Математички факултет, Универзитет у Београду

Катедра за рачунарство и информатику

Објектно оријентисано програмирање

вежбе

школска 2016/2017

Биљана Стојановић

Немања Мићовић

Никола Милев

Час 6: Апстрактне класе. Универзална суперкласа Object. Сва значења кључне речи final. [1]

Апстрактне класе

У примеру са површима, на ком смо учили полиморфизам, полиморфни метод `povrsina()` је у базној класи `Povrs` имао празно тело, из простог разлога што је присуство тог метода у базној класи неопходно, јер то захтева полиморфизам, а не знамо како бисмо га имплементирали јер не постоји универзални начин за рачунање површине површи када не знамо која површ је у питању.

То је чест случај у објектно оријентисаном програмирању: креира се суперкласа из које се изводе потклассе само да би се користиле предности полиморфизма, при чему полиморфни метод нема значење у контексту базне класе, те његово имплементирање у њој нема смисла.

Такав метод могуће је прогласити апстрактним. То се чини навођењем кључне речи `abstract` испред повратног типа метода и тачка-запете (;) уместо тела метода:

```
public abstract double povrsina();
```

`public abstract` ⇔ `abstract public` (битно је да буде испред повратног типа)

Чим класа има бар један апстрактни метод и сама постаје апстрактна, те се и у првом реду њене дефиниције, испред речи `class`, мора навести иста кључна реч, `abstract`.

Није могуће креирати конкретан објекат типа апстрактне класе, али могуће је декларисати променљиву типа апстрактне класе. Та променљива се, даље, може користити на исти начин као у примеру за полиморфизам.

Апстрактни метод не може бити `private`, пошто се `private` метод не може наследити, а тиме ни предефинисати у изведеној класи.

Приликом извођења потклассе из апстрактне базне класе, уколико се не предефинише неки од апстрактних метода базне класе и сама потклассе биће апстрактна, јер поседује апстрактни метод наслеђен од базне класе. У том случају, нећемо моћи да креирамо ни објекте те изведене апстрактне класе.

Ако је класа апстрактна, морамо користити кључну реч `abstract` када је дефинишемо, чак и када она само наслеђује апстрактни метод своје суперкласе. Пре или касније, морамо имати потклассу која има имплементацију свих метода које је наследила од своје (апстрактне) базне класе. Онда можемо креирати објекте те класе.

Једине измене у примеру са површима које је неопходно извршити како би базна класа била апстрактна су:

```
public abstract class Povrs{

      public abstract double povrsina();
      // sve ostalo isto
}
```

Сва значења кључне речи `final`

Кључна реч `final` има различита значења у зависности од тога на шта је примењена.

`final` променљива (атрибут, параметар метода, локална променљива)
вредност те променљиве не може да се мења (константна је)

`final` метод
није га могуће предефинисати у изведеној класи
Апстрактни метод не може бити дефинисан као `final` јер мора бити дефинисан у некој потклассе

`final` класа
није могуће извести потклассу из ње
Апстрактна класа не може бити дефинисана као `final` пошто ће то спречити да апстрактни методи класе икада буду дефинисани
Декларисање класе као `final` је драстичан корак, који спречава да функционалност класе буде проширена извођењем, па треба да будемо потпуно сигурни да је то оно што желимо.

1 Према материјалима Марије Милановић

`Object` – универзална суперкласа

Уколико се експлицитно не наведе да је класа изведена из неке друге, подразумева се да је изведена из класе `java.lang.Object`, тако да је `Object` директна или индиректна суперкласа сваке друге класе.

Последице:

- променљива типа `Object` може да чува референцу на објекат произвољног класног типа, што је корисно када желимо да пишемо метод који рукује објектима непознатог типа. Можемо да дефинишемо параметар метода типа `Object`, па се референца на објекат произвољног типа може проследити методу.
- наше класе наслеђују чланове класе `Object`. То су методи (7 `public` и 2 `protected`):
  - `public: toString(), equals(), hashCode(), getClass(), notify(), notifyAll(), wait()`
  - `protected: clone(), finalize()`

  О методима `toString()` и `getClass()` је већ било речи.

  Методи `getClass()`, `notify()`, `notifyAll()` и `wait()` су дефинисани као `final`, те се не могу предефинисати у изведеним класама.

  Метод `equals()` пореди референцу на текући објекат и референцу прослеђену као аргумент и враћа `true` акко су те две референце једнаке, тј. обе реферишу на исти објекат у меморији. Иначе враћа `false` (сматра се да су два објекта различита чак и када имају исте вредности одговарајућих атрибута)

  Метод `hashCode()` предефинише се у изведеним класама онда када се објекти тих класа користе као кључеви у хеш-табелама. Више речи о томе биће у наставку курса.

  Методи `notify()`, `notifyAll()` и `wait()` служе за синхронизацију нити. Више речи о нитима и њиховој синхронизацији биће у наставку курса.

  Метод `clone()` креира објекат који је копија текућег објекта, без обзира на његов тип.

  Метод `finalize()` позива се приликом уништавања објекта. Може се предефинисати како би се додао сопствени кôд за ослобађање ресурса које је објекат користио.