4

Objektno orijentisano programiranje vežbe

Staša Vujičić Stanković

Osnovni pojmovi. Podsećanje.

- **<u>Objekat</u>** – integralna celina podataka i procedura za rad sa njima.
- **<u>OOP</u>** – programska paradigma zasnovana na skupu objekata koji dejstvuju međusobno. Glavne obrade se zasnivaju na manipulisanju objektima.

Osnovni pojmovi. Podsećanje.

- **<u>Klasa</u>** – Skup objekata sa zajedničkim svojstvima. Npr. prevozno sredstvo je klasa.
- **<u>Potklasa</u>** – klasa koja nasleđuje sve osobine roditeljske klase (**<u>nadklase</u>**) i ima dodatnu specijalizaciju. Npr. automobil ima sve karakteristike klase prevozno sredstvo, ali ima i svoje dodatne karakteristike – npr. 4 točka...

Osnovni pojmovi. Podsećanje.

- **Instanca** – konkretan objekat iz klase.
- **Instancne promenljive ili atributi klase** – parametri koji definišu objekat neke klase. Mogu biti osnovni tipovi podataka ili objekti neke druge klase.
- Npr. automobil:
  - boja: bela, crvena, crna,...
  - proizvođač: Honda, Opel, BMW, ...
  - model: limuzina, sportski, karavan...

Osnovni pojmovi. Podsećanje.

- Primer:

```
class Krug {            ← klasa Krug
    int x, y;
    float r; …
    }
Krug k = new Krug(10,20,5);   ← instanca klase Krug
```

- x, y i r su instancne promenljive koje u instanci k imaju redom vrednosti 10, 20 i 5.

Osnovni pojmovi. Podsećanje.

- Klasa pored kolekcije podataka specifikuje i šta sve može da se radi sa objektima te klase. Definiše listu mogućih operacija nad objektima klase. To se postiže blokom programskog koda koji se naziva metoda.
- Na primer:
  - Startuj motor
  - Zaustavi auto
  - Ubrzaj…

Klase u Javi

- Klasa se može shvatiti i kao novi tip.
- Definicija klase se koristi da se kreiraju objekti tog klasnog tipa, tj. da se kreiraju objekti koji sadrže sve komponente koje su specifikovane u klasi.

Klase u Javi

- Definicija klase u sebi sadrži:
  - **atribute** – ovo su promenljive koje obično razlikuju jedan objekat klase od drugog
  - **metode** – one definišu operacije koje možemo da radimo za datu klasu – određuju šta možemo da radimo nad objektima klase ili sa objektima klase. Obično se izvršavaju nad atributima klase.

ATRIBUTI U DEFINICIJI KLASE

- Kada kreiramo objekat, on će sadržati sve atribute koji se nalaze u definiciji klase.
- Atributi klase mogu da budu primitivnog tipa ili mogu da referišu na objekte proizvoljnog klasnog tipa, uključujući i onaj koji definišemo.

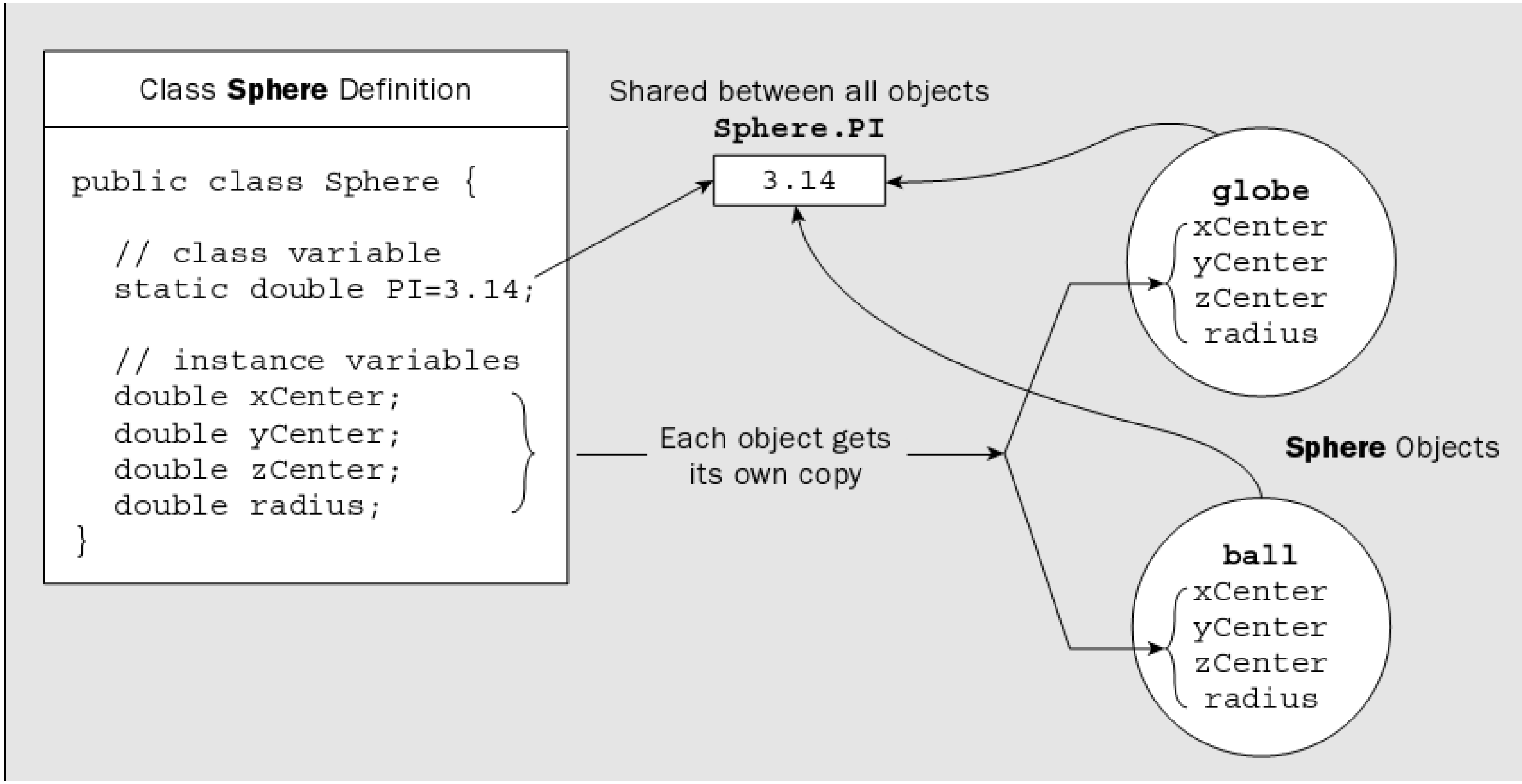
ATRIBUTI U DEFINICIJI KLASE

- Postoje 2 vrste atributa:
  - **Statički ili klasni atribut** –
    on je zajednički za sve objekte klase.
    Postoji samo jedna kopija tog atributa bez obzira na to koliko objekata te klase je kreirano i ta kopija postoji čak i kada nije kreiran nijedan objekat te klase.
    Deklarišu se ključnom rečju **static**.
  - **Nestatički atribut ili instancna promenljiva** -
    svaki objekat klase poseduje svoju sopstvenu kopiju atributa. Ovakvi atributi daju objektima individualnost, tj. razlikuju ih međusobno.

Statički atributi

- Statički atributi obično opisuju neko svojstvo koje je zajedničko za sve objekte te klase.
- Još jedna primena statičkih atributa je za čuvanje vrednosti koje su zajedničke svim objektima te klase.
- Npr: ako želimo da brojimo koliko je objekata neke klase kreirano u programu, najbolje rešenje je da se za to iskoristi statička promenljiva, zato što ona može da se koristi i kada nijedan objekat klase nije kreiran.

STATIČKI ATRIBUTI



Metodi u definiciji klase

- Analogno atributima, postoje dve vrste metoda:
  - **Statičke ili klasne metode** –
    mogu da se izvršavaju i kada ne postoji nijedan objekat klase. Deklarišu se ključnom rečju **static**.
  - **Nestatičke ili instancne metode** –
    mogu da se izvršavaju samo u odnosu na objekat.

Statički metodi

- Pošto statički metodi mogu da se izvršavaju i kada ne postoje objekti, oni ne mogu da referišu na instancne promenljive,
  jer u suprotnom može da dođe do toga da se operiše nad promenljivim koje ne postoje.
  U tom slučaju, kôd se neće prevesti, dobiće se poruka o grešci.

Statički metodi – main()

- Metod main() je, kao što smo videli ranije, uvek deklarisan kao statički. Pre nego što neka aplikacija počne da se izvršava, ne postoji ni jedan objekat.
  Da bismo startovali izvršavanje, neophodan nam je metod koji može da se izvrši čak i kada nema objekata, dakle, statički metod.

Statički metodi

- Pored main() metoda, verovatno najčešća situacija kada koristimo statičke metode je kada se klasa koristi kao <u>kontejner za gomilu korisnih metoda</u>
- Na primer, matematičke funkcije implementirane kao klasne metode u standardnoj klasi Math – ove metode nemaju nikakve veze sa objektima, već samo operišu nad primitivnim tipovima, a sadrže i neke klasne promenljive koje su zapravo korisne matematičke konstante – pi, e, itd.

Nestatički metodi

- <u>Napomena!</u>

  Iako se <u>nestatičke metode</u> vezuju za objekat, u memoriji postoji samo jedna kopija svake instancne metode koju dele svi objekti odgovarajuće klase, jer bilo bi jako skupo da se za svaki novi objekat kreira i nova kopija instancne metode.

  Poseban mehanizam omogućava da svaki put kada se pozove metod, njegov kôd se izvršava na način specifičan za konkretan objekat.

Pristup statičkim članovima klase

- Obično promenljivim i metodama koje su definisane u klasi pristupamo izvan nje.
- Statičkom članu klase pristupamo koristeći

  <u>ime klase.ime članice</u>

  Double koren = Math.sqrt(Math.PI);

Pristup statičkim članovima klase

- Statičkom članu se može pristupiti i preko promenljive koja referiše na konkretan objekat klase.

  <u>referenca na objekat.ime članice</u>

- Ovo je moguće jer svaki objekat uvek ima pristup svim statičkim članovima klase.

PRISTUP NESTATIČKIM ČLANOVIMA KLASE

- Instancnim promenljivim i metodima se pristupa **referenca na objekat.ime članice**:

  Double povrsinaKupe = kupa.povrsina();

Klase u Javi

- Za definisanje klase koristimo ključnu reč **class** koju prati ime klase i u okviru vitičastih zagrada detalji definicije.

```
class ImeKlase {
  //  članovi ( atributi i metodi )
 }
```

- Po konvenciji,
  imena klasa u Javi počinju velikim slovima.

Definisanje klase

- Za atribute, kako statičke tako i nestatičke, se u definiciji klase mogu postaviti inicijalne vrednosti.

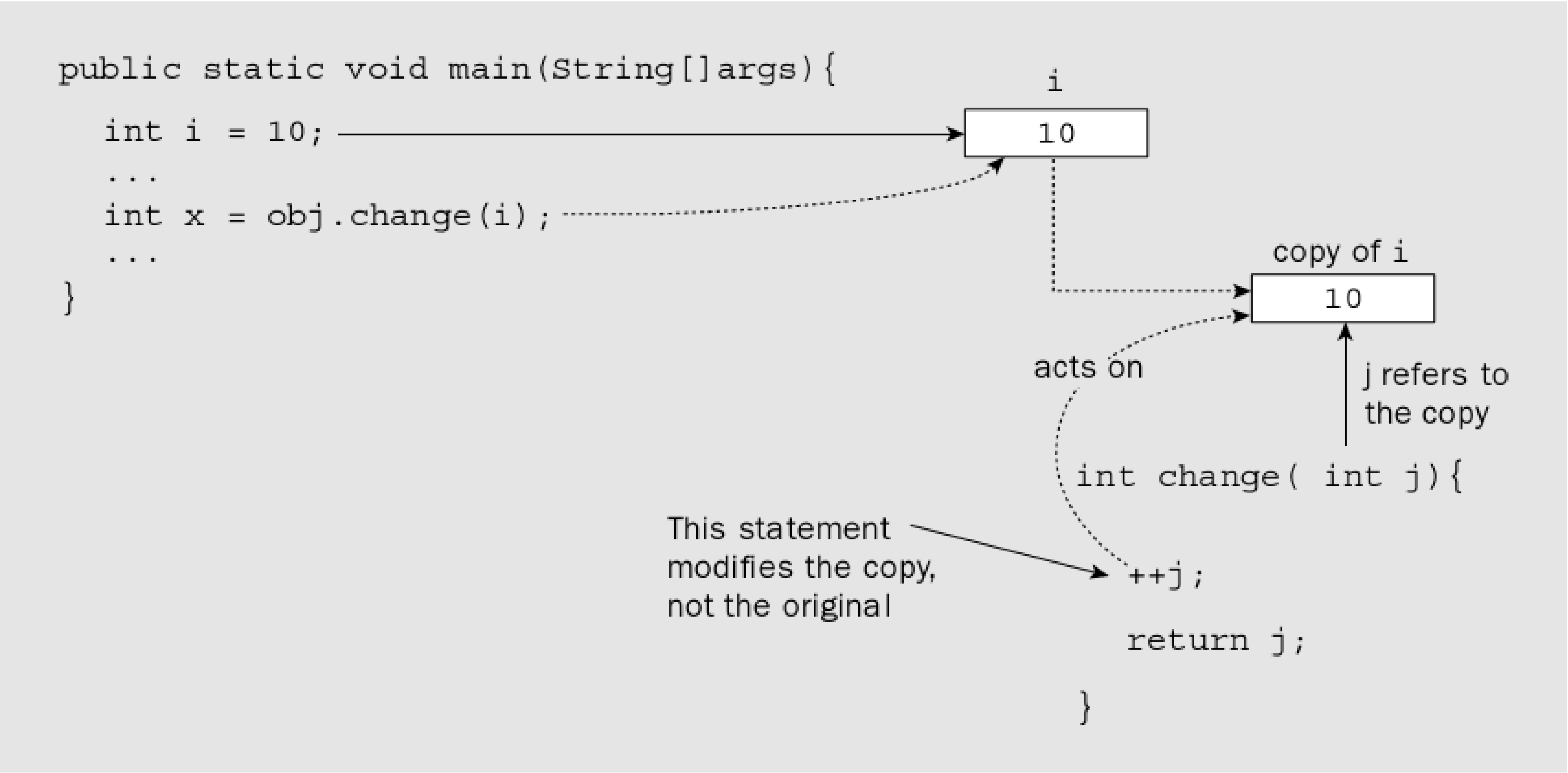
- Ukoliko se ne navede inicijalna vrednost, prilikom kreiranja objekta biće pridružena default vrednost i to:
  - 0 za numeričke tipove
  - '\u0000' za tip char
  - null za reference na objekte i reference na nizove

DEFINISANJE METODA

- Metodi su zapravo funkcije i tako se i definišu.
- Tip vrednosti koju metod vraća može biti *void*, proizvoljan tip ili klasa.
- Lokalne promenljive se ne inicijalizuju automatski, već se mora eksplicitno izvršiti njihova inicijalizacija
(inače ne prolazi kompajliranje)!

Prenošenje argumenata metodama

- Metodama se argumenti prenose po **vrednosti.**
- Dakle, za svaki argument koji se prenosi, pravi se njegova kopija i kopija se prenosi metodu preko imena parametra, a ne originalna vrednost, pa tako ako je argument promenljiva primitivnog tipa, funkcija ne može promeniti njenu vrednost.


public static void main(String[]args){
int i = 10;
...
int x = obj.change(i);
...
}
i
10
copy of i
10
acts on
j refers to the copy
int change(int j){
This statement modifies the copy, not the original
++j;
return j;
}

Prenošenje argumenata metodama

- Iako se mehanizam prenošenja argumenata po vrednosti primenjuje na sve tipove argumenata, efekat za objekte se razlikuje od onog za promenljive primitivnih tipova.
- Objekat se može promeniti jer promenljiva klasnog tipa sadrži referencu na objekat, a ne sâm objekat. Kada se takva promenljiva prenese kao argument metoda, prenosi se kopija reference na objekat, a ne kopija samog objekta.

Final parametri

- Svaki parametar metoda može se navesti i kao **final** čime se sprečava da funkcija promeni njegovu vrednost, o čemu će kompajler voditi računa.

- Pošto se parametri prenose po vrednosti, **final** ima smisla samo za parametre koji su reference na objekte klasa.

- Međutim, time se sprečava promena reference na objekat koja se prosleđuje metodu, a ne samog objekta.

Definisanje statičkih metoda

- Doda se ključna reč **static** ispred definicije metoda.
- Važno je zapamtiti da se unutar statičkih metoda ne mogu referisati nestatički atributi, jer se ovakvi metodi mogu pozivati i kada nije kreiran nijedan objekat klase, u kom slučaju i ne postoji nijedan nestatički atribut.

Promenljiva this

- Svaki nestatički metod ima promenljivu koja se zove **<u>this</u>** koja predstavlja referencu na tekući objekat za koji je metod pozvan.
- Kompajler implicitno koristi **this** kada metod referiše nestatički atribut klase.
- Možemo i sami napisati **<u>this.atribut.</u>**
- Nije dobra praksa na taj način nepotrebno opterećivati kod. Međutim, postoje i situacije kada je neophodno eksplicitno koristiti **<u>this.</u>**

Primer

- Kada izvršimo naredbu:

  double ballVolume = ball.volume();

  gde je ball objekat odgovarajuće klase,

  this u metodu volume() referisaće na objekat ball, tako da će upravo instancna promenljiva radius iz objekta ball biti iskorišćena u izračunavanju.

THIS I NESTATIČKI METODI

- Kako postoji samo jedna kopija nestatičkog metoda u memoriji, bez obzira na to koliko objekata date klase postoji, promenljiva **this** omogućuje da ta jedna instanca metoda radi sa različitim objektima klase.

- Prilikom svakog poziva nestatičkog metoda, promenljiva **this** se postavi da referiše na objekat za koji je metod pozvan i onda se kôd u metodu odnosi na članove upravo tog objekta.

Eksplicitna upotreba this-a

- Imena promenljivih koje se deklarišu unutar metoda, lokalna su za taj metod.
  Dozvoljeno je koristiti imena lokalnih promenljivih, kao i imena parametara metoda koja su ista kao i imena atributa.

  U tom slučaju, neophodno je koristiti **this** za referisanje člana klase unutar metoda.

  Sâmo ime promenljive će se uvek odnositi na lokalnu promenljivu metoda, ne na atribut klase.

Primer

```
void changeRadius( double radius ){
    this.radius = radius;
}
```

- **<u>this.radius</u>** se odnosi na atribut klase, a **<u>radius</u>** na parametar metoda

Inicijalizacija atributa klase

- I statički i nestatički atributi se mogu inicijalizovati unutar definicije klase.
- Neke stvari se ne mogu inicijalizovati jednim izrazom, npr. veliki niz koji želimo inicijalizovati vrednostima koje iziskuju neku vrstu izračunavanja.
- To je posao za **inicijalizacioni blok**.

Korišćenje inicijalizacionih blokova

- Inicijalizacioni blok je blok kôda između { i } koji se izvršava pri kreiranju objekta klase
- Postoje dve vrste inicijalizacionih blokova:
  - **Statički blok za inicijalizaciju** – blok definisan ključnom rečju static i izvršava se samo jednom kada se klasa učitava. On može da inicijalizuje samo statičke atribute klase.
  - **Nestatički blok za inicijalizaciju** – izvršava se za svaki objekat koji se kreira i stoga može da inicijalizuje instancne promenljive u klasi.
- Iako je moguće, normalno ne treba inicijalizovati statičke atribute nestatičkim inicijalizacionim blokom.

```java
// Initialization block
static {
  System.out.println("Running initialization block.");
  for(int i=0; i<values.length; i++) {
    values[i] = (int)(100.0*Math.random());
  }
}
```

Konstruktori

- Možemo imati više inicijalizacionih blokova i onda se oni izvršavaju onim redom kojim su navedeni u definiciji klase.
- Iako su inicijalizacioni blokovi korisni, za kreiranje objekata potrebni su nam i konstruktori.

Konstruktori

- Uvek kada kreiramo objekat neke klase poziva se specijalna vrsta metoda, tzv. **konstruktor.**
- Ukoliko ne definišemo nijedan konstruktor za našu klasu, kompajler će obezbediti podrazumevani konstruktor klase koji ne radi ništa.

Konstruktori

- Primarna svrha konstruktora jeste da za objekat koji se kreira izvrši inicijalizaciju nestatičkih atributa.
- Inicijalizacioni blokovi koji su eventualno definisani u klasi uvek se izvršavaju pre tela konstruktora.
- Konstruktor ima 2 specifične osobine:
  - **nikada ne vraća vrednost (ni void!)**
  - **uvek ima isto ime kao i klasa kojoj pripada**

KONSTRUKTORI

- Konstruktor može imati proizvoljan broj parametara, uključujući i 0 (bez parametara).
- Podrazumevani konstruktor nema parametre.

Podrazumevani konstruktor

ime_klase()

{ … }

- Podrazumevani konstruktor nema parametre i ne radi ništa.
- Konstruktor se poziva kad god kreiramo objekat.
- Ukoliko sami napišemo bar jedan konstruktor, kompajler ne pravi podrazumevani konstruktor.
Ako nam je i on potreban,
moramo ga eksplicitno definisati .

Kreiranje objekata klase

- Kada deklarišemo promenljivu tipa neke klase, ne poziva se konstruktor, jer se i ne kreira objekat.
  - Npr.

    Sfera ball; // Deklarisanje promenljive
- Na ovom mestu kreira se promenljiva ball koja može da čuva referencu na objekat tipa Sfera, ako i kada kreiramo neki.

Kreiranje objekata

- Da bismo kreirali objekat klase moramo koristiti ključnu reč **new** za kojom sledi poziv konstruktora.
- Da inicijalizujemo ball referencom na objekat:

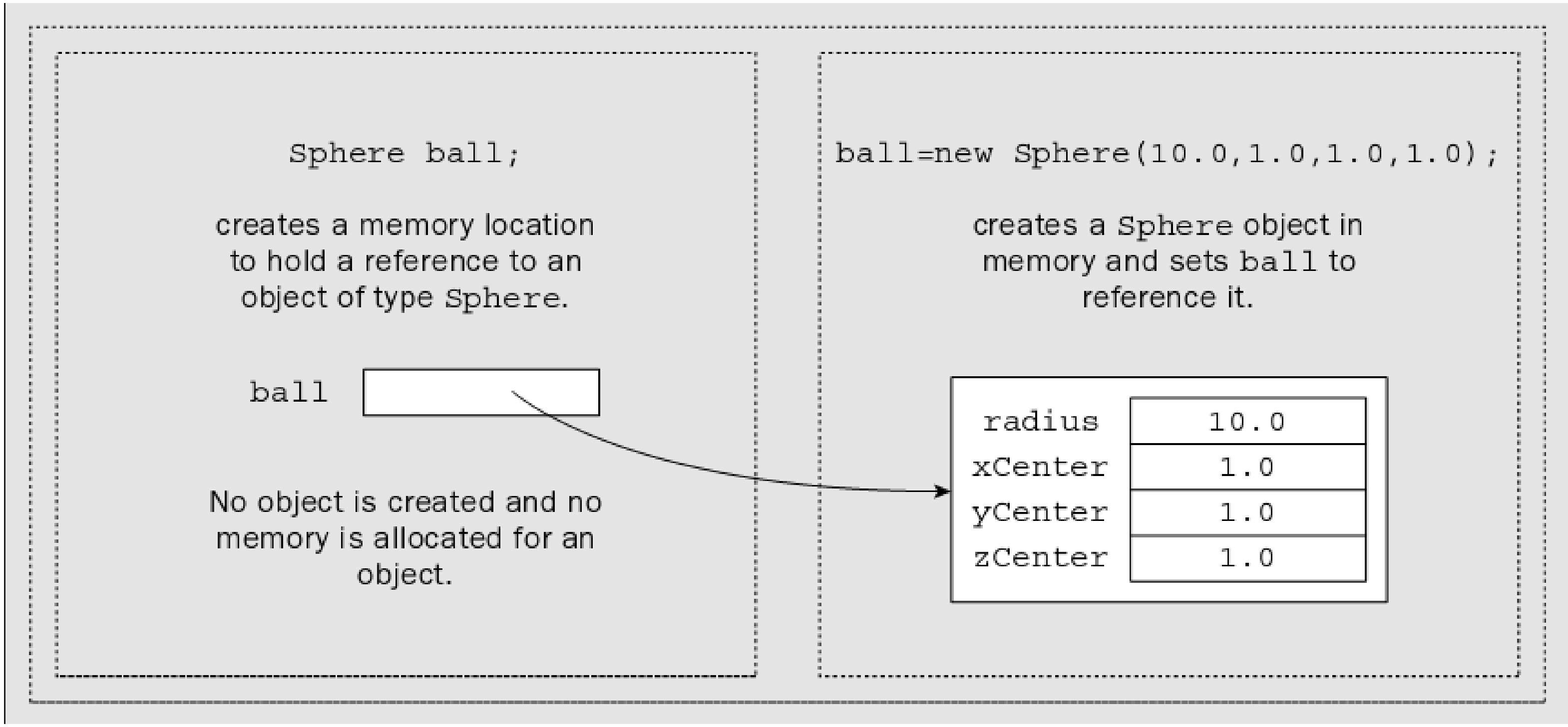
  ball = new Sphere(10.0, 1.0, 1.0, 1.0);
- Može i sve u jednom redu:

  Sphere ball = new Sphere(10.0, 1.0, 1.0, 1.0);

  // deklariše se promenljiva ball i definiše

  // objekat na koji ona referiše

KLASE U JAVI


Sphere ball;
creates a memory location to hold a reference to an object of type Sphere.
ball
No object is created and no memory is allocated for an object.
ball=new Sphere(10.0,1.0,1.0,1.0);
creates a Sphere object in memory and sets ball to reference it.
radius
10.0
xCenter
1.0
yCenter
1.0
zCenter
1.0

Životni vek objekata

- Životni vek objekta određen je promenljivom koja čuva referencu na njega, pod pretpostavkom da postoji samo jedna takva.
- Objekat na koji promenljiva referiše ”umire” na kraju bloka u kome je deklarisana.
- Ako postoji više promenljivih koje referišu isti objekat, objekat postoji sve dok postoji bar jedna od tih promenljivih.

Životni vek objekata

- Sa **ball = null**; resetujemo promenljivu tako da više ne pokazuje na objekat.

  Ukoliko je to bila jedina promenljiva koja je referisala na taj objekat, on će biti uništen.

  Promenljiva ball, međutim, nastavlja da postoji i može se koristiti da referiše na neki drugi objekat iste klase.

Garbage collector

- Uništavanje ”mrtvih” objekata vrši tzv. **Garbage Collector**. To se u Javi dešava automatski, ali ne znači da mrtvi objekti odmah nestaju iz memorije. To ni na koji način ne utiče direktno na naš program. Može da bude bitno jedino u situacijama kada su naši objekti ogromni ili kada odjednom kreiramo i osobađamo se velikog broja objekata. U tom slučaju možemo probati da zovemo statički metod **gc()** klase System da bi Java VM izvršila **garbage collection**
  - **System.gc();**
- Međutim, poziv gc-a može i da pogorša stvar i da uspori ceo taj proces.

PREKLAPANJE (OVERLOADING) METODA

- Moguće je definisanje više metoda klase sa istim imenom i jedinstvenim skupom parametara.
  To se naziva **preklapanjem metoda.**

- Ime metoda, zajedno sa tipovima njegovih parametara čini **potpis metoda.**

Preklapanje (overloading) metoda

- Potpis svakog metoda u klasi mora biti različit kako bi kompajler mogao u svakom trenutku da odredi koji metod pozivamo.
- Tip povratne vrednosti metoda nema uticaja na potpis metoda.

PREKLAPANJE KONSTRUKTORA

- Konstruktori se mogu preklapati, kao i bilo koji drugi metod klase.
- Kompajler odabira koji konstruktor će koristiti na osnovu prosleđenih argumenata.
- Za potpis metoda bitan je broj i tip parametara, a ne njihova imena.

Poziv konstruktora iz drugog konstruktora

- Jedan konstruktor klase može da poziva drugi konstruktor te klase u svojoj prvoj izvršnoj naredbi. Time se često može uštedeti na ponovnom pisanju koda.
- Za referisanje drugog konstruktora iste klase koristi se this kao ime metoda praćen odgovarajućim argumentima između zagrada.

Zadatak 1. – Ucenik.java

```
package Ucenik;
class Ucenik {
        String ime;
        String skola;
        int razred;

// Konstruktor sa argumentima
  Ucenik(String ime_uc, String skola_uc, int razred_uc){
        ime=ime_uc;
        skola=skola_uc;
        razred=razred_uc;
  }
```

```
// Default konstruktor
/* Ucenik(){
	ime="Novak Novicic";
	skola="Vuk Stefanovic Karadzic";
	razred=3;
} */

// Metod za ispis skole
void stampajSkolu(){
	System.out.println("Ime skole je: "+skola);
}
```

```
// Metod za ispis imena ucenika
void stampajIme(){
        System.out.println("Ime ucenika je: "+ime);
}
}
```

TestUcenik.java

```
package Ucenik;

class TestUcenik {

   public static void main (String args []){

         Ucenik prvi=new Ucenik("Milos
   Milosevic","Ivan Goran Kovacic",7);
         prvi.stampajSkolu();
         prvi.stampajIme();

         System.out.println("Ucenik "+prvi.ime+" ide
   u "+prvi.razred +". razred skole "+ prvi.skola);
```

```
/* Ucenik drugi=new Ucenik();
        System.out.println("Ucenik "+drugi.ime+" ide u "+drugi.razred +". razred skole "+ drugi.skola);
        */
    }
}
```

Zadatak 2.

- Napisati u paketu **Sfera** klasu **Sfera.java** takvu da sadrži:
  - Klasnu promenljivu PI sa fiksiranom vrednošću.
  - Klasnu promenljivu koja broji objekte klase.
  - Instancne promenljive: poluprecnik i koordinate centra (xCentar, yCentar i zCentar).
  - Konstruktor jedinične sfere u koordinatnom početku.
  - Konstruktor jedinične sfere u zadatoj tački.
  - Konstruktor sfere zadatog poluprečnika u zadatoj tački.
  - Statički metod koji vraća ukupan broj kreiranih objekata klase.
  - Instancni metod za računanje zapremine lopte.

- Napisati u paketu **Sfera** klasu **TestSfere.java** takvu da da sadrži:
  - Ispis broja objekata koji su formirani (pozivom odgovarajućeg metoda) pre nego što je formiran ijedan objekat.
  - Kreira objekte
    - *sfera* u koordinatnom početku, poluprečnika 4.0.
    - *globus* (12., 1.0, 1.0, 1.0)
    - *bilijarska_lopta* (10.0, 10.0, 0.0)
    - Jedinična sfera *neparna* u koordinatnom početku
  - Ispis broja objekata koji su formirani.
  - Štampa zapreminu svake od lopti.

ZADATAK 3.

- Napisati u paketu **Geometrija** klasu **Tacka.java** takvu da sadrži:
  - Instancne promenljive, x i y koodrinata tačke.
  - Konstruktor na osnovu zadate x i y koordinate tačke.
  - Konstruktor na osnovu zadatog objekta Tacka.
  - Metod za pomeranje Tacke po x i y osi za x_pomeraj, y_pomeraj.
  - Metod za računanje rastojanja do zadate tačke.

- Napisati u paketu **<u>Geometrija</u>** klasu **<u>Duz.java</u>** takvu da sadrži:
  - Instancne promenljive, početna i krajnja tačka duži.
  - Konstruktor na osnovu zadate početne i krajnje tačke duži.
  - Konstruktor na osnovu zadate x i y koordinate početne tačke i x i y koordinate krajnje tačke.
  - Metod za računanje dužine duži.
  - Metod za određivanje preseka dveju duži, zadatih parametarski:
    - x=x_pocetak+(x_kraj-x_pocetak)*t
    - y=y_pocetak+(y_kraj-y_pocetak)*t

- Napisati u paketu **Geometrija** klasu **TestGeometrija.java** takvu da sadrži:
  - Kreiramo dve tačke (pocetak (0.0, 1.0) i kraj (5.0, 6.0)).
  - Kreiramo duž čija su početna i krajnja tačka baš ove zadate tačke.
  - Kreiramo drugu duž sa zadatim koordinatama početne i krajnje tačke.
  - Ispisati koje duži su formirane.
  - Pomeramo tačku kraj za (1.0, -5.0).
  - Ispisati novu duž.
  - Metod presek se poziva za jedan objekat tipa Duz, a drugi objekat uzima kao argument. Izračunati presek.