

Objektno-orjentisano programiranje

1 Objektno-orjentisano programiranje

- Uvod
- Metode u klasi
- Default vrednosti atributa
- Reference na objekte
- getters & setters
- Konstruktori
- Modifikatori pristupa

2 Klasa String

3 Wrapper klase za primitivne tipove

4 Nizovi i objekti

5 Kolekcije

6 Asocijacija

7 Konvencije davanja imena

Objektno-orijentisani program

- Objektno orijentisano programiranje se svodi na identifikaciju entiteta u nekom domenu, navođenje njihovih osobina i pisanje operacija nad tim osobinama.
- U objektno orijentisanoj terminologiji, entiteti su opisani klasama, osobine su atributi, a operacije su metode.

Sve je objekat

- Dakle, klasa je model objekta i uključuje attribute i metode
- U Javi je sve predstavljeno klasama, nije moguće definisati funkcije i promenljive izvan neke klase, zato listing počinje ključnom reči **class**
- Instance neke klase se zovu objekti
- Objekti se kreiraju upotrebom ključne reči **new**

Primer klase

```
class Automobil {  
    boolean radi;  
    void upali() {  
        radi = true;  
    }  
    void ugasi() {  
        radi = false;  
    }  
}
```

Primer kako se instancira

```
class Test {  
    public static void main(String args[]) {  
        Automobil a;  
        a = new Automobil();  
        a.upali();  
    }  
}
```

Metode u klasi

- izdvojeni skup programskog koda koji se može pozvati (izvršiti) u bilo kom trenutku u programu
- najlakše objasniti ako se posmatraju kao podprogrami (podalgoritmi) specifične namene
- Definicija metode:

```
povratni_tip ime_metode (parametri) {  
    programski kod  
}
```

- U klasi može da postoji više metoda sa istim imenom
- Razlikuju se po parametrima, metode se nikada ne razlikuju samo po povratnoj vrednosti
- Parametri mogu biti: primitivni tipovi, reference na objekte
- Rezultat može biti: primitivni tip, referenca na objekat
- Ukoliko metoda ne vraća povratnu vrednost navodi se rezervisana reč void u deklaraciji funkcije
- Metoda vraća vrednost naredbom :
`return vrednost;`

Default vrednosti atributa

- Atributi klasa imaju podrazumevane vrednosti (lokalne promenljive u metodama nemaju podrazumevane vrednosti i izazivaju grešku prilikom kompajliranja)

| Primitivni tip | Default |
|----------------|----------|
| boolean | false |
| char | '\u0000' |
| byte | (byte)0 |
| short | (short)0 |
| int | 0 |
| long | 0L |
| float | 0.0f |
| double | 0.0d |

- Reference kao atributi klase imaju podrazumevanu vrednost null što može izazvati NullPointerException grešku u radu aplikacije (runtime exception).

Reference na objekte

- Na steku su uvek samo reference na objekat, koji su zapravo na heap-u
- U Javi je sve referenca, nema pokazivaca
Automobil a;
a = new Automobil();
lokalna promenljiva **a** nije objekat, već referenca na objekat
- Objekat takođe može imati atribut koji nije primitivni tip tj. može imati referencu ka drugom objektu koji je na heap-u

- Ako želimo da inicijalizujemo referencu tako da ona ne ukazuje ni na jedan objekat, onda takvoj promenljivoj dodeljujemo **null** vrednost, odn. **null** literal:
Automobil a = null;

Operator dodele vrednosti

```
Automobil a = new Automobil();  
Automobil b = new Automobil();  
b=a;
```

- Operator dodele vrednosti nad objektima samo kopira referencu, ne i objekat

Ključna reč this

- Ključna reč **this** je referenca na objekat nad kojim je pozvana metoda
- Ovako nešto se često koristi unutar klasa u Javi

```
/** Atribut koji opisuje visinu */  
double visina;  
/** Postavlja vrednost atributa */  
void setVisina(double visina) {  
    this.visina = visina;  
}
```

getters & setters

- Ponekad je potrebno obezbediti kontrolisan pristup atributima, kako za čitanje, tako i za pisanje.
- To se postiže pisanjem odgovarajućih metoda kroz koje se pristupa atributima:

```
public class Automobil {  
    int maksimalnaBrzina;  
    boolean radi;  
  
    public int getMaksimalnaBrzina() {  
        return maksimalnaBrzina;  
    }  
  
    public void setMaksimalnaBrzina(int  
        maksimalnaBrzina) {  
        this.maksimalnaBrzina =  
            maksimalnaBrzina;  
    }  
}
```

- Ova kombinacija atributa i njegovog getter-a i setter-a se još zove i svojstvo (*property*).
- Ovim je omogućeno da se čitanje vrednosti svojstva sprovodi kroz njegov getter, a izmena kroz setter.
- Ako izostavimo setter, dobijamo *read only* svojstvo.

Inicijalizacija objekata

- Ako želimo posebnu akciju prilikom kreiranja objekta neke klase, napravićemo konstruktor
- Konstruktor se automatski poziva prilikom kreiranja objekta
`Automobil a = new Automobil();`
- Ako ne napravimo konstruktor, kompajler će sam napraviti default konstruktor, koji ništa ne radi
- U konstruktoru inicijalizujemo attribute koji bi trebalo da su inicijalizovani
- Konstruktor može primiti i parametre

Konstruktor bez i sa parametrima

```
public class Automobil {  
    int maksimalnaBrzina;  
    boolean radi;  
  
    public Automobil(){  
        radi=true;  
    }  
  
    public Automobil(int maksimalnaBrzina,  
        boolean radi) {  
        this.maksimalnaBrzina = maksimalnaBrzina;  
        this.radi = radi;  
    }  
}
```

- Konstruktor sa jednim parametrom koji je istog tipa kao i klasa u kojoj je definisan

```
public Automobil(Automobil a) {  
    this.maksimalnaBrzina = a.maksimalnaBrzina;  
    this.radi = a.radi;  
}
```

Primer pozivanja konstruktora

```
public class TestAutomobil {  
    public static void main(String[] args) {  
        Automobil a1=new Automobil();  
        Automobil a2=new Automobil(200,true);  
        Automobil a3=new Automobil(a2);  
    }  
}
```

Modifikatori pristupa

- **public** – vidljiv za sve klase
- **protected** – vidljiv samo za klase naslednice i klase iz istog paketa
- **private** – vidljiv samo unutar svoje klase
- nespecificiran (*package-private*) – vidljiv samo za klase iz istog paketa (direktorijuma, foldera)

Modifikatori pristupa

| oblast modifikator | Klasa | Paket | Klasa naslednica (isti paket) | Klasa naslednica (drugi paket) | Svet |
|-----------------------|-------|-------|-------------------------------------|--------------------------------------|------|
| public | + | + | + | + | + |
| protected | + | + | + | + | - |
| default | + | + | + | - | - |
| private | + | - | - | - | - |

- Niz karaktera je podržan klasom String. String **nije** samo niz karaktera – on je klasa!
- Objekti klase String se ne mogu menjati (*immutable*)!
- Izmena stringa konkatencijom ili dodelom novog string literala kreira se novi objekat na heap-u – alternativa StringBuffer ili StringBuilder klasa.
- Za cast-ovanje String-a u neki primitivni tip koristi se wrapper klasa i njena metoda parseXxx():
`int i = Integer.parseInt(s);`

- Reprezentativne metode:

- `str.length()`
- `str.charAt(i)`
- `str.indexOf(s)`
- `str.substring(a,b)`, `str.substring(a)`
- `str.equals(s)`, `str.equalsIgnoreCase(s)` – **ne koristiti `==`, jer se porede reference, a ne vrednosti**
- `str.contains(s)`
- `str.startsWith(s)`
- `str.toLowerCase()`

Wrapper klase za primitivne tipove

- Za sve primitivne tipove postoje odgovarajuće klase:
 - `int` → `Integer`
 - `long` → `Long`
 - `boolean` → `Boolean`
 - `short` → `Short`
 - `float` → `Float`
 - `double` → `Double`
 - `char` → `Character`
 - `byte` → `Byte`
- Imaju statičku metodu `Xxxx.parseXxxx()`
 - `int i = Integer.parseInt("10")`
 - `long l = Long.parseLong("10")`
- Ove wrapper klase rade automatski boxing i unboxing, odnosno automatsku konverziju primitivnih tipova u objekte i obrnuto kada je to potrebno

Nizovi i objekti

```
int a[] a = new int[5]; //sve vrednosti u nizu  
    su 0  
int a[] = { 1, 2, 3, 4, 5 };
```

```
Automobil[] parking = new Automobil[20]; //sve  
    vrednosti u nizu su null  
//dodaj objekte u niz  
parking[0] = new Automobil();  
parking[1] = new Automobil();  
  
//ili  
for(int i = 0; i < parking.length; i++)  
    parking[i] = new Automobil();
```

Kreiranje i popunjavanje

```
Automobil[] parking = {  
    new Automobil(),  
    new Automobil(),  
    new Automobil()};
```

- ili:

```
Automobil[] parking;
```

```
parking = new Automobil[] {  
    new Automobil(),  
    new Automobil(),  
    new Automobil()  
};
```

- Nizovi imaju jednu manu – kada se jednom naprave nije moguće promeniti veličinu.
- Kolekcije rešavaju taj problem.
- Zajedničke metode:
 - dodavanje elemenata,
 - uklanjanje elemenata,
 - iteriranje kroz kolekciju elemenata

Kolekcije

| <div>Implementacija</div> <div>Interfejs</div> | Hash table | Resizable Array | Balanced tree | Linked List | Hash table + Linked List |
|--|---------------|--------------------|------------------|----------------|-----------------------------|
| Set | HashSet | | TreeSet | | LinkedHashSet |
| List | | ArrayList | | LinkedList | |
| Map | HashMap | | TreeMap | | LinkedHashMap |

Klasa ArrayList

- Predstavlja kolekciju, odn. dinamički niz
- Elementi se u ArrayList dodaju metodom `add()`
- Elementi se iz ArrayList uklanjaju metodom `remove()`
- Elementi se iz ArrayList dobijaju (ne uklanjaju se, već se samo čitaju) metodom `get()`

Tipizirane kolekcije - Generics

- Tipizirane kolekcije omogućavaju smeštaj samo jednog tipa podatka u kolekciju.
- Tipizirane kolekcije se tumače kao: „kolekcija Stringova“ ili „kolekcija double brojeva“, i sl.
- Na primer:

```
ArrayList<Integer> lista = new  
    ArrayList<Integer>();
```

- Netipizirana kolekcija, mozemo ubaciti objekte razlicitih tipova

```
ArrayList lista = new ArrayList();  
lista.add(5);  
lista.add("Tekst");
```

Klasa ArrayList

```
ArrayList<Integer> lista = new
    ArrayList<Integer>();
lista.add(5);
lista.add(new Integer(5));
lista.add(1, 15);
System.out.println("Velicina je: " +
    lista.size());
lista.remove(0);
int broj = lista.get(0);
System.out.println(broj);
System.out.println("Velicina je: " +
    lista.size());
```

Prolaz kroz kolekciju

- Koristeći for petlju

```
for (int i = 0; i < lista.size(); i++) {  
    System.out.println("Broj je: " +  
        lista.get(i));  
}
```

- Koristeći foreach petlju

```
for (Integer el : lista) {  
    System.out.println("Broj je: " + el);  
}
```


Asocijacija

- Predstavlja vezu između dve klase
- Definišu se posebni atributi koji opisuju kakva je veza između klasa
- Za klasu Automobil možemo definisati vezu asocijacije sa klasom Deo
- Automobil može u sebi da sadrži više delova, dok jedan Deo može da se doda samo jednom Automobilu



Asocijacija klasa Automobil

- Za automobil može da bude dodato više delova i taj kraj veze definišemo u vidu atributa koji za tip ima kolekciju
- Kolekcija se mora inicijalizovati na praznu kolekciju

```
public class Automobil {  
    //Kraj veze za Asocijaciju  
    private ArrayList<Deo> delovi;  
  
    public Automobil() {  
        delovi=new ArrayList<Deo>(); //Inicijalizacija kolekcije  
    }  
  
    public ArrayList<Deo> getDelovi() {  
        return delovi;  
    }  
  
    public void setDelovi(ArrayList<Deo> delovi) {  
        this.delovi = delovi;  
    }  
}
```

Asocijacija klasa Deo

- Svaki deo odgovara tačno jednom automobilu
- Koristi se klasa Automobil kao tip atributa

```
public class Deo {  
    //Kraj veze za Asocijaciju  
    private Automobil automobil;  
  
    public Automobil getAutomobil() {  
        return automobil;  
    }  
  
    public void setAutomobil(Automobil automobil) {  
        this.automobil = automobil;  
    }  
}
```

- Ovaj kraj veze se ne mora navesti, ali se u tom slučaju za deo ne zna kom automobilu je dodeljen, već samo automobil zna koje delove ima

Postavljanje referenci asocijacije

- Moraju se postaviti odgovarajuće reference između kreiranih objekata klasa Automobil i Deo

```
Automobil a1=new Automobil(54,true);  
Automobil a2=new Automobil(23,false);
```

```
Deo d1=new Deo("Volan");  
Deo d2=new Deo("Volan sportski");
```

```
a1.getDelovi().add(d2);  
d2.setAutomobil(a1);
```

Konvencije davanja imena

- Nazivi klasa pišu se malim slovima, ali sa početnim velikim slovom (npr. Automobil, ArrayList).
- Ukoliko se naziv klase sastoji iz više reči, reči se spajaju i svaka od njih počinje velikim slovom (npr. HashMap).
- Nazivi metoda i atributa pišu se malim slovima (npr. size, width). Ako se sastoje od više reči, one se spajaju, pri čemu sve reči počevši od druge počinju velikim slovom (npr. setSize, handleMessage).
- Nazivi paketa pišu se isključivo malim slovima. Ukoliko se sastoje iz više reči, reči se spajaju (npr. mojpaket, velikipaket.malipaket).
- Detaljan opis konvencija nalazi se na adresi <http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>