

OOP



- ζ Princíp: „Samospráva“ objekty sú zodpovedné za to, že sú v konzistentnom stave
 - ψ Nastavenie hodnôt atribútov XY a použitie hodnôt atribútov XY: **setXY**, **getXY**
 - ψ Korektá inicializácia: **Konštruktor**
- ζ **this**: odkaz na objekt samotný (adresa samotného objektu)
- ζ **static**: statické atribúty a metódy, ktoré existujú **práve raz** pre celú triedu

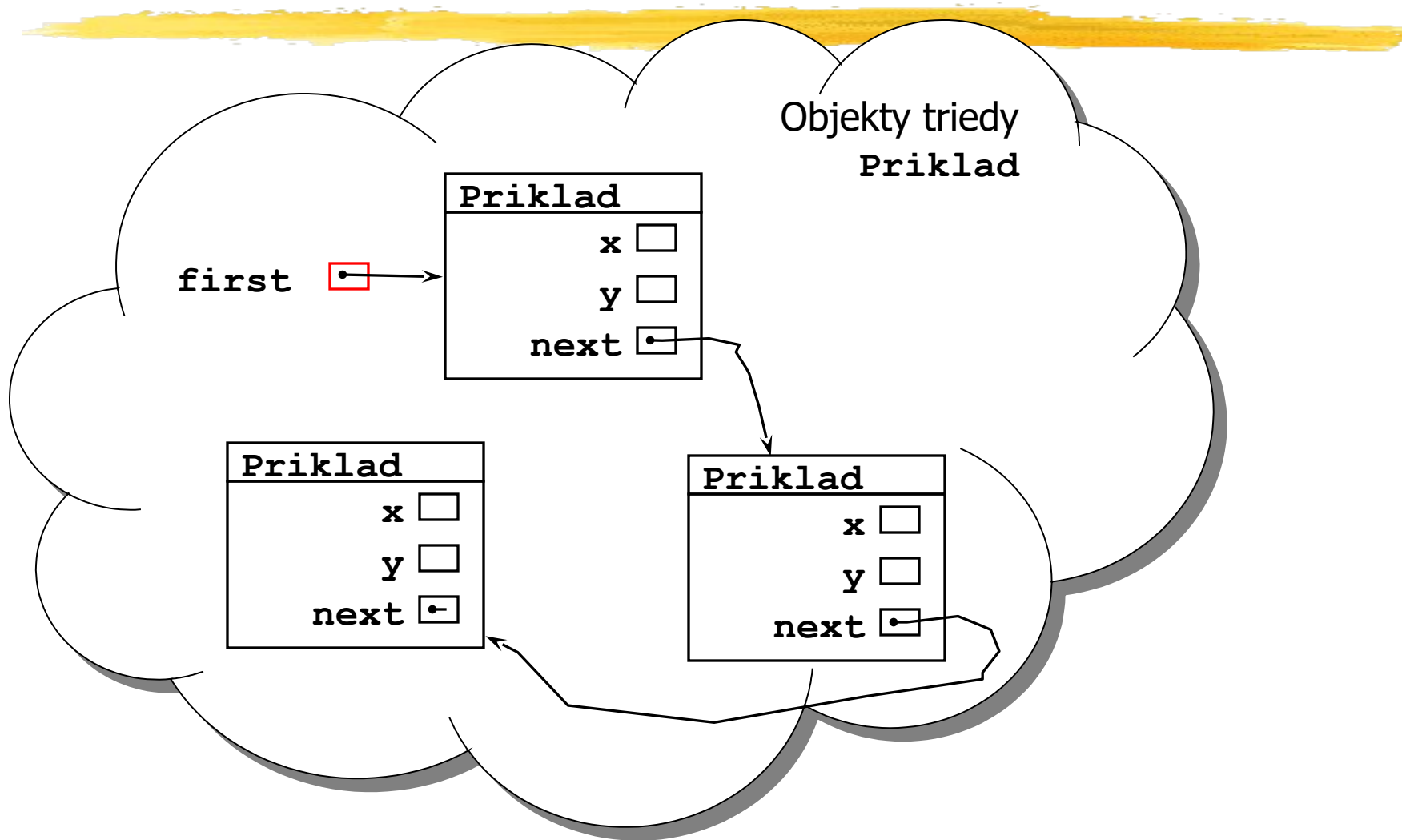
Príklad



```
public class Priklad
{   int x;
    int y;

    static Priklad first;
    Priklad next;

    public Priklad(int x, int y)
    {   this.x = x;
        this.y = y;
        next = first;
        first = this;
    }
    ...
}
```



OOP



Ďalšie princípy

- ζ Dedenie (špecializácia a generalizácia)
- ζ Pakety
- ζ Prístupové práva
- ζ Výnimky
- ζ Interfejsy

1. Dedenie (špecializácia - generalizácia)

1.1. Motivácia

Príklad: Správa používateľov (napr. Vo Výpočtovom stredisku) s viacerými kategóriami používateľov. Študenti, pracovníci (externí aj interní).

Osoby z rôznych skupín majú niektoré atribúty rovnaké a niektoré atribúty sú špecifické pre skupinu, podobne je to s metódami.

Príklad Skupiny užívateľov: externý používateľ Person

```
class Person
{ int    userId;
  String name;
  String telNr;

  Person(int id, String name, String tel)
  { this.userId = id;
    this.name    = name;
    this.telNr   = tel; }
}
```

Príklad Skupiny užívateľov:

Student

```
class Student
{ int    userId;
  String name;
  String telNr;
  String matrNr;

  Student(int id, String name, String tel, String matrNr)
  { this.userId = id;
    this.name    = name;
    this.telNr   = tel;
    this.matrNr  = matrNr; }
}
```

Príklad Skupiny užívateľov:

Pracovník

```
class Pracovník
{ int    userId;
  String name;
  String telNr;
  String titul;

  Pracovník(int id, String name,
             String tel, String titul)
  { this.userId      = id;
    this.name        = name;
    this.telNr       = tel;
    this.titul       = titul; }
}
```

Otázka:



Môžeme všetky tri skupiny spravovať v jednom poli (v jednom zret'azenom zozname) ?

Jednoduchá odpoveď: Ak sú takto definované, tak **nie**, keďže ide o tri rôzne triedy, objekty sú rôzneho typu.

Komplikovanejšia odpoveď:

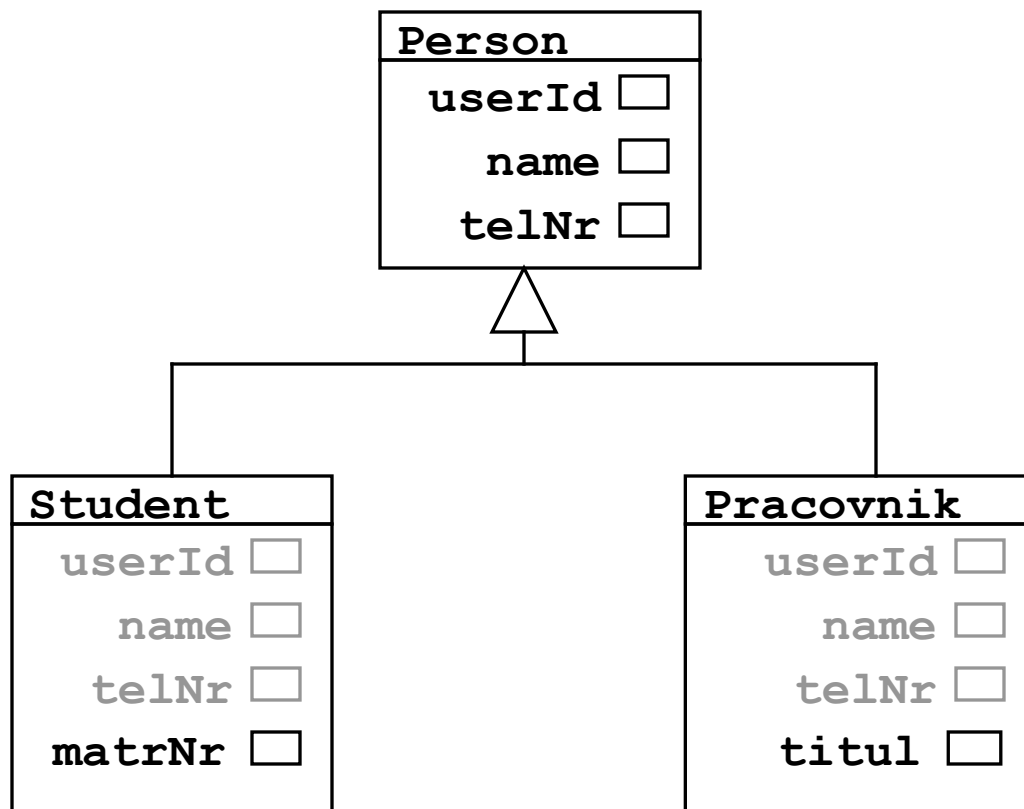


Môžeme pre objekty použiť jedno pole, ak by sme nejakým spôsobom zjednotily to, čo majú spoločné!

Pozorovanie:

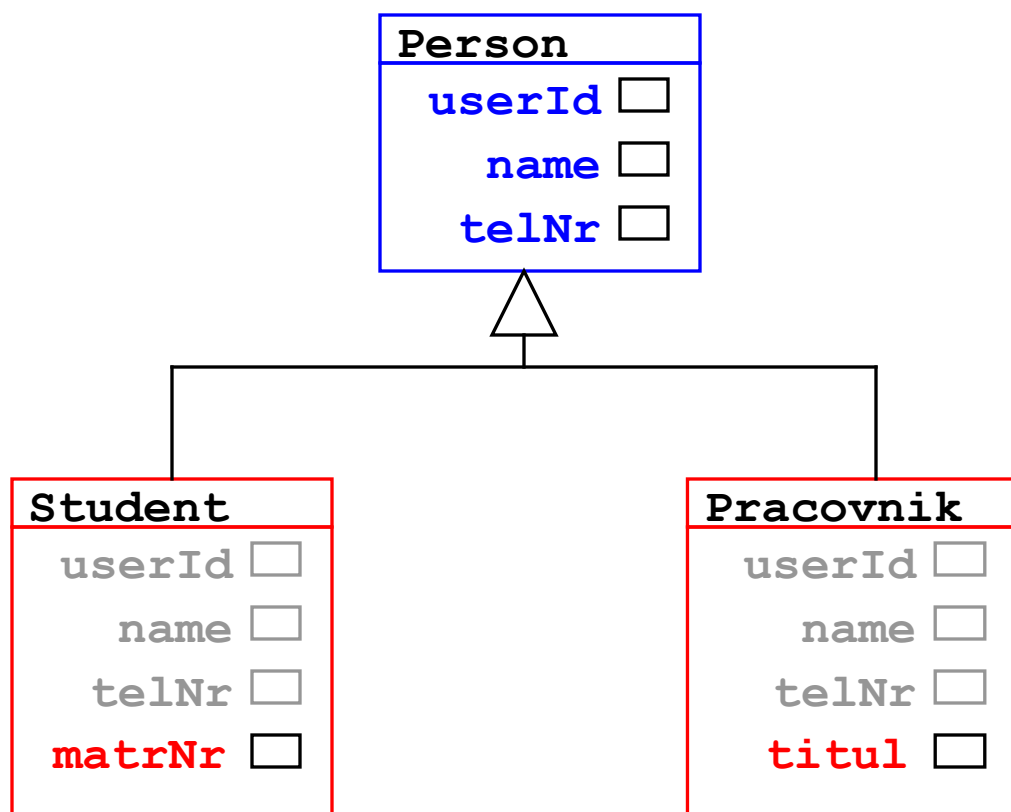
Spoločná informácia o študentoch a pracovníkoch je presne tá, ktorá je v triede **Person**.

1.2 Dedenie: Princíp



Student a Pracovnik **dedia** atribúty z triedy Person; všetko, čo môžeme robiť z inštanciami triedy Person, môžeme robiť aj s inštanciami tried Pracovnik a Student!

Dedenie



Špecializácia
↓
↑
Generalizácia

Dedenie v Java:

```
class Person
{ int    userId;
  String name;
  String telNr;

  Person()
  { }
```

Trieda **Person** bude
definovaná skoro rovnako!

```
Person(int id, String name, String telNr)
{ this.userId = id;
  this.name    = name;
  this.telNr   = telNr; }
}
```

Dedenie v Java:

```
class Student extends Person  
{
```

```
    String matrNr;
```

```
    Student(int id, String name,  
            String tel, String matrNr)
```

```
{    this.userId = id;  
    this.name    = name;  
    this.telNr   = tel;  
    this.matrNr  = matrNr; }  
}
```

extends je kľúčové slovo, ktoré znamená, že trieda **Student** dedí z triedy **Person**!

Kedže trieda **Student** dedí z triedy **Person**, bude mať trieda **Student** všetky atribúty triedy **Person** (bez toho, aby boli explicitne definované).

Dedenie v Java:

```
class Pracovnik extends Person
{
```

```
    String titul;
```

extends je kľúčové slovo, ktoré znamená, že trieda **Pracovnik** dedí z triedy **Person**!

```
    Mitarbeiter(int id, String name,
                  String tel, String titul)
```

```
{    this.userId      = id;
    this.name         = name;
    this.telNr        = tel;
    this.titul        = titul; }
```

Keďže trieda **Pracovnik** von der Klasse **Person** dedí z triedy **Person**, bude mať trieda **Pracovnik** všetky atribúty triedy **Person** (bez toho, aby boli explicitne definované).

```
}
```

Dedenie v Jave:



- ζ Tieto nové definície tried sú v podstate rovnaké s pôvodnými definíciami tried
- ζ Rozdiel je v tom, že teraz môžu byť adresy objektov triedy **Student** a **Pracovnik** priradené premennej typu **Person**, keďže špecializujú túto triedu!

Príklad



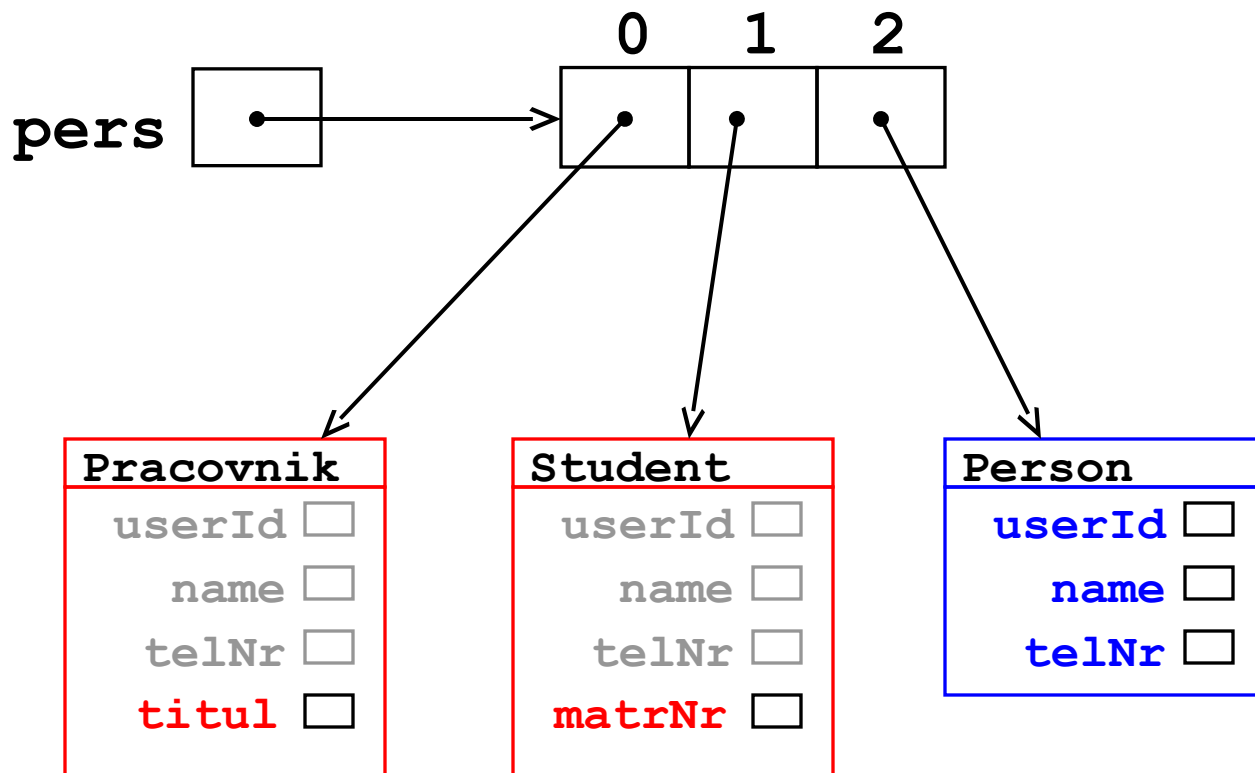
```
Person[] pers = new Person[3];
```

```
pers[0] = new Pracovnik(1, "Nový", "0905 ...", "Ing.");
```

```
pers[1] = new Student(2, "Prvák", "0907 ...", "0815");
```

```
pers[2] = new Person(5, "Externý", "0918 ...");
```

Príklad



Príklad



```
Person[] pers = new Person[3];
```

```
pers[0] = new Pracovnik(1, "Nový", "0905 ...", "Ing.");
```

```
pers[1] = new Student(2, "Prvák", "0907 ...", "0815");
```

```
pers[2] = new Person(5, "Externý", "0918 ...");
```

```
// Vypise na obrazovku telefony vsetkych objektov pola  
pers:
```

```
for (int i = 0; i < pers.length; i++)
```

```
{ System.out.println(pers[i].name + ":" + pers[i].telNr);
```

```
}
```

Princíp



- ζ Objekt triedy **B**, ktorá špecializuje triedu **A** (**dedí z A**), vlastní okrem atribútov a metód triedy **A** všetky atribúty a metódy triedy **B**.

Princíp

- ζ Adresa objektu triedy **B** môže byť uložená do premennej typu **A**, ak trieda **B** dedí z triedy **A**.
Príklad:

```
Person pers;  
pers = new Pracovnik(1, "Nový", "0905 ...", "Ing.");
```

- ζ Opačne to nie je možné! Každý **Pracovnik** je totiž **Person**, ale nie každý **Person** je aj **Pracovnik**

```
Pracovnik marb;  
marb = new Person(1, "Nový", "0905 ...");
```

Princíp



- ζ Objektom, ktorého adresa je uložená v premennej typu triedy **A**, môže byť ľubovoľný objekt z triedy **B**, ktorá dedí z triedy **A**.
- ζ Pre objekt ktorého adresa je uložená v premennej typu triedy **A** je možné pomocou bokovej notácie a premennej pristupovať ku všetkým atribútom a metódam triedy **A** —
ale iba k týmto! V prípade, ak je v premennej typu triedy **A** objekt triedy **B**, ktorá dedí z triedy, nie je možné cez premennú typu triedy **A** pristupovať k nezdedeným atribútom a metódam triedy **B**

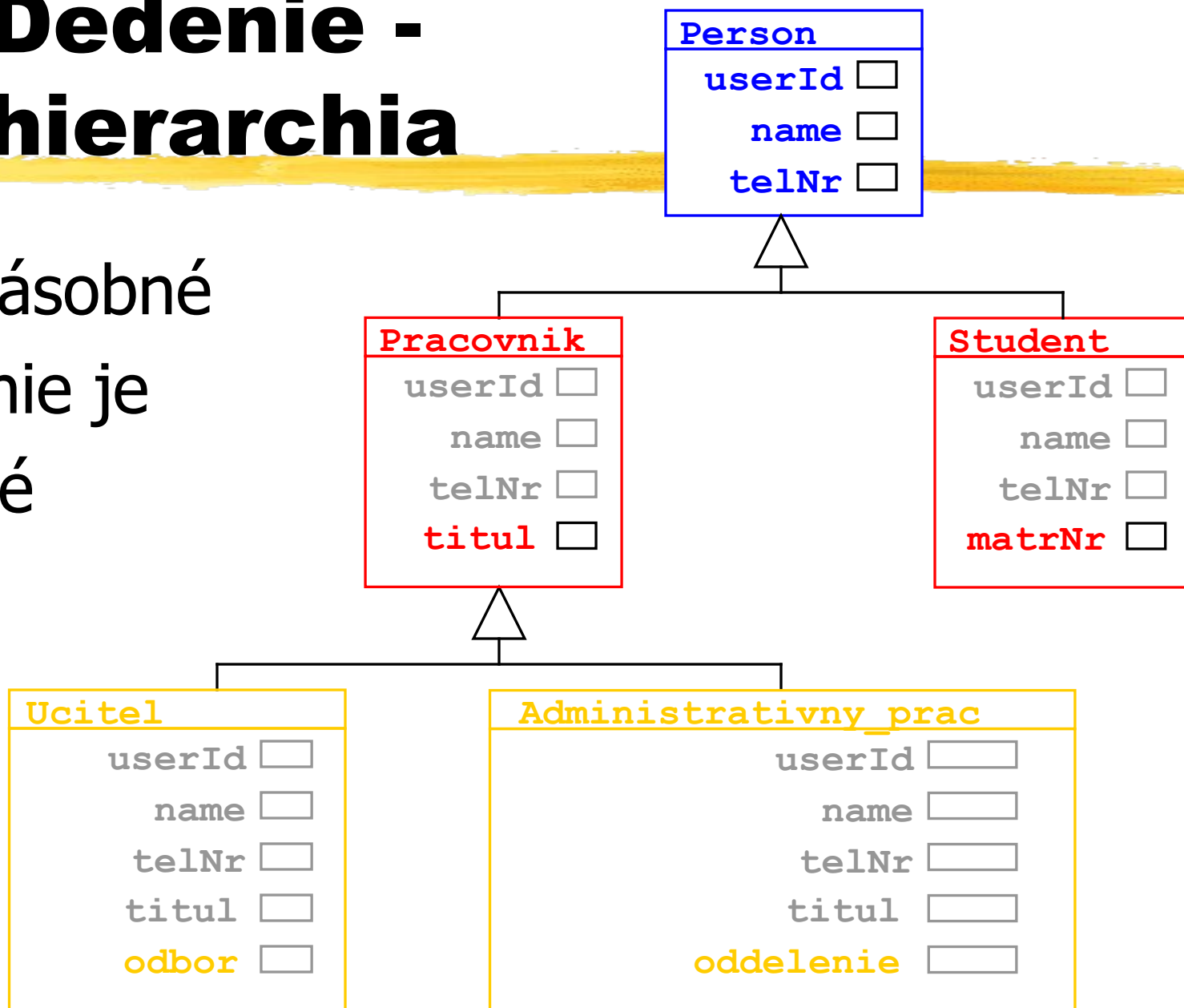
Príklad:

```
Person pers;  
pers = new Pracovnik(1, "Nový", "0905 ...", "Ing.");  
  
System.out.println(pers.name + ":" + pers.telNr);  
  
System.out.println(pers.titul);
```

Objekt, ktorého adresa je uložená v premennej **pers**, má síce atribút **titul**, ale premenná **pers** má typ **Person**!
Keďže trieda **Person** nemá atribút **titul**, nedá sa k nemu prístup pomocou premennej **pers**.

1.3 Dedenie - hierarchia

Viacnásobné
dedenie je
možné



Dedenie je tranzitívne



Ucitel dedí nepriamo cez **Pracovnika** atributy a metody od **Person**!

Adresa objektu triedy **Ucitel** môže byť uložená v premennej typu **Pracovnik** ale aj v typu **Person** a samozrejme aj premennej typu **Ucitel**.

V Java:



```
class Ucitel extends Pracovnik  
{
```

```
    String odbor;
```

```
    ...
```

```
}
```

Pozor:

Aby to skutočne fungovalo, musíme v triede **Pracovnik** definovať štandardný konštruktor **Pracovnik()**.

O chvíľu sa dozvieme prečo, resp. Kedy to nemusíme urobiť.

1.4 Dedenie metód



Podobne ako atribúty, metódy sa tiež dedia!

Príklad:



```
class Person
{ int    userId;
  String name;
  String telNr;

  Person() { }

  Person(int id, String name, String telNr)
  { /* ako doteraz */ }

  public String toString()
  { return name + ": " + telNr }
}
```

Metóda `toString` je použitá vždy, keď sa má objekt zmeniť na String: napr. Pri zavolaní metódy `println`.

Príklad:



```
Person pers = new Person(5, "Nieкто", "0907");
```

```
System.out.println(pers.toString());
```

```
/* Vypise na obrazovku:
```

```
* Nieкто: 0907
```

```
*/
```

Mohli sme tiež napísať:

```
System.out.println(pers);
```

Metóda `toString()` sa v takom prípade zavolá automaticky.

Príklad:



```
Pracovnik prac;  
prac = new Pracovnik(1, "Iný", "0918", "prof.");  
  
System.out.println(prac.toString());  
/* Vypise na obrazovku:  
 * Iný: 0918  
 */
```

Keďže **Pracovnik** dedí z triedy **Person**, budú objekty triedy **Pracovnik** dediť tiež metódu **toString**.

Samoyrejme, opäť stačí zavolať:
`System.out.println(prac);`

Príklad:

```
Person[] pers = new Person[3];
```

```
pers[0] = new Pracovník(1, "Nový", "0905 ...", "Ing.");
```

```
pers[1] = new Student(2, "Prvák", "0907 ...", "0815");
```

```
pers[2] = new Person(5, "Externý", "0918 ...");
```

```
for (int i = 0; i < pers.length; i++)
```

```
{ System.out.println(pers[i]); }
```

```
/* Vypise na obrazovku:
```

```
* Nový: 0905 ...
```

```
* Prvák: 0907 ...
```

```
* Externý: 0918 ...
```

```
*/
```

Nezabudnite:

`pers[i].toString()`
sa v Jave zavolá
automaticky!

1.5 Prepísanie – prekrytie metód = polymorfizmus

Doteraz: V triedach, ktoré dedia, sme pridávali nové atribúty (alebo metódy)

Teraz: V triedach, ktoré dedia, prepíšeme zdedené metódy.

Príklad:



```
class Pracovnik extends Person
{

    String titul;

    Pracovnik(int id, String name,
               String telNr, String titul)
    { \* ako doteraz *\ }

    public String toString()
    { return name + ":" + tel + ", " + titul; }
}
```

Príklad:



```
class Student extends Person
{

    String matrNr;

    Student(int id, String name,
            String telNr, String matrNr)
    { /* wie ako doteraz */ }

    public String toString()
    { return name + ":" + tel + ", " + matrNr; }
}
```

Príklad:

```
Person[] pers = new Person[3];
```

```
pers[0] = new Pracovník(1, "Nový", "0905 ...", "Ing.");
```

```
pers[1] = new Student(2, "Prvák", "0907 ...", "0815");
```

```
pers[2] = new Person(5, "Externý", "0918 ...");
```

```
for (int i = 0; i < pers.length; i++)
```

```
{ System.out.println(pers[i]); }
```

```
/* Vypise na obrazovku:
```

```
* Nový: 0905 ..., Ing.
```

```
* Prvák: 0907 ..., 0815
```

```
* Externý: 0918 ...
```

```
*/
```

Tu bude zavolaná **toString**-metóda triedy **Pracovník** resp. triedy **Student**!