Fakulta informatiky a informačných technológií,

Slovenská technická univerzita, Ilkovičova 2, Bratislava, 842 16

„Zadanie 3 –

Lesná farma – záverečné odovzdanie“

Základy objektovo orientovaného programovania

Cvičenia: Streda 10:00-11:50

Cvičiaci: Ing. Michal Hucko

autor:
Riccardo Kiss
ročník štúdia: **druhý**

Bratislava
2019

Zadanie 3 – Záverečné odovzdanie

- Funkčnosť

Projekt je spustiteľný klasicky z Main.java. Užívateľ zadáva na vstup čísla na prepínanie medzi menu. Po nájdení menu na pridanie drevorubača alebo správcu je vstup v nasledovnom tvare [krstné meno(**string**), priezvisko(**string**), pohlavie(boolean – **true** = muž / **false** = žena), vek(**int**)], medzi každým prvkom nechajte jednu medzeru.

Vzorové príklady vstupu pre *hireWoodcutter()* alebo *hireForestManager():*

Riccardo Kiss true 20 ***[ENTER]***

Eva Novakova false 34 ***[ENTER]***

Michal Hucko true 23 ***[ENTER]***

Andrea Slovakova false 19 ***[ENTER]***

Vzorové príklady vstupu pre *addDeciduousTree()* alebo *addConiferousTree():*

oak brown 10 14 ***[ENTER]***

birch white 12 15 ***[ENTER]***

spruce brown 11 16 ***[ENTER]***

sequoia brown 120 100 ***[ENTER]***

- Zmeny

Drevorubačovi (*Woodcutter.java*) som pridal „výplatnú pásku“ (*Paycheck.java*), t.j. za každý zoťatý strom dostane *n* peňazí. Pridaný balík s dvoma triedami pre ukážku upcasting-u a downcasting-u.

- Balíky

1. **treeFarm.main** – hlavné triedy, ktoré sú zodpovedné za chod programu
   1.1. Main.java
   1.2. Menu.java
2. **treeFarm.people** – triedy ľudí
   2.1. Worker.java
   2.2. ForestManager.java
   2.3. Woodcutter.java
   2.4. ListForestManager.java
   2.5. ListWoodcutter.java
3. **treeFarm.tree** – triedy stromov
   3.1. Tree.java
   3.2. Deciduous.java
   3.3. Coniferous.java
   3.4. ListDeciduous.java
   3.5. ListConiferous.java
4. **treeFarm.casting** – pomocné triedy pre ukážku upcastingu a downcastingu

4.1. WorkerCasting.java
4.2. WoodcutterCasting.java

5. *treeFarm_diagram.ucls* – diagram tried
6. *treeFarm_UMLdiagram*.uxf – diagram tried pomocou UMLet-u

- Použité OOP princípy

1. Trieda *(class)*

   V projekte mám aktuálne vytvorených 13 tried. Napr.

   *public class **Worker()** {...}*

2. Objekt *(object)*

   Každá trieda ma svoje vlastné inštancie, okrem Menu.java, ktorá je statická a nevytváram nikde taký objekt. Napr.

   *ListForestManager **fmList** = new ListForestManager(...)*

3. Zapuzdrenie *(encapsulation)*

   V rodičovských triedach (Worker.java, Tree.java) používam modifikátory atribútov *„protected“*, inak v ostatných triedach *„private“* a pristupujem k nim pomocou **get** a **set** metód, ktoré sú *„public“*. Napr.

   *private int **money**;*
   *public int **getMoney()** {return **money**;}*

4. Preťažovanie *(overloading)*

   Mám jednu preťaženú metódu *menuBack()* v Menu.java a to nasledovne:

   *private static void **menuBack(**String currentMenu**)** {...}*
   *private static void **menuBack(**String currentMenu, int key**)** {...}*

5. Prekonávanie *(overriding)*

   V triede Weather (*Weather.java*) mám metódu **toString()**, ktorá prekonáva metódu **toString()** z **java.lang.Object.toString();** .

   public String toString() { return "\nCurrent weather is " + getCurrentWeather();}

6. Dedičnosť *(inheritance)*

   V projekte som použil dedenie 4-krát: Triedy ForestManager (*ForestManager.java*) a Woodcutter (*Woodcutter.java*) dedia atribúty a metódy z triedy Worker (*Worker.java*), a triedy Coniferous (*Coniferous.java*) a Deciduous (*Deciduous.java*) dedia z triedy Tree (*Tree.java*). Napr.

   *public class Woodcutter **extends** Worker {...}*
   *public class Coniferous **extends** Tree {...}*

7. Agregácia *(aggregation)*

   Referenciu na objekt triedy Paycheck (*Paycheck.java*) v triede Woodcutter (*Woodcutter.java*) si vieme vrátiť pomocou metódy **get**. Napr.

   *public Paycheck **getMoney()** { return this.paycheck;}*

8. Asociácia *(association)*

   Trieda ListConiferous (*ListConferous.java*) používa pole objektov triedy Coniferous (*Coniferous.java*). Napr.

   *private static ArrayList<**Coniferous**> coniferousList = new ArrayList<**Coniferous**>();*

9. Kompozícia *(composition)*

   Trieda Woodcutter *(Woodcutter.java)* obsahuje objekt triedy Paycheck *(Paycheck.java)* a v konštruktore drevorubača sa vytvorí aj inštancia objektu triedy Paycheck pri každom novom drevorubačovi. Ak zanikne drevorubač, zanikne aj jeho výplata. Napr.

```
public class Woodcutter {
        private Paycheck paycheck;
        public Woodcutter(...) {
                super(...);
                this.paycheck = new Paycheck(0);
        }
}
```

10. Abstraktná trieda *(abstract class)*

    Trieda Worker (*Worker.java*) je abstraktnou triedou, lebo nikde v projekte sa nevytvára jejnová inštancia **new Worker(...)**.

```
public abstract class Worker {
    ...
}
```

11. Abstraktná metóda (*abstract method*)

    V triede Worker (*Worker.java*) sa nachádza napríklad abstraktná metóda **getFirstName()**, ktorá nemá definované telo a tým pádom ju trieda podtypu automaticky prekonáva (overriding).

```
public abstract String getFirstName();
```

12. Upcasting

    V triede Main (*Main.java*) mám uplatnený upcasting spôsobom, že referenciu triedy **WoodcutterCasting** si ukladám do inštancie triedy **WorkerCasting**, ktorá je jej nadtypom.

```
WorkerCasting testWoodcutter = new WoodcutterCasting();
```

13. Downcasting

    V triede Main (*Main.java*) mám uplatnený downcasting , keď sa snažím volať metódu z triedy WoodcutterCasting (*WoodcutterCasting.java*), ktorá bola v predchádzajúcom kroku upcastnutá do inštancie jej nadtypu.

```
((WoodcutterCasting)testWoodcutter).casting();
```

14. Rozhranie (*interface*)

    Rozhranie MyInfo (*MyInfo.*java) mám implementované v triedach Woodcutter (*Woodcutter.java*) a ForestManager (*ForestManager.java*) v hlavičke triedy pomocou kľúčového slova **implements**. Každá z tried, ktoré implementujú zadefinované metódy z rozhrania ich tak môžu prekonávať (*overriding*) podľa vlastnej potreby.

```
public interface MyInfo {
        public String myFirstName();
        ...
}

public class Woodcutter extends Worker implements MyInfo {
        ...
        public String myFirstName() { return getFirstName(); }
        ...
```

}

- Veci, na ktoré som pyšný

Spôsob akým som si vymyslel načítavanie a prepínanie menu. Každé menu má svoj vlastný „kódový stav“ (viď. ***Menu.java*** blokový komentár na začiatku). Napr. hlavné menu ma označenie „0“ a po výbere nejakej voľby „2“ (prepnutie do menu drevorubačov) sa k premennej stavu aktuálneho menu (typu String) pripíše zvolená voľba (t.j. „0“ + „2“ = „02“, menu s označením „02“ je už menu drevorubačov) a následne sa zmení menu podľa aktuálneho stavu tejto premennej.

- UML diagram