Programovanie, algoritmy, zložitosť (UINF / PAZ1c)

Diel IV.

Róbert Novotný
robert.novotny@upjs.sk
15. 10. 2008

Ako nevynájsť koleso nanovo

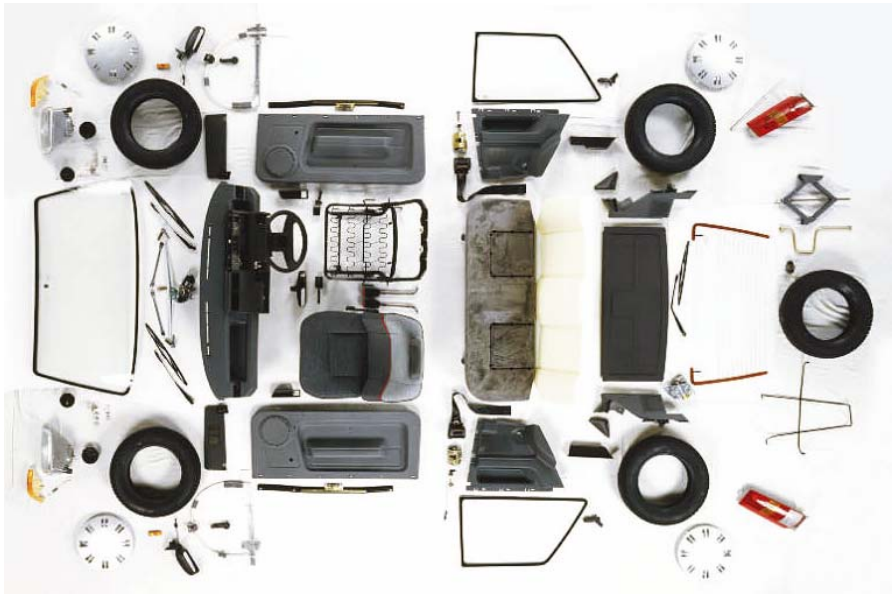
- znovupoužiteľnosť kódu je jedným z cieľov OOP
  - nebudeme vynachádzať ni teplú vodu, ni koleso, ni triedenie, ni spojový zoznam...
- dobre navrhnutú triedu možno prevziať a použiť v iných projektoch bez zmien
- vznikajú tak knižnice tried, kde megaprojekt vystaviame zo znovupoužiteľných súčastí

Ako vystavať babylonskú vežu

- v OOP jestvujú dva spôsoby ako vystavať zložitú triedu z jednoduchších
  - kompozícia: skladanie. Stav zložitej triedy je tvorený jednoduchšími triedami
  - dedičnosť: odvodzovanie/špecializácia z jednoduchšej/všeobecnejšej triedy
  - delegácia: hybrid

Kompozícia tried

- Automobil sa skladá z motora, prevodovky, volantu...
  - Motor pozostáva z valcov, sviečok, ...
    - Valec pozostáva z ...



Kompozícia v OOP

Kompozícia v OOP je jednoduchá.
Súčasti uvedieme ako stav triedy:

```
class Auto {
   Motor motor;
   Prevodovka prevodovka;
   Volant volant;
}
```

```
public class Motor {
   Valec[] valce;
   Sviečky[] sviečky
}
```

```
public class Valec {
   ...
}
```

V ľudskej reči je kompozícia vyjadrená slovom „má“ (*has a*). Auto má motor. Motor má valec.

Dedičnosť tried

- Dedičnosť umožňuje odvodiť objekt od iného, pridať mu nové schopnosti / stav a prípadne zmeniť existujúce.
- Auto *je* vozidlo, ktoré je motorové, dvojstopové a poháňané benzínom.
- Človek *je* dvojnožec, ktorý nemá perie.

  – *Platón, Politikus, 4. stor. pnl*
- Študent *je* človek, ktorý navštevuje školu, je v nejakom ročníku...

Dedičnosť v Jave

Chceme zaviesť psa-cirkusanta

```
class Pes {
   private int vek;
   String štekaj() {
      return "Haf haf";
   }
}
```

```
class CirkusovýPes extends Pes {
   void derivuj() {
        System.out.println("derivácia
        funkcie x je 1");
   }
}
```

- `CirkusovýPes` je `Pes`, ktorý vie navyše derivovať.
- Cirkusový pes *zdedí* všetky metódy
  - teda vie štekať
  - a vie derivovať

Poznámky k dedičnosti a kompozícii

- Pôvodné triedy sú často ako binárky (niekto nám ich venuje, predá…) a nemáme ich zdrojový kód, a teda nemôžeme ich meniť.
- Výhodou kompozície a dedičnosti je to, že pôvodné triedy nemusíme meniť.
- Kedy kompozícia a kedy dedičnosť?
  - mnoho hádok a debát
  - **„uprednostňujte kompozíciu pred dedičnosťou“**

    -- Gang of Four, autori Design Patterns
  - niekedy je hranica nejasná

Poznámky k dedičnosti a kompozícii

Kontrolné otázky:

- *kompozícia* (has a?): objekt má / pozostáva z / je tvorený z iného objektu
  - Auto má motor.
- *dedičnosť* (is a?): objekt je špeciálnym prípadom / odvodený z iného objektu
  - Pleso je jazero, ktoré… Každé pleso je jazerom.
  - Žirafa je zviera, ktoré… Každá žirafa je zvieraťom.

Protipríklady:

- Auto nemôže dediť z motora, lebo nie každé auto je motorom.
- Žirafa nie je kompozíciou zvieraťa, lebo nepozostáva zo zvieraťa.

Slyšte teoretika – fundamentálne pojmy OOP

- **triedy**

  V našom informačnom systéme žijú psy a veľryby.

- **objekty**

  Dunčo a Lajka sú psami

- **abstrakcia**

  Ak chcem, aby Dunčo štekal, nemusím ovládať prenos nervových signálov z mozgu psa do hlasiviek.

- **zapúzdrenie**

  Krotiteľa psov nezaujíma, že na to, aby pes zaštekal, musíme mať nažratého a bdelého a nezachrípnutého a... psa

- **dedičnosť**

  Zvieratá vydávajú zvuky. Psy a veľryby sú zvieratá.

  Psy a veľryby teda vydávajú zvuky.

- **polymorfizmus**

  Zvieratá vydávajú zvuky. Každé zviera (či už pes alebo veľryba) vydáva zvuky svojským spôsobom.

Dirigenti a orchestrovia

- **Úloha:** Ste dirigentom minikomorného orchestra, v ktorom sú nasledovné nástroje:
  - husle (20)
  - lesné rohy (2)
  - flauty (2)
  - tympany (1)

- Každý nástroj dokáže zahrať tón. Navrhnite postup, ktorým dokáže orchester naraz zahrať *Kohútik jarabý*.

Dirigenti a orchestrovia

```
public class Husle {
    void zahraj(int tón) {
        System.out.println("Husličky hrajú tón " +
      tón);
    }
}
```

```
public class Flauta {
    void zahraj(int tón) {
        System.out.println("Flauta hrajú tón " + tón);
    }
}
```

Dirigenti a orchestrovia

- Idea triedy pre orchester:
  - orchester je tvorený mnohými nástrojmi
  - ak má orchester zahrať 1 tón, prejdeme každý zoznam a zavoláme metódu `zahraj()` na konkrétnom nástroji.

Dirigenti a orchestrovia

- Idea triedy pre orchester:
  - orchester je tvorený mnohými nástrojmi
  - budeme mať pre každý typ nástroja jeden zoznam
    - zoznam huslí (20 prvkový)
    - zoznam lesných rohov (2 prvk
    - zoznam fláut (2 prvky)
    - zoznam tympanov (1 prvok)

Dirigenti a orchestrovia

```
public class Orchester {
    private ArrayList<Husle> husle = new
      ArrayList<Husle>();
    private ArrayList<Flauta> flauty = new
      ArrayList<Flauta>();
    …
    public void zahraj(int tón) {
        for(Husle h : husle) {
              h.zahraj(tón);
        }
        for(Flauta f : flauty) {
              f.zahraj(tón);
        }
        ...
    }
}
```

Dirigenti a orchestrovia

- Problémy tohto návrhu
  - čo keď chceme pridať nový typ nástroja?
    - musíme nadefinovať novú triedu
    - v triede definovať metódu hraj
    - **do orchestra musíme pridať nový zoznam**
  - čo keď chceme pre každý nástroj evidovať meno jeho hráča?
    - v každej z tried musíme definovať inštančnú premennú `menoHráča` a dodať getter a setter.

Dirigenti a orchestrovia

- Riešenie:

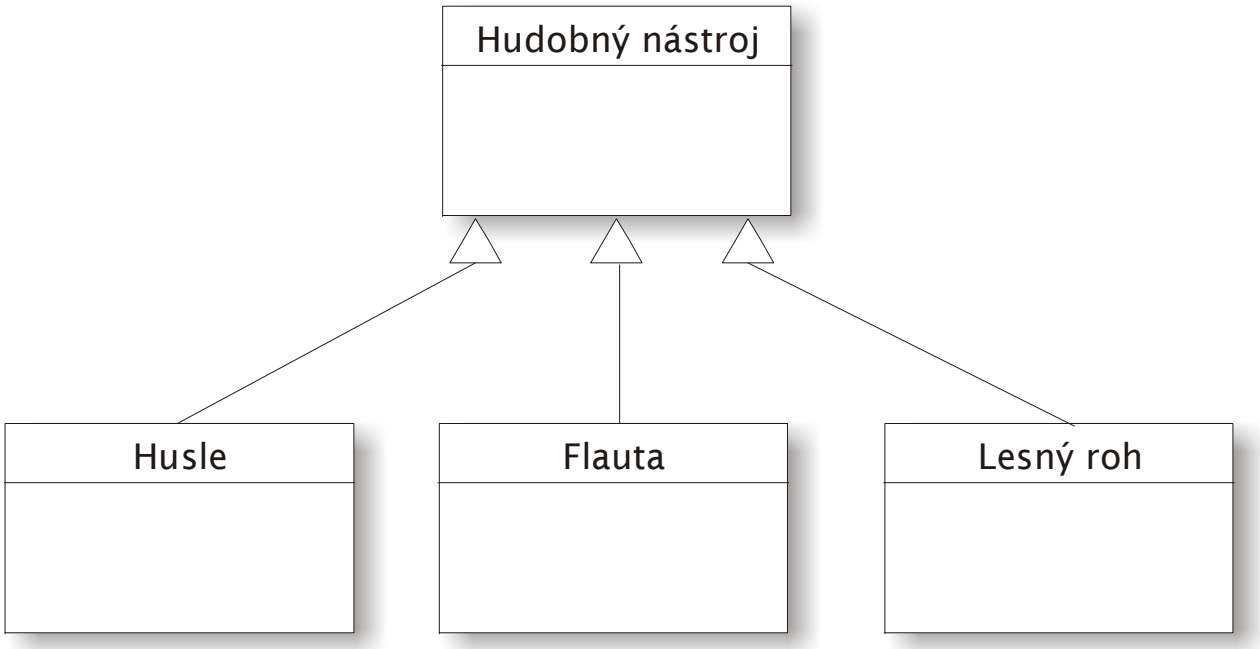
dedičnosť (inheritance)

  - Husle sú **hudobný nástroj**
  - Flauta je **hudobný nástroj**
  - Lesný roh je **hudobný nástroj**
  - Hŕba hudobných nástrojov tvorí orchester.

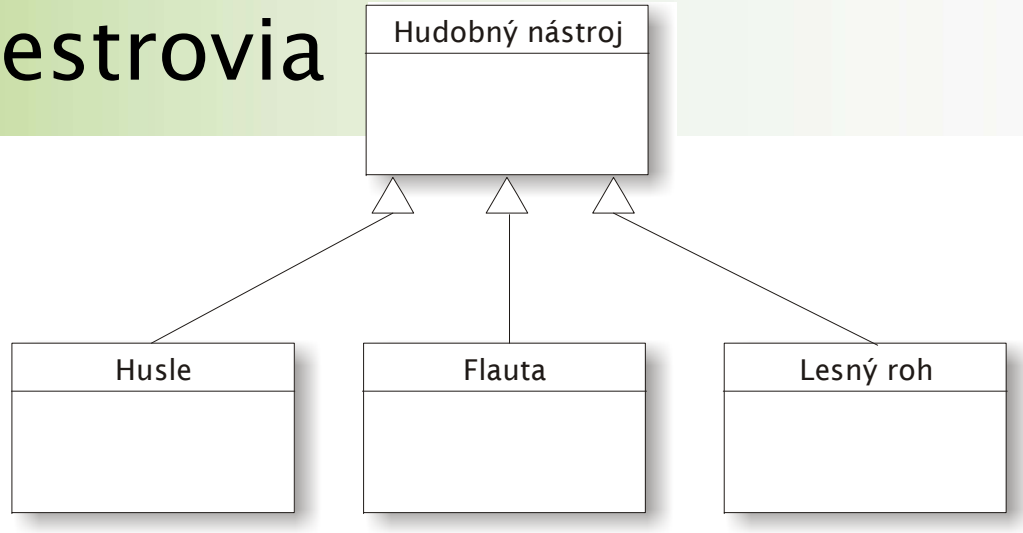
  - Hierarchia *is-a ("je")*
    - každé pleso je jazerom
    - každý lev je mačkovitou šelmou
    - každý druhák je študentom

Dirigenti a orchestrovia

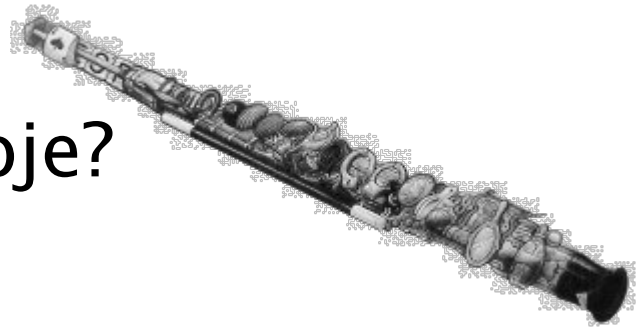
- Zápis v škatuľkovom jazyku UML



Dirigenti a orchestrovia



- keďže `HudobnýNástroj` je trieda, zamyslime sa nad stavom a schopnosťami



- aký spoločný stav majú nástroje?
  - naše nástroje nemali stav
- aké spoločné schopnosti majú nástroje?
  - dokážu hrať

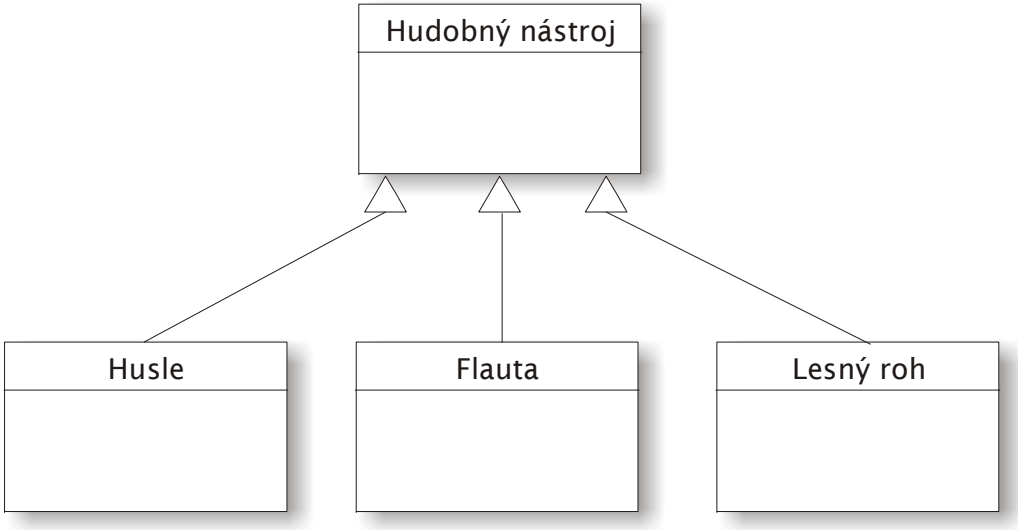
Dirigenti a orchestrovia

- aký spoločný stav majú nástroje?
  - naše nástroje nemali stav
- aké spoločné schopnosti majú nástroje?
  - dokážu hrať

```
public class HudobnýNástroj {
    void zahraj(int tón) {
      //tu sa nič nedeje
    }
}
```

Dirigenti a orchestrovia

- Flauta, husle, tympany... sú hudobné nástroje.
- V reči OOP: flauta
  - je **podtriedou** (*subclass*) triedy `HudobnýNástroj`
  - je **odvodenou triedou** (*derived class*) zo základnej triedy (*base class*) `HudobnýNástroj`.



Dirigenti a orchestrovia

- V reči OOP: flauta
  - je podtriedou (*subclass*) triedy `HudobnýNástroj`

```
public class Flauta extends HudobnýNástroj {
}
```

```
public class Tympany extends HudobnýNástroj {
}
```

Dirigenti a orchestrovia

- Čo znamená, že `Flauta` je `HudobnýNástroj`?
  - môžem vytvárať inštancie flauty

    ,,to sme vedeli aj bez dedičnosti"

    ```
    Flauta flauta = new Flauta();
    ```

  - môžem deklarovať premenné typu HudobnýNástroj

    ```
    HudobnýNástroj nástroj;
    ```

  - na flautu sa môžem dívať ako na hudobný nástroj

    ```
    HudobnýNástroj nástroj = flauta;
    ```

Dirigenti a orchestrovia

- na flautu sa môžem dívať ako na hudobný nástroj

```
HudobnýNástroj nástroj = flauta;
```

HudobnýNástroj

Flauta

- ide to aj naopak?

```
HudobnýNástroj nástroj =
Flauta flauta = nástroj;
```

čo keď mám v hudobnom nástroji uložený lesný roh?

nie

Dirigenti a orchestrovia

- na flautu sa môžem dívať ako na hudobný nástroj, ale nie každý hudobný nástroj musí byť nutne flauta!

```
HudobnýNástroj nástroj = flauta;
```

- do premennej typu `HudobnýNástroj` **viem** nastrkať flauty, trombóny, violy, ...
- do premennej typu `Husle` **neviem** nastrkať ľubovoľný hudobný nástroj

Kedy budú nástroje hrať?

- Potrebujeme, aby nástroje hrali – pretože každý hudobný nástroj musí vedieť hrať.

```
public class HudobnýNástroj {
    void zahraj(int tón) {
    …
```

Flauta má takú istú metódu ako HudobnýNástroj

- Dodáme metódy `zahraj()`

```
public class Flauta extends HudobnýNástroj {
    void zahraj(int tón) {
        System.out.println(
          "Flauta hrá tón " + tón);
    }
}
```

Flauta prekryla metódu `zahraj()`

Polymorfizmus

- Trieda sa môže chovať raz tak, raz onak

```
public class HudobnýNástrojTester {
   public static void main(String[] args) {
        Flauta flauta = new Flauta();
        Husle husle = new Husle();
        HudobnýNástroj nástroj = flauta;
        nástroj.zahraj(440);
        nástroj = husle;
        nástroj.zahraj(440);
   }
}
```

```
Flauta hrá tón 440
Husle hrajú tón 440
```

Polymorfizmus


```
Flauta flauta = new Flauta();
HudobnýNástroj nástroj = flauta;
nástroj.zahraj(440);
```

- Pozrieme sa na **skutočný typ premennej** `nástroj` (`Flauta`)
- Zavoláme metódu na skutočnom type (na `Flaute`)
  - pozrieme sa do triedy Flauta a vykonáme metódu `zahraj()`
- Ak metóda v skutočnom type neexistuje, pozrieme sa do rodiča (`HudobnýNástroj`)

Orchester zas a znova

- nepotrebujeme 3 zoznamy – stačí jeden
- polymorfizmus zabezpečí, že každý nástroj zahrá svojou metódou

```
public class Orchester {
  private ArrayList<HudobnýNástroj> nástroje
     = new ArrayList<HudobnýNástroj>();

  void zahraj(int tón) {
    for(HudobnýNástroj nástroj : nástroje) {
      nástroj.zahraj(tón);
    }
  }
}
```

husle budú hrať
ako husle
flauty ako flauty

Pes-matematik a prekrývanie metód

```
class CirkusovýPes extends Pes {
   void derivuj() {
        System.out.println("derivácia funkcie x je 1");
   }
}
```

```
CirkusovýPes cirkusant = new CirkusovýPes();
cirkusant.štekaj();
```

- pozrieme sa do `CirkusovéhoPsa`
- má uvedenú metódu `štekaj()`?
  - áno – zavoláme ju a šlus
  - nie – pozrieme sa do predka
- má predok (`Pes`) metódu `štekaj()`?
  - áno – zavoláme ju a šlus
  - ak náhodou nie – pozrieme sa do predk

- keďže prapra...predkom je `Object`, určite raz skončíme.
- ak sa metóda nenájde ani v `Object`-e, tak máme syntaktickú chybu

Metódy na nájdenie zdedenej metódy

- pozrieme sa do danej triedy
- má uvedenú danú metódu?
  - áno – zavoláme ju a šlus
  - nie – pozrieme sa do nadradenej triedy a opakujeme algoritmus
- prinajhoršom skončíme v `Object`-e

- typickým príkladom sú metódy `equals()`, `hashCode()` a `toString()`;

Pes-cirkusant a dedičnosť metód

- Metódy môžeme prekrývať (*override*)

```
class Pes {
   String štekaj() {
     return "Haf haf";
   }
}
```

```
class CirkusovýPes extends Pes {
   String štekaj() {
     return "Baf baf baf";
   }
}
```

```
Pes dunčo = new Pes();
dunčo.štekaj();
```

```
CirkusovýPes cirkusant = new CirkusovýPes();
cirkusant.štekaj();
```

```
Pes cirkusant2 = new CirkusovýPes();
cirkusant2.štekaj();
```

```
Haf haf
Baf baf baf
Baf baf baf
```

Ešte zložitejšia dedičnosť metód

- niekedy chceme len rozšíriť správanie z predka
- napr. chceme, aby cirkusový pes štekal ako každý iný pes, ale na koniec štekania vložil reklamu na svoj cirkus.

```
class CirkusovýPes extends Pes {
 String štekaj() {
    String stekanie = super.štekaj();
    return stekanie + " Príďte do nášho Cirkusu™.";
  }
}
```

zavolá sa metóda uvedená v triede `Pes`

super je zvláštna premenná takého typu, aký má predok

v tomto prípade je **super** typu `Pes`

Hudobné nástroje z minulého dielu

```
HudobnýNástroj n = new HudobnýNástroj();
```

- Má význam dovoliť používateľovi vytvárať inštancie hudobných nástrojov?
  - náš úplne všeobecný hudobný nástroj, zovšeobecňujúci flauty, rohy... v skutočnosti nemôže existovať.

Hudobné nástroje jednoducho neexistujú...

Riešenie: ***abstraktné triedy***

- abstraktné triedy reprezentujú *abstraktné* pojmy, či entity

„Divím sa, že sa divím, že sa divím..."

- z abstraktnej triedy nemožno vytvárať inštancie

```
public abstract class HudobnýNástroj {
   ...
}
```

```
HudobnýNástroj n = new HudobnýNástroj();
```

Cannot instantiate type / Nemôžem vytvoriť inštanciu typu

Abstraktné triedy i metódy

- Aj metódy môžu byť abstraktné
- Abstraktná metóda = ,,ja nerobím nič, ale moji dedičia musia túto metódu prekryť (*override*)".

```
public abstract class HudobnýNástroj {
   abstract void zahraj(int tón);
}
```

- každý hudobný nástroj (flauta, trúba...) *musí* vedieť zahrať tón

žiadne kučeravé zátvorky!

len hlavička s bodkočiarkou

Abstraktné triedy i metódy

- Pravidlo: **ak sa v triede abstraktnú metódu, celá trieda musí byť abstraktná.**
- Príklad:
  - biológ: každý vták lieta
  - informatik: trieda `Vták` má abstraktnú metódu `lietaj()`. Trieda `(Európska)Lastovička` lieta tak, že 15krát za sekundu zamáva krídlami.

```
public class Vták {

   abstract void lietaj();

}
```

```
Vták f = new Vták();

f.lietaj();
```

Čo sa podľa vás má stať, vážení diváci?

`abstract class Vták`!

Načo je to dobré?

- abstraktná trieda môže vo svojich metódach robiť mnoho vecí, ale niektoré druhy správania chce nechať na zodpovednosť dedičom
- niektoré druhy správania chce vyžadovať od dedičov
- používané ako doplnok k rozhraniam (interface-om) – o tom neskôr