



# Programiranje I – RIN Računalništvo I – MA

## Osnove predmetno naravnane programiranja

# Kazalo

- Razred
- Predmet
- Dostopnost
- Dedovanje
- Večobličnost (polimorfizem)

# Razred

- Razred je pravzaprav predloga, ki definira nek tip predmeta.
- Razrede sestavljajo spremenljivke – lastnosti ter funkcije – metode.
- Metode omogočajo delo z lastnostmi, podatki v nekem razredu.
- Posebna metoda, tvoritelj ali konstruktor, služi za tvorjenje predmetov na osnovi razreda.
- Posebna statična metoda `main` je definirana samo v razredu, ki služi kot zagonski razred (ta razred lahko poženemo kot program)

# Predmet

- Predmet je realizacija razreda v programu.
- Vsak predmet v Javi moramo eksplicitno ustvariti z uporabo operatorja `new`.
- Dokler predmet ni ustvarjen ima vrednost `null`.
- Ustvarimo ga z eksplicitnim klicem konstruktorja (glej nižje)

# Dostopnost

- Posameznim delom določamo način dostopnosti,
- Poznamo tri načine dostopnosti:
  - public – javen, dostopa lahko vsak
  - private – lasten, dostopa lahko samo isti razred
  - protected – omejen, dostopa lahko isti razred oziroma dedovane izpeljanke (razredi, ki razširjajo ta razred)

# Dedovanje

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

- Razred definiran na osnovi obstoječega razreda
  - nadrazred – podrazred
  - podrazred je poseben primer nadrazreda
  - podrazred ima definirane dodatne lastnosti
  - podrazred **podeduje** vse lastnosti nadrazreda
- Večkratno dedovanje
  - Java – ni dovoljeno !
  - dovoljeno v C++, Smalltalk, ...
  - problemi: konceptualni, implementacijski, ...

# Dedovanje

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

- Pri dedovanju lahko pride do konfliktov imen
  - **prekrivanje** (angl. overriding)
    - nadrazred: <dostopnost> <tip> <ime-metode>(<parametri>);
    - podrazred: <dostopnost> <tip> <ime-metode>(<parametri>);
- Kocka in kvadrat imata metodo `narisi()`;
  - vsaka izriše svoj objekt

# Dinamično povezovanje

## ■ Problem

```
Kvadrat objs[] = new Kvadrat[100];
```

...

```
for (int i = 0; i < objs.length; i++) {  
    objs[i] je lahko Kocka ali Kvadrat;  
    prevajalnik zve kateri v času izvajanja;  
}
```

## ■ Rešitev

- dinamično povezovanje (angl. dynamic binding)
- med izvajanje se metoda dinamično poveže s kodo



# Dedovanje

```
public interface Petkotnik extends NKotnik {  
    ...  
}
```

- **Dedovanje med vmesniki**
- Enaka pravila kot pri razredih
- Java dovoljuje večkratno dedovanje med vmesniki
  - v primeru da so parametri definirani znotraj večih nad-vmesnikih morajo imeti isti tip

# Večobličnost

- Grško: **več oblik**
  - metode, objekti, ... imajo več oblik
  - odvisno iz katerega zornega kota jih gledaš
- Imamo več različnih oblik polimorfizma
  - **ad hoc, podtipi, parametričen**
- V literaturi je več različnih pojmovanj polimorfizma

# Polimorfizem

- “Ad hoc” polimorfizem
  - različni razredi imajo lahko metodo z isto signaturo
  - razredi ni potrebno, da so med seboj povezani
  - prekrivanje - prej predstavljeno
  - v času prevajanja ne vemo vedno za katero metodo gre
- Primer
  - kvadrat in tudi krog imata metodo
  - nariši();

# Polimorfizem

- **Polimorfizem zaradi podtipov**
  - angl. subtype polymorphism
  - Toneta lahko vidimo kot osebo, krojača, ...
  - objekt je član vseh nad-razredov  $\Rightarrow$  ima različne tipe
  - razred, nad-razredi po pod-razredi imajo lahko več metod z istim imenom in enakimi ali različnimi parametri
- **Primer**
  - primerek kocke lahko obravnavamo kot kocko ali kot kvadrat
  - kvadrat in kocka imata več metod nariši()

# Polimorfizem

- **Parametrični polimorfizem**
  - metodi z istim imenom in tipom vendar z različnim naborom parametrov
  - podpis metod se razlikuje samo v parametrih
  - metodi sta lahko definirani znotraj istega razreda ali v hierarhiji dedovanja
  - v času prevajanja vemo za katero metodo gre
- **Primer**
  - razred kocka ima dve metodi
    - izpiši();
    - izpiši(int rob); // debelina roba

# Implementacija vmesnika

- Razred implementira vmesnik
  - mora implementirati vse metode vmesnika

```
public class Trikotnik implements NKotnik {  
    ...  
}
```

- Vmesnik je lahko orodje za skupno delo

# Primeri

- Poglejmo si zdaj bolj obsežen primer
- Definirajmo hierarhijo geometrijskih objektov
- Reševanje problema sestoji iz dveh delov:
  - reševanje problema kot takšnega
    - za to potrebujemo samo papir in svinčnik ter idejo za rešitev
  - zapis rešitve na formalen način
    - to je lahko s psevdo-kodo, programsko kodo ali z matematičnimi izrazi
- Pri računalništvu počnemo oboje
- **Moramo ločiti med obema deloma**

# Primeri

- Reševanje problema sestoji iz dveh delov:
  - reševanje problema kot takšnega
    - za to potrebujemo samo papir in svinčnik ter idejo za rešitev
  - zapis rešitve na formalen način
    - to je lahko s psevdo-kodo, programsko kodo ali z matematičnimi izrazi



# Primeri

- Pri računalništvu počnemo oboje
  - ločimo med obema deloma!
- Primere bomo zato namenoma reševali postopoma:
  - najprej rešitev problema - definicija razreda in njegovih lastnosti
  - nato šele kodiranje

# Večkotnik

- Recimo, da sta lastnosti  $n$ -kotnika:
  - število stranic
  - obseg
- Definirajmo kako izgleda podpis razreda predmetov, ki so  $n$ -kotniki
  - opustimo tvoritelja

# Večkotnik

```
• public interface NKotnik {  
  
•     int stStranic();  
•     int obseg();  
  
• } // NKotnik
```

# Skupinsko delo

- Ker smo definirali podpis večkotnikov, jih lahko pričnemo uporabljati
  - Silvester lahko v svoji kodi že uporablja večkotnike, kot da bi jih že imeli
  - za testiranje si lahko pripravi preprosto izvedbo:

# Skupinsko delo

```
- public class MojNkotnik implements NKotnik {  
-     public NKotnik() {};  
-     public int stStranic() { return 0; };  
-     public int obseg()      { return 0; };  
- } // MojNkotnik
```

- Temu n-Kotniku rečemo tudi **točka**

# Skupinsko delo

- Njegovo resnično delo izgleda takole:

```
- public class MojeDelo {  
-     ...  
-     public int povpStranica(NKotnik ngon) {  
-         return ngon.obseg() / ngon.stStranic();  
-     }; // povpStranica  
-     ...  
- } // MojeDelo
```

- Manjka lovljenje sporočila o napaki pri napačnem izračunu - deljenje z nič

# Skupinsko delo

- **Medtem** Nikolaj v miru prične s programiranjem resničnega večkotnika - kvadrata
- Recimo, da ima kvadrat samo lastnosti n-kotnika
  - to, da ima štiri stranice in
  - da so te stranice enake, bo implicitno zapisano v kodi

# Kvadrat - ideja

- Nikolaj si zamisli kvadrat takole:
  - obe metodi, ki ju zahteva podpis *nKotnik* se da narediti in izračunati, če vemo, kako dolga je stranica kvadrata;
  - zato bo predmet iz razreda kvadrat pomnil le dolžino stranice;
  - metodi pa bo potem izračunal

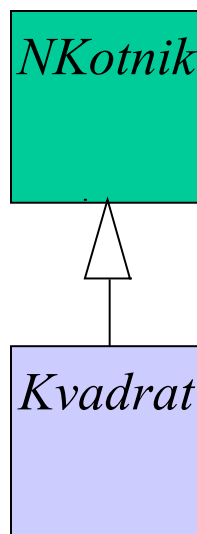


# Kvadrat

```
• public class Kvadrat implements NKotnik {  
•   protected final static int stranica= 4;  
•   protected int a;    /* to je dolzina stranice !! */  
•   /* ----- */  
•   /* -----[ create / destroy ]--- */  
•   public Kvadrat(int stranica) { a= stranica; };  
  
•   /* ----- */  
•   /* -----[ query ]--- */  
•   public int stStranica() { return stranica; };  
•   public int obseg()      { return stranica * a; };  
  
• } // kvadrat
```

# Izdelava razreda

- Na podlagi vmesnika smo izdelali razred



# Vmesniki lahko dedujejo

- Podobno kot razredi, lahko tudi podpisi podedujejo lastnosti po drugih podpisih
- Definirajmo  $n$ -kotnik, ki bo:
  - poleg lastnosti  $n$ -Kotnika
  - imel še lastnost vsote notranjih kotov

# n-koti n-kotnik

```
public interface NKoti extends NKotnik {  
    int vsotaKotov();  
} // nKoti
```

# Trikotnik

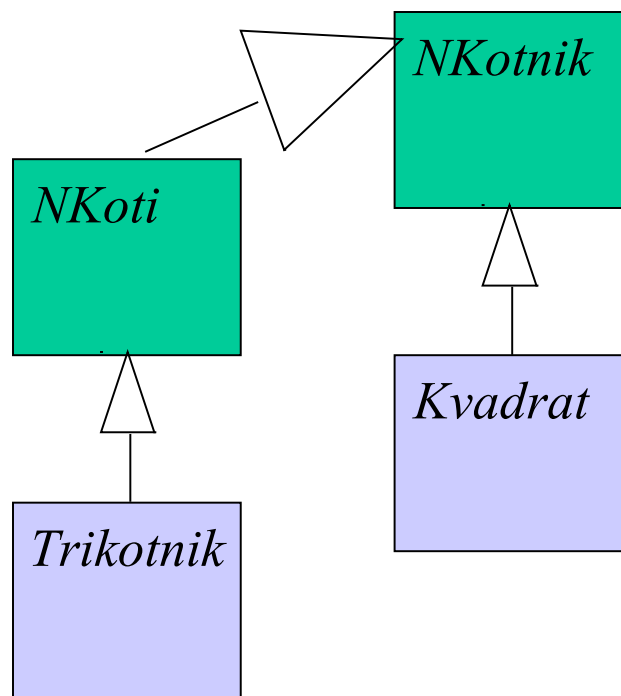
- Naredimo kot primera  $n$ -kotege  $n$ -kotnika (enakostranični) *trikotnik*
- Trikotnik ima podpis  $n$ -kotege  $n$ -kotnika
- Zato ima:
  - lastnosti  $n$ -kotege  $n$ -kotnika, ki
    - ima pa tudi vse lastnosti  $n$ -kotnika

# Trikotnik

```
• public class Trikotnik implements NKoti {  
•     protected final static int stranica= 3;  
•     protected int a;  
•     /* ----- */  
•     /* -----[ create / destroy ]--- */  
•     public Trikotnik(int stranica) { a= stranica; };  
•     /* ----- */  
•     /* -----[ query ]--- */  
•     public int stStranica() { return stranica; };  
•     public int obseg() { return stranica * a; };  
•     public int vsotaKotov() { return 180; }  
  
• } // trikotnik
```

# Hierarhija

- Odvisnost med podpisi in razredi



# Telesa

- Želimo imeti tudi 3D telesa
- Ponovno, najprej načrtovanje in na koncu kodiranje
- Lastnosti 3D teles:
  - prostornina
  - ...



# 3D stvari

```
public interface Telo {  
    float prostornina();  
} // telo
```

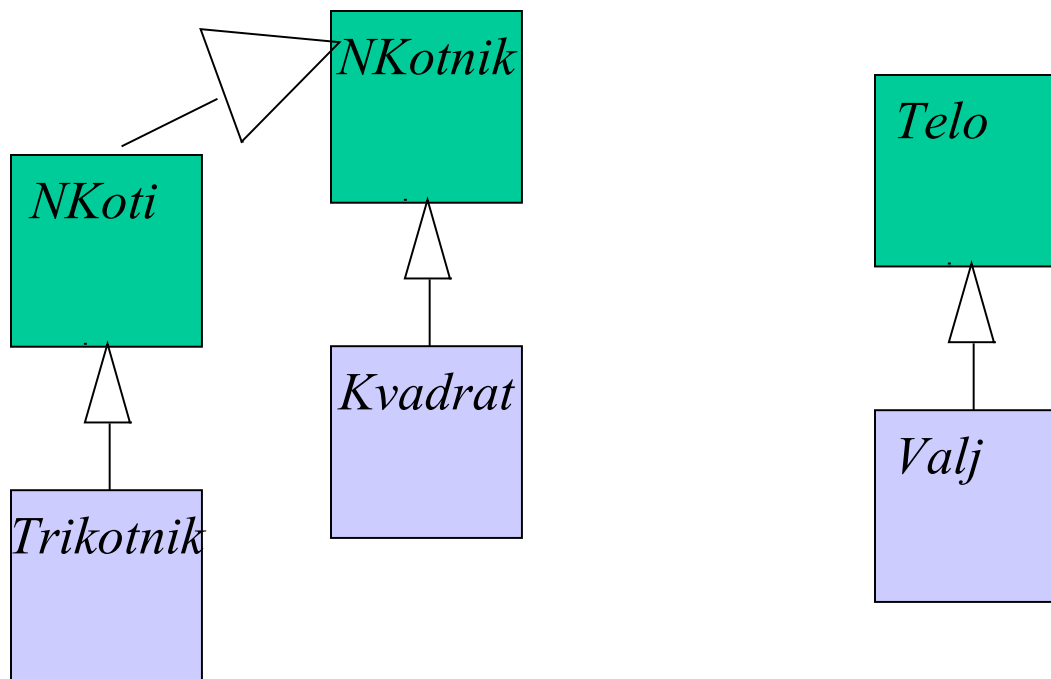
# Valj kot primer telesa

- Recimo, da bomo naredili razred valjev, kot primer teles
- Načrt:
  - za definicijo valja potrebujemo
    - njegovo višino
    - polmer osnovne ploskve

# Razred *valj*

```
• public class Valj implements Telo {  
•     protected float polmer;  
•     protected float visina;  
•     /* ----- */  
•     /* -----[ create / destroy ]--- */  
•     public Valj(float r, float v) {polmer=r; visina=v;};  
•     /* ----- */  
•     /* -----[ query ]--- */  
•     public float prostornina() {  
•         float rezultat;  
•         rezultat = Math.PI * Math.pow(polmer, 2.0) * visina;  
•         return rezultat;  
•     } // prostornina  
• } // valj
```

# Kaj smo dobili



# Hkratna izvedba večih vmesnikov

- V bistvu dedovanje po dveh vmesnikih
- Definirajmo *pravokotnik*
- Pravokotnik je *n-kotnik* in hkrati (izrojeno) *telo*
- Ima lastnosti tako *n-kotnika* kot *telesa*
- Zato ga lahko uporabljamo kot eno ali drugo - ***večobličnost*** ali ***polimorfizem***

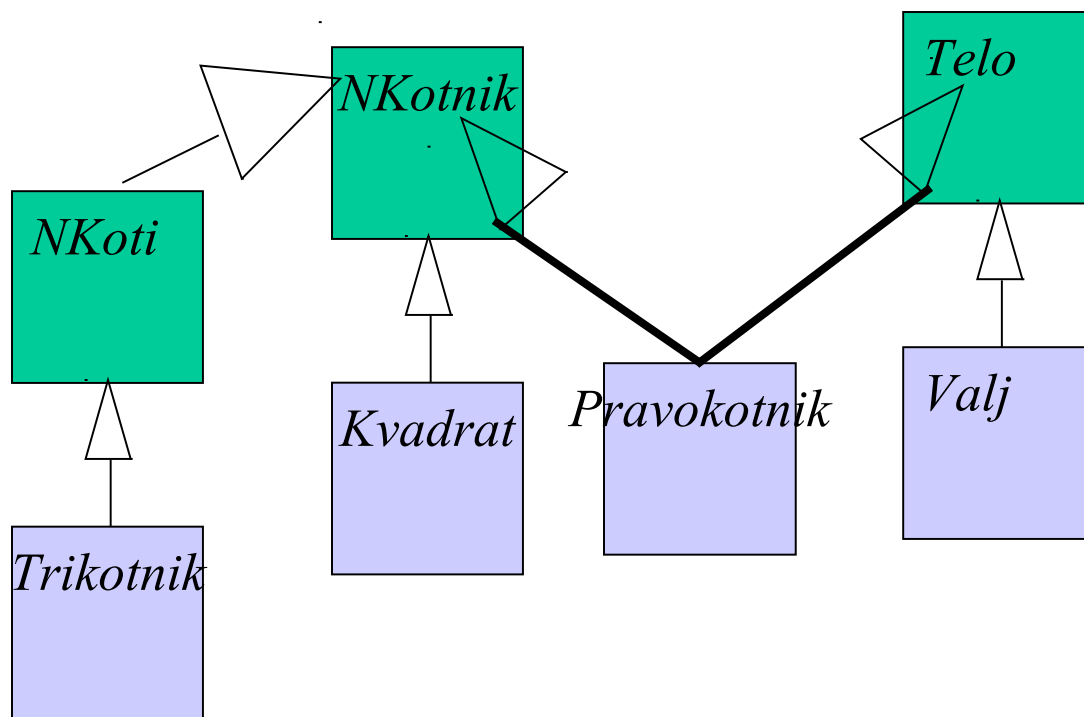
# Pravokotnik

```

• public class Pravokotnik implements NKotnik, Telo {
•     protected final static int stranic= 4;
•     protected int a, b;
•     /* ----- */
•     /* -----[ create / destroy ]--- */
•     public Pravokotnik(int strA, int strB) { ... };
•     /* ----- */
•     /* -----[ query ]--- */
•     public int    stStranic()    { return stranic;    };
•     public int    obseg()        { return 2 * (a + b); };
•     public float  prostornina() { return 0.0F;        };
•
• } // pravokotnik

```

# Kaj smo dobili



# Dedovanje po razredu in vmesniku

- Imamo dve obliki dedovanja:
  - pravo dedovanje: razširjanje ali dodajanje lastnosti (*extends*)
  - kot izvedba vmesnikov (*implements*)
    - lahko dedujemo po **večih** predhodnikih
  - kombinacija obeh



# Dedovanje po razredu in vmesniku

- V Javi:
  - se imenuje dedovanje samo pravo dedovanje
  - zaradi drugih pravil jezika pravo dedovanje ne dovoljuje nasledstva po večih razredih
- Kot primer kombinacije dedovanja naredimo kocko
  - podeduje lastnosti po kvadratu
  - in telesu

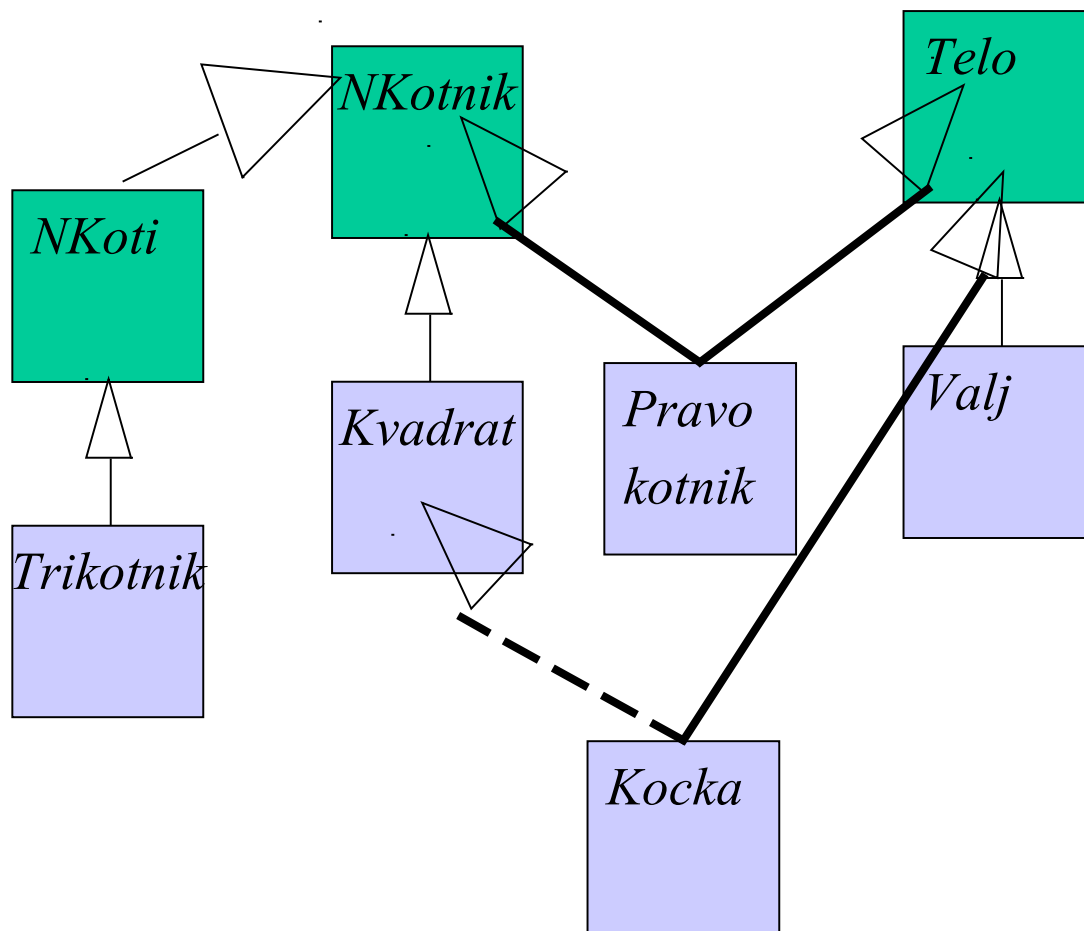
# Kocka

```

• public class Kocka extends Kvadrat implements Telo {
•
•   /* ----- */
•   /* -----[ create / destroy ]--- */
•   public Kocka(int stranica) { super(stranica); };
•
•   /* ----- */
•   /* -----[ query ]--- */
•   public int    stStranic()
•       { return 3 * super.stStranic(); };
•   public float prostornina() { ... };
• } // kocka

```

# Kaj smo dobili



# Abstraktni razredi in metode

- Poleg pravih razredov in vmesnikov obstajajo še **abstraktni** razredi
- Abstraktni razredi so:
  - razredi, torej jih lahko razširjamo, vendar naslednik lahko deduje samo po enem razredu
  - imajo metode, za katere ni nujno, da so narejene - so abstraktne
    - nasledniki jih morajo udejaniti
    - ne obstajajo predmeti iz abstraktnih razredov

# Abstraktni razred

## ■ Primer:

```
public abstract class Spremenljivka {  
    protected int sprem;  
    public int vrednost() { return sprem; }  
    public abstract void popravi(int vrednost);  
}
```

## ■ In potem:

```
public class MojaSprem implements Spremenljivka {  
    public void popravi(int vrednost)  
        { sprem= vrednost};  
}
```

# Abstraktni razred

- Abstraktna mora biti najmanj ena metoda
- Vsi izpeljani razredi iz razreda *spremenljivka* bodo imeli:
  - spremenljivko *sprem*
  - metodo za poizvedovanje o vrednosti *vrednost*
- **Morali** bodo narediti svojo metodo za popravljanje vrednosti *popravi*
- Primerjaj s splošno zamenjavo (*overloading*), kjer ta ni obvezna

# Vmesniki in abstraktni razredi

- Oboji služijo za **abstrakcijo** (*abstraction*) in **zakrivanje** (*encapsulation*)
- Razdeljevanje problemov na posamezne podprobleme je primer (razrede) je **modularizacija** (*modularization*)
- Vse tri tehnike so temeljni tehnike (orodja) v programerskem inženirstvu

# Programersko inženirstvo

- Najprej naredimo vmesnike
  - definiramo lastnosti predmetov, s katerimi se bomo ukvarjali
  - bolje kot abstraktne razrede
- Nato razdelimo delo med člane skupine
  - vsak od članov prične z delom na svojem kosu
  - za svoje potrebe si lahko naredi preprosto izvedbo drugih vmesnikov



# Povzetek

- Vmesniki
- Dedovanje
  - Vmesnika po (večih) vmesnikih
  - Razreda po enem razredu in/ali (večih) vmesnikih
- Abstraktni razredi – nekje med vmesnikom in razredom
- Dedujemo lahko samo po enem razredu
  - ker razred že ima kodo metode, pa ni jasno, katero kodo naj pri isti lastnosti podedujemo pri večkratnem dedovanju