

Nasleđivanje

Objektno-orjentisano programiranje 1

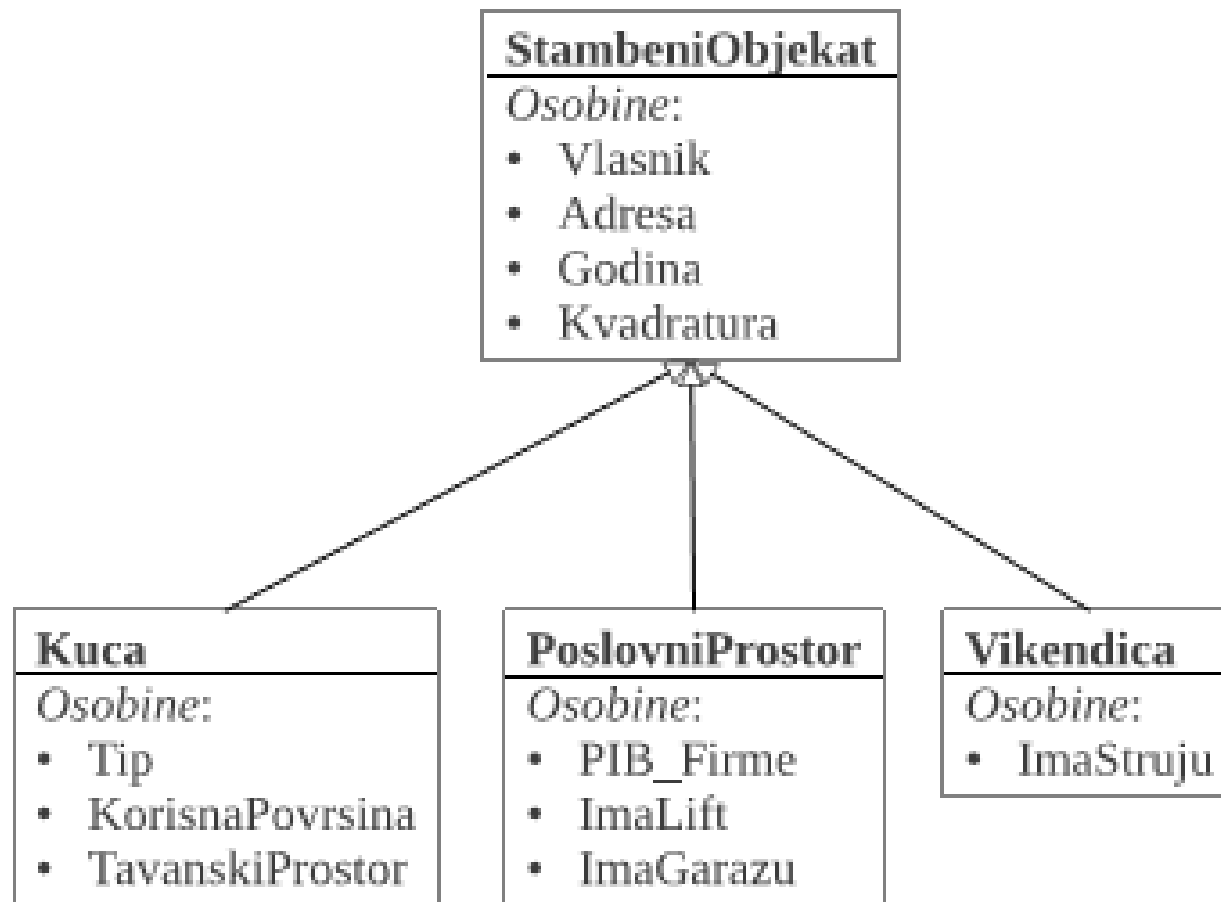




Nasleđivanje

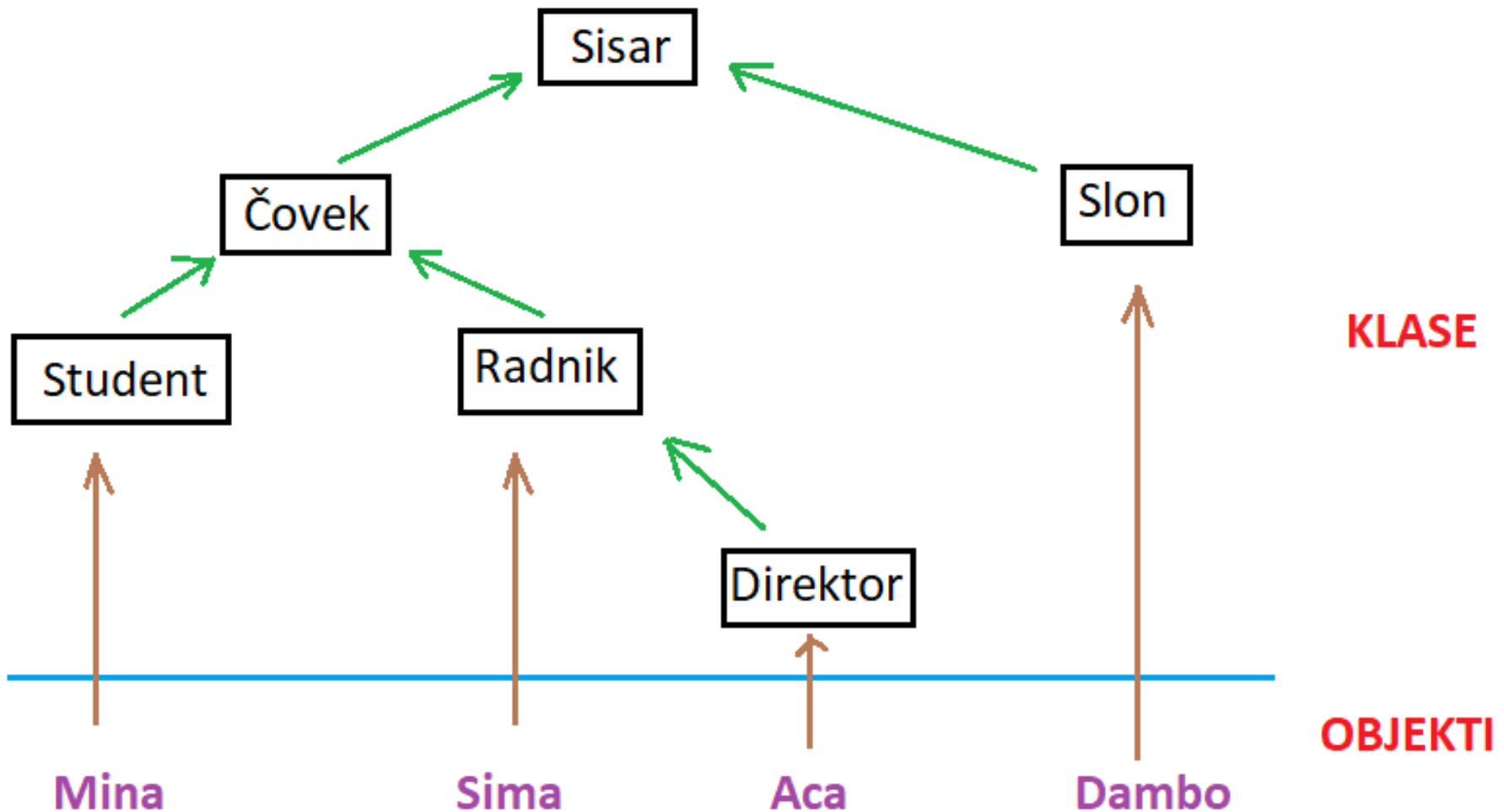
- Jedan od OO mehanizama za ponovno iskorišćenje koda
- U PJ Java imamo jednostruko nasleđivanje – klasa može naslediti tačno jednu klasu
- Ako klasa A nasleđuje klasu B tada ona nasleđuje sve njene attribute i metode
 - i privatni članovi se nasleđuju ali nisu vidljivi
 - modifikator **protected** – vidljivo u izvedenim klasama
- Izvedena klasa može dodavati nove attribute i metode, ali i **redefinisati (override)** nasleđene attribute i metode
- Direktno i indirektno nasleđivanje
- Ako klasa u Javi ne nasleđuje nijednu klasu tada ona implicitno nasleđuje klasu **Object** iz paketa **java.lang**

Nasleđivanje



Slika 2.4: Hijerarhija klasa stambenih objekata.

Hijerarhija klasa





Ključna reč extends

- Iza ključne reči **extends** u zaglavlju klase navodimo ime bazne klase

```
// ova klasa implicitno nasledjuje klasu Object
```

```
public class GeomFigura {  
    ...  
}
```

```
public class Pravugaonik extends GeomFigura {  
    ...  
}
```

```
public class Kvadrat extends Pravugaonik {  
    ...  
}
```

```
public class ObojenKvadrat extends Kvadrat {  
    ...  
}
```



this i super

- Dva atributa (polja) implicitno prisutna u svim objektima
- **this** – referenca na samog sebe
- **super** – referenca na delove objekta koji su nasleđeni
- Koristeći referencu **super** možemo pristupati nasleđenim metodama i atributima u slučaju da su redefinisani u izvedenoj klasi
- Konstruktor klase koristeći ključnu reč
 - **this** – poziva drugi konstruktor iz klase
 - **super** – poziva odgovarajući konstruktor iz nadklase
 - Koji konstruktor se poziva se određuje na osnovu broja i tipova argumenata

Konstruktori

- **Ako bazna klasa definiše bar jedan konstruktor tada izvedena klasa mora imati bar jedan konstruktor sa `super` pozivom**
 - super poziv sa odgovarajućim brojem argumenata mora biti prva naredba konstruktora u izvedenoj klasi
 - ili poziv nekog drugog konstruktora iz iste klase, ali lanac poziva konstruktora iz iste klase mora da se završi konstruktorom sa `super` pozivom
- **Ako klasa ne definiše nijedan konstruktor tada joj se dodaje podrazumevani konstruktor**
 - nema prametre i sadrži samo jednu naredbu: `super()`
- **Ako klasa definiše konstruktor i prva naredba konstruktora nije `super/this` poziv tada kompajler automatski dodaje `super()` kao prvu naredbu**
 - Greška tokom kompajliranja ako bazna klasa nema konstruktor bez argumenata



```
public class Foo {  
    private int x;  
  
    public Foo(int x) {  
        this.x = x;  
    }  
}  
  
public class Bar extends Foo {  
    private int y;  
  
    public Bar(int x, int y) {  
        super(x);  
        this.y = y;  
    }  
  
    public Bar(int y) {  
        this(0, y);  
    }  
}
```




```
public class Foo {  
    private int x;  
    public Foo(int x) { this.x = x; }  
}
```

```
public class Bar extends Foo {  
    private int y;  
  
    // ne prolazi kompajliranje!  
    public Bar(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

- (1) U **Bar(int, int)** kompajler automatski dodaje **super()**, ali **Foo** nema definisan konstruktor bez parametara
- (2) Nasleđeno polje **x** je privatno i nije vidljivo u klasi **Bar**



Indirektno nasleđivanje

```
public class Foo {  
    private int x;  
    public void m1() { ... }  
}
```

```
public class Bar extends Foo {  
    private int y;  
    public void m2() { ... }  
}
```

```
public class Baz extends Bar {  
    private int z;  
    public void m3() { ... }  
}
```

- Objekat klase Baz sadrži attribute **x**, **y** i **z**, i metode **m1**, **m2** i **m3**



Modifikator pristupa *protected*

```
public class Foo {  
    private int a;  
    protected int b;  
  
    private void m1()    { ... }  
    protected void m2() { ... }  
}  
  
public class Bar extends Foo {  
    public void m3() {  
        // a = 1;    ne kompajlira se  
        // m1();      ne kompajlira se  
  
        b = 4;  
        m2();  
    }  
}
```



Redefinisanje

- Izvedena klasa može redefinirati nasleđene attribute i metode

```
public class Foo {  
    protected int a = 2;  
  
    protected void m() {  
        a /= 2;  
        System.out.println(a);  
    }  
}
```

```
public class Bar extends Foo {  
    // redefinisan nasledjen atribut  
    private String a = "Pera";  
  
    // redefinisan nasledjen metod  
    public void m() {  
        System.out.println(a);  
    }  
}
```



Anotacija **@Override**

- Anotacije su dodatne informacije o klasi i delovima klase
- Počinju sa @ i daju se pre definicije klase, metoda, atributa, itd.
- Anotacije nemaju nikakav efekat na izvršavanje programa
- **Anotacije mogu biti zgodne kompajleru**
 - Detektovanje nekih vrsta grešaka na osnovu anotacija
 - Zanimarivanje nekih upozorenja (*warning*)
- Anotacija **@Override** data pre definicije metoda ukazuje da će taj metod redefinisati nasleđeni metod
 - Ako metod takve signature (zaglavlja) ne postoji u baznoj klasi generiše se greška u vremenu kompajliranja



Anotacija @Override

```
public class Foo {  
    protected int a = 2;  
  
    protected void m() {  
        a /= 2;  
        System.out.println(a);  
    }  
}  
  
public class Bar extends Foo {  
    // redefinisan nasledjen atribut  
    private String a = "Pera";  
  
    @Override  
    public void m() {  
        System.out.println(a);  
    }  
}
```



Anotacija @Override

```
public class Foo {  
    protected int a = 2;  
  
    protected void m() {  
        a /= 2;  
        System.out.println(a);  
    }  
}
```

```
public class Bar extends Foo {  
    // redefinisan nasledjen atribut  
    private String a = "Pera";
```

```
@Override
```

```
public void blabla() {  
    System.out.println(a);  
}
```

blabla() ne postoji u Foo
→ ne može se redefinisati

Klasa Bar ne prolazi kompajliranje!



Redefinisanje i super

- Referencu super možemo iskoristiti da pristupimo redefinisanim poljima i pozovemo redefinisane metode

```
class Foo {  
    protected int a = 2;  
  
    protected void m() {  
        a /= 2;  
        System.out.println(a);  
    }  
}  
  
class Bar extends Foo {  
    private String a = "Pera";  
  
    @Override  
    public void m() {  
        System.out.println(a);  
        System.out.println(super.a);  
        super.m();  
    }  
}
```




Nasleđivanje – primer

```
public class Pravugaonik {  
    protected double a, b;  
  
    public Pravugaonik(double a, double b) {  
        this.a = a;  
        this.b = b;  
    }  
  
    public double obim() {  
        return 2 * (a + b);  
    }  
}  
  
public class Kvadrat extends Pravugaonik {  
    public Kvadrat(double a) {  
        super(a, a);  
    }  
}
```



Nasleđivanje – primer

```
Pravugaonik p = new Pravugaonik(2, 3);  
System.out.println(p.obim());
```

```
Kvadrat k = new Kvadrat(4);  
System.out.println(k.obim());
```

```
// Liskov princip supstitucije  
// Svaki kvadrat je pravugaonik!
```

```
Pravugaonik q = new Kvadrat(5);  
System.out.println(q.obim());
```

```
// Nije svaki pravugaonik kvadrat!  
// ne prolazi kompajliranje  
// Kvadrat r = new Pravugaonik(4, 3);
```



```
public class Pravugaonik {  
    protected double a, b;  
  
    public Pravugaonik(double a, double b) {  
        this.a = a;  
        this.b = b;  
    }  
  
    public double obim() {  
        return 2 * (a + b);  
    }  
}
```

```
public class Kvadrat extends Pravugaonik {  
    public Kvadrat(double a) {  
        super(a, a);  
    }  
}
```

```
@Override  
public double obim() {  
    return 4 * a;  
}  
}
```



**Redefinisan metod
obim iz bazne klase**

Dinamičko vezivanje

- Na osnovu tipa objekta u vremenu izvršavanja se određuje koja od redefenisanih metoda se poziva

```
Pravugaonik p = new Pravugaonik(2, 3);  
double op = p.obim();  
// poziva se obim iz klase Pravugaonik
```

```
Kvadrat k = new Kvadrat(4);  
double ok = k.obim();  
// poziva se obim iz klase Kvadrat
```

```
Pravugaonik q = new Kvadrat(5);  
double oq = q.obim();  
// poziva se obim iz klase Kvadrat
```



Liskov princip supstitucije

```
public class Pravugaonik {  
    protected double a, b;  
  
    public Pravugaonik(double a, double b) {  
        this.a = a;  
        this.b = b;  
    }  
  
    public boolean podudaranSa(Pravugaonik drugi) {  
        return a == drugi.a && b == drugi.b;  
    }  
}
```

```
public class Kvadrat extends Pravugaonik { ... }
```

```
Pravugaonik p = new Pravugaonik(4, 4);  
Kvadrat k = new Kvadrat(4);  
if (p.podudaranSa(k))  
    System.out.println("Podudarni");
```



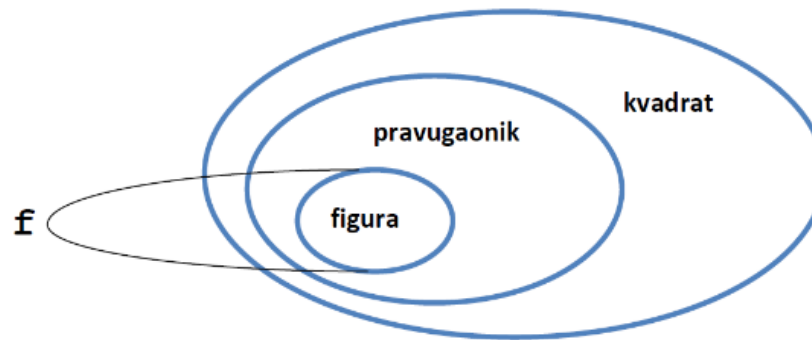
instanceof operator

- **Binarni infiksni operator kojim proveravamo da li je objekat instanca neke klase**
 - Binarni – dva operanda
 - Infiksni – operator između operanada
- `p instanceof C`
 - `p` – promenljiva referencijalnog tipa
 - `C` – ime klase (ime referencijalnog tipa)
- Rezultat: vrednost logičkog tipa (true/false)
- **Uzima u obzir nasleđivanje**

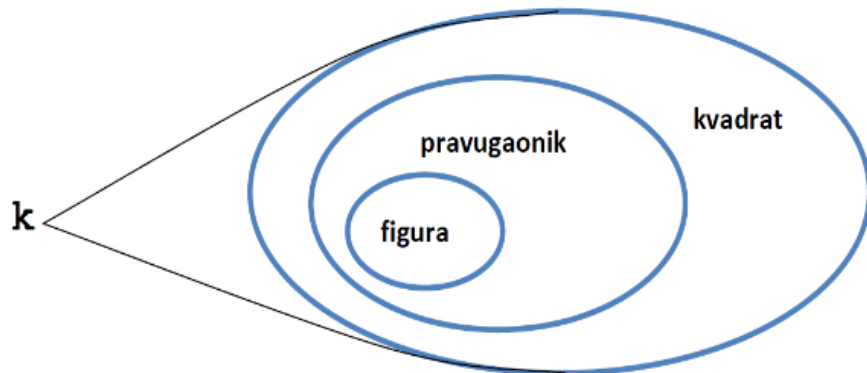
```
Kvadrat k = new Kvadrat(5);  
if (k instanceof Kvadrat) System.out.println("jeste");  
if (k instanceof Pravugaonik) System.out.println("jeste");  
if (k instanceof Object) System.out.println("jeste");
```

Eksplisitne konverzije referenci

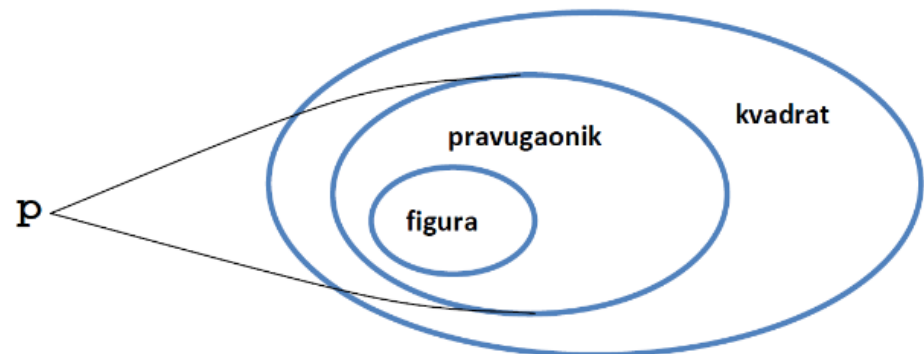
```
class Figura {}
class Pravugaonik extends Figura {}
class Kvadrat extends Pravugaonik {}
Figura f = new Kvadrat();
```



```
Kvadrat k = (Kvadrat) f;
```



```
Pravugaonik p = (Pravugaonik) f;
```





Klasa Object

- Klasa na vrhu hijerarhije nasleđivanja
- Neke od metoda klase Object
 - **boolean equals(Object o)**
 - Proverava da li su dva objekta identična po sadržaju
 - == i != su operatori kojima se porede reference ne objekti
 - **String toString()**
 - Vraća string reprezentaciju objekta
 - Poziva se kada konkatenujemo string objekat sa objektom nekog drugog tipa
 - Automatski se poziva od System.out.println()
 - **int hashCode()**
 - Vraća heš kod objekta, dva objekta identična po sadržaju moraju imati isti heš kod



```
public class Kvadrat extends Pravugaonik {
    public Kvadrat(double a) {
        super(a, a);
    }

    @Override
    public int hashCode() {
        return (int) a;
    }

    @Override
    public String toString() {
        return "Kvadrat stranice " + a;
    }

    @Override
    public boolean equals(Object o) {
        if (o instanceof Kvadrat) {
            Kvadrat k = (Kvadrat) o;
            return a == k.a;
        }

        return false;
    }
}
```



```
Kvadrat k1 = new Kvadrat(4);  
System.out.println(k1);  
// Kvadrat stranice 4
```

```
Kvadrat k2 = new Kvadrat(4);  
System.out.println(k1 == k2);  
// false
```

```
System.out.println(k1.equals(k2));  
// true
```

```
System.out.println(k1.hashCode() == k2.hashCode());  
// true
```

```
Kvadrat k3 = new Kvadrat(5);  
System.out.println(k3.equals(k1));  
// false
```



Apstraktne klase

- Apstraktne klase se deklarirajo s ključno rečjo **abstract**
- **Apstraktne klase se ne mogu instancirati**
 - Ne možemo primeniti new operator na apstraktnu klasu
- Apstraktne klase mogu da sadrže dve vrste metoda
 - Apstraktne metode – samo zaglavlje bez implementacije
 - Konkretne metode – zaglavlje i telo
- Apstraktne klase ne moraju da sadrže nijednu apstraktnu metodu
- Ako klasa sadrži barem jednu apstraktnu metodu tada ona mora biti apstraktna
- **Apstraktne klase nemaju smisla ako se ne naslede**

Apstraktne klase

- U apstraktnim klasama implementiramo opšte funkcionalnosti
- Specifične funkcionalnosti u apstraktnim klasama su apstraktne i implementiraju se u izvedenim klasama
- **Opšte funkcionalnosti mogu koristiti specifične funkcionalnosti**
 - Konkretna metoda iz apstraktne klase može pozvati apstraktnu metodu
 - **Izvršiće se neki konkretna metoda iz neke od izvedenih klasa po pravilu dinamičkog vezivanja**
- **Ako klasa nasleđuje apstraktnu klasu tada ona mora implementirati sve nasleđene apstraktne metode ili i sama mora biti apstraktna**
- **Hijerarhija klasa u velikim OO programima: od apstraktnih ka sve specifičnijim i specifičnijim klasama**



Apstraktna klasa – primer

```
public abstract class Nastavnik {  
    private String ime;  
  
    public Nastavnik(String ime) {  
        this.ime = ime;  
    }  
  
    // apstraktan metod  
    public abstract int brojCasova();  
  
    // konkretan metod (redefinisan iz Object)  
    // poziva apstraktan metod  
    public String toString() {  
        return ime + " ima " + brojCasova() + " casova";  
    }  
}
```



```
public class Asistent extends Nastavnik {  
    public Asistent(String ime) {  
        super(ime);  
    }  
  
    @Override  
    public int brojCasova() {  
        return 10;  
    }  
}
```

```
public class Profesor extends Nastavnik {  
    public Profesor(String ime) {  
        super(ime);  
    }  
  
    @Override  
    public int brojCasova() {  
        return 6;  
    }  
}
```



```
public class Katedra {  
    // niz ciji tip je apstraktna klasa!  
    private Nastavnik[] nastavnici;  
  
    public Katedra(Nastavnik[] nastavnici) {  
        this.nastavnici = nastavnici;  
    }  
  
    public double prosekCasova() {  
        if (nastavnici.length == 0)  
            return 0;  
  
        int ukupnoCasova = 0;  
        for (int i = 0; i < nastavnici.length; i++)  
            ukupnoCasova += nastavnici[i].brojCasova();  
  
        return (double) ukupnoCasova / nastavnici.length;  
    }  
  
    public void ispisiClanove() {  
        for (int i = 0; i < nastavnici.length; i++)  
            System.out.println(nastavnici[i]);  
    }  
}
```



```
public class KatedraDemo {  
    public static void main(String[] args) {  
        Nastavnik[] nastavnici = {  
            new Asistent("Mika"),  
            new Profesor("Zika"),  
            new Asistent("Ana"),  
            new Profesor("Mina"),  
            new Profesor("Mara")  
        };  
  
        Katedra k = new Katedra(nastavnici);  
        System.out.println(k.prosekCasova());  
  
        k.ispisiClanove();  
    }  
}
```

7.6

Mika ima 10 casova

Zika ima 6 casova

Ana ima 10 casova

Mina ima 6 casova

Mara ima 6 casova



Ključna reč final

- Ako je klasa definisana sa ključnom reči **final** tada se ona **ne može nasleđivati**

```
public final class Kvadrat {  
    ...  
}
```

- Ako je metod definisan sa ključnom reči **final** tada se on **ne može redefinisati** u izvedenim klasama

```
public final void izracunaj() { ... }
```

- Ako je atribut definisan sa ključnom reči **final** tada **on ne može menjati vrednost** (konstanta) nakon inicijalizacije

```
private final double pi = 3.14;
```



Modifikatori vidljivosti

Modifikator	Klasa / interfejs	Član klase
public	vidljivi u svim paketima	vidljivi za sve klase svih paketa
protected	/	vidljivi za sve klase svog paketa i za nasleđene klase ili interfejse iz bilo kog paketa
default (<i>bez modifikatora</i>)	vidljivi samo u okviru svog paketa	vidljivi za sve klase u istom paketu
private	/	vidljivi samo u okviru svoje klase