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

- inheritance
- polymorphism
- abstract classes

Warm-up

You have read from our textbook about inheritance and polymorphism in Java and we covered in class abstract classes. For today you may have read the tutorial listed below. As you finish the reading complete the small exercise to confirm your understanding of the fundamental principles.

- Read the following tutorial until Section 6.
- Download, open and read through ABCDemo.java.
- Complete the exercise which is to predict the output before to execute ABCDemo.java, uncommenting a line at a time. Ask us about anything you are not sure about. Feel free to experiment by adding more lines of code.
- Finish to read the tutorial: Section 6 is about abstract classes, which you use next.

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:

```java
Employee
 /    \
Programmer   Tester
```

Specifically the overall code structure and each class responsibilities 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 has a range \([\text{low}-\text{high}]\) of lines of code s/he can write per day. Each work day s/he will write a random number in between that specified range (you know some days are not as good as others, and some tasks are harder/easier than others).

A programmer’s work method returns the number of lines of code s/he wrote on a given day.

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 specified range.

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, \(\text{linesOfCodeTestedPerDay}\).

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 \(\text{linesOfCodeTestedPerDay}\).

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 the specified number of line of code has been written and tested. It is fine if during the simulation the number of lines tested is greater than the number of lines that have been written.
yet. However `linesOfCodeWritten` and `linesOfCodeTested` need to above a project `linesOfCode` for the project to be completed.

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. The method should return once the number of line of code has been achieved.

To guide you there is the output of a `Project` simulation:

Employee: [Programmer] Aida (ID: 102) line range: 40-500 was added.
Employee: [Tester] Mike (ID: 103) avg lines: 400 was added.
Employee: [Tester] Elodie (ID: 104) avg lines: 300 was added.
Employee: [Programmer] Joel (ID: 105) line range: 105-1110 was added.
Employee: [Programmer] Bill (ID: 106) line range: 1500-5550 was NOT added.

*** DAY 1 ***
[Programmer] Aida (ID: 102) line range: 40-500, Completed 219 lines
[Tester] Mike (ID: 103) avg lines: 400, Completed 325 lines
[Tester] Elodie (ID: 104) avg lines: 300, Completed 251 lines
[Programmer] Joel (ID: 105) line range: 105-1110, Completed 1048 lines
Day 1/10: lines written = 1267, tested = 576 (out of 2000)

*** DAY 2 ***
[Programmer] Aida (ID: 102) line range: 40-500, Completed 478 lines
[Tester] Mike (ID: 103) avg lines: 400, Completed 467 lines
[Tester] Elodie (ID: 104) avg lines: 300, Completed 316 lines
[Programmer] Joel (ID: 105) line range: 105-1110, Completed 951 lines
Day 2/10: lines written = 2696, tested = 1359 (out of 2000)

*** DAY 3 ***
[Programmer] Aida (ID: 102) line range: 40-500, Completed 130 lines
[Tester] Mike (ID: 103) avg lines: 400, Completed 451 lines
[Tester] Elodie (ID: 104) avg lines: 300, Completed 342 lines
[Programmer] Joel (ID: 105) line range: 105-1110, Completed 939 lines
Day 3/10: lines written = 3765, tested = 2152 (out of 2000)

After 3 days, the project was completed.
Submit

Show the lab instructor the work you completed before leaving the lab.
Submit on Moodle by the deadline a zip file called `lab05` containing the four classes you implemented and tested with three `main` methods included.

Credit: Lab adapted from Colorado State University