COSC 102 **Spring 2015**

Software Company — Lab 5

In this lab you will work on the following OOP concepts:

- inheritance
- polymorphism
- abstract classes

Pre-Lab

You have read from our textbook about inheritance and polymorphism in Java. We have covered in class *abstract classes*. Before the lab I want you to read a review about those concepts and to complete a small exercise to make sure you understand the fundamental principles.

- Read the following tutorial until Section 6.
- Download, open and read through `ABCDemo.java`.
- Complete the exercise which is predict the output before to execute `ABCDemo.java`, uncommenting a line at a time. Ask us in the lab if anything that you didn't understand. Feel free to experiment by adding more lines of code.
- Finish to read the tutorial: Section 6 is about abstract classes, which you use in the lab.

Design Overview

During this week lab you design, implement and test classes that concern a software development company, as they aim to model **employees** and **software projects**. The company has the following types of employees:

1. programmers, and
2. testers,

represented by the following Java class hierarchy:

```
         Employee
         /      \
Programmer     Tester
```

Specifically the overall code structure and each class responsabilities is as follow.

- `Employee` is an `abstract class` that represents the notion of an employee.
- The `class Employee` has an abstract method called `work()`, that the classes `Programmer` and `Tester` will provide an implementation for.
- A `Programmer` writes code.
- A `Tester` tests code.

We provide the following starter file: Employee.java

Class Employee

The `Employee` class contains functionality common to types of employees in the company. For example, all employees have a name and ID number. **Note that the provided class is missing a constructor**.

Your first task can be to add a constructor. How can you test it? Do you need to do something else to check those few lines of code?

Class Programmer

The `Programmer` class is a sub-class of `Employee`.

A given programmer can write a certain number of lines of code per day, `linesOfCodePerDay`.

A programmer's `work` method returns the number of lines of code s/he wrote on a given day, which is a random number between 50% and 150% of `linesOfCodePerDay` (some days are not as good as others, and some tasks are harder/easier than others).

Implement the `Programmer` class, one part at a time. Add a `main` method to this class to test each part as you develop them. For example, make sure that

- a `Programmer` object can be created and
- the integer returned by the `work` method is correct with respect to the `linesOfCodePerDay` value.

Class Tester

The `Tester` class is a sub-class of `Employee`.

A given tester can test a given number of lines of code per day, `linesOfCodeTestedPerDay`.

A tester's `work` method returns the number of lines of code s/he tested that day, which is a random number between 75% and 125% of `linesOfCodeTestedPerDay`.

Implement the `Tester` class and similarly check that it is correct before continuing with the next class.

Class Project

The class `Project` models a software development project. Each project has **among other things** at least the following attributes:

- `linesOfCode`: an estimate of the number of lines of code the project will require (that's never known ahead of time, but let's just assume that)
- `employees`: an array that stores the employees (programmers and testers) associated with the project
- `linesOfCodeWritten`: the number of lines of code that have been written by the programmers working on the project
- `linesOfCodeTested`: the number of lines of code that have been tested so far by the tester working on the project
- `duration`: how many days managers have given for completion of the project

The project is complete once all the code is written and tested (it's fine if during the simulation the number of lines tested is greater than the number of lines that have been written yet).

The `Project` class should have at least the following constructor(s) and methods.

- constructor(s): set the number of lines of code for the project, the maximum number of employees and the required duration.
- `boolean addEmployee`: adds an employee to the project if possible; returns if the given `Employee` could be added to `employees`.
- `boolean runProject`: each day, as long as the project is not complete or the project duration is not expired, each employee's `work()` method is called. Write sufficient and relevant output to watch the simulation unfolding: each day output the current progress. `runProject` returns if the project could be completed within the allotted number of days, print that final information.

The `main` method of this class should create a new instance of `Project`, add a few programmers and testers, call the `runProject()` method.

Submit

Show the lab instructor the work you completed before leaving the lab.

Submit on Moodle by the deadline a `zip` file called `lab04` containing the four classes you implemented and tested with three `main` methods included.

Credit: Lab adapted from Colorado State University