna parametra i dodeliti mu sledeći naziv i adresu: "Reumatoloski institut Beograd", "Resavska 67". Posle kreiranja svakog objekta, ispisati vrednosti njegovih atributa na ekranu.

Rešenje:

```
class ZdravstvenaUstanova {

    String naziv;
    String adresa;

    ZdravstvenaUstanova() {
        naziv = "nepoznato";
        adresa = "nepoznato";
    }

    ZdravstvenaUstanova(String naziv1) {
        naziv = naziv1;
    }

    ZdravstvenaUstanova(String naziv1, String adresa1) {
        naziv = naziv1;
        adresa = adresa1;
    }

    void ispisi() {
        System.out.println("Naziv: "+naziv+" Adresa: "+adresa);
    }
}

class TestZdravstvenaUstanova {

    public static void main(String[] args) {

        ZdravstvenaUstanova z1 = new ZdravstvenaUstanova();

        z1.naziv = "KCS";
        z1.adresa = "Pasterova 2";
        z1.ispisi();

        ZdravstvenaUstanova z2 =
            new ZdravstvenaUstanova("Dom zdravlja Vozdovac");

        z2.ispisi();

        ZdravstvenaUstanova z3 =
            new ZdravstvenaUstanova("Reumatoloski institut Beograd", "Resavska 67");

        z3.ispisi();
    }
}
```

Zadatak 2

Napraviti klasu **Proizvod**. Ova klasa bi trebalo da ima:

- Atribut **sifra**. Primer sifre: 123999
- Atribut **naziv**.
- Konstruktor koji kao ulazni parametar prima naziv proizvoda i postavlja vrednost atributa naziv. Naziv ulaznog parametra konstruktora bi trebalo da bude "naziv" (isto kao ime atributa).
- Konstruktor koji kao ulazne parametre prima naziv i šifru proizvoda i postavlja vrednosti oba atributa. Nazivi ulaznih parametara bi trebalo da budu isti kao nazivi atributa - "naziv" i "sifra".

- Metodu ispisi koja na ekranu ispisuje vrednosti oba atributa u jednom redu u formatu "Sifra: ##### Naziv: #####".

Napraviti i klasu **TestProizvod** koja kreira dva objekta klase Proizvod. Prvi objekat kreirati korišćenjem konstruktora koji ima jedan ulazni parametar i proslediti mu naziv "Stolica". Kreirati drugi objekat korišćenjem parametrizovanog konstruktora koji ima dva ulazna parametra i dodeliti mu sledeći naziv i šifru: "Drveni sto", 1234. Posle kreiranja svakog objekta, ispisati vrednosti njegovih atributa na ekranu.

Rešenje:

```
class Proizvod {

    int sifra;
    String naziv;

    Proizvod(String naziv) {
        //Mora da se koristi rec this jer se atribut
        //i ulazni argument isto zovu - naziv.
        this.naziv = naziv;

        //Komanda ne bi imala nikakav efekat ako bi glasila ovako
        //naziv = naziv;
    }

    Proizvod(String naziv, int sifra){
        this.naziv = naziv;
        this.sifra = sifra;
    }

    void ispisi() {
        System.out.println("Sifra: "+sifra+" Naziv: "+naziv);
    }

}

class TestProizvod {

    public static void main(String[] args) {

        //Ova komanda ne bi funkcionisala jer oba
        //konstruktora imaju parametre.
        //Proizvod p1 = new Proizvod();

        Proizvod p1 = new Proizvod("Stolica");

        p1.ispisi();

        Proizvod p2 = new Proizvod("Drveni sto", 1234);

        p2.ispisi();

    }

}
```

Globalne promenljive i globalne metode (ključna reč STATIC)

Objekti imaju atribute i metode koji se mogu pozvati i koristiti tek kada se objekat inicijalizuje. Takođe, svaki objekat ima svoje, zasebne vrednosti atributa. U nekim situacijama je, međutim,

korisno da više objekata (iz istih ili različitih klasa) deli jednu promenljivu. Takve promenljive su vidljive na nivou celog programa, pa se zovu **globalne promenljive**. U Javi se globalne promenljive označavaju ključnom reči “static” i pišu se u okviru tela klase kao atributi klase:

```
static tip_vrednosti nazivPromenljive;
```

Zbog toga što globalna promenljiva nije vezana ni za jedan objekat, korišćenje ove promenljive ne podrazumeva inicijalizaciju objekta klase kojoj pripada već joj se pristupa direktno preko naziva klase u okviru koje je deklarirana:

```
NazivKlase.nazivPromenljive
```

Primer 28

Napraviti klasu **Hotel** i u okviru nje definisati globalnu promenljivu *preostaliBrojSoba* (početna vrednost je 100). Napraviti klasu **TestHotel** koja menja vrednost ove globalne promenljive.

```
class Hotel {  
  
    static int preostaliBrojSoba = 100;  
  
}  
  
class TestHotel {  
  
    public static void main(String[] args) {  
  
        //Nije potrebna inicijalizacija objekta klase  
        //Hotel da bi se koristila globalna promenljiva  
        //vec se poziv vrši preko naziva klase.  
        Hotel.preostaliBrojSoba = 5;  
  
        System.out.println("Kapacitet je: "  
                            +Hotel.preostaliBrojSoba);  
    }  
  
}
```

Ponekad postoji i potreba za metodama koje pružaju neku opštu funkcionalnost koja nije striktno vezana za neku klasu. Ovakve metode su često dobri kandidati da postanu **globalne metode**. Globalne metode se u Javi označavaju ključnom reči “static” pa se često zovu i **statičke metode**. Ove metode se deklariraju u okviru tela klase na isti način kao i bilo koja druga metoda:

```
static tip_vrednosti nazivMetode(... ulazni parametri...) {  
  
    //Telo metode  
  
}
```

Razlika u odnosu na običnu metodu je ta što se ne mora inicijalizovati objekat te klase da bi se metoda koristila, već se poziv vrši preko naziva klase:

```
NazivKlase.nazivMetode(...ulazni parametri...);
```

U stvari, ako se malo bolje pogleda, može se videti da je i “System.out.println” komanda poziv statičke metode jer nije potrebno napraviti objekat klase System da bi se pozvala println metoda.

Main metoda je takođe statička metoda. Ovo je neophodno jer se poziva na početku rada programa pre nego što je bilo koji objekat napravljen.

Primer 29

Napraviti klasu **Kalkulator** i u okviru nje definisati dve globalne metode: *saberi* i *oduzmi*. Obe metode primaju dva cela broja kao ulazne parametre a vraćaju rezultat operacije sabiranja tj. oduzimanja. Napraviti klasu **TestKalkulator** koja poziva ove dve metode.

```
class Kalkulator {  
  
    static int saberi(int a, int b) {  
        return a+b;  
    }  
  
    static int oduzmi(int a, int b) {  
        return a-b;  
    }  
  
}  
  
class TestKalkulator {  
  
    public static void main(String[] args) {  
  
        int a = 5;  
        int b = 10;  
  
        int rezultat1 = Kalkulator.saberi(a, b);  
  
        int rezultat2 = Kalkulator.oduzmi(a, b);  
  
        System.out.println(rezultat1);  
        System.out.println(rezultat2);  
  
    }  
  
}
```

Pri korišćenju statičkih metoda treba imati na umu i to da statičke metode **ne mogu pozivati nestatičke elemente klase u kojoj su deklarisane** (obične atribute i metode).

Zadaci

Zadatak 1

Napraviti klasu **StatickiKonvertorTemperature**. Ova klasa bi trebalo da ima:

- Statičku metodu **konvertujCUF** koja prima kao ulazni parametar temperaturu izraženu u Celzijusima (realan broj npr. 27.9), pretvara je u temperaturu u Farenhajtima i vraća kao rezultat. Konverzija se vrši po formuli $T(F) = (T(C) * 9) / 5 + 32$.
- Statičku metodu **konvertujFUC** koja prima kao ulazni parametar temperaturu izraženu u Farenhajtima (realan broj npr. 100.4), pretvara je u temperaturu u Celzijusima i vraća kao rezultat. Konverzija se vrši po formuli $T(C) = (T(F) - 32) * (5/9)$.

Napraviti klasu **ProveraStatickogKonvertoraTemperature** koja kreira jedan objekat klase **StatickiKonvertorTemperature** i, pomoću njega, konvertuje temperaturu 0 C u Farenhajte i temperaturu od 41 Farenhajta u Celzijuse. U oba slučaja potrebno je ispisati rezultat konvertovanja na ekranu uz odgovarajuću poruku.

Rešenje:

```
class StatickiKonvertorTemperature {

    static double konvertujCUF (double tc){
        double tf;
        tf = ((tc*9.0)/5.0)+32;
        return tf;
    }

    static double konvertujFUC (double tf){
        double tc;
        tc = (tf-32)*(5.0/9.0);
        return tc;
    }
}

class ProveraStatickogKonvertoraTemperature {

    public static void main (String[] args){

        //Nije potrebna inicijalizacija objekta da bi
        //se pozvale staticke metode za konvertovanje

        System.out.println ("Temperatura 0 C je "+
            StatickiKonvertorTemperature.konvertujCUF(0)+" F");

        System.out.println ("Temperatura 41 F je "+
            StatickiKonvertorTemperature.konvertujFUC(41)+" C");

    }
}
```

Konstante (ključna reč *FINAL*)

Ukoliko je potrebno definisati neku konstantu u okviru Java klase, koristi se **ključna reč “final”**. Konstante se definišu kao atributi, sa tim što se ispred tipa atributa stavi reč “final”:

```
final tip_vrednosti NAZIV_KONSTANTE = vrednost;
```

Prema nepisanom pravilu, **nazivi konstanti se pišu svim velikim slovima** (“PI”, “KAMATA”). Ako naziv konstante ima više reči, reči se odvajaju znakom donje crte (“MIN_KAMATA”). Konstantama se **moraju dodeliti vrednosti odmah pri deklaraciji** jer to kasnije nije moguće (čak ni u okviru konstruktora).

Pozivanje konstante se vrši na isti način kao i pozivanje atributa klase. Jedina razlika je u tome što konstantama **nikada ne može da se dodeli nova vrednost**.

Za neke konstante je poželjno da budu globalno vidljive pa se, pored reči “final”, označavaju i ključnom reči “static”. Pozivanje ovih konstanti je isto kao i pozivanje globalnih promenljivih.

```
static final tip_vrednosti NAZIV_KONSTANTE = vrednost;
```

Ključna reč “final” može da se koristi i uz deklaracije klasa i metoda, ali u tim slučajevima ima potpuno drugačije značenje. Ova vrsta upotrebe je usko vezana za nasleđivanje klasa, pa će biti detaljno objašnjena u poglavlju o nasleđivanju.

Zadaci

Zadatak 1

Napraviti klasu **MatematickeKonstante**. Ova klasa bi trebalo da ima:

- Statičku konstantu PI koja iznosi 3,141592.
- Statičku konstantu E koja iznosi 2,71.

Napraviti klasu **TestMatematickihKonstanti** koja na ekranu ispisuje vrednosti dve konstante iz klase MatematickeKonstante uz odgovarajuću poruku.

Rešenje:

```
class MatematickeKonstante {  
  
    static final double PI = 3.141592;  
    static final double E = 2.71;  
  
}  
  
class TestMatematickeKonstante {  
  
    public static void main(String[] args) {  
  
        System.out.println("PI = "+  
                             MatematickeKonstante.PI);  
  
        System.out.println("E = "+  
                             MatematickeKonstante.E);  
  
        //PI i E su konstante pa im se ne moze  
        //dodeljivati nova vrednost. Sledece  
        //komande ce se shvatiti kao greske.  
        //MatematickeKonstante.PI = 2;  
        //MatematickeKonstante.E = 2.7177;  
  
    }  
  
}
```

Relacije

Predmeti i pojave iz realnog sveta često formiraju i neke odnose. Klase, kao uprošćene slike realnih predmeta i pojava, takođe mogu da formiraju međusobne odnose koji reflektuju realne odnose. Odnosi između klasa se nazivaju **relacijama**. Kao što objekti preuzimaju i koriste attribute i metode klase kojoj pripadaju, tako preuzimaju i relacije.

Postoje tri vrste relacija:

- **Asocijacija** (“USING” relacija)
- **Kompozicija - dekompozicija** (“HAS-A” relacija, “PART-OF” relacija)
- **Generalizacija - specijalizacija** (“IS-A” relacija, “nasleđivanje”)

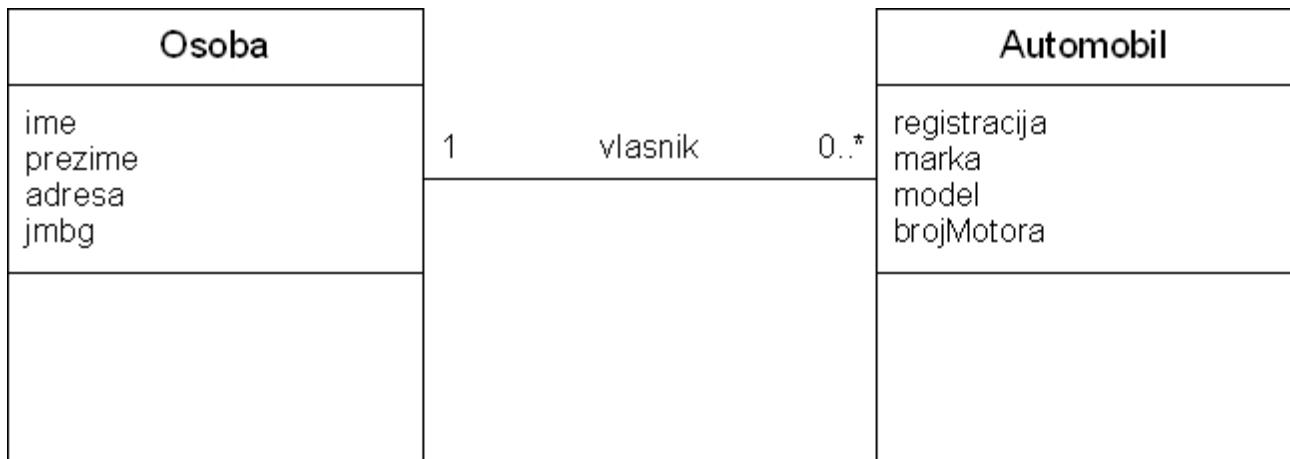
Asocijacija predstavlja najopštiji oblik relacije u kome objekti jedne klase koriste objekte druge klase. Pod korišćenjem se podrazumeva pozivanje atributa i metoda klase, ili prenošenje objekata klase kao ulaznih parametara ili rezultata metode. U opštem slučaju, asocijacije su dvosmerne, ali postoje i slučajevi u kojima asocijacije mogu imati samo jedan smer: objekti prve klase koriste objekte druge klase, ali ne važi obrnuto. Asocijacija može da ima naziv ali i ne mora.

Još jedan pojam koji je potrebno odmah uvesti i objasniti je **kardinalnost**. Kardinalnost je broj koji označava koliko objekata jedne klase može da bude u relaciji sa jednim objektom druge klase i obrnuto. Većina asocijacija je dvosmerna, pa se može reći da svaka relacija ima dve kardinalnosti u odnosu na smer relacije. Pored toga, kardinalnost ne mora da bude samo jedan broj već može da bude raspon, pa prema tome postoji **minimalna i maksimalna kardinalnost**.

U Javi, **asocijacije se najčešće implemetiraju kao obični atributi klase, sa tim što je tip atributa sama klasa** - navodi se naziv klase. U opštem slučaju, implementacija zavisi od kardinalnosti, pa se tako asocijacije “više na prema više” mogu implemetirati uvođenjem nove klase isl. Na graphicima se asocijacije najčešće predstavljaju običnom linijom između dve klase na kojoj može biti napisana kardinalnost ili naziv asocijacije.

Primer 30

Primer asocijacije može da bude relacija između klasa Automobil i Osoba u kojoj je osoba vlasnik vozila (Slika 7). Prvo, potrebno je primetiti kardinalnost ove asocijacije. Odmah pored klase Automobil, na slici se nalazi oznaka “0..”. To znači da je minimalna kardinalnost asocijacije u ovom smeru nula (“0”) a maksimalna više (“*”) - jedna osoba može biti vlasnik više automobila (maksimalna kardinalnost) a ne mora biti vlasnik nijednog automobila (minimalna kardinalnost). Kardinalnost u drugom smeru asocijacije je 1 i znači da tačno jedna osoba može da bude vlasnik jednog automobila (automobil mora da ima tačno jednog vlasnika).*



Slika 7: Primer asocijacije

Implementacija ove asocijacije u Javi bi izgledala ovako:

```

class Osoba {

    String ime;
    String prezime;
    String adresa;
    String jmbg;

}

class Automobil {

    String marka;
    String model;
    String registacija;
    int brojMotora;

    //Asocijacije mogu da se implementiraju kao
    //atributi klase, samo je tip atributa klasa.
    Osoba vlasnik;

}
  
```

Bitno je primetiti da se asocijacije implementiraju kao atributi klase, ali se kao tip vrednosti navodi naziv klase.

Pozivanje objekta preko asocijacije se vrši na sličan način kao i pozivanje običnog atributa sa tim što je objekat potrebno inicijalizovati pre prvog korišćenja.

Primer 31

Napisati klasu *TestAutomobil* koja kreira jedan objekat klase *Automobil* (iz prethodnog primera) marke “Ford”, model “Focus” registracije “BG123-456” i broj motora “123456”. Postaviti da vlasnik ovog automobila bude Pera Perić JMBG: 2112980710018 koji živi u Resavskoj 40.

```

class TestAutomobil {

    public static void main(String[] args) {

        Automobil a = new Automobil();

        a.marka = "Ford";
        a.model = "Focus";
        a.registacija = "BG123-456";
        a.brojMatora = 123456;

        //Atribut vlasnik je objekat klase Osoba
        //pa prvo mora da se inicijalizuje
        a.vlasnik = new Osoba();

        //Atributima osobe koja je vlasnik se
        //pristupa preko naziva objekta klase
        //Automobil i preko naziva atributa
        a.vlasnik.ime = "Pera";
        a.vlasnik.prezime = "Peric";
        a.vlasnik.jmbg = "2112980710018";
        a.vlasnik.adresa = "Resavska 40";

    }

}

```

Atributima objekta klase Osoba koji je vlasnik automobila se pristupa posredno - prvo pozivanjem objekta klase Automobil, pa pozivanjem atributa vlasnik. Između svaka dva elementa u okviru poziva je tačka.

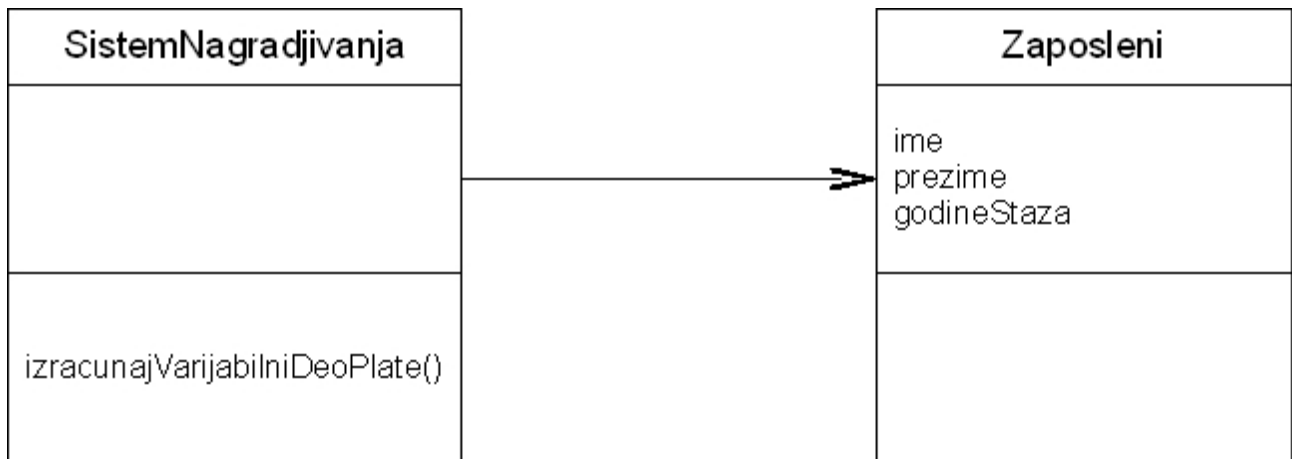
```
a.vlasnik.ime = "Pera";
```

Kada jedna klasa koristi neki objekat druge klase u okviru neke svoje metode, i to se smatra asocijacijom. Nije bitno da li je objekat te druge klase ulazni parametar metode, povratna vrednost metode ili se samo koristi u okviru tela metode. Ovakve asocijacije najčešće nemaju eksplicitno označenu kardinalnost niti ime, a grafički se predstavljaju linijom usmerenom ka klasi koja se koristi.

Primer 32

*Napisati klasu Zaposleni koja ima attribute: ime, prezime i godineStaza. Napraviti klasu SistemNagradjivanja koja ima metodu izracunajVarijabilniDeoPlate. Ova metoda kao ulazni parametar prima objekat klase Zaposleni i izračunava i vraća procenat koji predstavlja varijabilni deo plate po formuli: godineStaza*1.2.*

Na slici (Slika 8) se može videti kako bi se ova vrsta asocijacije grafički prikazala. Kao i druge asocijacije i ova je predstavljena linijom između klasa, ali je ovaj put linija usmerena - klasa SistemNagradjivanja koristi klasu Zaposleni. Za ovu vrstu asocijacija se obično ne piše naziv niti kardinalnost.



Slika 8: Primer asocijacije

```

class Zaposleni {

    String ime;
    String prezime;
    int godineStaza;

}

class SistemNagradjivanja {

    double izracunajVarijabilniDeoPlate(Zaposleni z){
        double var;
        var = z.godineStaza * 1.2;
        return var;
    }

}
  
```

U ovom primeru, klasa SistemNagradjivanja koristi objekat klase Zaposleni kao ulazni parametar jedne svoje metode. Metode takođe mogu i da vraćaju objekte kao rezultat.

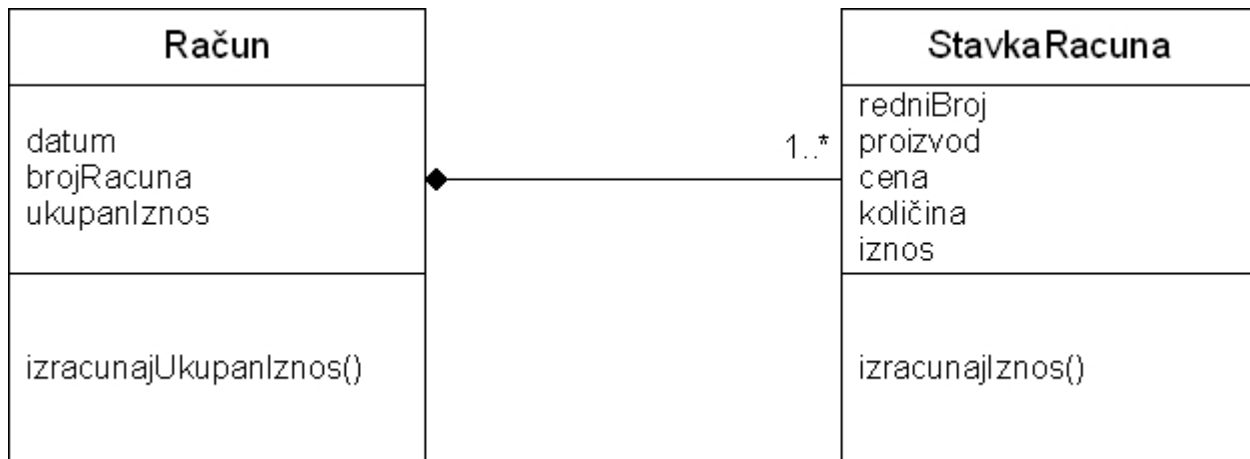
Kompozicija - dekompozicija predstavlja posebnu vrstu asocijacije u kojoj objekat jedne klase, kao sastavni deo, sadrži jedan ili više objekata druge klase. Postoje dve vrste relacija kompozicija-dekompozicija.

Ako objekat (ili više njih) koji je sadržan ne može da postoji nezavisno od objekta koji ga sadrži, u pitanju je **kompozicija**, a u suprotnom je u pitanju **agregacija**. Kod kompozicije, kada se obriše objekat koji sadrži druge objekte, brišu se i ti objekti. Kod agregacije to nije slučaj.

Relacija kompozicije-dekompozicije se grafički predstavlja linijom koja na jednom kraju ima romb. Ako je romb ispunjen u pitanju je kompozicija, a ako je prazan onda je agregacija. Kompozicija-dekompozicija može da ima naziv, a kardinalnost se piše samo sa jedne strane jer je sa druge strane uvek 1.

Primer 33

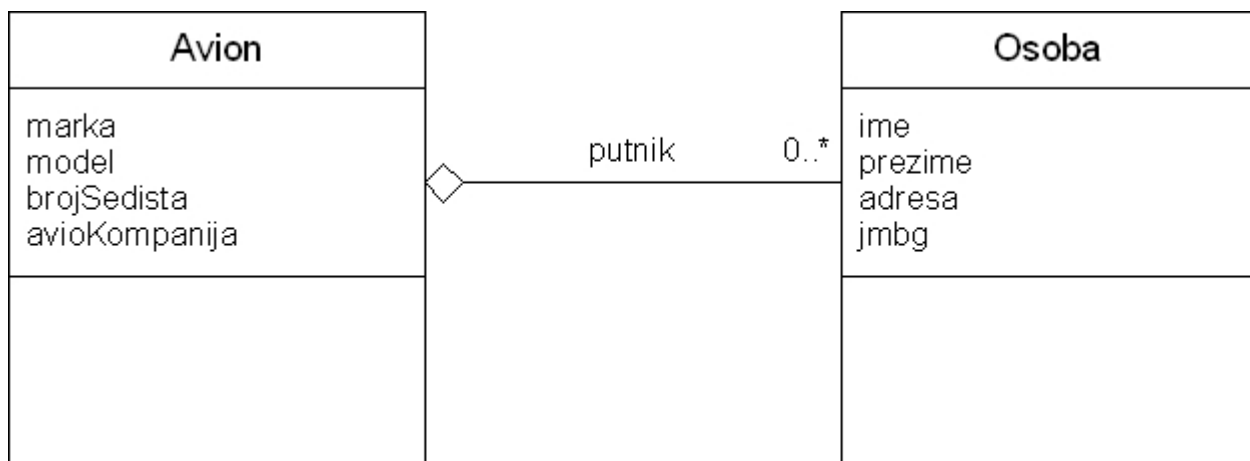
Primer koji se često navodi da opiše relaciju kompozicije je odnos između klasa “Račun” i “StavkaRačuna” (Slika 9). Svaki račun sadrži jednu ili više stavki (minimalna kardinalnost je 1 a maksimalna kardinalnost više), a svaka stavka pripada tačno jednom računu (minimalna i maksimalna kardinalnost je 1 pa se ni ne piše), i ne bi imalo nikakvog smisla ako bi postojale stavke računa koje ne pripadaju nijednom računu. Takođe, kada se obriše račun, brišu se i sve njegove stavke.



Slika 9: Primer kompozicije

Primer 34

Relacija agregacije može da se demonstrira preko odnosa klasa Avion i Osoba u kojima su osobe putnici u avionu (Slika 10). Za razliku od kompozicije, agregacija podrazumeva a agregirani objekti mogu da postoje i bezobjekta koji ih sadrži. U ovom slučaju, Osobe postoje i van konteksta putnika u avionu.



Slika 10: Primer agregacije

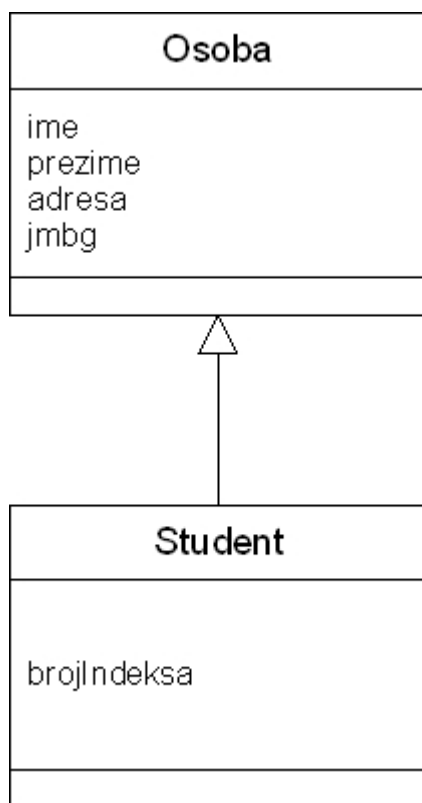
Kompozicija-dekompozicija se implementira na isti način kao i obična asocijacija - preko atributa klase. Jedina razlika je u načinu na koji se tumači asocijacija.

Generalizacija - specijalizacija je vrsta relacije u kojoj jedna klasa “nasleđuje” drugu klasu, pa se zato često naziva i **nasleđivanje**. Nasleđivanje podrazumeva preuzimanje svih atributa i metoda **nadklase** (klasa koja je nasleđena) i njihovo prenošenje u **podklasu** (klasa koja nasleđuje). Krajnji efekat je taj da podklasa poseduje sve karakteristike i ponašanja nadklase, ali takođe sadrži i još

neke karakteristike i ponašanje. Nasleđivanje će detaljno biti objašnjeno u jednom od narednih poglavlja. Za sada, dovoljno je reći da se grafičko predstavljanje relacije nasleđivanja vrši usmerenom linijom čiji vrh predstavlja praznu trouglastu strelicu i dase uz ovu relaciju ne navodi kardinalnost niti naziv.

Primer 35

Neka klasa Osoba ima atribute ime, prezime, adresa i jmbg. Neka je potrebno napraviti klasu Student koja ima iste ove attribute, ali i broj indeksa. U ovom slučaju, najjednostavnija solucija je da se ove klase naprave tako da klasa Student nasleđuje klasu Osoba. Student će da nasledi attribute ime, prezime, adresa i jmbg iz klase Osoba, pa će biti potrebno samo dodati atribut broj indeksa (Slika 11).



Slika 11: Primer nasleđivanja

Zadaci

Zadatak 1

Napraviti klasu Osoba koja ima:

- Atribut ime.
- Atribut prezime.
- Metodu ispiši koja ispisuje sve podatke o osobi.

Napraviti klasu Formula1Tim koja ima:

- Atribut naziv.
- Atribut menadzer koji je objekat tipa Osoba i predstavlja menadžera trkačkog tima.
- Atribut prviVozac koji je objekat tipa Osoba i predstavlja prvog vozača.
- Atribut drugiVozac koji je objekat tipa Osoba i predstavlja prvog vozača.
- Metodu ispiši koja ispisuje sve podatke o timu uključujući i imena i prezimena menadžera i oba vozača. Ova

metoda bi trebalo da poziva metodu ispisi klase Osoba u cilju ispisivanja podataka o menadzeru i vozačima.

Napraviti klasu TestFormula1Tim koja kreira tim formule 1 “Scuderia Ferrari Marlboro” čiji vlasnik je “Berny Ecklestone”. Prvi vozač tima je “Kimi Raikkonen” a drugi “Felipe Massa”. Ispisati na ekranu sve podatke o timu.

Rešenje:

```
class Osoba {

    String ime;
    String prezime;

    void ispisi() {
        System.out.println("Ime: "+ime);
        System.out.println("Prezime: "+prezime);
    }

}

class Formula1Tim {

    String naziv;

    Osoba menadzer;
    Osoba prviVozac;
    Osoba drugiVozac;

    void ispisi() {
        System.out.println("Naziv tima: "+naziv);

        System.out.println("Menadzer");
        //Direktno se poziva metoda ispisi klase Osoba
        //da bi se ispisalo ime i prezime menadzera.
        menadzer.ispisi();

        System.out.println("Prvi vozac");
        //Direktno se poziva metoda ispisi klase Osoba
        //da bi se ispisalo ime i prezime prvog vozaca.
        prviVozac.ispisi();

        System.out.println("Drugi vozac");
        //Direktno se poziva metoda ispisi klase Osoba
        //da bi se ispisalo ime i prezime drugog vozaca.
        drugiVozac.ispisi();
    }

}

class TestFormula1Tim {

    public static void main(String[] args) {

        Formula1Tim t = new Formula1Tim();

        t.naziv = "Scuderia Ferrari Marlboro";

        t.menadzer = new Osoba();
        t.menadzer.ime = "Berny";
        t.menadzer.prezime = "Ecklestone";
```



```
t.prviVozac = new Osoba();
t.prviVozac.ime = "Kimi";
t.prviVozac.prezime = "Raikkonen";

t.drugiVozac = new Osoba();
t.drugiVozac.ime = "Felipe";
t.drugiVozac.prezime = "Massa";

t.ispisi();
}

}
```