

Java

nasleđivanje, kolekcije, obrada izuzetaka

Nasleđivanje – osnove

- ▶ Postoji samo jednostruko nasleđivanje
 - Jedna klasa može samo jednu naslediti, ali više klasa može nasleđivati istu klasu
- ▶ Ako ništa ne napišemo klasa nasleđuje Object klasu
- ▶ Ključna reč **extends**
- ▶ Primer nasleđivanja: student je osoba koja studira, profesor je osoba koja predaje na fakultetu. Imamo osnovnu klasu osoba (JMBG, ime i prezime, grad) i specijalizacije bi bile student (osoba koja ima indeks i ocene) i profesor (osoba koja radi na fakultetu i ima zvanje, platu, radno mesto, predmete koje drži).

Nasleđivanje – osnove

- ▶ Klasa koja nasleđuje drugu klasu ima sve metode i attribute klase koju nasleđuje i može dodavati nove attribute i metode, ali može i redefinisati postojeće metode
- ▶ Zavisno od modifikatora pristupa metode i atributi klase pretka su:
 - vidljivi unutar metoda klasa naslednica i mogu se pozivati nad objektima klasa naslednica – public
 - vidljivi unutar metoda klasa naslednica i ne mogu se pozivati nad objektima klasa naslednica – protected
 - nisu vidljivi unutar metoda klasa naslednica i ne mogu se pozivati nad objektima klasa naslednica – private
- ▶ Pored nasleđivanja veza između dve klase može biti i asocijacija i agregacija

Redefinisanje metoda – overriding

- ▶ Method overriding
- ▶ Pojava da u klasi naslednici postoji metoda istog imena i parametara kao i u baznoj klasi
- ▶ Anotacija `@Override`
- ▶ Primer:
 - klasa A ima metode **metoda1()** i **metoda2()**
 - klasa B nasleđuje klasu A i takođe ima metode **metoda1()** i **metoda2()**, ali samo **metoda1()** je redefinisana

Redefinisanje metoda – overriding

```
class A {  
    int metoda1() {  
        System.out.println("metoda1 klase A");  
    }  
    int metoda2() {  
        System.out.println("metoda2 klase A");  
    }  
}  
class B extends A {  
    @Override  
    int metoda1() {  
        System.out.println("metoda1 klase B");  
    }  
}  
...  
A varA = new A();  
B varB = new B();  
varA.metoda1();  
varB.metoda1();  
varA.metoda2();  
varB.metoda2();
```

Redefinisanje metoda – overriding

- ▶ Na konzoli će pisati

```
metoda1 klase A  
metoda1 klase B  
metoda2 klase A  
metoda2 klase A
```

primer 01



Ključna reč *super*

- ▶ Ključna reč *super* označava roditeljsku klasu. Ona se može koristiti i u metodama i u konstruktorima:

```
class BorbeniAvion extends Avion {  
    Top top;  
    Bomba[] bombe;  
    @Override  
    void sleti() {  
        System.out.println("BorbeniAvion odbacuje bombe.");  
        System.out.println("BorbeniAvion slece.");  
        super.sleti();  
    }  
    void pucaj() { ... }  
}
```

Ključna reč *super* u konstruktoru

- ▶ Ključna reč *super* u konstruktoru označava da pozivamo konstruktor roditeljske klase i tada se **mora** napisati na samom početku konstruktora klase naslednice:

```
public BorbeniAvion() {  
    super();  
    System.out.println("Konstruktor borbenog  
aviona.");  
    top = new Top();  
    bombe = new Bomba[] { new Bomba(),  
                           new Bomba() };  
}
```


Apstraktne klase

- ▶ Klase koje ne mogu imati svoje objekte, već samo njene klase naslednice mogu da imaju objekte (ako i one nisu apstraktne)

```
abstract class A {  
    int i;  
    public void metoda1() { ... }  
    public abstract void metoda2();  
    ...  
}
```

```
class B extends A {  
    @Override  
    public void metoda2() { ... }  
}
```

- ▶ Ako klasa ima makar jednu apstraktnu metodu, mora da se deklariše kao apstraktna.
- ▶ Apstraktna klasa ne mora da ima apstraktne metode!

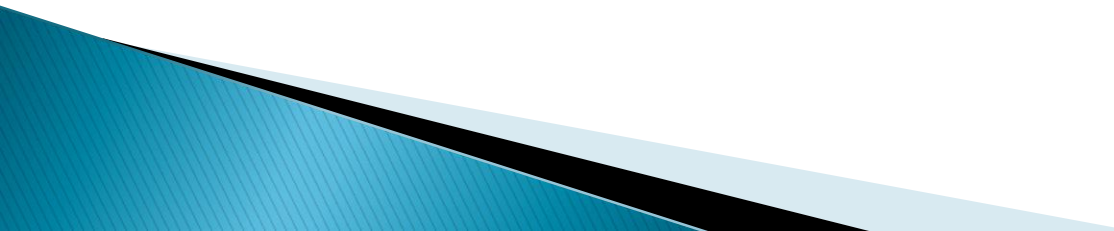
Polimorfizam

- ▶ Situacija kada se poziva metoda nekog objekta, a ne zna se unapred kakav je to konkretan objekat
 - ono što se zna je koja mu je bazna klasa
- ▶ Tada je moguće u programu pozivati metode bazne klase, a da se zapravo pozivaju metode konkretne klase koja nasleđuje baznu klasu

Polimorfizam

```
abstract class Vozilo {  
    abstract void vozi();  
}  
class Automobil extends Vozilo {  
    @Override  
    void vozi() { ... }  
}  
class Vozac {  
    void vozi(Vozilo v) {  
        v.vozi();  
    }  
}  
...  
Vozac v = new Vozac();  
v.vozi(new Automobil());
```

Interfejsi

- ▶ Omogućavaju definisanje samo apstraktnih metoda, konstanti i statičkih atributa
 - ▶ Ključna reč implements
 - ▶ Interfejs nije klasa! On je spisak metoda i atributa koje klasa koja implementira interfejs mora da poseduje.
 - ▶ Interfejsi se ne nasleđuju, već implementiraju
 - ▶ Da bi klasa implementirala interfejs, mora da redefiniše sve njegove metode
 - ▶ Jedan interfejs može da nasledi drugog
- 

Interfejsi

Jedna klasa može da implementira jedan ...

```
public class Racunar {  
    public HardDisk hardDisk;  
    public int upali() {  
    }  
}  
public interface HardDisk {  
    int pomeriGlavu();  
}  
public class SATAHardDisk implements HardDisk {  
    @Override  
    public int pomeriGlavu() {  
        ...  
    }  
}
```

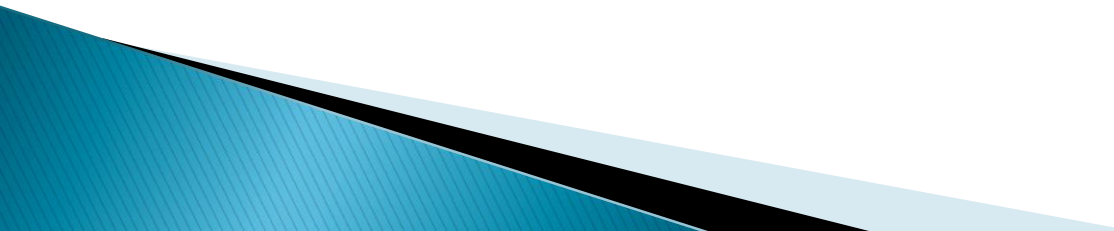
Interfejsi

► ... ili više interfejsa

```
interface USB {  
    void init();  
    byte[] getData();  
}  
  
interface Camera {  
    void init();  
    Picture getPicture();  
}
```

Interfejsi

```
class WebCam implements USB, Camera {  
    @Override  
    void init() { ... }  
    @Override  
    byte[] getData() { ... }  
    @Override  
    Picture getPicture() { ... }  
}
```



Kolekcije

- ▶ 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

Implementacija Koncept	Hash table	Resizable Array	Balanced Tree	Linked List	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

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.
- ▶ Primer:

```
ArrayList<String> kolekcija1 = new ArrayList<String>();  
kolekcija1.add("tekst");  
String s = kolekcija1.get(0);
```

foreach kroz kolekcije

```
ArrayList<String> kolekcija = new ArrayList<String>();  
kolekcija.add("tekst1");  
kolekcija.add("tekst2");  
kolekcija.add("tekst3");  
for (String s : kolekcija) {  
    System.out.println(s.length());  
}
```

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()`

Klasa ArrayList

```
ArrayList<Integer> lista = new ArrayList<Integer>();  
lista.add(5);  
lista.add(10);  
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());
```

Asocijativne mape

- ▶ Memorijske strukture koje omogućuju brzu pretragu sadržaja po ključu
- ▶ Element se ubacuje u paru sa svojim ključem, koji mora da bude jedinstven

Klasa HashMap

- ▶ Predstavlja asocijativnu mapu
- ▶ U HashMap se stavljaju dva podatka:
 - ključ po kojem će se pretraživati
 - vrednost koja se skladišti u HashMap i koja se pretražuje po ključu
- ▶ Metodom put() se ključ i vrednost smeštaju u HashMap
- ▶ Metodom get() se na osnovu ključa dobavlja (samo čita) vrednost iz HashMap
 - ako se ne nađe ključ, vratiće null

Tipizirana klasa HashMap

```
HashMap<String, String> ht =  
    new HashMap<String, String>();  
ht.put("E10020", "Marko Markovic");  
ht.put("E10045", "Petar Petrovic");  
ht.put("E10093", "Jovan Jovanovic");  
String indeks = "E10045";  
System.out.println("Student sa brojem indeksa " +  
    indeks + " je " + ht.get(indeks));  
indeks = "E10093";  
System.out.println("Student sa brojem indeksa " +  
    indeks + " je " + ht.get(indeks));
```


Izuzeci

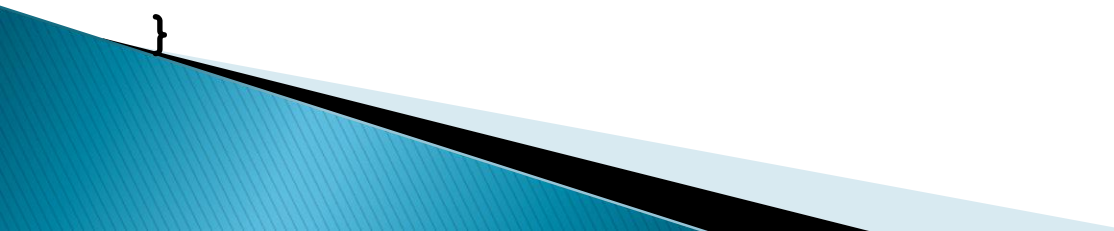
- ▶ Mehanizam prijavljivanja greške
- ▶ Greška se signalizira "bacanjem" izuzetka
- ▶ Metoda koja poziva potencijalno "grešnu" metodu "hvata" izuzetak
- ▶ Hijerarhija:
 - Throwable – roditeljska klasa
 - Error – ozbiljne sistemske greške
 - Exception – bazna klasa za sve standardne izuzetke
 - unchecked: RuntimeException i njene naslednice – ne moraju da se obuhvate try/catch blokom
 - checked: Ostale klase koje nasleduju Exception klasu i koje moraju da se obuhvate try/catch blokom

Izuzeci

- ▶ Checked (Exception i njene naslednice) – moraju da se uhvate
 - `EOFException`
 - `SQLException`
 - ...
- ▶ Unchecked (RuntimeException i njene naslednice) – ne moraju da se uhvate, jer mogu da se programski spreče
 - `NullPointerException`
 - `IndexOutOfBoundsException`
 - ...

Izuzeci

```
try {  
    // kod koji može da izazove  
    // izuzetak  
}  
catch (java.io.EOFException ex) {  
    System.out.println("Kraj datoteke pre vremena!");  
}  
catch (IndexOutOfBoundsException ex) {  
    System.out.println("Pristup van granica niza");  
}  
catch (Exception ex) {  
    System.out.println("Svi ostali izuzeci");  
}  
finally {  
    // kod koji se izvršava u svakom slučaju  
}
```



Izuzeci

- ▶ Programsko izazivanje izuzetka

```
throw new Exception("Ovo je jedan izuzetak");
```

- ▶ Korisnički definisani izuzeci

```
class MojException extends Exception {  
    MojException(String s) {  
        super(s);  
    }  
}
```

Izuzeci

▶ Ključna reč **throws**

```
void f(int i) throws MojException { ... }
```

▶ Propagacija izuzetaka

- ne moramo da obuhvatimo try-catch blokom, već da deklariramo da i pozivajuća metoda takođe baca izuzetak
- tako možemo da prebacujemo odgovornost hvatanja izuzetka na gore