PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET
METROPOLITAN
BEOGRAD

KI103 - JAVA 1: OSNOVE PROGRAMIRANJA U JAVI

Metodi

Lekcija 06

PRIRUČNIK ZA STUDENTE

KI103 - JAVA 1: OSNOVE PROGRAMIRANJA U JAVI

Lekcija 06

METODI

- Metodi
- Poglavlje 1: Definisanje metoda
- Poglavlje 2: Pozivanje metoda
- Poglavlje 3: Parametri metoda
- Poglavlje 4: Vraćanje rezultata metoda
- Poglavlje 5: Preopterećenje metoda
- Poglavlje 6: Domaći zadaci
- Zakljucak

UVOD

Primena metoda omogućuje modularizaciju softvera.

Cilj ove lekcije je da naučite:

1. Definisanje metoda sa formalnim parametrima,
2. Pozivanje metoda sa stvarnim vrednostima parametara.
3. Definisanje metoda koji vraća vrednost.
4. Definisanje metoda koji ne vraćaju vrednost.
5. Prenošenje vrednosti (argumenata) po vrednosti.
6. Razvoj višestruko upotrebljivog programa koji je lak za razumevanje i održavanje.
7. Upotrebu preopterećenih metoda
8. Određivanje domena upotrebljivosti promenljivih.
9. Primenu apstrakcije metoda u razvoju softvera.
10. Projektujete i razvijate metode primenom poboljšanja, korak po korak.

Poglavlje 1

Definisanje metoda

KONCEPT POTPROGRAMA

Potprogram se sastoji od niza naredbi i promenljivih koje predstavljaju neku funkcionalnu celinu sa svojim posebnim imenom

Koncept potprograma je sveprisutan u programskim jezicima, ali se pojavljuje pod različitim imenima: metod, procedura, funkcija, rutina i slično. Njegova glavna svrha je to da se složeni program podeli u manje delove koji se mogu lakše i nezavisno razvijati. Osim što je deo programa i što obično obavlja jednostavniji zadatak, potprogram ima sličnu logičku strukturu kao glavni program.

Potprogram se sastoji od niza naredbi i promenljivih koje predstavljaju neku funkcionalnu celinu sa svojim posebnim imenom. Ime potprograma se može koristiti bilo gde u programu kao zamena za ceo niz naredbi od kojih se potprogram sastoji. Pored toga, ova mogućnost se može koristiti više puta, na različitim mestima u programu.

Potprogrami se mogu čak koristiti unutar drugih potprograma. To znači da se mogu napisati jednostavni potprogrami, koji se mogu iskoristiti u složenijim potprogramima, a i ovi se dalje mogu koristiti u još složenijim potprogramima. Na taj način, vrlo složeni programi se mogu postupno razvijati korak po korak, pri čemu je svaki korak relativno jednostavan.

Potprogrami imaju, baš kao i programi, ulazne i izlazne podatke preko kojih komuniciraju sa okolnim (pot)programom.

Ulazni podaci su parametri koje potprogram dobija od okolnog (pot)programa radi izvršenja svog zadatka, a izlazni podaci predstavljaju rezultat rada potprograma. Ulazni i izlazni podaci su deo veze jednog potprograma sa drugim potprogramima koja se često naziva interfejs potprograma. Interfejs pored ovoga uključuje ime i opis funkcije potprograma, odnosno sve što treba znati radi ispravne upotrebe potprograma u drugim potprogramima bez poznavanja same implementacije potprograma. U tom smislu se za potprogram kaže da predstavlja "crnu kutiju" čija je unutrašnjost , tj. implementacija, sakrivena od spoljašnjosti, a interfejs je granica koja povezuje ova dva dela.

Korišćenje potprograma se u programiranju tehnički naziva pozivanje potprograma za izvršavanje. Svakim pozivom potprograma se dakle izvršava niz naredbi od kojih se potprogram sastoji.

Implementacija potprograma, odnosno niz naredbi i ostali elementi koji čine potprogram, naziva se definicija (ili deklaracija) potprograma.

Termin koji se u Javi koristi za potprogram je metod, u smislu postupaka ili načina za rešavanje nekog zadatka.

NAČIN DEFINISANJA METODA

Termin koji se u Javi koristi za potprogram je metod, u smislu postupaka ili načina za rešavanje nekog zadatka.

Telo metoda ima oblik običnog bloka naredbi, odnosno telo metoda se sastoji od niza proizvoljnih naredbi između vitičastih zagrada.

Zaglavlje metoda sadrži sve informacije koje su neophodne za pozivanje metoda. To znači da se u zaglavlju metoda, između ostalog, nalazi:

- Ime metoda
- Tipovi i imena parametara metoda
- Tip vrednosti koju metod vraća kao rezultat
- Razni modifikatori kojima se dodatno određuju karakteristike metoda

Preciznije, opšti oblik definicije metoda u Javi je:

```
modifikatori tipRezultata imeMetoda(listaParametara)

    {
        naredba1;
        ...
        naredbaN;
    }
```

Na primer, definicija glavnog metoda ***main()*** u programu obično ima sledeći oblik:

```
public static void main (String[] args) { // Zaglavlje metoda
        // Telo metoda
    }
```

U zaglavlju ove definicije metoda ***main(),*** rezervisane reči public i static su primeri modifikatora, void je tip rezultata, main je ime metoda i, na kraju,***String[]*** args u zagradi čini listu od jednog parametra, tj niz stringova pod nazivom args.

Deo modifikatori na početku zaglavlja nekog metoda nije obavezan, ali se može sastojati od jedne ili više službenih reči koje su međusobno razdvojene razmacima. Modifikatorima se određuju izvesne karakteristike metoda.

Tako, u primeru metoda ***main(),*** rezervisana reč***public*** ukazuje da se metod može slobodno koristiti u bilo kojoj drugoj klasi; službena reč ***static*** ukazuje da je metod statički, a ne objektni, i tako dalje. Za metode je u Javi na raspolaganju ukupno oko desetak modifikatora. O njima, kao i drugim elementima definicije metoda, govorićemo u narednim predavanjima.

ZAGLAVLJE METODA

Prva reč imena metoda počinje malim slovom, a ako se to ime sastoji od nekoliko reči, onda se svaka reč od druge piše sa početnim velikim slovom

Ako metod izračunava jednu vrednost koja se vraća kao rezultat poziva metoda, onda deo **tipRezultata** u njegovom zaglavlju određuje tip podataka kojem pripada rezultujuća vrednost metoda. S druge strane, ako metod ne vraća nijednu vrednost, onda se deo tipRezultata sastoji od rezervisane reči***void***. Time se želi označiti da tip nepostojeće vraćene vrednosti predstavlja tip podataka koji ne sadrži nijednu vrednost.

```
modifikatori tipRezultata imeMetoda(listaParametara)
    {
        naredba1;
        ...
        naredbaN;
    }
```

Formalna pravila za davanje imena metodu u delu **imeMetoda** jesu uobičajena pravila za imena u Javi o kojima smo govorili u odeljku o identifikatorima. Dodajmo ovde samo to da je neformalna konvencija za *imena metoda u Javi ista kao i za imena promenljivih: prva reč imena metoda počinje malim slovom, a ako se to ime sastoji od nekoliko reči, onda se svaka reč od druge piše sa početnim velikim slovom*. Naravno, opšte nepisano pravilo je da imena metoda treba da budu smislena, odnosno da imena metoda dobro opisuju šta je funkcija pojedinih metoda.

Na kraju zaglavlja metoda, deo **listaParametara** u zagradi određuje sve parametre metoda. Parametri su promenljive koje se inicijalizuju vrednostima argumenata prilikom pozivanja metoda za izvršavanje. ***Parametri*** *predstavljaju dakle ulazne podatke metoda koji se obrađuju naredbama u telu metoda.* Lista parametara metoda može biti prazna (ali se zagrade moraju pisati), ili se ta lista može sastojati od jednog ili više parametara. Budući da parametar konceptualno predstavlja običnu promenljivu, *za svaki parametar u listi navodi se njegov tip i njegovo ime u obliku:*

```
tipParametra imeParametra
```

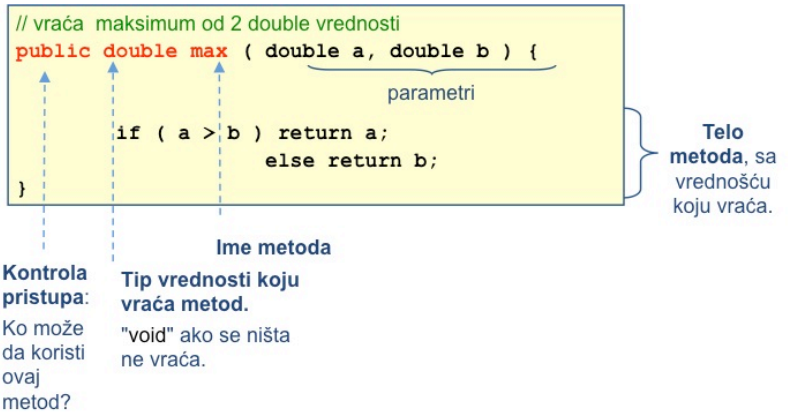
Tako lista sadrži više od jednog parametra, onda se parovi specifikacija tih parametara međusobno razdvajaju zapetama. Primetimo da *svaki par za tip i ime parametra mora označavati samo jedan parametar.*

To znači da ako neki metod ima, recimo, dva parametra x i y tipa double, onda se u listi parametara mora pisati double x, double y, a ne skraćeno double x, y.

MODIFIKATORI

Modifikatori private, protected, public i default se moraju pisati ispred tipa rezultata metoda.

Na slici 1 prikazan je primer definisanja metoda max() koji daje veći od dva uneta broja.



Slika 1.1 Primer definisanja metoda max()

Kao što vidimo, pre naziva metoda, treba navesti modifikator vidljivosti, tj. pristupa i tip povratne vrednosti; ako se ne vraća nijedna vrednost, onda se stavlja reč void. Između malih zagrada se upisuju parametri, tj. argumenti preko kojih se prenose vrednosti u metod. Za svaki argument je potrebno navesti i tip podataka koji koristi svaki argument. Između velikih zagrada je blok Java iskaza (komandi, instrukcija) kojim se

programira šta metod treba da uradi. Iza iskaza return navodi se povratna veličina koju vraća metod u program iz kojeg je pozvan metod.

Vi određujete koji objekti ili delovi programa mogu da pristupe metodima nekog objekta. Imate 4 izbora:

1. **private:** metod može da bude aktiviran samo od objekta ove klase;
2. **protected:** metod može da bude aktiviran iz programa istog paketa i objekta ove klase i njenih podklasa;
3. **public:** metod može biti aktiviran bilo kojim Java programom;
4. **default:** isto kao "protected" + program istog paketa.

Modifikatori se u zaglavlju metoda moraju pisati ispred tipa rezultata metoda, ali njihov međusobni redosled nije bitan. Konvencija je ipak da se najpre piše eventualni specifikator pristupa, zatim eventualno reč ***static i***, na kraju, eventualno ostali modifikatori.

.

PRIMER KORIŠĆENJA METODA

Primeri korišćenja modifikatora public i static, kao i vraćenih vrednosti tipa int i boolean.

Slika 2 prikazuje metod ***uplati()*** klase ***BankovniRacun***.

```
// definiši ponašanje "ulaganja" novca na račun u banci.
public class BankovniRacun {
        private long stanje;   // atribut stanja

        public void uplati(long suma ) {
                stanje = stanje + suma;
        }
                //... Više metoda i atributa
}
```

Slika 1.2 Metod uplati()

U nastavku su prikazani još neki primeri definicija metoda u Javi, bez naredbi u telu tih metoda kojima se realizuje njihova funkcija:

```
int iscrtajTekst(int n,int m, String naslov) {
        // Telo metoda
    }
```

U ovom primeru nema modifikatora u zaglavlju metoda; tip vraćene vrednosti metoda je int; ime metoda je***iscrtajTekst***, lista parametara sadrži tri parametra: n i m tipa***int*** i naslov tipa ***Strin***g

U sledećem primeru je***static*** jedini modifikator; tip vraćene vrednosti je ***boolean***; ime metoda je ***manjeOd()***; lista parametara sadrži dva parametra x i y tipa ***float***.

```
static boolean manjeOd(float x, float y) {
        // Telo metoda
    }
```

Drugi metod iscrtajTekst() u prethodnim primerima je objektni (nestatički) metod, jer njegovo zaglavlje ne sadrži rezervisanu reč***static***. Generalno, podrazumeva se da je metod objektni ukoliko nije statički, što se određuje modifikatorom static.

Ukoliko u zaglavlju metoda nije naveden nijedan specifikator pristupa (kao u drugom i trećem od prethodnih primera), onda se metod može pozivati u svakoj klasi istog paketa kome pripada klasa u kojoj je metod definisan, ali ne i izvan tog paketa.

PRIMER 1

Pisanje statičke metode za računanje pentagonalnih brojeva

Napraviti program koji računa pentagonalni broj za brojeve od 1 do 100. Potrebno je napraviti metodu koja prima određeni broj, a vraća pentagonalni broj tog broja.

Pentagonalni broj se računa kao: (broj*((3*broj)-1))/2

Metoda koja računa pentagonalni broj datog broja se može napisati na sledeći način:

```
public static int pentagonalniBroj(int i) {
     int resenje = (i * ((3 * i) - 1)) / 2;
     return resenje;
}
```

Metoda prima jedan ceo broj kao parametar i, a zatim po formuli računa odgovarajući pentagonalni broj i vraća rezulutat. Povratna vrednost je takođe ceo broj.

Kada je metoda definisana, možemo je pozvati iz main metode sledećom naredbom:

```
pentagonalniBroj(broj);
```

Kako bismo rešili zadatak, metodu pozivamo za svaki broj od 1 do 100 pomoću for petlje.

Rešenje:

```
package primer1;

public class Primer1{

    public static void main(String[] args) {
       for (int i = 1; i <= 100; i++) {
            System.out.println(pentagonalniBroj(i));
        }
    }
    //Metoda koja vraca pentagonalni broj na osnovu prosledjenog broja i
    public static int pentagonalniBroj(int i) {
        int resenje = (i * ((3 * i) - 1)) / 2;
        return resenje;
    }
}
```

Primetiti static ključnu reč koja stoji pre povratne vrednosti metode. Metoda je statička kako bi mogla da se pozove pre kreiranja objekta klase Zadatak1. Takva metoda može da se pozove i iz druge statičke metode, kao što je metoda main u prikazanom rešenju.

Obrisati ključnu reč static i pogledati koju grešku prijavljuje NetBeans.

PRIMER 2

Pisanje metode za obrtanje stringa. Pretvaranje stringa u broj i obrnuto.

Napisati metod koji rotira cifre u broju. Testirati metod na 3 broja.

Primer: Ukoliko je broj 1234 treba da bude 4321.

Potpis tražene metode:

```
public static int obrniBroj(int broj)
```

Objašnjenje:

Obrtanje stringa može se izvršiti pomoću metode reverse klase StringBuilder. Ako možemo da obrnemo string, kako bismo obrnuli broj dovoljno je da znamo kako da pretvorimo string u broj i obrnuto.

String.valueOf(broj) pretvara broj u string, dok Integer.parseInt(string) pretvara string u ceo broj.

StringBuilder pravimo pomoću konstruktora koji prima string. StringBuilder možemo posle da pretvorimo u običan string pomoću metode toString();

Rešenje:

```
package primer2;

public class Primer2 {

    public static void main(String[] args) {
        new Primer2();
    }

    public Primer2() {
        System.out.println("Obrnuti broj od: 1245 je " + obrniBroj(1245));
        System.out.println("Obrnuti broj od: 2345 je " + obrniBroj(2345));
        System.out.println("Obrnuti broj od: 3455 je " + obrniBroj(3455));
    }
    /**
     * Metoda koja prima broj Pretvara ga u String pravi StringBuilder objekat
     * od string-a poziva metodu StringBuildera za obrnuti String a potom string
     * vraca u int.
     *
     * @param broj
     * @return
     */
    public static int obrniBroj(int broj) {
        return Integer.parseInt(new
StringBuilder(String.valueOf(broj)).reverse().toString());
    }
}
```

ZADATAK 1

Samostalno vežbanje kreiranja metoda

Kreirati NetBeans projekat KreiranjeMeroda po sledećim zahtevima:

- Kreirati klasu VezbanjeMetoda;
- Klasa sadrži metodu main();
- Klasa ima dva polja tipa String: ime i prezime;
- Kreirati metodu spojiUVelika() povratnog tipa String koja kao argumente uzima ime i prezime i kao rezultat vraća spojen String, sa razmakom između imena i prezimena, prikazan sa svim velikim slovima;
- Kreirati metodu spojiUMala() povratnog tipa String koja kao argumente uzima ime i prezime i kao rezultat vraća spojen String, sa razmakom između imena i prezimena, prikazan sa svim malim slovima;
- U metodi main() omogućiti učitavanje ovih stringova sa tastature, poziv navedenih metoda i prikazivanje rezultata.

Poglavlje 2

Pozivanje metoda

EFEKAT POZIVANJA METODA

Pozivom metoda počinje izvršavanje naredbi u njegovom telu, pri čemu se unose vrednosti definisane kao paremtri metoda.

Definicijom novog metoda se uspostavlja njegovo postojanje i određuje kako on radi. Ali metod se uopšte ne izvršava sve dok se ne pozove na nekom mestu u programu - tek pozivom metoda se na tom mestu izvršava niz naredbi u telu metoda. Ovo je tačno čak i za metod main() u nekoj klasi, iako se taj glavni metod ne poziva eksplicitno u programu, već ga implicitno poziva JVM na samom početku izvršavanja programa.

Generalno, poziv nekog metoda u Javi može imati tri oblika. Najprostiji oblik naredbe poziva metoda je:

```
imeMetoda(listaArgumenata)
```

Prema tome, na mestu u programu gde je potrebno izvršavanje zadatka koji neki metod obavlja, navodi se samo njegovo ime i argumenti u zagradi koji odgovaraju parametrima metoda.

Na primer, naredbom:

```
iscrtajTekst(100, 200, "Moja slika");
```

poziva se metod ***iscrtajTekst()*** za izvršavanje sa argumentima koji su navedeni u zagradi.

Obratite pažnju na to da parametar u definiciji metoda mora biti neko ime, jer je parametar konceptualno jedna promenljiva. S druge strane, argument u pozivu metoda je konceptualno neka vrednost i zato argument može biti bilo koji izraz čije izračunavanje daje vrednost tipa odgovarajućeg parametra. Zbog toga, pre izvršavanja naredbi u telu metoda, izračunavaju se izrazi koji su navedeni kao argumenti u pozivu metoda i njihove vrednosti se prenose metodu tako što se dodeljuju odgovarajućim parametrima.

Pozivom metoda se dakle izvršava telo metoda, ali sa inicijalnim vrednostima parametara koje su prethodno dobijene izračunavanjem vrednosti odgovarajućih argumenata. Na primer, poziv metoda:

```
iscrtajTekst(i + j, 2 * k, s);
```

izvršava se tako što se izračunava vrednost prvog argumenta

i + j i dodeljuje prvom parametru, zatim se izračunava vrednost drugog argumenta 2 * k i dodeljuje drugom parametru, zatim se izračunava vrednost trećeg argumenta s, čiji je rezultat aktuelna vrednost promenljive s i dodeljuje se trećem parametru, pa se s tim početnim vrednostima parametara najzad izvršava niz naredbi u telu metoda ***iscrtajTekst()***.

POZIVANJE STATIČKIH METODA

Statički metodi, koji se definišu modifikatorom static, se mogu pozivati i izvan klase u kojoj su definisani.

Statički metodi se definišu upotrebom modifikatora ***static***. Statički metodi se mogu pozivati i izvan klase u kojoj je definisan. Zato, u pozivu metoda se mora navesti imeklase u kojoj je definisan statički metod:

```
ImeKlase.imeMetoda(listaArgumenata)
```

Ovde je ***ImeKlase*** ime one klase u kojoj je metod definisan. Statički metodi pripadaju celoj klasi, pa upotreba imena klase u tačka-notaciji ukazuje na to u kojoj klasi treba naći definiciju metoda koji se poziva. Na primer, naredbom:

```
Math.sqrt(x + y);
```

poziva se statički metod sqrt() koji je definisan u klasi***Math***. Argument koji se prenosi metodu ***sqrt()*** je vrednost izraza x+y, a rezultat izvršavanja tog metoda je kvadratni koren vrednosti njegovog argumenta. Evo drugih primera poziva statićkih metoda

```
Math.sqrt( 25.0 )           //  sqrt Math klase
MyClass.main( arg[ ] )  //  glavni metod klase
Integer.parseInt("123") // konvertuje String u  int
System.exit( 0 )            // izlaz iz programa.
```

Statički metodi su servisi koje obezbeđuje klasa. Oni nemaju nikakav efekat na objekte te klase. Statički metodi se koriste za funkcije koje ne zahtevaju pristup određenom objektu.

Na slici 1 je dat primer statičkog metoda ***getSledeciBrojRacuna*** koji vraća sledeći raspoloživi broj računa. ***Klasa BankovniRacun*** sadrži poslednje izdat broj računa u formi vrednosti atributa ***sledeciBrojRacuna***. Pri pozivu, metod ***getSledeciBrojRacuna*** poveća vrednost tog atributa za jedan, i tu vrednost vrati preko naredbe ***return***, kao novu vrednost atributa

Definisanje statičkog metoda getSledeciBrojRacuna:

```
public class BankovniRacun{
    private static long sledeciBrojRacuna; // stačka promenljiva
    private long stanje;                   // promenljiva objekta
                    // statički metod vraća sledeći razpoloživ broj računa
    private static long getSledeciBrojRacuna( ) {
        sledeciBrojRacuna++;
        return sledeciBrojRacuna;
    }
```

Pozivanje statičkog metoda getSledeciBrojRacuna:

```
BankovniRacun.getSledeciBrojRacuna
```

Slika 2.1 Primer pozivanja statičkog metoda getSledeciBrojRacuna()

OGRANIČENJA STATIČKIH METODA

Statički metodi ne smeju da pristupaju nijednom članu klase, sem statičkim podacima.

Statički metodi ne mogu da direktno priđu atributima objekata ili metodima objekata neke klase. Statički metodi ne smeju da pristupaju nijednom članu klase, sem statičkim podacima. Statički metodi ne smeju da pozivaju nijedan metod objekata klase. To ćemo pokazati u sledećem primeru:

```
public class TestProgram {
    int max( int m, int n ) {  return (m>n)? m : n ;  }

    public static void main( String [] args ) {
        int n = 100;
        int m = 200;
        System.out.println("max of m, n is " + max(m,n) );
    // main je statički metod. Ne može da poziva metod objekta max
    }
} // kraj programa za testiranje
```

Slika 2.2 Nepravilno pozivanje promenljive objekta iz statičkog metoda

U ovom primeru se koristi promenljiva objekta stanje u statičkom metodu **getSledeciBrojRacuna**. Nije dozvoljeno da se u statičkom metodu koriste promenljive atributa, jer u suprotnom, to bi bilo u koliziji definicijom statičkih metoda, jer oni ne smeju da zavise od stanja obajekata. Vezani su samo za određenu klasu, te važe za sve objekte.

Metod ***main( )*** je statički metod (metod klase). Zbog toga, on ne može da poziva nijedan metod objekata dok ne kreira taj objekt. Takva greška je pokazana u sledećem primeru:

```
public class BankovniRacun {
       long stanje;
       long brojRacuna;
       String ime;
       static long sledeciBrojRacuna;   // promenljiva klase

       pubic static getSledeciBrojRacunar( ) {
              sledeciBrojRacuna++;  // OK. Ovo je static atribut
              brojRacuna = 0;  // pristupa atributu koji nije static
              return sledeciBrojRacuna
       }
       // ...itd...
```

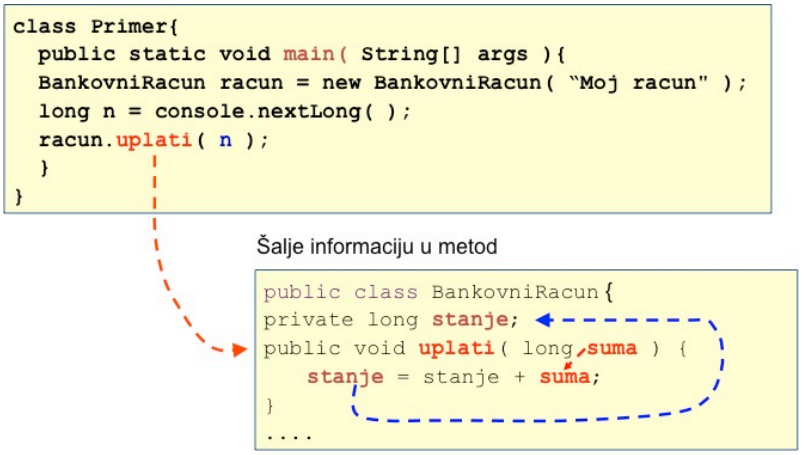
Slika 2.3 Pogrešno korišćenje obhekta u statičkom metodu pre nego što je objekat i kreiran

U gornjem primeru, u statičkom metodu main() poziva se metod objekta getReplay a objekat nije prethodno kreiran.

PRIMERI GREŠAKA U PRIMENI STATIČKIH METODA

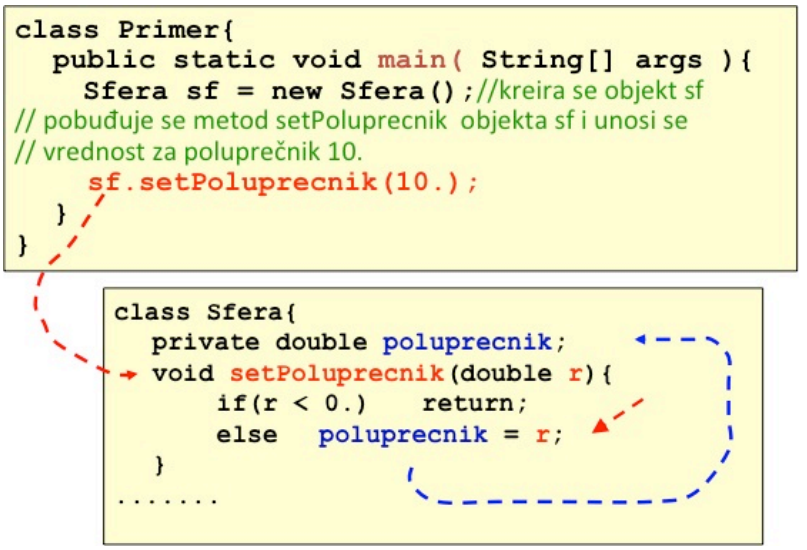
Statički metod ne može da poziva promenljive objekata niti metode objekata

Statički metod ne može da poziva ni atribut ni metod objekta. Takvu vrste greške pokazuje sledeći primer:



Slika 2.4 Greška zbog pozivanja metoda objekta iz statičkog metoda

Donji primer pokazuje pogrešno korišćenje promenljive objekta u statičkom metodu.



Slika 2.5 Greška zbog pozivanja promenljive objekta iz statičkog metoda

POZIVANJE METODA OBJEKATA

Metod objekta ima efekta samo na određenom objektu te se poziva navođenjem određenog objekta. On se definiše u okviru klase, ali važi samo sa određeni objekat te klase.

Metod objekta je deo objekta. Metod objekta je onaj koji određuje ponašanje jednog objekta. Metod objekta ne pripada celoj klasi u kojoj je definisan, te se poziva obavezno uz navođenja imena metoda kome pripada i na koji ima efekta:

```
imeObjekta.imeMetoda(listaArgumenata);
```

Ovde je ***imeObjekta*** zapravo jedna promenljiva koja sadrži referencu na onaj objekat za koji se poziva metod. Na primer, naredbom:

```
rečenica.charAt(i);
```

poziva se objektni metod ***charAt()*** u klasi String, sa vrednošću promenljive i kao argumentom, za objekat klase ***String*** na koji ukazuje promenljiva **rečenica**. Rezultat ovog poziva je i-ti znak u stringu koji je referenciran promenljivom **rečenica**.

Metodi objekta:

- definišu ponašanje objekta;
- pristupaju atributima objekta;
- mogu da pozivaju statičke metode

Na slici 2 prikazan je primer upotreba metoda objekta ***racun*** koji pripada klasi ***BankovniRacun***. U klasi ***Primer*** kreiran je objekat***racun*** klase***BankovniRacun***. Primenom metoda ***uplati*** unosi se učitana vrednost preko tastature. Metod **uplati** uvećava postojeću vrednost atributa stanje objekta sa nazivom “Moj racun” za iznos suma. Na ovaj način,

uvećava se vrednost atributa ***stanje*** samo u objektu sa nazivom “Moj racun”, jer je metod ***uplati()*** povezan samo sa određenim objektom, jer nema modifikator ***static.***

```
public class BankovniRacun {
    private static long sledeciBrojRacuna;   // statička promenljiva
    private long stanje;                     // promenljiva objekta (instance)
    private static long getSledeciBrojRacuna( ) {
        sledeciBrojRacuna++;                 // OK
        stanje = 0;                  // NEPRAVILNO
        return sledeciBrojRacuna;
    }
```

SLika-6 Pozivanje metoda uplati() objekta racun

PRIMER 3 - POZIVANJA METODA OBJEKTA

Poziv metoda iz statičkog metoda je moguć samo ako je prethodno formiran objekt čiji se metod poziva.

Na slici 7 prikazan je drugi primer podsticanja jednog metoda. U klasi **Primer** kreira se objekat sf klase Sfera. Onda se koristi njen metod **setPoluprecnik** da bi se unela vrednost poluprečnika veličine 10. Metod **setPoluprecnik** prvo proverava da nije slučajno uneta negativna vrednost za poluprečnik (što nije dozvoljeno), i ako to nije slučaj (kao u našem slučaju), vraća učitanu vrednost kao vrednost atributa poluprecnik klase **Sfera.**

Važno je uočiti da se metod objekta **setPoluprecnik()** poziva iz statičkog metoda **main(),** ali je prethodno kreiran objekat sf klase Sefera. Tek onda se može pozvati metod koji definiše vrednost poluprečnika formiranog objekta sf.

```
class Primer{
  public static void main( String[] args ){
    Sfera sf = new Sfera();//kreira se objekt sf
// pobuđuje se metod setPoluprecnik  objekta sf i unosi se
// vrednost za poluprečnik 10.
    sf.setPoluprecnik(10.);
  }
}
```

```
class Sfera{
  private double poluprecnik;
  void setPoluprecnik(double r){
      if(r < 0.)    return;
      else   poluprecnik = r;
  }
.......
```

Slika 2.6 Pozivanje metoda setPoluprecnik prethodno formiranog objekta sf klase Sfera

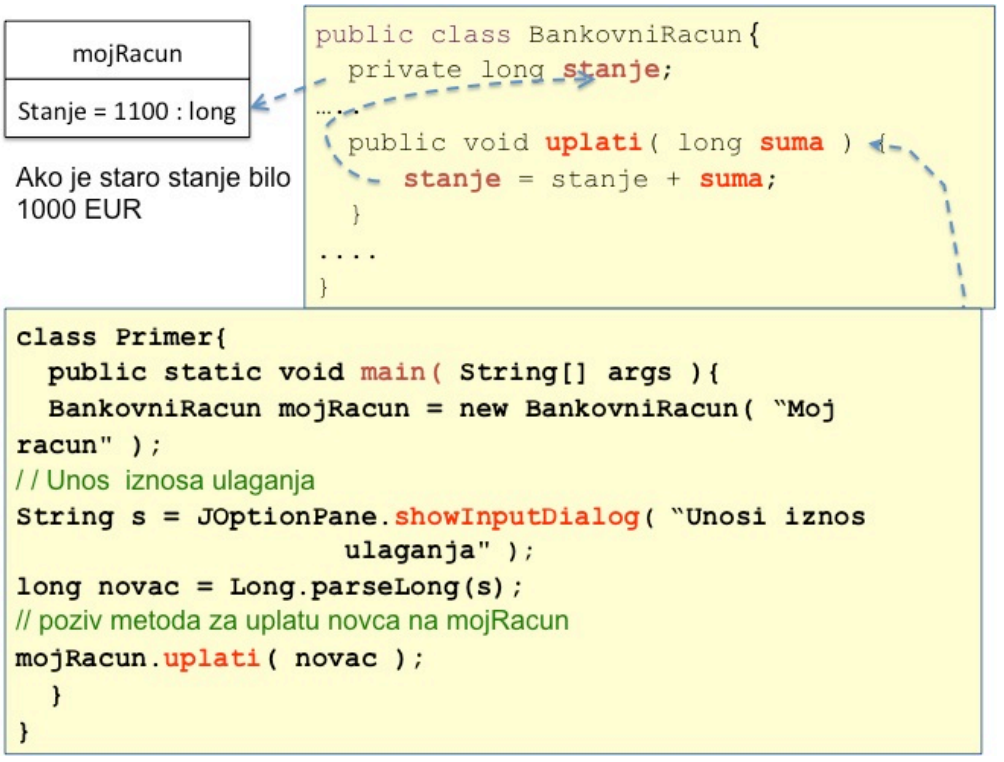
PRIMER 4 - POZIVANJA METODA OBJEKTA

Metod uplati objekta mojRacun se poziva posle kreiranja objekta mojRacun u metodu main().

Ovaj primer pokazuje kako se pozivanjem metoda uplati() vrši promena stanja na računu mojRacun.

Najpre je u okviru metoda **main()** kreiran objekat **mojRacun**, kao objekat klase **BankovniRacu**n, koji ima naziv "Moj racun". Onda je primenom metoda **showInputDialog()** klase **JOptionPane** unet iznos koji se želi da uplati na račun (npr. 100 EUR**).** Pomoću njega je ovaj niz oznaka (klase ***String***) konvertovan u broj tipa ***long*** pozivom metoda **parseLong(),** koji je statički metod klase Long. I konačno, pozivom metoda ***uplati()*** objekta **mojRacun**, vrši se povećanje vrednost njegovog atributa stanje za unet iznos /npr, za 100 EUR).

Ovde treba napomenuti da je pre poziva metod objekta **uplati(),** izvršeno kreiranje njegovog objekta **mojRacun** u statičkom metodu **main().**



Slika 2.7 Pozivom metoda uplati objekta mojRacun, povećava se se stanje na računu

PRIMER 5: RAČUNANJE SVIH DELIOCA UNETOG BROJA

Objekat Scanner poziva metodu nextInt()

Tekst zadatka:

Napisati program koji na osnovu unetog broja n ispisuje sve delioce broja n.

Na primer, ako je n = 28 treba da ispiše: 1 2 4 7 14

Program treba da bude deo NetBeans projekta pod nazivom KI103-L06 i deo paketa cs101.L06. Naziv klase treba da bude Primer4.

Za učitavanje podataka od korisnika treba koristiti objekat klase***Scanner***, a za ispis podataka objekat ***System.out***.

Programski kod:

```
package cs101.L06;

    import java.util.Scanner;

    public class Primer5 {
        public static void main(String[] args) {
            Scanner ulaz = new Scanner(System.in);
            System.out.println("Unesi broj: ");
            int broj = ulaz.nextInt();
            for (int i = 1; i <= broj / 2; i++) {
                if (broj % i == 0) {
                    System.out.print(i + " ");
                }
            }
        }
    }
```

UPOREĐENJE METODA KLASE I METODA OBJEKTA

Statički metodi su povezani sa klasom, a metodi objekta samo sa određenim objektom, iako su definisano u okviru određene klase.

Na slici 3 dato je poređenje metoda klase (statičkog metoda) i metoda objekta.

```
public class TestProgram {
      // metod objekta (instance):
      public  String getReply( ) {
            Scanner console = new Scanner( System.in );
            return console.readLine( );
      }
      public static void main( String [] args ) {
            System.out.print("Koje je Vaše ime? ");
            String ime = getReply( );    // ** GREŠKA **
            ...itd
      }
}
```

Slika 2.8 Upoređenje statičkih metoda i metoda objekata

PRIMER 6: KONVERZIJA HEKSADEKADNOG U DEKADNI BROJ

String objekat poziva metodu to.UpperCase; poziv statičke metode hex2Decimal()

Tekst zadatka:

Napisati program koji konvertuje broj iz heksadekadnog brojevnog sistema u odgovarajući broj dekadnog sistema. Kreirati posebnu metodu koja vrši konverziju.

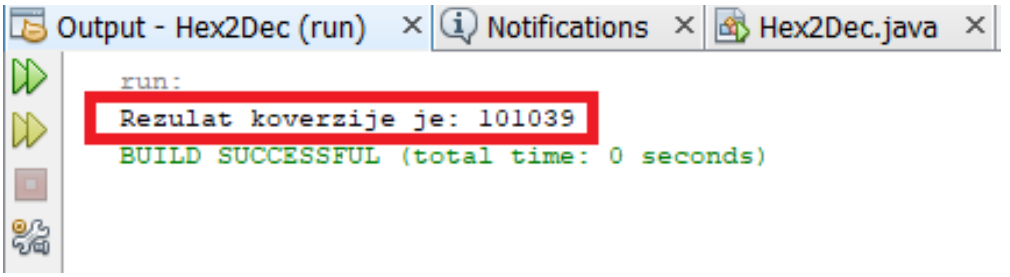
Program treba da bude deo NetBeans projekta pod nazivom Hex2Dec i **deo paketa hex2dec.** Naziv klase treba da bude **Hex2Dec.java.**

Za prikazivanje podataka od korisnika treba koristiti objekat **System.out.**

Rešenje:

Kreirana klasa sadrži metodu hex2Decimal() koja preuzima String argument (heksadekadni broj) i ukoliko u njemu postoje karakteri, koji odgovaraju slovima, prvo ih prevodi u velika slova. Povratna vrednost je inicijalizovana na nula i kroz petlju, u svakoj iteraciji, uvećava se za vrednost tekućeg karaktera uvećanog za proizvod njegove pozicije u zadatom stringu i broja 16 (osnova brojevnog sistema).

```
package hex2dec;

/**
 *
 * @author Vladimir Milicevic
 */
public class Hex2Dec {

    public static int hex2Decimal(String s) {
        String digits = "0123456789ABCDEF";
        s = s.toUpperCase();
        int val = 0;
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            int d = digits.indexOf(c);
            val = 16 * val + d;
        }
        return val;
    }

    public static void main(String[] args) {
        int decimal = hex2Decimal("18AAF");
        System.out.println("Rezulat koverzije je: " + decimal);

    }

}
```



Slika 2.9 Rezultat konverzije

DOMEN METODA

Iz unutrašnjosti klase, metod se poziva samo upotrebom njegovog imena. Van klase, upotrebite ime klase (za statičke metode) ili ime objekta (za metode objekata) da bi pozvali metod.

Svi metodi (funkcije) moraju biti deo neke klase. U Javi nema globalnih funkcija! Ovo eliminiše konflikte sa imenima. Metod **max( )** klase **MojMath** nije u konfliktu sa **max( )** metodom neke druge klase. Metod mogu da nose isti naziv, ali pošto pripadaju različitim klasama, oni jedn drugom ne ulaze u domen izvršavanja. Svaki metod se definiše samo u okviru jedne klase. Metod sa istim nazivom, a koji je deo (dvojstvo) druge klase, je neki drugi objekat. Evo primera:

```
/** Moja matematička biblioteka */
public class MojMath {
    /** max metod */
    public static double max( double x, double y)  {
        if ( x >= y ) return x;
        else return y;
    }
    //... Drugi metodi i atributi
}
```

Slika 2.10 Metod max() klase MojMath nije u konfliktu sa metodom max() klase Math

Iz unutrašnjosti klase, možete pozvati metod samo upotrebom njegovog imena. Van klase, morate upotrebiti ime klase (za statičke metode) ili ime objekta (za metode objekata) da bi pozvali metod.

```
public class TestMojMath {
    public static void main( String [] args ) {
        Scanner console = new Scanner(System.in);
        // poziva "nextDouble"  objekta console:
        double x = console.nextDouble( );
        double y = console.nextDouble( );
        // poziva "max" klase MyMath:
        double r1 = MojMath.max( x, y );
        // poziva "max"  Javine Math klase:
        double r2 = Math.max( x, y );
}
```

Slika 2.11 Poziv metoda objekta i statičkih metoda bez konflikata

PRIMER 7

Pisanje metode za prikazivanje n x m matrice slučajnih brojeva

Potrebno je napraviti program koji prikazuje n x m matricu. Korisnik unosi n i m preko grafičkog interfejsa.

Korisnik unosi koliko redova i kolona želi da matrica ima, a potom program treba da pozove metodu koja prima broj redova i kolona i ispisuje na konzolu random matricu sa tim brojem kolona i redova.

Potpis tražene metode je sledeći:

```
public static void displayMatrix(int rows, int cols)
```

Da bismo inicijalizovali ili ispisali vrednosti svih elemenata matrice, koriste se ugnježdene petlje:

```
for (int i = 0; i < rows; i++)
    for (int j = 0; j < cols; j++)
```

Prva i petlja prolazi kroz sve redove, a druga j petlja prolazi kroz sve kolone u odgovarajućem i-tom redu.

Objašnjenje:

Prolazimo kroz sve redove i kolone i ispisujemo random vrednost za svaku kolonu unutar jednog reda. Kada završimo sa svim kolonama jednog reda ispisujemo prelaz za novi red na konzolu. Tako su redovi matrice ispisani jedan ispod drugog.

Rešenje:

```
package primer7;

import java.util.Random;
```

```
import javax.swing.JOptionPane;

public class Primer7 {

    public static void main(String[] args) {
        new Primer7();
    }

    public Primer7() {
        int rows = Integer.parseInt(JOptionPane.showInputDialog("Unestie broj
redova"));
        int cols = Integer.parseInt(JOptionPane.showInputDialog("Unestie broj
kolona"));
        displayMatrix(rows, cols);
    }

    /**
     * Metoda koja preko petlje u pelji prikazuje n redova sa n kolona Random
     * nam sluzi da bi generisali random broj od 10 do 100
     *
     * @param rows
     * @param cols
     */
    public static void displayMatrix(int rows, int cols) {
        Random r = new Random();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                System.out.print((r.nextInt(90) + 10) + " ");
            }
            System.out.println("");
        }
    }
}
```

Dodati metodu koja prima samo jedan parametar n i na osnovu njega ispisuje kvadratnu matricu n x n.

PRIMER 7- NASTAVAK

Preopterećenje metode

Dodajemo metodu koja prima samo jedan parametar n, a zatim na osnovu njega ispisuje kvadratnu n x n matricu.

Primetite da metoda ima isto ime, ali različitu listu parametara, što znači da je došlo do preopterećenja metoda. Pri pozivanju metode displayMatrix, ako se prosledi jedan argument n, izvršiće se nova metoda:

```
public static void displayMatrix(int n) {
        Random r = new Random();
```

```
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print((r.nextInt(90) + 10) + " ");
            }
            System.out.println("");
        }
}
```

U main metodi testiramo metodu displayMatrix, prvo sa dva argumenta rows i cols, a zatim sa svakim pojedinačno.

Rešenje:

```
package primer7;

import java.util.Random;
import javax.swing.JOptionPane;

public class Primer7Ext {

    public static void main(String[] args) {
        new Primer7Ext();
    }

    public Primer7Ext() {
        int rows = Integer.parseInt(JOptionPane.showInputDialog("Unestie broj
redova"));
        int cols = Integer.parseInt(JOptionPane.showInputDialog("Unestie broj
kolona"));

        displayMatrix(rows,cols);
        System.out.println("");
        displayMatrix(rows);
        System.out.println("");
        displayMatrix(cols);

    }

    /**
     * Metoda koja preko petlje u pelji prikazuje n redova sa n kolona Random
     * nam sluzi da bi generisali random broj od 10 do 100
     *
     * @param rows
     * @param cols
     */
    public static void displayMatrix(int rows, int cols) {
        Random r = new Random();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                System.out.print((r.nextInt(90) + 10) + " ");
            }
            System.out.println("");
        }
```

```
    }

     public static void displayMatrix(int n) {
        Random r = new Random();
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print((r.nextInt(90) + 10) + " ");
            }
            System.out.println("");
        }
    }
}
```

PRIMER 8

Pisanje metode sa parametrima različitog tipa

Napisati metodu koji prima dva argumenta. Prvi je string skoji unosi korisnik, a drugi je broj n. Metoda treba da napravi novi string koji nastaje nadovezivanjem stringa s n puta.

Potpis metode:

```
public static String repeat(String s, int n)
```

Za nadovezivanje stringova možemo da koristimo klasu StringBuilder. Nadovezivanje na postojeći string se vrši metodom **append(string).** Na kraju od StringBuilder objekta pravimo objekat klase String sa metodom toString();

Na početku kreiramo prazan StringBuilder i zadajemo kapacitet na veličinu novog stringa, što doprinosi efikasnosti izvršavanja naše metode.

```
StringBuilder b = new StringBuilder(n * s.length());
```

```
package primer8;

import java.util.Scanner;

public class Primer8 {
     public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      System.out.println("Unesite string: ");
      String s = input.nextLine();
      System.out.println("Unesite koliko puta zelite ponavljanje: ");
      int n = input.nextInt();
      System.out.println(repeat(s,n));
      input.close();
    }
    //Metoda koja vraca novi string nastao ponavljanjem string s n puta
    public static String repeat(String s, int n) {
         StringBuilder b = new StringBuilder(n * s.length());
```

```
            for (int i = 0; i < n; i += 1) {
                b.append(s);
            }
          return b.toString();
    }
}
```

Izmeniti metod tako da se prilikom spajanja stavlja razmak između.

Napisati preopterećeni metod koji prima još jedan parametar m. Posle svakih m nadovezivanja preći u novi red.

PRIMER 9

Parsiranje jedinstvenog matičnog broja građana.

Potrebno je napraviti program koji na osnovu unetog JMBG-a prepoznaje datum rođenja i pol građanina.Treba napraviti posebnu metodu za pol (vraća String), a posebnu metodu za vraćanje datuma rođenja. Takođe treba validirati da li je korisnik uneo 13 karaktera (brojeva) za JMBG.

Format JMBG-a je DDMMGGGRRBBBK gde DD predstavlja dan rođena, MM mesec rođenja, GGG godina rođenja a BBB je pol. Ako je BBB manji od 500 onda je pol Muški dok ukoliko je veći od 500 onda je pol ženski.

Objašnjenje:

Treba koristiti metodu substring String klase kako bismo dobili određeni deo stringa. Koristimo lenght() metodu klase String za utvrđivanje dužine stringa.

Prvo proveravamo da li je dužina stringa 13 karaktera, ako to nije slučaj ispisuje se poruka o grešci. Ako je JMBG ispravan, onda se izdvajaju njegovi delovi i na osnovu njih pravi novi string.

Rešenje:

```
package primer9;

import javax.swing.JOptionPane;

public class Primer9 {

    public static void main(String[] args) {
        new Primer9();
    }

    public Primer9() {
        //JMBG JE U FORMATU DD MM GGG RR BBB K
        //BBB U JMBGU PREDSTAVLJA POL DD.MM.GGG predstavlja datum rodjenja
        String jmbg = JOptionPane.showInputDialog("Unesite JMBG");
        System.out.println(getPolNaOsnovuJMBG(jmbg));
```

```
        System.out.println(getDatumRodjenjaNaOsnovuJMBG(jmbg));
    }

    public static String getPolNaOsnovuJMBG(String jmbg) {
        if (jmbg.length() == 13) {
            int bbb = Integer.parseInt(jmbg.substring(jmbg.length() - 4,
jmbg.length() - 1));
            if (bbb < 500) {
                return "Pol: Muško";
            } else {
                return "Pol: Žensko";
            }
        } else {
            return "Loš unos matičnog broja";
        }
    }

    public static String getDatumRodjenjaNaOsnovuJMBG(String jmbg) {
        if (jmbg.length() == 13) {
            String dan = jmbg.substring(0, 2);
            String mesec = jmbg.substring(2, 4);
            String god = jmbg.substring(4, 7);
            return "Datum rodjenja je: " + dan + "." + mesec + "." + god;

        } else {
            return "Loš unos matičnog broja";
        }
    }

}
```

PRIMER 10

Pisanje metode koja bilo koji String pretvara u String sa prvim velikim slovom i ostalim malim.

Napisati metodu koja string koji prima kao argument pretvara u string u kojem je prvo slovo veliko, a sva ostala mala.

Da bismo dobili String sa svim malim slovima osim prvog prvo pretvaramo sva slova u mala, a potom uzimamo prvi karakter i pretvaramo ga u veliko slovo. Nakon ovoga spajamo taj karakter sa Stringom bez prvog velikog slova.

Potpis metode:

```
public static String dajMiStringSaPrvimVelikim(String hocuVelikoSlovo)
```

- Pretvaranje u mala slova - toLowerCase()
- Pretvaranje u veliko slovo - toUppercase()
- Izdvajanje prvog karaktera stringa - charAt(0)

- String bez prvog karaktera - substring(1)

Rešenje:

```
package primer10;

public class Primer10 {

    public static void main(String[] args) {
        new Primer10();
    }

    public Primer11() {

        System.out.println(dajMiStringSaPrvimVelikim("maRKo"));
    }

    /**
     * Sve pretvaramo u mlasa slova sa metodom toLowerCase potom vracamo prvi
     * karakter uvecan spojen sa substring-om koji je string bez prvog karaktera
     * .
     *
     * @param hocuVelikoSlovo
     * @return
     */
    public static String dajMiStringSaPrvimVelikim(String hocuVelikoSlovo) {
        String novi = hocuVelikoSlovo.toLowerCase();
        return Character.toUpperCase(novi.charAt(0)) + novi.substring(1);
    }

}
```

Izmeniti kod tako da prva dva slova budu velika, a ostala mala.

ZADATAK 2

Samostalno vežbanje poziva metoda

Kreirajte klasu koja poseduje:

- statička polja;
- nestatička polja;
- jednu statičku metodu;
- jednu objektnu metodu.
- demonstrirajte razliku u pozivu navedenih metoda.

ZADATAK 3

Razlikovanje poziva klasne i objektne metode

- Vratite se na primer *Primer 4: Konverzija heksadekadnog u dekadni broj;*
- Obeležite komentarima mesta u kodu gde je pozvana objektna, a gde klasna (statička) metoda.

Poglavlje 3

Parametri metoda

UVOD

Sadržaj ovog objekta učenja

U ovom odeljku govorićemo o

- prenosu parametara
- konverziji tipova promenljivih,
- Lokalnim promenljivim.

3.1 Prenos parametara

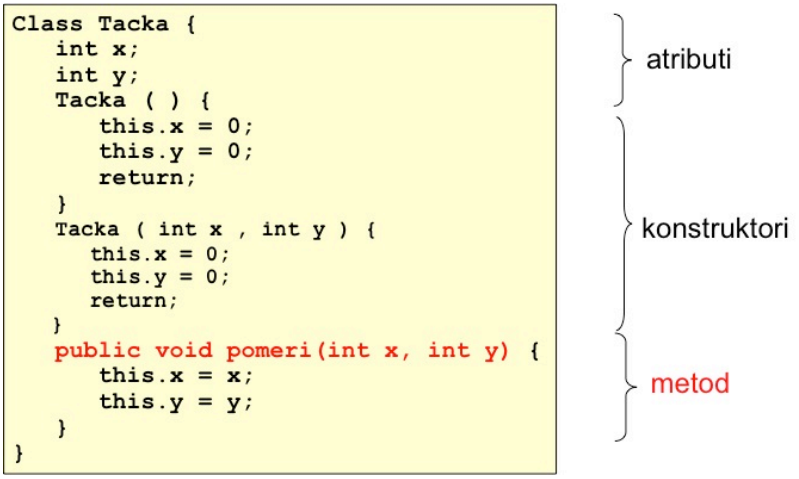
PRENOS VREDNOSTI PREKO PARAMETARA

Vrednosti promenljivih u metodu se unose u metod preko njegovih parametara.

Parametri modela obezbeđuju argumente koji nose početne vrednosti promenljivih metoda. Metod može i ne mora da ima parametre. Parametri modela se uose između malih zagrada iza imena metoda:

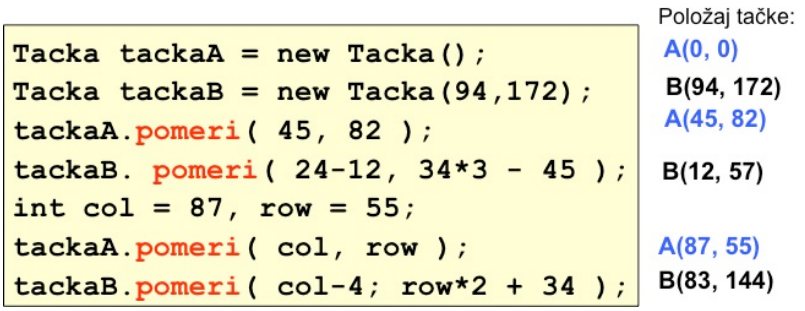
modifikatori tipRezultata imeMetoda(listaParametara)

Na slici 1 prikazana je klasa **Tacka**, koja sadrži metod **pomeri(),** čijim pozivom vrši se pomerenja u novu poziciju definisanu unetim koordinatama x i y..



Slika 3.1.1 Klasa Tacka sa svojim atributima, konstruktorima i metodom pomeri()

Klasa **Tačka** se koristi da se definiše objekat čiji se metod poziva, kao i metod koji treba pozvati. Lista argumenata snabdeva metod potrebnim podacima. Evo nekoliko primera:



Slika 3.1.2 Deo programa u kome se pozivom metoda pomeri() pomeraju tačke A i B

Ukoliko su argumenti dati u vidu matematičkih operatora, tj. izraza, onda se prvo oni izračunavaju, a onda se izvršava metod, u gornjem slučaju, metod **pomeri().**

NAČIN PRENOSA PARAMETARA PO VREDNOSTI

Promene koje nastaju u metodu usled unetih argumenata ostaju ograničene samo u okviru metoda.

Ako je tip argumenta koji se šalje u metod neki od primitivnih tipova, onda promene koje napravite nad tim argumentom u metodu, nemaju uticaja izvan samog metoda. Taj argument je lokalna kopija onog što je prosleđeno u metod. Promene utiču samo na tu lokalnu kopiju. To je pokazano u premeru koji je ovde prikazan.

Evo kako funkcioniše prosleđivanje po vrednosti:

- Kada se metod pozove, na mestu poziva se zadaje lista vrednosti (stvarne vrednosti argumenata).
- Kada pozvani metod počne sa radom, te vrednosti se vezuju za listu imena (formalni argumenti) koja je definisana u metodu.

- Pozvani metod koristi te vrednosti u svom radu.
- Pozvani metod ne može da preko argumenata, pošalje vrednost nazad do mesta poziva.

```
class BancinRacun {
  private int   stanje;
  . . . .
  void  uplati( int suma) {  // ovde počinje domen promenljive suma
    stanje = stanje + suma;
    // ovde završava domen promenljive suma
  }
  void prikazi() {
    System.out.println(balance+"\t" + suma); //greška  sintakse
  }
}
class CheckingTester{
  BancinRacun act;
  public static void main ( String[] args )  {
    int suma = 5000;    // sredstva koja se uplaćuju
    act = new Racun( "123-345-99",   "Mirko Petrovic",  100000 );
    // štampa se "5000"
    System.out.println( "suma:" + suma );
    // poziva se metod  uplati  sa sumom 5000
    act.uplati(suma);
    // štampa "5000" - suma je nepromenjena , jer je van domena
    System.out.println( " suma:" + suma);
  }
}
```

PRENOS REFERENCI OBJEKATA

Kao parametar, prosleđuje se referenca objekta (adresa), a ne sam objekat.

Argumenti metoda mogu biti i objekti. Prosleđivanje argumenata funkcioniše slično kao kod prosleđivanja promenljivih primitivnog tipa. Razlika je u tome što se **sada šalju reference na objekat**. Pošto pozvani metod ima referencu na objekat, on može da pristupa tom objektu i da ga koristi.

```
class ObjectPrinter{

  public void print( String st )  {

    System.out.println("Vrednost    parametra: " + st );

  }

}

class OPTester{
```

```
  public static void main ( String[] args )  {
    String poruka ="Samo jedan objekat";
    ObjectPrinter op=new ObjectPrinter();
    System.out.println("Prva vrednost poruke:   " + poruka);
    op.print(poruka);
    System.out.println("Druga vrednost          poruke: " + poruka);
  }
}
```

Sva tri puta će se odštampati tekst "Samo jedan objekat". To je i logično očekivati jer sva tri puta pristupamo istom objektu.

U prethodnom primeru, kada je u metod print poslat objekat poruka, u stvari je poslata referenca na objekat tipa String. Iako se pravi kopija reference i ona šalje u metod, u pitanju je referenca koja ukazuje na isti objekat.

Pravilo: Ako su objekti argumenti metoda, oni se u metod šalju po referenci. To znači da se u metodu može da pristupi originalnom objektu. Promene stanja objekta u metodu dovode do promena i u originalnom objektu, koji je postojao na mestu poziva. Ovo ne važi za argumente koji su primitivni tipovi.

PRIMER 11- PRENOS REFERENCI OBJEKATA

Prenos objekata kao argumenti vrši se korišćenjem njihovih referenci (adresa), a ne samih objekata.

Pogledajmo sada jedan primer u kojem se u metodu menja stanje objekta koji je poslat kao argument (slika 9.16).

Ako se ovaj program izvrši, dobija se sledeći izlaz:

x = 3; y = 5

Ulaz u DupliranjeTacke

x = 3; y = 5

x = 6; y = 10

Napustanje metoda dvaPuta

x = 6; y = 10

Kao što se vidi u metodu **dvaPuta**, klase **DupliranjeTacke**, došlo je do promene stanja objekta koji je poslat kao argument. Ta promena stanja je promenila originalan objekat (to je ustvari, isti objekat), što se vidi iz poslednjeg štampanja.

Stvar je programera da li će u metodu menjati objekat koji je prosleđen kao argument. Ponekad to može biti potrebno, a ponekad ne. Programer samo treba da bude svestan da te promene utiču i na originalan objekat i da se ponaša u skladu sa tim.

```
class Tacka{
  public int x=3, y=5 ;
  public void print(){
    System.out.println("x = " + x + "; y = " + y );
  }
}
class DupliranjeTacke{
...../// kreirana je referenca ka tački t sa t = new Tacka()
  public void dvaPuta( Tacka t )  {
    System.out.println("Ulaz u DupliranjeTacke");
    t.print() ;
    t.x = t.x * 2 ;
    t.y = t.y * 2 ;
    t.print() ;
    System.out.println("Napuštanje metoda dvaPuta");
  }
}
class Test{
  public static void main ( String[] args )  {
    Tacka t = new Tacka ();
    DupliranjeTacke dbl = new DupliranjeTacke ();
    t.print();
    dbl.dvaPuta( t );
    t.print();
  }
}
```

PRIMER 12

Metoda može da ima varijabilan broj argumenata

Napraviti program koji može da sabere n brojeva koristeći varijabilnu metodu za sabiranje. Ova metoda bi trebala da primi od 0 do n double vrednosti i da ih sabere.

Objašnjenje:

Kada pravimo varijabilne metode u atribute promenjive stavljamo *Tip podataka* zatim tri tačke, a potom naziv parametra.

Potpis metode sa varijabilnim brojem doubleargumenata je sledeći:

```
double saberiBrojeve(double... params)
```

Nakon toga možemo proći kroz parametre tako što ćemo for petlji reći da prođe svaku vrednost tog tipa podataka u varijabilnom delu i dodati vrednost na rezultat. Na kraju metoda vraća rezultat.

Rešenje:

```
package primer12;
public class Primer12 {

    public static void main(String[] args) {
        new Primer12();
    }

    public Primer12() {
        System.out.println("Rezultat: " + saberiBrojeve(1, 2, 34, 34.3));
        System.out.println("Rezultat: " + saberiBrojeve());
        System.out.println("Rezultat: " + saberiBrojeve(23, 23, 11, 12, 32, 22.2));
    }

    /* Ovde imamo primer varijabilne metode. Ova metoda moze da primi n double
     * vrednosti Zatim prolazimo kroz sve double vrednosti koje su unete i
     * sabiramo ih Rezultat vracamo *
     */
    public static double saberiBrojeve(double... params) {
        double rezultat = 0;
        for (double i : params) {
            rezultat += i;
        }
        return rezultat;
    }
}
```

Izmeniti metod tako da nadovezuje prosleđene stringove umesto što sabira brojeve.

PRIMER 13

Izmena stanja objekta u metodi. Prenos po referenci.

Napisati metodu koja kao argument dobija bojui povećava njenu B komponentu za 50.

Napomena: Metoda ne treba da pravi novu boju već da izmeni vrednost u postojećoj.

Koristimo našu klasu za definisanje boje. Više o pravljenju klasa će biti u lekciji 8.

```
package zadatak8;

public class Boja{
    public int r,g,b;

    @Override
    public String toString() {
        return r +" "+ g +" "+ b;
    }

}
```

Pravimo objekat klase Boja u main metodi. Zatim pozivamo naš metod. Boja se u metod prenosi po referenci, što znači da izmene koje izvršimo nad bojom u metodi možemo da

vidimo i u main-u. Štampamo boju pre i posle pozivanja metode moreBlue. Vidimo da izmena vrednosti **boja.b** u okviru moreBlue metode utiče na vrednost **color.b** u main metodi. Ove dve promenljive, boja i color su reference na isti objekat.

Rešenje:

```
package primer13;
import java.util.Random;

public class Primer13 {

    public static void main(String[] args) {
        new Primer13();
    }

    public Primer13() {
        Random r = new Random();
        Boja color = new Boja();
        color.r = r.nextInt(255);
        color.g = r.nextInt(255);
        color.b = r.nextInt(255);

        System.out.println(color);

        moreBlue(color);

        System.out.println(color);
    }

    public static void moreBlue(Boja boja){
        int novaVrednost = boja.b + 50;
        if(novaVrednost > 255){
            novaVrednost = 255;
        }
        boja.b = novaVrednost;
    }

}
```

ZADATAK 4

Samostalno vežbanje predavanja varijabilnog broja argumanata metodi.

- Modifikujte prethodni primer;
- Metoda vraća String i preuzima argumente tipa String;
- Poziv metode bi trebalo da bude u nekom od sledećih formata:

```
System.out.println("Rezultat: " + spojiStringove("1"," 2", "3","4"));
System.out.println("Rezultat: " + spojiStringove("Vlada","Milicevic",
"Kragujevac","Srbija"));
```

- Prevedite, pokrenite program i pratite rezultate izvršavanja poziva metoda.

3.2 Konverzija tipova

KONVERZIJA TIPOVA REZULTATA METODA

Koverzija tipova se dešava kada se pre početka rada metoda neki od argumenta konvertuju u traženi tip. Konverzija može biti eksplicitna i implicitna.

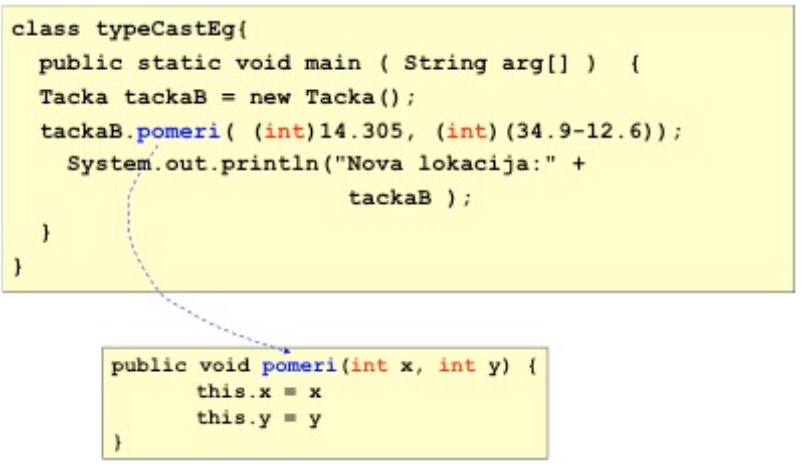
Kada metod počne sa radom, on mora imati tačan broj parametara i svaki parametar mora biti traženog tipa. Ponekad se dešava da se pre početka rada metoda neki od argumenata konvertuju u traženi tip. Ta konverzija se može desiti na jedan od dva načina:

- **Eksplicitno,** kada programer traži da se vrednost konvertuje, preko operatora konverzije.
- **Implicitno,** kada kompajler može da uradi konverziju, bez gubitka informacija, ili sa vrlo malim gubitkom preciznosti; on će to i uraditi.

Operator eksplicitne konverzije radi na sledeći način:

```
(traženiTip)(izraz)
```

(traženiTip) je nešto poput***(int).*** (izraz) je običan izraz. Ako je to samo jedna promenljiva, ne morate je stavljati u zagrade. Sledeći primer na slici 3 pokazuje kako se koristi eksplicitna konverzija. Na engleskom se za eksplicitnu konverziju tipova koristi termin ***type castin***g



Slika 3.2.1 Primer konverzije promenljivih tipa real u tip int

U ovom primeru je potrebna eksplicitna konverzija za oba argumenta, pošto se konvertuje realan broj u ceo broj. Kod konverzije realnog broja u ceo broj, decimalni deo se odbacuje (ne zaokružuje se).

U prethodnom primeru je konverzija realnog broja u ceo broj dovodila do gubitka informacija, tako da je programer morao da eksplicitno zatraži konverziju.

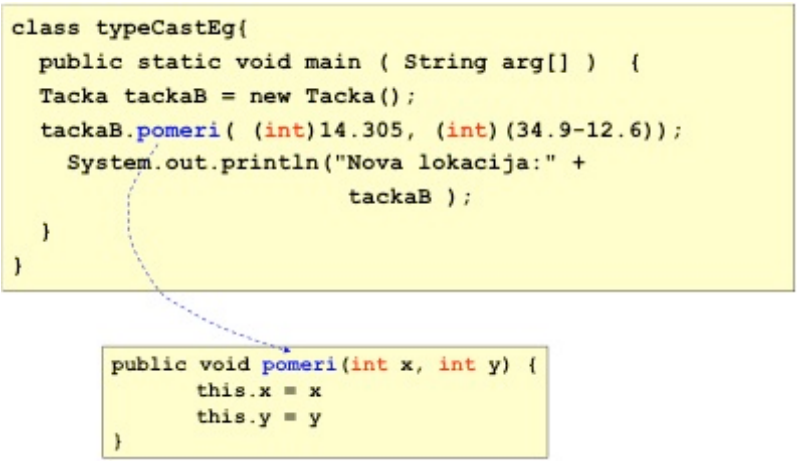
PRIMER 14 - IMPLICITNA KONVERZIJA TIPOVA PROMENLJIVIH

Kada se konverzija iz jednog u drugi tip može obaviti bez gubitka informacija, kompajler to radi automatski.

Kada se konverzija iz jednog u drugi tip može obaviti bez gubitka informacija, kompajler to radi automatski. Na primer, metod move() traži dva parametra tipa int:

```
public void pomeri(int x, int y);  // menja se (x,y) objekta tačka
```

Vrednost tipa intsadrži 32 bit-a. Vrednost tipa ***short*** ima 16 bit-ova i može se konvertovati u 32-bitnu vrednost bez gubitka informacija. Program na slici 4 će prema tome, raditi kako treba. Promenljive a i b se ne menjaju. One su samo upotrebljene da ukažu na vrednosti koje se konvertuju.



Slika 3.2.2 Primer implicitne konverzije, sa automatskom konverzijom argumenata

Pravila konverzije:

Generalno govoreći, kada postoji mogućnost da se informacije izgube, konverzija iz jednog tipa u drugi se ne može obaviti automatski. Konverzija tipa koji koristi N bit-ova u podatak koji koristi manje od N bit-ova nosi rizik gubljenja informacija i neće se obaviti automatski. Kompajler odluku o konverziji donosi na osnovu tipa, a ne na osnovu vrednosti koje su tog trenutka prisutne.

Kompajler će automatski izvršiti konverziju u sledećim slučajevima:

- ako se konvertuje celobrojni tip u drugi celobrojni tip koji koristi više bit-ova;
- ako se konvertuje realan broj u drugi realan broj koji koristi više bit-ova;

- ako se konvertuje celobrojni tip u realan broj koji koristi isti broj bit-ova, može da dođe do gubitka preciznosti, ali se to ipak radi automatski;
- ako se konvertuje celobrojni tip u realan tip sa više bit-ova.

ZADATAK 5

Samostalno vežbanje konverzije tipova rezultata metoda

- Kreirajte klasu SabiranjeBrojeva;
- Kreirajte dva polja tipa double: prvi i drugi;
- Kreirajte metodu ove klase na sledeći način:

```
public static int saberi (int a, int b)
```

- Na koji način je moguće sabiranje vrednosti prvi i drugi pozivom metode saberi()?
- Implemetirajte ovaj poziv u klasu;
- Pokrenuti klasu i pratiti rezultate izvršavanja.

3.3 Lokalne promenljive

ŠTA SU LOKALNE PROMENLJIVE

Lokalne promenljive su promenljive koje se definišu unutar samog metoda

Može se desiti da je za rad metoda potrebno da deklarišete i neke nove promenljive. Promenljive koje se definišu unutar samog metoda se nazivaju lokalnim promenljivama. Domen lokalnih promenljivih je metod u kojem su deklarisane promenljive. Izvan tog metoda im se ne može pristupiti. Te promenljive postoje samo u toku izvršavanja metoda. Posle napuštanja metoda one se brišu iz memorije.

U prethodnom primeru je u metodu skiniSaRacuna() deklarisana lokalna promenljiva skini, koja se koristi samo u tom metodu (označava cenu usluga banke). Oblast važenja te promenljive je samo metod skiniSaRacuna().

U okviru metoda možete definisati lokalnu promenljivu sa istim imenom, kao što je neka promenljiva deklarisana na nivou objekta (slika 1). Skoro uvek je greška da se koristi lokalna promenljiva sa istim imenom kao promenljiva definisana na nivou klase

Lokalne promenljive su promenljive koje se definišu unutar samog metoda. One su neophodne za rad metoda. One su deklarisane unutar domena metoda. Van njega, njima se ne može prostupiti. One postoje samo u toku izvršenja programa u glavnoj memoriji računara. Posle napuštanja metoda se briše iz memorije

.

```
class Racun{   . . . .
  private int stanje;
  . . . .
  void uplati( int suma){
      // ovde počinje domen promenljive suma
     stanje = stanje + suma;
     // ovde završava domen promenljive suma
  }
```

Slika 3.3.1 Lokalna promenljiva suma i atribut stanje

Ovo nije sintaktička greška (iako je verovatno logička greška). Kompajler će ovaj kod iskompajlirati. Druga deklaracija promenljive stanje (plave boje) kreira lokalnu promenljivu u metodu **uplati().** Domen ove promenljive počinje njenom deklaracijom i završava se na kraju metoda. To znači da se u narednom redu koristi lokalna promenljiva, a ne promenljiva objekta

Pravilo: Skoro uvek je greška, ako se koristi lokalna promenljiva sa istim imenom kao promenljiva definisana na nivou klase. Ipak, ovo nije sintaktička greška, tako da vam kompajler neće pomoći u otkrivanju takvih grešaka.

Atribut stanje važi u okviru klase, a suma je lokalna promenljiva i važi samo u okviru metoda.

DOMEN LOKALNO PROMENLJIVIH, PARAMETARA I ATRIBUTA

Domen je prostor u programu u kome se prepoznaje ime promenljive

Domen (engl. scope) je prostor u programu u kome se prepoznaje ime promenljive (ili bilo koji identifikator). Svaki programski jezik ima svoja pravila za određivanje opsega. U Javi...

- Za atribute domen je cela klasa, ali se može prekriti nekom lokalnom promenljivom ili parametrom koji imaju isto ime.
- Za parametre metoda : domen je metod (važi samo unutar metoda)
- Kod lokalna promenljive domen se kreće od tačke u kojoj se deklariše do kraja { ... } bloka u kome je deklarisana.

Argumenti koji se šalju u metod su vidljivi samo u metodu za koji su deklarisani. Svaka promenljiva ima svoj domen, odnosno oblast važenja. Oblast važenja argumenata metoda je samo taj metod.

Domen atributa je cela klasa, bez obzira gde se deklariše atribut.

Preporučljivo je da se atributi definišu na početku definisanja klase.

```
public class BankovniRacun {
  private String brojRacuna
  private static long kamata; /
  **kreira novi objekt za bankovni racun
   */
  public BankovniRacun( String
    broj,
    String ime )   {
    brojRacuna = broj;
    imeRacuna = ime;
    stanje = 0;
  }
  // možete definisati atribute bilo gde
  private long stanje;
  private String brojRacuna;
  private int domicilnaFlijala;
}
```

Slika 3.3.2 Atributi se mogu definisati bilo gde u klasi, ali je preporučljivo na početku

ARGUMENTI METODA

Domen (engl. scope) je prostor u programu u kome se prepoznaje ime promenljive

Argumenti koji se šalju u metod su vidljivi samo u metodu za koji su deklarisani. Svaka promenljiva ima svoj domen, odnosno oblast važenja. Oblast važenja argumenata metoda je samo taj metod. Ako pogledamo sledeći primer na slici 5:

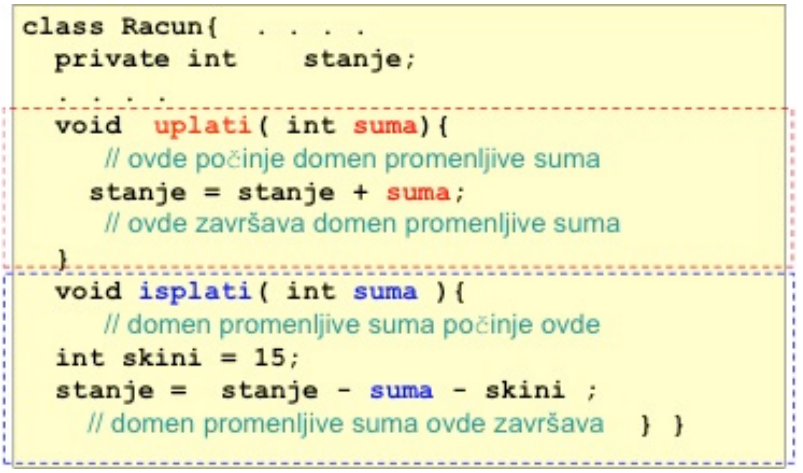
```
class Racun{   . . . .
  private int stanje;
  . . . . .
  void  uplati( int suma){
          // ovde počinje domen promenljive suma
    stanje = stanje + suma;
          // ovde završava domen promenljive suma
  }
  void prikazi() {
  System.out.println(stanje+"\t" + suma);
  // sintaktička greška (korišćenje lokalne promenljive
  // suma van domena, Java VM ovde ne zna šta je suma )
  }
}
```

Slika 3.3.3 Primer sintaktičke greške zbog korišćenja promenljive metoda van domena metoda

U metodu prikazi ne može se pristupiti argumentu suma, pošto je on izvan domena te promenljive. Kompajler neće kompajlirati ovaj program

U okviru klase može postojati više metoda sa istim argumentima, što je dozvoljeno, jer se domeni ovih promenljivih ne preklapaju (slika 5)

Dva metoda koriste isti identifikator, suma, za dva različita argumenta. Svaki metod ima svoj argument potpuno odvojen od drugog metoda. Domeni se ne preklapaju, tako da iskazi iz jednog metoda ne mogu da vide promenljivu – iz drugog.



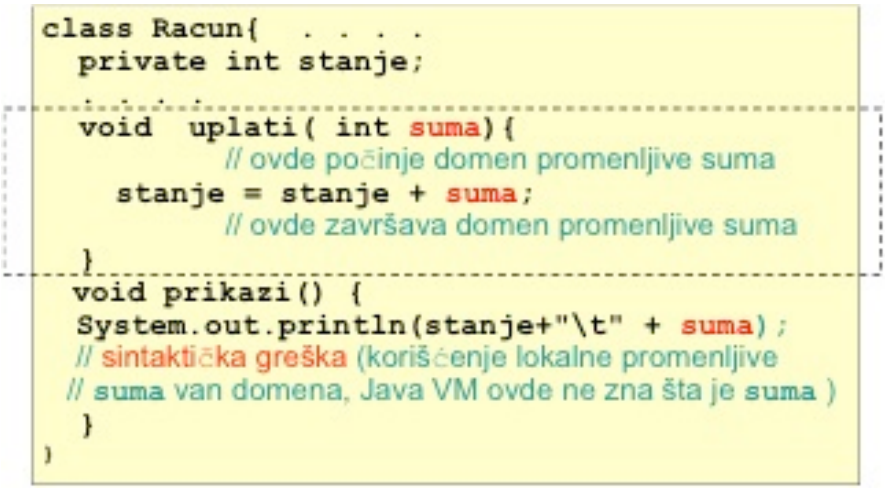
Slika 3.3.4 Korišćenje istih naziva argumenata u različitim metodima

DOMEN ATRIBUTA

Ključna reč this povezuje atribut i lokalnu promenljivu.

Unutar metoda, lokalna promenljiva ili parametar može da “prekrije” (shadow) neki atribut. U tom slučaju, poziv atributa se vrši upotrebom “this”:

this.imeAtributa



Slika 3.3.5 Upotreba this.imeAtributa

Opseg za parametre je ceo metod. Parametar može da “prekrije” neki atribut koji ima isto ime. U Javi, lokalna promenljiva ne sme da ima isto ime kao neki parametar.

```
class Racun{  . . . .
  private int    stanje;
  . . . .
  void  uplati( int suma){
   // ovde pocinje domen promenljive suma
     stanje = stanje + suma;
   // ovde završava domen promenljive suma
  }
  void isplati( int suma ){
      // domen promenljive suma počinje ovde
  int skini = 15;
  stanje =  stanje - suma - skini ;
    // domen promenljive suma ovde završava  } }
```

Slika 3.3.6 Parametar metoda može da prekrije neki atribut sa istim imenom

DOMEN LOKALNIH PROMENLJIVIH

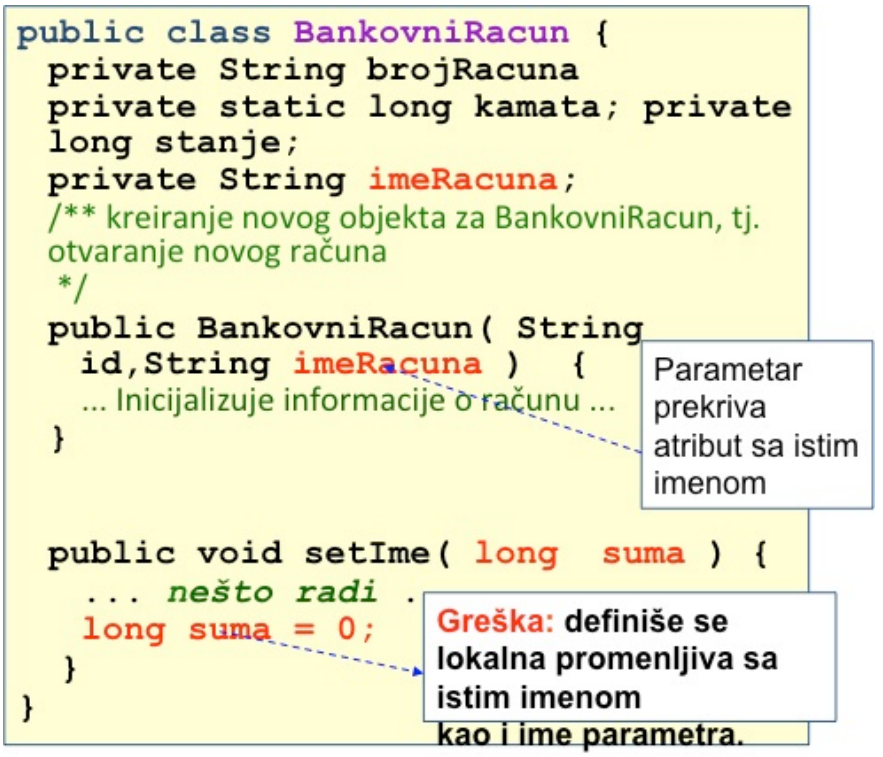
Domen lokalne promenljive važi od tačke u kojoj je defininisana, pa do kraja bloka.

Domen lokalne promenljive važi od tačke u kojoj je defininisana, pa do kraja bloka { ... }, kao što je pokazano na donoj slici.

```
public class BankovniRacun {
  private String brojRacuna;
  private static long kamata;
  private long stanje;
  private String imeRacuna;

 public void setStanje ( long
  stanje )  {
  // parametar stanje prekriva
  // atribut stanje.

  if ( stanje >= 0 )
    this.stanje = stanje;
  }
}
```

Slika 3.3.7 Domen lokalno promenljivih

Donja slika pokazuje primer primene lokalne promenljive x van njenog domena, što je greška.



Slika 3.3.8 Promenljiva x se koristi van dozvoljenog domena

GLOBALNE I LOKALNE PROMENLJIVE U JAVI (VIDEO)

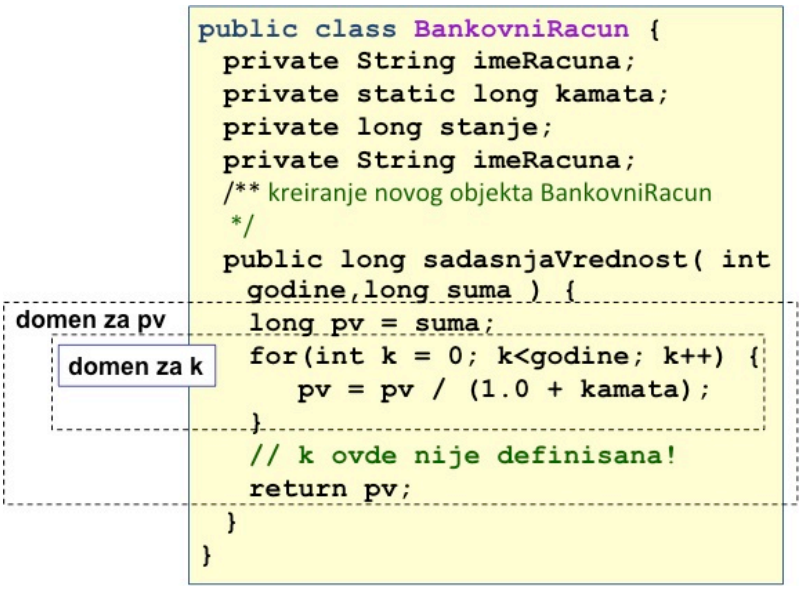
Video objašnjava razliku globalnih i lokalnih promenljivih u Javi.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PRIMER 15 - ISPRAVNO I NEISPRAVNO KORIŠĆENJE LOKALNIH PROMENLJIVIH

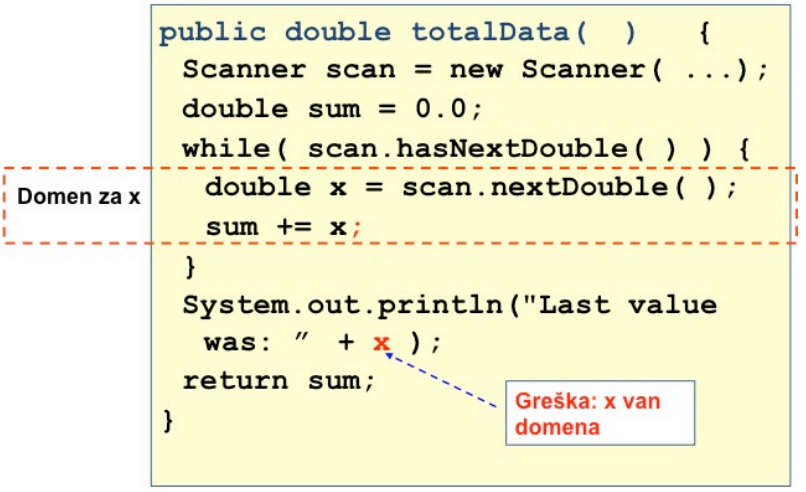
Lokalna promenljiva se može koristiti samo unutar metoda, od mesta definisanja do kraja bloka metoda.

Donja slika pokazuje grešku u korišćenju lokalne promenljive. Promenljiva suma se koristi van dozvoljenog domena.



Slika 3.3.9 Promenljiva suma se koristi van dozvoljenog domea

Ako se promenljiva sa istim imenom koristi u dva odvojena metoda u okviru iste klase, onda je to dozvoljeno, jer se svaka od njih koristi isključivo unutar svog domena, a to je domen metoda u kome su definisane. To ilustruje donji primer:



Slika 3.3.10 Dozvoljeno korišćenje promenljive suma u dva metoda

ZADATAK 6

Samostalno vežbanje ispravnog korišćenja lokalnih promenljivih

Šta će biti rezultat izvršavanja ovog koda?

```
class Test {

    static double rezultat = 0;

    public static double saberiBrojeve(double... params) {
```

```
        for (double i : params) {
            rezultat += i;
        }
        return rezultat;

    }

    public static void main(String[] args) {

        System.out.println(rezultat);
        System.out.println(saberiBrojeve(1.2,2.3,3.4));
    }
}
```

Šta nije u redu u sledećem listingu?

```
class Test {



    public static double saberiBrojeve(double... params) {
        double rezultat = 0;
        for (double i : params) {

            rezultat += i;
        }
        return rezultat;

    }

    public static void main(String[] args) {

        System.out.println(rezultat);
        System.out.println(saberiBrojeve(1.2,2.3,3.4));
    }
}
```

Poglavlje 4

Vraćanje rezultata metoda

VRAĆANJE JEDNE VREDNOSTI

Metod može vraćati jednu vrednost koja se dobija kao rezultat poziva metoda, a ako ne vraća ništa onda koristi kljunu reč void.

Metod može vraćati **jednu vrednost** koja se dobija kao rezultat poziva metoda. Ako je to slučaj, onda vraćena vrednost metoda mora biti onog tipa koji je naveden kao tip rezultata u definiciji metoda.

Druga mogućnost je da metod ne vraća nijednu vrednost. U tom slučaju se tip rezultata u definiciji metoda sastoji od rezervisane reči ***void***. Primetimo da metod u Javi može vraćati najviše jednu vrednost. Pozivi metoda koji vraćaju jednu vrednost se navodi tamo gde je ta vrednost potrebna u programu. To je obično na desnoj strani znaka jednakosti kod naredbe dodele, ili kao argument poziva drugog metoda ili unutar nekog složenijeg izraza. Pored toga, poziv metoda koji vraća logičku vrednost može biti deo logičkog izraza kod naredbi grananja i ponavljanja. Dozvoljeno je navesti poziv metoda koji vraća jednu vrednost i kao samostalnu naredbu, ali u tom slučaju se vraćena vrednost prosto zanemaruje.

U definiciji metoda koji vraća jednu vrednost mora se navesti, pored tipa te vrednosti u zaglavlju metoda, bar jedna naredba u telu metoda čijim izvršavanjem se ta vrednost upravo vraća. Ta naredba se naziva naredba povratka i ima opšti oblik

```
return izraz;
```

Ovde se tip vrednosti izraza iza službene reči return mora slagati sa navedenim tipom rezultata metoda koji se definiše. U ovom obliku, naredba return ima dve funkcije: vraćanje vrednosti izraza i prekid izvršavanja pozvanog metoda return izraz u telu pozvanog metoda se odvija u dva koraka:

1. Izračunava se navedeni izraz u produžetku na uobičajeni način.
2. Završava se izvršavanje pozvanog metoda i izračunata vrednost izraza i kontrola toka izvršavanja prenose se na mesto u programu gde je metod pozvan.

:

PRIMER 16: PRORAČUN OBIMA PRAVOUGAONOG TROUGLA

Definisanjem metoda se ništa ne izvršava, nego se tek pozivanjem metoda, u tački programa gde je potreban rezultat metoda, izvršavaju naredbe u telu metoda.

Da bismo bolje razumeli sve aspekte rada sa metodima, razmotrimo definiciju jednostavnog metoda za izračunavanje obima pravouglog trougla ukoliko su date obe katete trougla:.Preciznije, izvršavanje naredbe:

```
double obim(double a, double b) {
        double c = Math.sqrt(a * a + b * b);
        return a + b + c;
    }
```

Iz zaglavlja metoda obim() se može zaključiti da taj metod ima dva parametra a i b tipa double koji predstavljaju vrednosti kateta pravouglog trougla. Tip rezultata metoda obim() je takođe double, jer ako su katete realne vrednosti, onda i obim u opštem slučaju ima realnu vrednost. Primetimo i da nije naveden nijedan modifikator u zaglavlju ovog metoda, jer to ovde nije bitno i samo bi komplikovalo primer. Podsetimo se da se u tom slučaju smatra da je metod objektni i da se može pozivati u svim klasama iz istog paketa u kome se nalazi klasa koja sadrži definiciju metoda. U telu metoda obim() se najpre izračunava hipotenuza pravouglog trougla po Pitagorinoj formuli, a zatim se vraća rezultat metoda kao zbir kateta i hipotenuze pravouglog trougla.

Kao što je ranije napomenuto, definisanjem metoda se ništa ne izvršava, nego se tek pozivanjem metoda, u tački programa gde je potreban rezultat metoda, izvršavaju naredbe u telu metoda uz prethodni prenos argumenata. Zbog toga, pretpostavimo da se na nekom mestu u jednom drugom metodu u programu izvršavaju sledeće dve naredbe:

```
double k1 = 3, k2 = 4;
    double O = obim(k1, k2);
```

Prvom naredbom deklaracije promenljivih k1 i k2 se, naravno, rezerviše memorijski prostor za promenljive k1 i k2 i u odgovarajućim memorijskim lokacijama se upisuju vrednosti, redom, 3 i 4.

Drugom naredbom se rezerviše memorijski prostor za promenljivu O, a zatim joj se dodeljuje izračunata vrednost izraza koji se nalazi na desnoj strani znaka jednakosti. U okviru ovog izraza se nalazi poziv metoda **obim()** tako da je vrednost tog izraza, u stvari, vraćena vrednost poziva metoda **obim().** Poziv tog metoda se dalje izvršava na način koji smo prethodno opisali: vrednosti argumenata u pozivu metoda se dodeljuju parametrima metoda u njegovoj definiciji, a sa tim inicijalnim vrednostima parametara se izvršava telo metoda u njegovoj definiciji.

DOPUNSKO RAZMATRANJE O NAREDBI RETURN

Ukoliko metod vraća jednu vrednost, u njegovom telu se mora nalaziti bar jedna naredba povratka u punom obliku: return izraz

U konkretnom slučaju dakle, argumenti poziva metoda obim() su promenljive k1 i k2, pa se njihove trenutne vrednosti 3 i 4 redom dodeljuju parametrima ovog metoda a i b. Zatim se sa ovim inicijalnim vrednostima parametara a i b izvršava telo metoda obim(). Kontrola toka izvršavanja se zato prenosi u telo metoda obim() i redom se izvršavaju sve njegove naredbe dok se ne naiđe na naredbu return. To znači da se najpre izračunava hipotenuza izrazom Math.sqrt(a * a + b * b) čija se vrednost 5 dodeljuje promenljivoj c. Zatim se izvršava naredba return tako što se izračunava izraz u produžetku a + b + c i dobija njegova vrednost 12. Nakon toga se prekida izvršavanje metoda obim() i izračunata vrednost 12 i kontrola izvršavanja se vraćaju tamo gde je taj metod pozvan. To znači da se nastavlja izvršavanje naredbe dodele O = obim(k1, k2) koje je bilo privremeno prekinuto radi izvršavanja poziva metoda obim() na desnoj strani znaka jednakosti. Drugim rečima, vraćena vrednost 12 tog poziva se konačno dodeljuje promenljivoj O.

Iako u ovom primeru to nije slučaj, obratite pažnju na to da generalno naredba return ne mora biti poslednja naredba u telu metoda. Ona se može pisati u bilo kojoj tački u telu metoda u kojoj se želi vratiti vrednost i završiti izvršavanje metoda. Ako se naredba return izvršava negde u sredini tela metoda, izvršavanje

metoda se odmah prekida. To znači da se sve eventualne naredbe iza***return*** naredba preskaču i kontrola se odmah vraća na mesto gde je metod pozvan. Naredba povratka se može koristiti i kod metoda koji ne vraća nijednu vrednost. Budući da tip rezultata takvog metoda mora biti "prazan" tip, tj. ***void,*** u zaglavlju metoda, naredba povratka u telu metoda u ovom slučaju ne sme sadržati nikakav izraz iza reči return, odnosno ima jednostavan oblik:

```
return;
```

Efekat ove naredbe je samo završetak izvršavanja nekog metoda koji ne vraća nijednu vrednost i vraćanje kontrole na mesto u programu gde je taj metod pozvan. Upotreba naredbe povratka kod metoda koji ne vraća nijednu vrednost nije obavezna i koristi se kada izvršavanje takvog metoda treba prekinuti negde u sredini. Ako naredba povratka nije izvršena prilikom izvršavanja tela takvog metoda, njegovo izvršavanje se normalno prekida (i kontrola izvršavanja se vraća na mesto gde je metod pozvan) kada se dođe do kraja izvršavanja tela metoda.

Ukoliko metod vraća jednu vrednost, u njegovom telu se mora nalaziti bar jedna naredba povratka u punom obliku: ***return*** izraz.Ako ih ima više od jedne, sve one moraju imati ovaj pun oblik, jer metod uvek mora vratiti jednu vrednost.

METODI (VIDEO)

Video daje objeašnjenje metoda i primere njihovog korišćenja.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PRIMER 17

Pisanje metode koja vraća random boju

Metoda treba da vrati random boju. Boja je predstavljena javinom klasom Color.

Potpis metode:

```
public static Color dajMiRandomBoju()
```

Konstruktor klase Color koji mi koristimo prima 3 celobrojna argumenta koji predstavljaju RGB format. Maksimalna vrednost svake od RGB komponenti boje jeste 255, a minimalna 0.

Na sledeći način pravimo novu random boju.

```
new Color(r.nextInt(255), r.nextInt(255), r.nextInt(255))
```

U nasoj metodi kreiramo novi objekat klase Color i vraćamo referencu na kerirani objekat. U main metodi pravimo 3 random boje i ispisujemo vrednosti njihovih RGB komponenti.

Rešenje:

```
package primer17;

import java.awt.Color;
import java.util.Random;

public class Primer17{

    public static void main(String[] args) {
        new Primer17();
    }

    public Primer17() {
        Color color = dajMiRandomBoju();
        Color color2 = dajMiRandomBoju();
        Color color3 = dajMiRandomBoju();
        System.out.println(color + "\r\n" + color2 + "\r\n" + color3);

    }

    /**
     * Pravimo random boju (Color klasa) gde prosledjujemo Random R, G i B
     * vrednosti.
     *
```

```
     * @return
     */
    public static Color dajMiRandomBoju() {
        Random r = new Random();
        return new Color(r.nextInt(255), r.nextInt(255), r.nextInt(255));
    }

}
```

ZADATAK 7

Samostalno vežbanje pisanja metoda koje vraćaju vrednost

- Napravi metodu koja prima n brojeva.
- Svaki broj koji metoda primi treba da pomnoži sa 2 i dodati na zbir.
- Zbir vratiti.
- Prikazati rad metode preko main().

Poglavlje 5

Preopterećenje metoda

ŠTA JE PREOPTEREĆENJE METODA?

Preopterećenje metoda nastaje kada dva ili više metoda u klasi imaju isto ime, a različitu listu argumenata.

Da bi se neki metod mogao pozvati radi izvršavanja, mora se poznavati njegovo ime, kao i broj, redosled i tipovi njegovih parametara. Ove informacije se nazivaju **potpis metoda.** Na primer, metod:

```
public int maksimum(int a, int b, int c) {
        // Telo metoda
    }
```

ima potpis:

```
maksimum(int a, int b, int c) {
```

Obratite pažnju na to da potpis metoda ne obuhvata imena parametara metoda, nego samo njihove tipove. To je zato što za pozivanje nekog metoda nije potrebno znati imena njegovih parametara, već se odgovarajući argumenti u pozivu mogu navesti samo na osnovu poznavanja tipova njegovih parametara. Primetimo i da potpis metoda ne obuhvata tip rezultata metoda, kao ni eventualne modifikatore u zaglavlju.

Definicija svakog metoda u Javi se mora nalaziti unutar neke klase, ali jedna klasa može sadržati definicije više metoda sa

istim imenom, pod uslovom da svaki takav metod ima različit potpis. Metodi iz jedne klase sa istim imenom i različitim potpisom se nazivaju **preopterećeni** (engl.overloaded) metodi. Na primer, u klasi u kojoj je definisan prethodni metod maksimum() može se definisati drugi metod sa istim imenom, ali različitim potpisom:

```
public int maksimum(int[] niz) {
        // Telo metoda
    }
```

Potpis ovog metoda je:

```
maksimum(int[])
```

što je različito od potpisa prvog metoda **maksimum().**

Potpis dva metoda u klasi mora biti različit kako bi Java prevodilac tačno mogao da odredi koji metod se poziva. To se lako određuje na osnovu broja, redosleda i tipova argumenata navedenih u pozivu metoda. Na primer, naredbom:

```
maksimum(9, 5, 7);
```

očigledno se poziva prva, a ne druga, verzija metoda maksimum(), jer su u pozivu navedena dva argumenta tipa int.

PRIMER 18 - PREOPTEREĆENJA METODA

Preopterećenje metoda nastaje kada dva ili više metoda u klasi imaju isto ime, a različitu listu argumenata

Preopterećenje metoda nastaje kada dva ili više metoda u klasi imaju isto ime, a različitu listu argumenata. Koji metod će se pozvati, određuje se u trenutku izvršenja uparivanjem liste deklarisanih argumenata sa listom stvarnih argumenata, koji se u tom trenutku prosleđuju.

Na slici 1 prikazano je nekoliko metoda max() sa sa različitim listama argumenata (i po tipu i po broju).

```
// metodi max sa različitim parametrima:
public int max( int m, int n ){return ( m>n ) ? m : n;}
float max(float x, float y ){return ( x>y ) ? x : y;}
public float max(float x, float y, float z ){
          return max(max(x,y),z);
}
```

Slika 5.1 Preopterećen metod max()

Kada nekoliko metoda imaju isto ime, pravilo za određivanje koji se metod poziva glasi*: Koristiće se onaj metod čiji stvarni argumenti odgovaraju deklarisanim argumentima*.

Pogledajmo primer na slici 2.

```
class Racun{  . . . .
  private int    stanje;
  . . . .
  // Metod uplati sa jednim argumentom
  void uplati( int suma){
    stanje = stanje + suma;
  }

  // Metod uplati sa dva argumenta
  void  uplati( int suma, int cenaUsluge){
   stanje = stanje + suma - cenaUsluge;
  }
}
.....
```

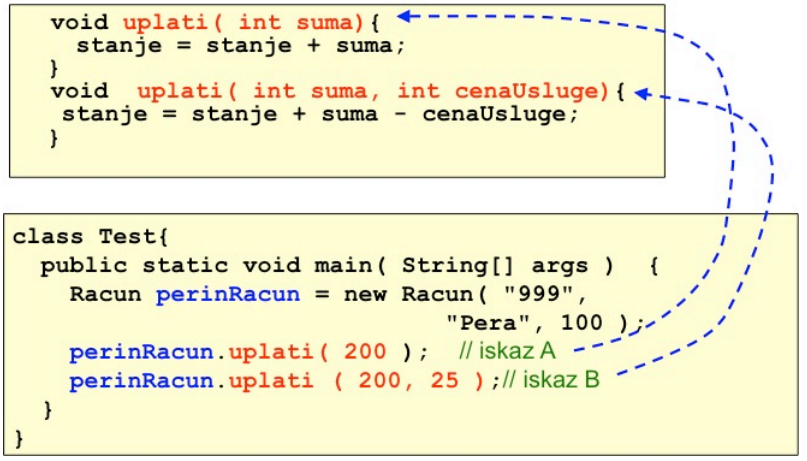
Slika 5.2 Primer sa preopterećenim metodom uplati()

Ovo je promenjena klasa od ranije. Pretpostavka je da su nam bila potrebna dva metoda za stavljanje na račun: jedan se koristi za uobičajeno postavljanje novca na račun kada banka ne skida novac za svoje usluge, i drugi – za ostale slučajeve kada banka uzima svoj deo.

UPOTREBA PREOPTEREĆENJA METODA

Povratni tip nije deo potpisa metoda, isto kao ni identifikatori argumenata u deklaraciji metoda

Sledeća slika pokazuje kako bi izgledala upotreba ovih metoda:



Slika 5.3 Program koji poziva preopterećeni metod uplati()

Kao što je pokazano, koristi se onaj metod koji ima istu listu argumenata kao i metod koji se poziva.

Povratni tip nije deo potpisa, isto kao ni identifikatori argumenata u deklaraciji metoda. To znači da nije dovoljno da se dva metoda u istoj klasi razlikuju samo po povratnom tipu.

Pogledajmo sledeća dva metoda:

int racunaSumu(int a)

float racunaSumu(int b)

Pošto je potpis ova dva metoda isti, a tip povratnog rezultata (koji nije deo signature) je različit, kompajler neće dozvoliti kompilaciju, odnosno *prijaviće grešku*, jer se navode dva metoda sa istim potpisom.

U prethodnom primeru treba obratiti pažnju na to da imena argumenata nisu deo potpisa, tako da nije bitno što su u pitanju promenljive a i b. Važan je njihov tip i broj.

.

PRIMER 19 - PREOPTEREĆENJE KONSTRUKTORA

Konstruktori su specijalni po tome što imaju isto ime kao klasa, i što se za njih – ne zadaje povratni tip.

Da se podsetimo: Konstruktori su specijalni metodi koji se pozivaju prilikom kreiranja novih objekata klase. Oni su specijalni po tome što imaju isto ime kao klasa, i što se za njih – ne zadaje povratni tip. Osim ovih razlika, konstruktori su isti kao i drugi metodi. To znači da i oni mogu da se preopterećuju. Preopterećenje konstruktora se mnogo često koristi, jer je to mogućnost da se na različit način inicijalizuju atributi objekta.

Pogledajmo sledeći primer:

```
class Tacka{
  public int x, y ;
  Tacka(int xu, int yu){// prvi konstuktor
  x = xu;
  y = yu;
  }
  Tacka(Tacka A){// drugi konstruktor
  x = A.x;
  y = A.y;
  }
}
```

Kao što vidimo, klasa Tacka ima dva konstruktora: jedan se koristi za kreiranje nove instance na osnovu dva cela broja koji predstavljaju koordinate nove tačke, a drugi konstruktor se koristi za kreiranje novog objekta kopiranjem starog. Koji konstruktor će se pozvati, određuje se u trenutku poziva na osnovu prosleđenih argumenata.

```
class Test{
    public static void main( String[] args )  {
        Tacka A = new Tacka(2, 3);
        Tacka B = new Tacka(A);
    }
}
```

Kod kreiranja tačke A poziva se prvi konstruktor (sa dva cela broja). Kod kreiranja tačke B se poziva drugi konstruktor, koji novu tačku pravi na osnovu koordinata stare.

PREOPTEREĆENJE (OVERLOADING) METODA (VIDEO)

Video objašnjava koncept preopterećenja metoda, tj. metoda s aistim imena, a sa različitim parametrima).

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PRIMERI KORIŠĆENJA PREOPTEREĆENJA METODA (VIDEO)

I ovaj video objašnjava koncept preopterećenja metoda korišćenjem više primera.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADATAK 8

Samostalno vežbanje preopterećenja metoda

- Kreirajte klasu SabiranjeBrojeva;
- Klasa ima dva polja tipa int: prvi i drugi;
- Klasa ima dva konstruktora: podrazumevani i konstruktor koji preuzima dva argumenta tipa int;
- Drugi konstruktor sabira argumente i prikazuje rezultat sabiranja;
- Izvršiti sabiranje vrednosti koje se čuvaju u poljima prvi i drugi, koristeći oba konstruktora.

Poglavlje 6

Domaći zadaci

ZADACI 1,2 I 3

Zadaci 1 - 4 za individualne vežbe

Zadatak 1

Napraviti dve metode za izračunavanje površine jednakostraničnog i jednakokrakog trougla. Prikazati rad metoda preko Main-a.

Zadatak 2

Napraviti metodu koja vraća N-ti element fibonačijevog niza. Svaki element fibonačijevog niza je jednak zbiru prethodna dva (Preskočiti 1 i 2). Koristiti petlju unutar metode. Prikazati rad metode preko Main-a.

Zadatak 3

Napravi metodu koja prima n brojeva. Svaki paran broj koji metoda primi treba da pomnoži sa 3 i dodati na zbir. Zbir vratiti. Prikazati rad metode preko Main-a.

ZADACI 4 I 5

Zadaci 5 - 6 za individualne vežbe

Zadatak 5

Napraviti metodu koja štampa kvadrat karakterom (*) veličine prosleđenog parametra (int). Primer ako je unos 3:

***

***

***

Prikazati rad metode preko Main-a.

Zadatak 6

Napraviti metodu koja menja prvi i zadnji karakter String-a i metodu koja uzima prva tri karaktera String i potom vraća 3 puta ta tri karaktera. Primer: Java Rezultat: JavJavJav. Prikazati rad metode preko Main-a.

Zakljucak

ŠTA SMO NAUČILI?

Primena metoda je način modularizacije objektno-orijentisanih softverskih sistema.

1. Modularzacija programa je jedan od osnovnih ciljeva softverskog inženjerstva. Java obezbeđuje tehike koja to omogućvaju. Jedna od njih je primena metoda.
2. Zaglavlje metode sadrži modifikatore, tip povratne vrednosti, ime (naziv) metoda i parametre metoda.
3. Metod može da vraća vrednost (rezultat) određenog (naznačenog) tipa. Ukoliko metod ne vraća nijednu vrednost, onda se unosi ključna reč void.
4. Lista parametra daje tip, redosled i broj parametra metoda. Ime (naziv) metoda i lista parametra se naziva potpisom metoda. Parametri su opcioni, jer metod i ne mora da ima parametre.
5. Naredba return može da se i kod void metoda iskoristi za završetak metoda i povraćaj programske kontrole na objekt koji je izvršio pozv metoda.
6. Argumenti koji se prenose u metod treba da su po broju, tipu i redosledu istovetni kao i u potpisu metoda.
7. Kada se iz progama poziva neki metod, kontrola programa se prenosi na metod. Pozvan metod vraća kontrolu programu (objektu) odakle je pozvan kada se dođe do njegove poslednje velike zagrade koja ga zatvara.
8. Metod koji vraća vrednost može da se poziva iz neke Java naredbe. U takvom slučaju, objekat koji ga poziva ignoriše povratnu vrednost.
9. Metod može da se preoptereti. To znači da dva metoda mogu da imaju ista imena, ali liste parametra moraju da se razlikuju.
10. Promenljive metoda se nazivaju lokalnim promenljivima. Domen važenja lokalne promenljive važi od mesta poziva pa da kraja bloka u kom je pozvan. Pre upotrebe, lokalna promenljiva mora da se deklariše.

REZIME

Programi koji se pišu korišćenjem metoda su koncizniji i lakši za razumevanje

1. Apstrakcija metoda se vrši tako što se odvaja upotreba metode od njene implementacije (izrade). Objekt koji poziva metod ga moće koristiti neznajući kako je on implementiran (izrađen). Detalji implementacije u očaureni u metodu i skriveni od klijenta koji je pozvao metod. Ovo se zove skrivanje informacija ili učaurenje.
2. Apstrakcija metoda modularizuje program na neki hijerarhijski način. Programi koji se pišu korišćenjem metoda su koncizniji i lakši za razumevanje.Ponovna upotrebivost metoda je yavisna od stila pisanja metoda.

3. Kada pišete velike programe pripremite jedan od dva pristupa kodiranju: odozgo nadole, ili od dozdo na gore. Ne pišite program odjednom, kao jednu celinu. To produžava vreme razvoja,ali smanjuje vreme u održavanju softvera, jer je modularizovan program jednostavniji za razumevanje.