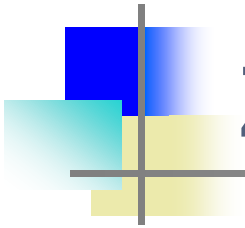
Prednáška 5: Organizácia tried do balíkov, balíky, príslušnosť triedy k balíku, prístupové práva

Ján Lang

kanc. 4.34, lang@fiit.stuba.sk, http://www2.fiit.stuba.sk/~lang/zoop/

Ústav informatiky, informačných systémov a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave

17. október 2023

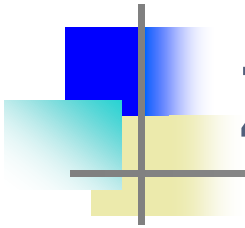


Zadanie 2.

- **Odovzdáva sa:**
  - zdrojový kód vrátane adresárovej štruktúry celého projektu
  - s diagramom tried
  - a súbor pouziteOopPrincipy.pdf do miesta odovzdania v AIS
- **Termín:** 12. november 2023 23:59

Zadanie - vzor

- Pilot riadi lietadlo alebo helikoptéru a zodpovedá za presné, bezpečné a hospodárne vykonanie letu v súlade s platnými predpismi. Interaguje pomerne pravidelne s lekárom/lekármi, ktorí sledujú jeho vybrané charakteristiky. Na palube komunikuje s viacerými a v rôznych jazykoch. Sleduje a riadi let prostredníctvom prvkov dostupných priamo na palube lietadla. Komunikuje s riadiacimi vežami letísk dotknutého vzdušného priestoru. Medzi jeho charakteristiky patrí napr. počet (úspešne) vykonaných letov.



Zadanie - vzor

- Lokalizácia podstatných mien z opisu
- Identifikácia potenciálnych tried, možno atribútov tried, objektov

Zadanie – vzor – Letisko?

- Pilot riadi lietadlo alebo helikoptéru a zodpovedá za presné, bezpečné a hospodárne vykonanie letu v súlade s platnými predpismi. Interaguje pomerne pravidelne s lekárom/lekármi, ktorí sledujú jeho vybrané charakteristiky. Na palube komunikuje s viacerými a v rôznych jazykoch. Sleduje a riadi let prostredníctvom prvkov dostupných priamo na palube lietadla. Komunikuje s riadiacimi vežami letísk dotknutého vzdušného priestoru. Medzi jeho charakteristiky patrí napr. počet (úspešne) vykonaných letov.

Zadanie – vzor – Letisko?

- Pilot riadi lietadlo alebo helikoptéru a zodpovedá za presné, bezpečné a hospodárne vykonanie letu v súlade s platnými predpismi. Interaguje pomerne pravidelne s lekárom/lekármi, ktorí sledujú jeho vybrané charakteristiky. Na palube komunikuje s viacerými a v rôznych jazykoch. Sleduje a riadi let prostredníctvom prvkov dostupných priamo na palube lietadla. Komunikuje s riadiacimi vežami letísk dotknutého vzdušného priestoru. Medzi jeho charakteristiky patrí napr. počet (úspešne) vykonaných letov.

Letisko

- Pilot riadi lietadlo alebo helikoptéru a zodpovedá za presné, bezpečné a hospodárne vykonanie letu v súlade s platnými predpismi. Interaguje pomerne pravidelne s lekárom/lekármi, ktorí sledujú jeho vybrané charakteristiky. Na palube komunikuje s viacerými a v rôznych jazykoch. Sleduje a riadi let prostredníctvom prvkov dostupných priamo na palube lietadla. Komunikuje s riadiacimi vežami letísk dotknutého vzdušného priestoru. Medzi jeho charakteristiky patrí napr. počet (úspešne) vykonaných letov.

- Triedy, objekty resp. atribúty? Abstrahovanie vlastností.

Zadanie – vzor. Triedy

```
class Pilot {
    String meno, priezvisko;
    int vek, pocetLetov;

    Pilot(String meno, String priezvisko, int vek, int pocetLetov) {
        this.meno = meno;
        this.priezvisko = priezvisko;
        this.vek = vek;
        this.pocetLetov = pocetLetov;
    }
```

```
class Letuska {
    String meno, priezvisko;
    int vek, pocetJazykov;

    Letuska(String meno, String priezvisko, int vek, int pocetJazykov) {
        this.meno = meno;
        this.priezvisko = priezvisko;
        this.vek = vek;
        this.pocetJazykov = pocetJazykov;
    }
}
```

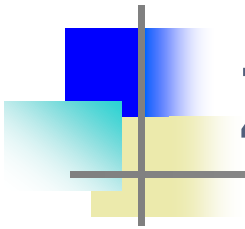
Zadanie – vzor. Triedy

```
class Let {
    Pilot hlavny;
    Letuska prva, druha;

    Let(Pilot hlavny, Letuska prva, Letuska druha) {
        this.hlavny = hlavny;
        this.prva = prva;
        this.druha = druha;
    }

    public static void main(String[] args) {
        Pilot p = new Pilot("Peter", "Rychly", 35, 52);
        Letuska l1 = new Letuska("Alena", "Pekna", 23, 3);
        Letuska l2 = new Letuska("Petra", "Sikovna", 21, 2);
        Let nikam = new Let(p, l1, l2);
        //nikam.vypis();
    }
}
```

- metóda `vypis();`

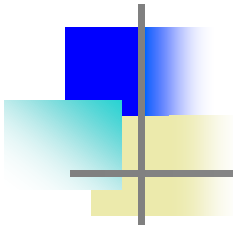
Zadanie – vzor. Triedy

- **package** sk.stuba.fiit.zadanieVzor_3;
- **public class** Let {
- String letID;
- P……
-
- ar.remove(1);
- ...že lietadlo odletelo
-
- ar.get(1).setPorucha(**true**);
- ...pridam do letu Boolean porucha = false;
- ...t.j. lietadlo, ktore je v poruche neodleti
-
- ....pridam do letu aj metodu odlet, kde skontroluje co ma a ak nie je v poruche tak odleti. Zmazem ho zo zoznamu letov na letisku....



Základné OOP pojmy

- Trieda (class) – definícia abstraktného typu dát
- Objekt (object) – inštancia triedy – implementuje stav entity, poskytuje navonok funkcionalitu prostredníctvom metód a ich implementácií, istá forma rozhrania
- Preťažovanie (overloading)
- **Zapuzdrenie, obalenie (encapsulation)**
- Prekonávanie (overriding)
- Dedičnosť (inheritance)
- Polymorfizmus (polymorphism)



„High Cohesion and Low Coupling“

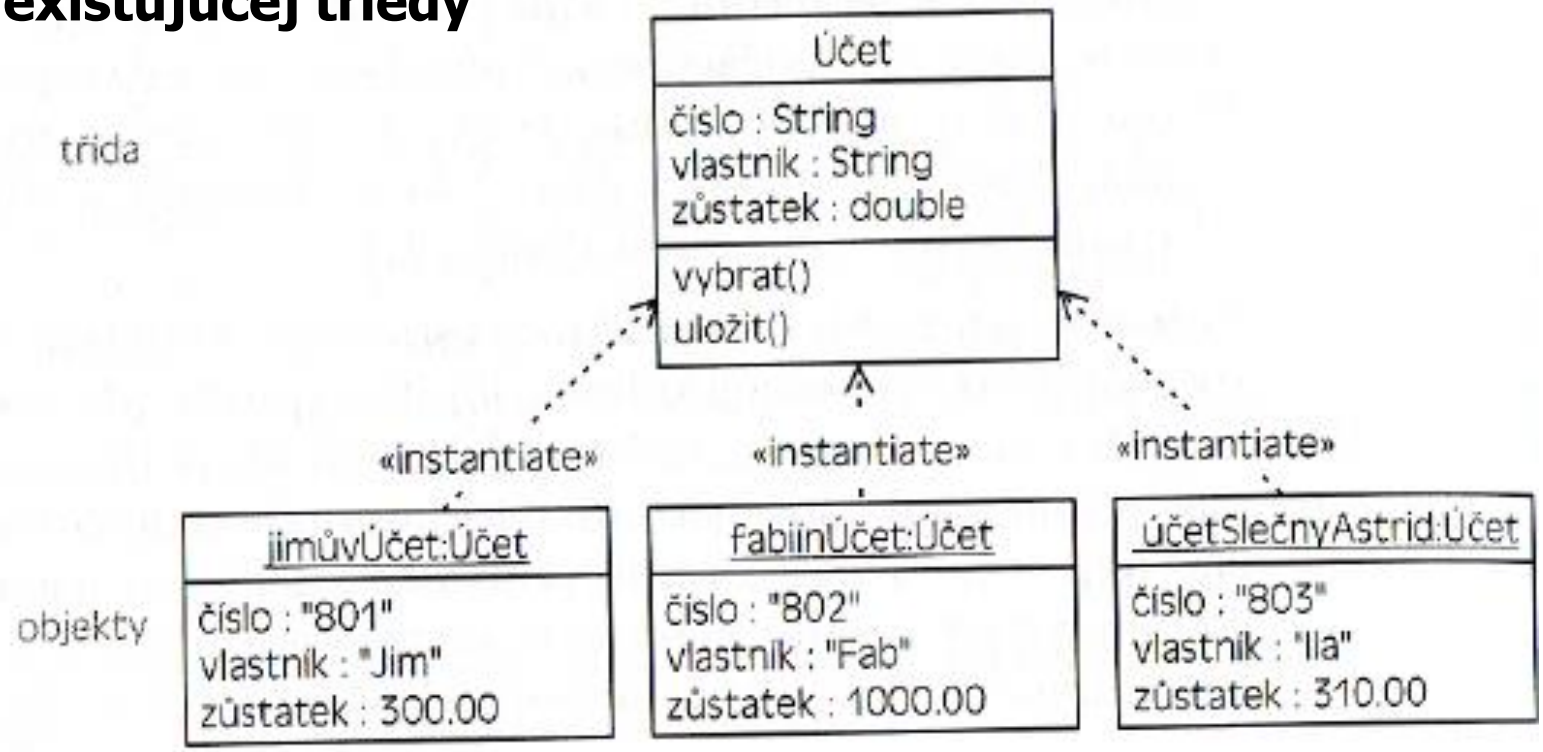
- Súdržnosť – Cohesion
  - Predstavuje vlastnosť sebestačnosti, autonómie.
- Previazanosť - Coupling
  - Predstavuje vlastnosť závislosti, je určená množstvom väzieb medzi softvérovými entitami. Mala by byť čo najmenšia. Minimalizácia množstva väzieb medzi entitami znižuje ich vzájomnú závislosť.
- Snaha organizovať triedy, ktorých objekty navzájom komunikujú do jedného balíka

„High Cohesion and Low Coupling“

```
public class Account {
    private String name;
    private double amount;

    Account(String name, double amount) {
        this.name = name;
        this.amount = amount;
        System.out.println(
                "Message from Acount
     constructor: Account was created for: " + this.name + " on: " + this.amount);
    }

    public boolean isOverdrawn() {
        return this.amount < 0;
    }

    public void addFunds(double amount) {
        this.amount += amount;
    }

    public String getName() {
        return name;
    }

    public double getAmount() {
        return amount;
    }
}
```

Znovu použitie tried

- Princíp znovu použitia programového kódu
- Jedna z najväčších predností objektovo-orientovaných jazykov
- Najjednoduchší spôsob znovu použitia je **v použití objektu existujúcej triedy**



Jim Arlow and Ila Neustadt. *UML~2 and the Unified Process: Practical Object-Oriented Analysis and Design*. Addison-Wesley, 2nd edition, 2005.

Znovu použitie tried

- Použitie existujúceho kódu - vytvorením novej triedy, alebo aplikáciou už existujúcej triedy
- Spôsob znovu použitia bez zásahu do existujúceho kódu (rešpektovanie princípu zapuzdrenia)
- Zatiaľ dva možné spôsoby:
  - Vytvorenie inštancie existujúcej triedy v novej triede – skladanie. Nová trieda je poskladaná z inštancií existujúcej triedy/tried. Jednoduché znovu použitie funkcionality, nie formy
  - Vytvorenie novej triedy ako typu už existujúcej triedy. Doslovné prevzatie formy existujúcej triedy a jej ďalšie možné rozšírenie bez modifikácie existujúcej triedy - dedenie - nabudúce

Znovu použitie tried

- Použitie existujúceho kódu - vytvorením novej triedy, alebo aplikáciou už existujúcej triedy
- Spôsob znovu použitia **bez zásahu do existujúceho kódu** (rešpektovanie princípu zapuzdrenia)
- Zatiaľ dva možné spôsoby:
  - Vytvorenie inštancie existujúcej triedy v novej triede – skladanie. Nová trieda je poskladaná z inštancií existujúcej triedy/tried. Jednoduché znovu použitie funkcionality, nie formy
  - Vytvorenie novej triedy ako typu už existujúcej triedy. Doslovné prevzatie formy existujúcej triedy a jej ďalšie možné rozšírenie bez modifikácie existujúcej triedy - dedenie - nabudúce

Znovu použitie tried

- Použitie existujúceho kódu - vytvorením novej triedy, alebo aplikáciou už existujúcej triedy
- Spôsob znovu použitia **bez zásahu do existujúceho kódu** (rešpektovanie princípu zapuzdrenia)
- Zatiaľ dva možné spôsoby:
  - Vytvorenie inštancie existujúcej triedy v novej triede – **skladanie**. Nová trieda je poskladaná z inštancií existujúcej triedy/tried. Jednoduché znovu použitie funkcionality, nie formy
  - Vytvorenie novej triedy ako typu už existujúcej triedy. Doslovné prevzatie formy existujúcej triedy a jej ďalšie možné rozšírenie bez modifikácie existujúcej triedy - dedenie - nabudúce

Znovu použitie tried

- Použitie existujúceho kódu - vytvorením novej triedy, alebo aplikáciou už existujúcej triedy
- Spôsob znovu použitia **bez zásahu do existujúceho kódu** (rešpektovanie princípu zapuzdrenia)
- Zatiaľ dva možné spôsoby:
  - Vytvorenie inštancie existujúcej triedy v novej triede – **skladanie**. Nová trieda je poskladaná z inštancií existujúcej triedy/tried. Jednoduché znovu použitie funkcionality, nie formy
  - Vytvorenie novej triedy ako typu už existujúcej triedy. Doslovné prevzatie formy existujúcej triedy a jej ďalšie možné rozšírenie bez modifikácie existujúcej triedy - **dedenie** - nabudúce

Vybrané vzťahy medzi triedami

Vzťah pomenovaný:

- *Asociácia* - Association: **uses a - používa**
  trieda Človek používa triedu Auto
- *Agregácia* - Aggregation: **has a - má**
  trieda Človek má triedu Domček (Domček zvyčajne prežije Človeka)
- *Kompozícia* - Composition: **owns a - vlastní**
  trieda Človek vlastní triedu Mozog (Keď Človek zomrie, zomrie aj Mozog)
- *Dedenie* - Inheritance: **is a - je**
  Človek je živá bytosť (Školník je Zamestnanec)

- Príklad class Clovek

Vybrané vzťahy medzi triedami

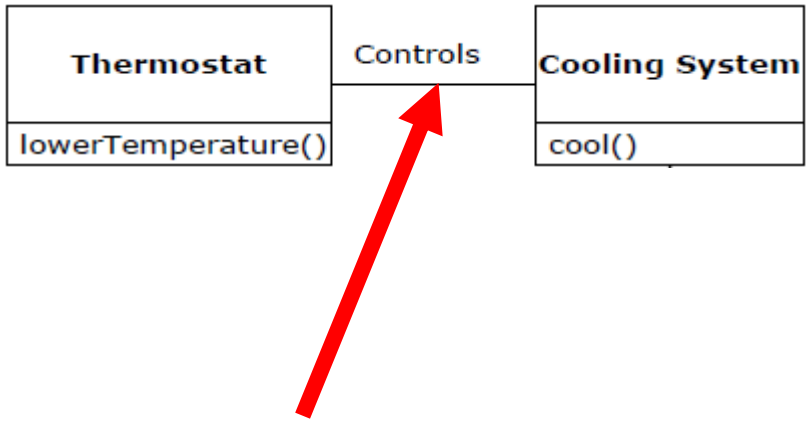
Vzťah pomenovaný:

- ***Asociácia*** - Association: **uses a - používa**
  trieda Človek používa triedu Auto
- *Agregácia* - Aggregation: **has a - má**
  trieda Človek má triedu Domček (Domček zvyčajne prežije Človeka)
- *Kompozícia* - Composition: **owns a - vlastní**
  trieda Človek vlastní triedu Mozog (Keď Človek zomrie, zomrie aj Mozog)
- *Dedenie* - Inheritance: **is a - je**
  Človek je živá bytosť (Školník je Zamestnanec)

- Príklad class Clovek

Vybrané vzťahy medzi triedami

- Asociácia - všeobecný vzťah (TIJ, *Chapter 1: Introduction to Objects*)
- Medzi dvoma triedami prostredníctvom ich objektov



- Všetky objekty majú svoj vlastný životný cyklus. Neexistuje vlastník. Napr. aj viacerí študenti sa môžu spájať s jedným učiteľom a jeden študent sa môže spájať s viacerými učiteľmi, ale medzi nimi neexistuje vlastníctvo a obaja majú svoj vlastný životný cyklus. Môžu sa vytvárať a mazať nezávisle.

Vybrané vzťahy medzi triedami

Vzťah pomenovaný:

- *Asociácia* - Association: **uses a - používa**
  trieda Človek používa triedu Auto
- ***Agregácia*** - Aggregation: **has a - má**
  trieda Človek má triedu Domček (Domček zvyčajne prežije Človeka)
- *Kompozícia* - Composition: **owns a - vlastní**
  trieda Človek vlastní triedu Mozog (Keď Človek zomrie, zomrie aj Mozog)
- *Dedenie* - Inheritance: **is a - je**
  Človek je živá bytosť (Školník je Zamestnanec)

- Príklad class Clovek

Vybrané vzťahy medzi triedami

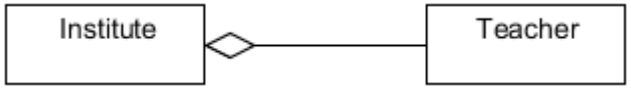
- Agregácia - predstavuje vzťah skladania celku z častí, celok zodpovedá za vytvorenie a zrušenie častí, je to vzťah celku k jednej časti, definujeme násobnosť, značí sa šípkou so symbolom diamantu pri triede, ktorá predstavuje celok.
- Podobne ako asociácia s tým, že tu existuje vlastníctvo. Konkrétny objekt nemôže v tom istom čase patriť inému objektu. Jeden učiteľ nemôže patriť k viacerým ústavom súčasne, ale ak odstránime ústav, objekt učiteľa nebude zničený.

Vybrané vzťahy medzi triedami

Agregácia

- Hodnotou (kompozícia) – plný kosoštvorec. Zodpovednejšia forma agregácie, pri ktorej agregujúci objekt nesie zodpovednosť za existenciu a uloženie agregovaných objektov (composite) - vzťah **vlastní**

- Referenciou (agregácia) – prázdny košoštvorec. Zdielaná agregácia, pri ktorej aj iné objekty môžu agregovať rovnaký objekt (shared) – vzťah **má**

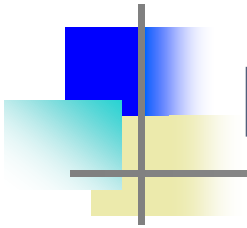


Vybrané vzťahy medzi triedami

Vzťah pomenovaný:

- *Asociácia* - Association: **uses a - používa**
  trieda Človek používa triedu Auto
- *Agregácia* - Aggregation: **has a - má**
  trieda Človek má triedu Domček (Domček zvyčajne prežije Človeka)
- ***Kompozícia*** - Composition: **owns a - vlastní**
  trieda Človek vlastní triedu Mozog (Keď Človek zomrie, zomrie aj Mozog)
- *Dedenie* - Inheritance: **is a - je**
  Človek je živá bytosť (Školník je Zamestnanec)

- Príklad class Clovek



Kompozícia

- Kompozícia je opäť špecializovanou formou agregácie a môžeme ju nazvať vzťahom "smrti". Je to silný typ agregácie.
- Agregovaný objekt nemá svoj životný cyklus a ak sa odstráni objekt, ktorý ho vlastní všetky jeho inštancie budú tiež vymazané.
- Predstavme si dom s miestnosťami. Dom môže obsahovať viac miestností, ale miestnosť nemá samostatný život.
- Žiadna izba nemôže patriť do dvoch rôznych domov. Ak odstránime dom, miestnosť sa automaticky odstráni. Odstrániť miestnosť v dome ešte neznamená odstrániť dom.

Kompozícia

```java
public class Auto {
    //referencia motor nesmie opustiť inštanciu = kompozícia
    private Motor motor;

    public Auto() {
        motor = new Motor();
    }
}
```

```java
public class Motor {
    private String type;
}
```

https://coderanch.com/t/522414/java/Association-Aggregation-Composition

Agregácia

```
public class Auto {
    //referencia motor nesmie opustiť inštanciu = kompozícia
    private Motor motor;

    public Auto() {
        motor = new Motor();
    }
    //referencia motor už môže opustiť inštanciu
    public Motor getMotor() {
        return this.motor;
    }
}
```

```
import java.util.ArrayList;

public class ServiceStation {
    private ArrayList<Motor> usedEngines;

    Motor provideUsedEngine(Auto starySrot) {
        return starySrot.getMotor();
    }
}
```

Vybrané vzťahy medzi triedami

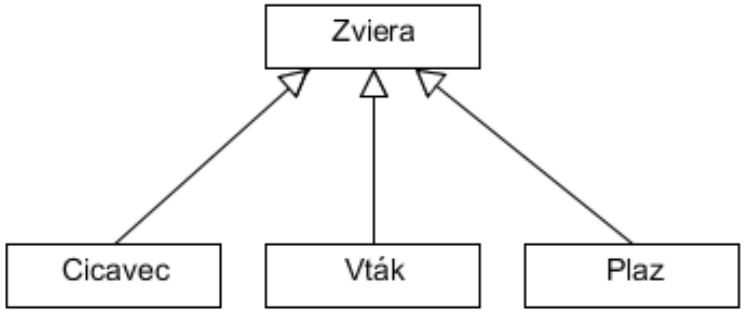
Vzťah pomenovaný:

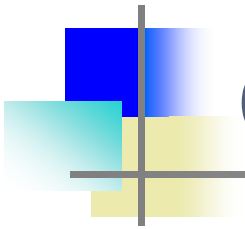
- *Asociácia* - Association: **uses a - používa**
  trieda Človek používa triedu Auto
- *Agregácia* - Aggregation: **has a - má**
  trieda Človek má triedu Domček (Domček zvyčajne prežije Človeka)
- *Kompozícia* - Composition: **owns a - vlastní**
  trieda Človek vlastní triedu Mozog (Keď Človek zomrie, zomrie aj Mozog)
- ***Dedenie*** - Inheritance: **is a - je**
  Človek je živá bytosť (Školník je Zamestnanec)

- Príklad class Clovek

Vybrané vzťahy medzi triedami

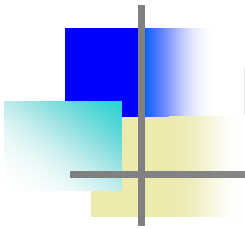
- Generalizácia/špecializácia – dedenie - nabudúce





Časté chyby/otázky/problémy

- Kontext aplikácie atribútov, globálny, lokálny
- Namiesto agregácie považlivý pokus o dedenie
- Namiesto referencie na novú inštanciu referencia už na existujúcu. Z toho vyplývajúce dve referencie na ten istý objekt
- Problém s konštrukciou: Stav inštancie
- Workspace. Problém s lokalizáciou pracovného priestoru
- Volanie metódy inej triedy
- Inštancia triedy ako parameter metódy
- Dve triedy v jednom súbore – iná vážnosť chyby s narastajúcim dátumom v rámci semestra
- Kde delegovať zodpovednosť hlavný program/trieda?
- Návrhové vzory



Úloha

- Vytvorte triedu S. V tejto triede demonštrujte príklad aplikácie
  - statickej konštanty v nestatickom kontext-e
  - statickej premennej v nestatickom kontexte
  - statickej metódy v nestatickom kontexte

- Kontext [1]:
  - the words that are used with a certain word or phrase and that help to explain its meaning [1]
  - the situation in which something happens : the group of conditions that exist where and when something happens [1]

[1] http://www.merriam-webster.com/dictionary/

Riešenie

```
3  //nestatická trieda, nestatický kontext
4  public class S {
5      // klucove slov final a velké písmená
6      private static final int KONSTANTA = 7;
7      private static int atribute = 7;
8
9      private static int metoda() {
10         return 0;
11     }
12
13     // nestatický kontext
14     public void demonstraciaApikacie() {
15         // aplikácia statickej konštanty v nestatickom kontexte
16         System.out.println(this.KONSTANTA);
17         // aplikácia statickej premennej v nestatickom kontexte
18         System.out.println(this.atribute);
19         // aplikácia statickej metódy v nestatickom kontexte
20         System.out.println(this.metoda());
21     }
22
23     // statický kontext
24     public static void main(String[] args) {
25         new S().demonstraciaApikacie();
26     }
27 }
28
```

Problems @ Javadoc Declaration Console

```
<terminated> S [Java Application] C:\Program Files\Java\jre1.8.0_74\bin\javaw.exe (
7
7
0
```

Úloha

- Vytvorte triedu O s atribútmi meno a vek. V hlavnom programe zabezpečte aby program vytvoril inštancie na základe údajov uvedených ako argumenty príkazového riadku (napr.: ak budú argumentmi príkazového riadku nasledovné dáta, tak program vytvorí dve inštancie triedy O: Peter 20 Maria 19). Zabezpečte tiež aby pri vzniku novej inštancie nás systém informoval o aktuálnom počte vytvorených inštancií.
  - Statický atribút, ktorého hodnota bude inkrementovaná v konštruktore

Riešenie

```
3  public class O {
4      public String meno;
5      public int vek;
6      public static int pocet = 0; // pocet instancii
7
8      // konštruktor bez parametrov
9      public O() {
10     }
11
12     // konštruktor s parametrami
13     public O(String meno, int vek) {
14         this.pocet++; // počítanie inštancií
15         this.meno = meno;
16         this.vek = vek;
17     }
18 }
```

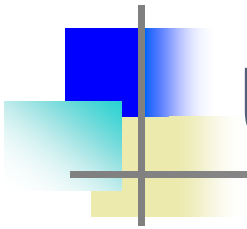
Riešenie

```
3  public class Hlavny {
4      public static void main(String[] args) {
5          // pole objektov
6          O o[] = new O[args.length];
7          for (int i = 0; i < (args.length / 2); i++) {
8              // inicializacia novej instancie
9              o[i] = new O((args[i * 2]), (Integer.parseInt(args[i * 2 + 1])));
10             // vypis:
11             System.out.println(
12                     "Vytvoril som '" + o[i].meno + "' s vekom " + o[i].vek +
13                     " (celkovy pocet: " + O.pocet + ").");
14         }
15     }
16 }
```

Problems @ Javadoc Declaration Console

<terminated> Hlavny (2) [Java Application] C:\Program Files\Java\jre1.8.0_74\bin\javaw.exe (27. 10. 2

```
Vytvoril som 'Jano' s vekom 10 (celkovy pocet: 1).
Vytvoril som 'Peter' s vekom 20 (celkovy pocet: 2).
```



Úloha

- Vytvorte triedu Obyvatel s atribútmi meno, výška, váha.
  V hlavnom programe vytvorte troch rôznych obyvateľov, jedného defaultného a jeden klon ľubovoľného z nich.

- Záležitosť konštruktorov...
- Čo je to defaultny obyvateľ? Ako vzniká?
- Čo je to klon? Ako vzniká? (Klonovací konštruktor)

Riešenie

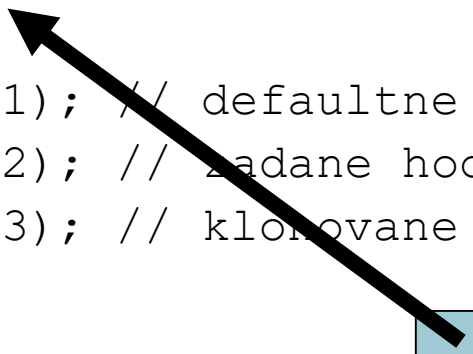
```
public class Obyvatel {
      String meno;
      int vyska, vaha;
      // defaultny konstruktor
      public Obyvatel() {
            meno = "DefaultnyObyvatel";
            vyska = vaha = 0;
      }
      // konstruktor
      public Obyvatel(String meno, int vyska, int vaha) {
            this.meno = meno;
            this.vyska = vyska;
            this.vaha = vaha;
      }
      // klonovaci konstruktor
      public Obyvatel(Obyvatel original) {
            this.meno = original.meno;
            this.vyska = original.vyska;
            this.vaha = original.vaha;
      }
      // prekonaná metóda toString pre vypis informacii
      public String toString() {
         return "Meno: '" + this.meno + "', vyska/vaha: " + this.vyska + "cm/" +
         this.vaha + "kg";
      }
}
```

Default hodnoty

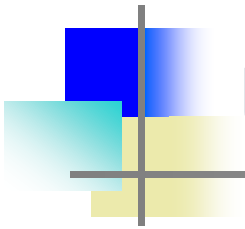
Klonovanie

Riešenie

```
public class Hlavny {
    public static void main(String[] args) {
    Obyvatel o1, o2, o3;

    o1 = new Obyvatel(); // defaultne hodnoty
    o2 = new Obyvatel("Janko Mrkvicka", 175, 75); //
      zadane hodnoty
    o3 = new Obyvatel(o2); // klonovane hodnoty

    System.out.println(o1); // defaultne hodnoty
    System.out.println(o2); // zadane hodnoty
    System.out.println(o3); // klonovane hodnoty
    }
}
```



Klon inštancie referovaný o2



Úloha

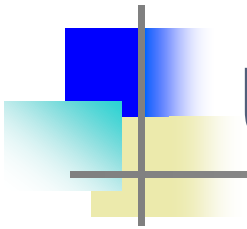
- Vytvorte triedu Ulica, ktorá bude mať atribúty: názov - reťazec max 20 znakov, počet stavebných pozemkov, ktorých bude 100 čo je konečný počet - ten vyjadrite konštantou. Ďalej vytvorte triedu Mesto. Napokon vytvorte triedu Domcek. Zabezpečte aby atribúty triedy Ulica boli viditeľné pre triedu Domcek ale nie pre triedu Mesto. V triede Ulica vytvorte jeden atribút, ktorý nebude viditeľný ani pre triedu Domcek. V tomto príklade uplatnite vzťah agregácie.
- ...diagram v UML. Tiedy Clovek, Poloha, Domcek

Riešenie

```
package sk.stuba.fiit.streda.ulicaA;
public class Ulica {
    protected final String nazov; //atribut viditelny v baliku
    protected static final int POCETSTAVEBNYCHPOZEMKOV = 100;
      //atribut viditelny v baliku
    private int neviditelnyAtribut = 7; //atribut viditelny len
      v ramci instancie triedy

    private Domcek d = new Domcek(); //kompozícia

    public Ulica(String nazov) { // max 20 znakov
        if(nazov.length() > 20) {
            System.out.println("Názov je dlhší ako je povolené");
            Status = "rozpracovana";
        }
        this.nazov = "";
    }
}
```

Riešenie

```
package sk.stuba.fiit.streda.ulicaB;
public class Mesto {
    Mesto () {
    //System.out.println(Ulica.POCETSTAVEBNYCHPOZEMKOV );
      //staticky atribut Ulice neviditelny
    //System.out.println(Ulica.neviditelnyAtribut); //
      neviditelnyAtribut Ulice neviditelny
    }
}

package sk.stuba.fiit.streda.ulicaA;
public class Domcek {

    Domcek () {
    System.out.println(Ulica.POCETSTAVEBNYCHPOZEMKOV);
      //staticky atribut Ulice viditelny
    //System.out.println(Ulica.neviditelnyAtribut); //
      neviditelnyAtribut Ulice neviditelny
    }
}
```



Úloha

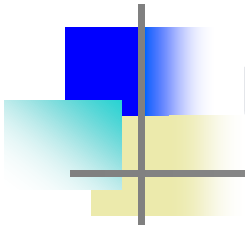
1. Vytvorte triedu Clovek s atribútmi meno a rokNarodenia a metódou vypisInformacie(Clovek c), ktorá vypíše stav konkrétnej inštancie triedy. V hlavnom programe vytvorte troch Clovek-ov s zavolajte metódu vypisInformacie(Clovek c).

Riešenie

```
public class Clovek {
    private String meno;
    private int rokNarodenia;

    Clovek (String meno, int rokNarodenia) {
    this.meno = meno;
    this.rokNarodenia = rokNarodenia;
    }

    public void vypisInformacie(Clovek c) {
    System.out.println("Vznikol clovek s menom: '" +
      c.meno + "' a rokom narodenia " + c.rokNarodenia +
      ".");
    }
}
```

Riešenie

```
public class Hlavny {
    public static void main(String[] args) {
        Clovek c1, c2, c3;

        c1 = new Clovek("Jano", 2000);
        c2 = new Clovek("Fero", 2001);
        c3 = new Clovek("Miso", 2013);

        c1.vypisInformacie(c1);
        c2.vypisInformacie(c2);
        c3.vypisInformacie(c3);
    }
}
```

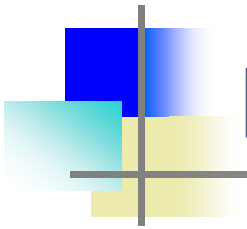


Úloha

- Vytvorte triedu H ktorá bude mať dva ľubovoľné atribúty. V cykle vytvorte 10 inštancií tejto triedy. Zabezpečte aby pri vzniku novej inštancie nás systém informoval o aktuálnom počte vytvorených inštancií. Nositeľom tejto informácie nemá byť počítadlo iterácii (cyklov).
- Statický atribút inkrementovaný v konštruktore
- O výpis sa stará konštruktor

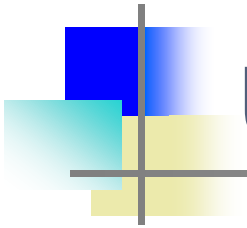
Riešenie

```
public class H {
    private String name;
    int age;
    private static int numberOfInst = 0;

    H() {
        this.numberOfInst++;
        System.out.println(this.numberOfInst);
    }
}
```



Riešenie

```
// prilis jednoduchy priklad a zlé riešenie
for (int i = 0; i < 10; i++)  {
    new H();
}

// správne riešenie
H h[] = new H[10];
for (int i = 0; i < 10; i++)  {
    h[i] = new H();
    h[i].age = i;
}
System.out.println(h[1].age);
System.out.println(h[5].age);
```

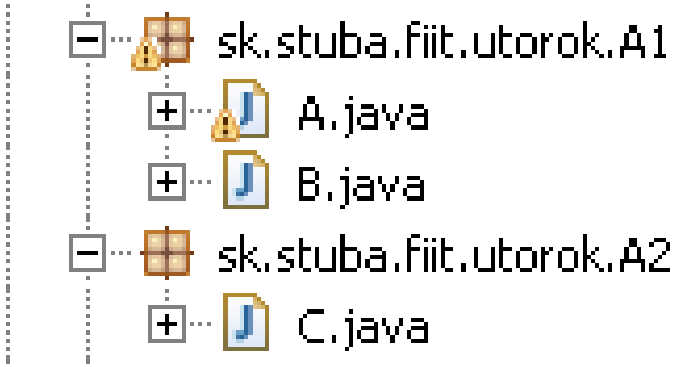


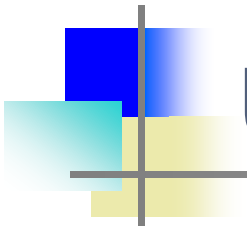
Úloha

- Vytvorte triedu A takú aby jej tri atribúty boli viditeľné pre triedu B ale nie pre triedu C. V triede A vytvorte jeden parameter ktorý nebude viditeľný ani pre triedu B. V triede A aplikujte agregáciu.

Riešenie

```
public class C {

}

public class B {

}

public class A {
    protected String s;
    protected int i;
    private B b = new B();

}
```



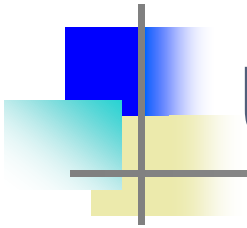


Úloha

- Vytvorte triedu F s jediným atribútom farba. V hlavnom programe zabezpečte aby program vytvoril inštancie na základe farieb uvedených ako argumenty príkazového riadku.

Riešenie

```
public class F {
    public String farba;

    F(String farba) {
        this.farba = farba;
    }
}
public static void main(String[] args) {
    int i = 0;
    F pole[] = new F[args.length];
    for (String s : args){
        pole[i] = new F(s);
        System.out.println(pole[i].farba);
        i++;
    }
}
```



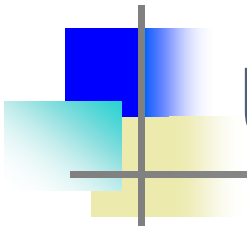
Úloha

- Vytvorte triedu T, ktorá bude mať štyri atribúty (dĺžka, šírka, výška a meno). V tejto triede vytvorte tri konštruktory a demonštrujte vlastnosť preťaženia konštruktorov. Túto skutočnosť vysvetlite priamo v kóde prostredníctvom komentárov.

Riešenie

```
public static void main(String[] args) {
    System.out.print(new T().dlzka);
}
```

```
public class T {
    int dlzka, sirka, vyska;
    String meno;

    // konstruktor s parametrami
    T(int d, int s, int v, String m) {
        this.dlzka = d;
        this.sirka = s;
        this.vyska = v;
        this.meno = m;
    }

    // bezparametrický konstruktor - nastaví atributy instancie na
       defaultne hodnoty
    // zavola originalny konstruktor s defaultnymi hodnotami
    T(){
        this(100, 0, 0, "");
    }

    // Konstruktor s parametrom - instaciou tej istej triedy. Opäť
    T(T t){
        this(t.dlzka, t.sirka, t.sirka, t.meno);
    }
}
```

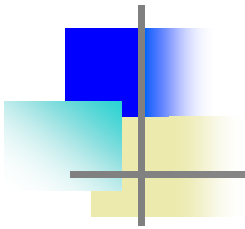


Úloha

- Vytvorte triedu E. V tejto triede deklarujte referenciu statickej konštanty. Tiež jednu statickú metódu, ktorá bude vracať reálne číslo. V hlavnom programe uplatnite možné prístupy k statickému atribútu a statickej metóde.

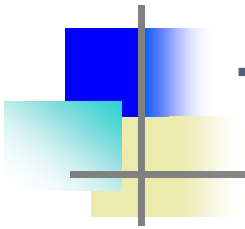
Riešenie

```
public class E {
    static final int POCET = 1; //referencia statickej
      konštanty s inicializáciou
    static double metoda(){ //staticka metoda
    return 0.7;
    }
}
```



Riešenie

```
public static void main(String[] args) {

    System.out.println(E.POCET);
    System.out.println(E.metoda());

    E e = new E();
    System.out.println(e.POCET);
    System.out.println(e.metoda());
}
```



TODO

- Aj vy môžete pomôcť vylepšiť tento predmet študentom pre nasledujúci akademický rok. Vaše odporúčanie, komentár či otázka.