

\$ OOP 05

\$ OOP - statičke promenljive i funkcije, nasleđivanje

Name: Ognjen Milinković[†]

Date: 15. mart 2021.

[†]ognjen_milinkovic@matf.bg.ac.rs

1. Implementacija klase

- Konstruktori

- Specifikatori vidljivosti

- Statičke promenljive i funkcije

- Prenos argumenata

2. Nasleđivanje

- Koncept nasleđivanja

- Nasleđivanje u Javi

- Dobre i loše strane

1. Implementacija klase

- Konstruktori

- Specifikatori vidljivosti

- Statičke promenljive i funkcije

- Prenos argumenata

2. Nasleđivanje

- Koncept nasleđivanja

- Nasleđivanje u Javi

- Dobre i loše strane

§ Konstruktor

- > Specijalna funkcija koja se poziva pri instanciranju objekta
- > Povratnu vrednost ne navodimo u definiciji
- > Njen zadatak je da **inicijalizuje stanje** objekta (dodeli vrednost poljima)
- > Mora se zvati **isto** kao i klasa u kojoj se nalazi
- > Klasa mora imati **barem jedan** konstruktor
 - * Ukoliko ne implementiramo konstruktor, java koristi podrazumevani

§ Podrazumevani konstruktor

- > Dostupan kada korisnik ne implementira konstruktor
- > Inicijalizuje sva polja na podrazumevane vrednosti (0, 0.0, false, null)
- > Kada implementiramo neki konstruktor, podrazumevani više ne postoji

§ Konstruktor kopije

- > Konstruiše kopiju prosleđenog objekta
- > Prosleđeni objekat je instanca iste klase čiji konstruktor definišemo

§ Konstruktor - this

- > Specijalnu promenljivu `this` možemo koristiti da u konstruktoru pozovemo već postojeći konstruktor (videti sledeći primer)
- > Na ovaj način možemo smanjiti ponavljanje koda

§ Konstruktori - primeri

```
class Tacka {  
    private double x, y;  
    public Tacka(double x, double y) {  
        this.x = x; this.y = y;  
    }  
  
    public Tacka() {  
        this(0, 0);  
    }  
  
    // Konstruktor kopije  
    public Tacka(Tacka t) {  
        this.x = t.x;  
        this.y = t.y;  
        // alternativno: this(t.x, t.y)  
    }  
}
```


§ Specifikatori vidljivosti

- > Uz deklaraciju funkcije ili promenljive možemo navesti **specifikator pristupa**
- > Ako ne navedemo specifikator, koristi se **podrazumevana** vidljivost

Tabela: Specifikatori pristupa

Specifikator	Klasa	Podklasa	Paket	Svet
private	da	ne	ne	ne
(podrazumevano)	da	ne	da	ne
protected	da	da	da	ne
public	da	da	da	da

- > Npr ako je **protected int x**, onda je pristup:
 - * Dozvoljen unutar klase
 - * Dozvoljen unutar podklase
 - * Dozvoljen unutar istog paketa
 - * Nije dozvoljan šire od paketa

§ Statičke promenljive

- > Nalaze se unutar klase
- > Doseg im možemo kontrolisati specifikatorima pristupa
- > Nisu vezane za objekat
 - * Ne opisuju stanje objekta
 - * Možemo ih koristiti da opišemo stanje klase

§ Statičke promenljive i funkcije - primer

```
public static void main(String[] args) {  
    System.out.println("pi = " + Math.PI);  
    System.out.println("e  = " + Math.E);  
    double k = Math.sqrt(16);  
    double p = Math.pow(2, 10);  
    System.out.println(k);  
    System.out.println(p);  
}
```

§ Statičke promenljive i funkcije - primer

```
class Tacka {  
    private double x, y;  
  
    // Privatna staticka promenljiva  
    private static int brojacTacki = 0;  
  
    public Tacka(double x, double y) {  
        this.x = x;  
        this.y = y;  
        brojacTacki++;  
    }  
  
    // Javno dostupna staticka funkcija  
    public static int kolikoJeNapravljenoTacaka() {  
        return brojacTacki;  
    }  
}
```

§ Inicijalizacija statičkih promenljivih

- > Ukoliko je inicijalizacija jednostavna može se uraditi tokom deklaracije
- > Inače, koristimo `inicijalizacioni blok`

§ Inicijalizacioni blok

```
class Primer {  
    // Inicijalizacija sa deklaracijom  
    private static String opis = "Brojaci cifara";  
  
    private static int[] brojaci;  
  
    // Inicijalizacioni blok  
    static {  
        brojaci = new int[10];  
        for (int i = 0; i < brojaci.length; i++)  
            brojaci[i] = 0;  
    }  
  
    public static int getBrojac(int i) {  
        return brojaci[i];  
    }  
}
```

§ Funkcija main

Zašto je main statička funkcija?

> Pretpostavimo da nije

§ Funkcija main

Zašto je main statička funkcija?

- > Pretpostavimo da nije
- > Onda je metod

§ Funkcija main

Zašto je main statička funkcija?

- > Pretpostavimo da nije
- > Onda je metod
- > Da bi pozvali metod, moramo instancirati objekat

§ Funkcija main

Zašto je main statička funkcija?

- > Pretpostavimo da nije
- > Onda je metod
- > Da bi pozvali metod, moramo instancirati objekat
- > Da bi instancirali objekat, program mora biti pokrenut

§ Funkcija main

Zašto je main statička funkcija?

- > Pretpostavimo da nije
- > Onda je metod
- > Da bi pozvali metod, moramo instancirati objekat
- > Da bi instancirali objekat, program mora biti pokrenut
- > Da bi program bio pokrenut treba pozvati main

§ Funkcija main

Zašto je main statička funkcija?

- > Pretpostavimo da nije
- > Onda je metod
- > Da bi pozvali metod, moramo instancirati objekat
- > Da bi instancirali objekat, program mora biti pokrenut
- > Da bi program bio pokrenut treba pozvati main
- > Ali main se ne može pozvati jer program nije pokrenut

§ Funkcija main

Zašto je main statička funkcija?

- > Pretpostavimo da nije
- > Onda je metod
- > Da bi pozvali metod, moramo instancirati objekat
- > Da bi instancirali objekat, program mora biti pokrenut
- > Da bi program bio pokrenut treba pozvati main
- > Ali main se ne može pozvati jer program nije pokrenut
- > Kontradikcija

§ Funkcija main

Zašto je main statička funkcija?

- > Pretpostavimo da nije
- > Onda je metod
- > Da bi pozvali metod, moramo instancirati objekat
- > Da bi instancirali objekat, program mora biti pokrenut
- > Da bi program bio pokrenut treba pozvati main
- > Ali main se ne može pozvati jer program nije pokrenut
- > Kontradikcija
- > Dakle, main mora biti statički kako bi ga pozvala JVM

§ Konstante

- > Koristi se ključna reč `final`
- > Ponašanje se razlikuje u zavisnosti od tipa (primitivni ili klasni)

§ Konstante - primitivni tipovi

Ako je promenljiva `primitivnog` tipa označena sa `final` onda:

> Njen sadržaj `ne možemo` menjati

```
public static void main(String[] args) {  
    final int x = 10;  
    x = 20;  
    // GRESKA!  
}
```


§ Konstante - klasni tipovi

Ako je promenljiva `klasnog` tipa označena sa `final` onda:

- > Njenu referencu `ne možemo` menjati
- > Njen sadržaj `možemo` menjati

```
public static void main(String[] args) {  
    final Tacka t = new Tacka(2, 3);  
    t.setX(100);  
    t.setY(200);  
    System.out.println(t);  
    // > (100, 200)  
}
```

§ Prenos argumenata

Argumenti se funkcijama u javi prenose:

- > Po vrednosti (primitivni tipovi)
- > Po referenci (klasni tipovi)

§ Prenos argumenata - primer

```
public static void primer1(int x, Tacka t) {
    x++;
    t.setX(100);
}

public static void primer2(final Tacka t) {
    t.setY(120);
}

public static void main(String[] args) {
    int a = 10;
    Tacka b = new Tacka(1, 1);
    primer1(a, b);
    System.out.println("a = " + a);
    System.out.println("b = " + b.toString());
    // > a = 10 b =(100, 1)
    primer2(b);
    System.out.println("b = " + b.toString());
    // > b = (100, 120)
}
```

§ Kako preneti objekat po vrednosti?

```
public static void prenosPoVrednosti(Tacka t) {  
    t.setX(100);  
    t.setY(120);  
}  
  
public static void main(String[] args) {  
    Tacka a = new Tacka(1, 1);  
    System.out.println("a = " + a.toString());  
  
    // poziva se konstruktor kopije koji  
    // pravi kopiju tacke 'a' i kopiju  
    // prenosi funkciji po referenci  
    prenosPoVrednosti(new Tacka(a));  
  
    System.out.println("a = " + a.toString());  
    // > a = (1, 1)  
}
```

1. Implementacija klase

- Konstruktori

- Specifikatori vidljivosti

- Statičke promenljive i funkcije

- Prenos argumenata

2. Nasleđivanje

- Koncept nasleđivanja

- Nasleđivanje u Javi

- Dobre i loše strane

§ Nasleđivanje

Zamislimo sledeću hijerarhiju za Zaposlene:

> Zaposleni

* Programer

- * Junior programer

- * Senior programer

* Dizajner

- * Grafički dizajner

* Direktor

- * Generalni direktor

- * Izvršni direktor

* Sekretar

> Da li uočavate neke sličnosti koje poseduju junior i senior programer?

> Šta je to što ima senior a nema junior?

> Šta je to što imaju junior i senior a nema (teorijski) niko drugi

§ Nasleđivanje

Nasleđivanje

- > Klasa B nasleđuje klasu A i to:
 - * Klasa B je **istovremeno** A
 - * Klasa B je i šira od klase A
 - * Dodatno ponašanje
 - * Dodatno stanje
 - * Klasa B je **specijalizacija** klase A
 - * Klasa A je **generalizacija** klase B

Primer

- > Junior programer nasleđuje klasu Programer
 - * Junior jeste programer
 - * Junior programer jeste specijalizacija pojma programer
 - * Programer je generalizacija pojmova junior programer i senior programer

§ Nasleđivanje

- > Omogućava nam da lako proširimo funkcionalnosti
- > Prirodno opisuje hijerarhije (jer i jeste hijerarhija)
- > Omogućava hijerarhijski polimorfizam (više o tome kasnije tokom kursa)

§ Nasleđivanje - Java

- > Klasa u javi nasleđuje najviše jednu klasu
- > Zapravo, klasa u Javi **uvek nasleđuje** jednu klasu
- > Ukoliko mi ne navedemo nasleđivanje eksplicitno, nasleđuje se Javina klasa **Object**
- > Metod `String toString()` uvek postoji u klasi jer je definisan u klasi **Object**
- > Svaki objekat u javi je istovremeno i instanca klase **Object**

§ Nasleđivanje - Java

Pretpostavimo:

- > Da imamo izuzetno bogatu klasu `Tacka`
- > Tokom rada shvatamo da želimo da naša tačka ima i svoje ime
- > Kako dodati ovu funkcionalnost?

Rešenje 1:

- > Menjamo klasu `Tacka`
- > Šta ako posle nekog vremena, ipak ne želimo da `Tacka` ima ime?

Rešenje 2:

- > Definišemo novu klasu `ImenovanaTacka`
- > Ona `nasleđuje` klasu `Tacka`
- > Ima sve funkcije koje je imala i klasa `Tacka`

§ Nasleđivanje - Java

```
class ImenovanaTacka extends Tacka {
    private String ime;
    public ImenovanaTacka(double x, double y, String ime) {
        super(x, y);
        this.ime = ime;
    }

    public ImenovanaTacka(String ime) {
        super();
        this.ime = ime;
    }

    public String toString() {
        return ime + " " + super.toString();
    }
}
```

§ Šta je super?

Šta je `super`?

- > Referenca na nadklasu
- > Slično kao `this`
- > Omogućava da `pristupimo` nadklasi
 - * Na primer da pozovemo `toString()` nadklase

Kako nasleđujemo?

- > Dodajemo konstrukciju `extends` ImeKlase
- > Prva instrukcija konstruktora mora pozivati konstruktor nadklase
 - * Koristimo `super`
 - * Na primer, `super()` poziva podrazumevani konstruktor nadklase

§ Nasleđivanje - dobre strane



- > Lako proširenje funkcionalnosti
- > Prirodno opisuje hijerarhije
- > Hijerarhijski polimorfizam
- > Olakšava održavanje koda

§ Nasleđivanje - loše strane



- > Duboke/široke hijerarhije
- > Nekada nije najbolje rešenje za dati problem¹
- > Loše projektovana hijerarhija je teška za održavanje i korišćenje
- > Postoji možda i jednostavnije i efikasnije rešenje?

¹Pročitati šta je composite obrazac za projektovanje.

Zahvalnica

Ovu prezentaciju je napravio kolega Nemanja Mićović.
Hvala kolegi Mićoviću što se saglasio da koristimo ovu prezentaciju.