Laboratory Sheet 7

This Lab Sheet contains material based on Lectures 1 – 14 (up to *5 November 2014*), and contains the submission information for Laboratory 7 (week 8, *10 – 14 November 2014*).

The deadline for submission of the lab exercise is 48 hours after the end of your scheduled laboratory session in week 8.

Aims and objectives

- Understanding of, and practice with, inheritance, polymorphism and abstract classes
- Continuing to apply principles of object-oriented design
- Simple use of packages and import statements

Set up

When you download Laboratory7.zip from moodle, please unzip this file. You will obtain a folder Laboratory7, containing a subfolder entitled Submission7_1. Remember that for this Laboratory you will have to switch your Eclipse workspace to the Laboratory7 folder.

In the folder Submission7_1 will be the following files:

- workers/Waged.java which defines an interface
- UniversityPayRoll.java which defines a `main` method

In Eclipse, you should create a new project entitled Submission7_1; the given files will automatically become part of this project.

You will need to create new Java classes in the workers package for your solution to this exercise. These are:

- Employee
- SalariedEmployee
- HourlyEmployee
- PermanentEmployee
- TemporaryEmployee

Submission material

You are to design classes to model the employees of a company. The first task is to gain an understanding of the problem description, by drawing a UML class diagram that illustrates the hierarchical relationships between your proposed classes. This drawing must be submitted, either as a JPEG or a PDF file. (You can hand-draw the class diagram and take a photo, or produce it in MSWord etc and save a PDF.)

Every employee is either a *salaried* employee or an *hourly paid* employee, and every hourly paid employee is either *permanent* or *temporary*. Every employee has an employee number, a family name, a given name, and a starting date, this last field is represented as an instance variable of type java.util.Calendar.

Each salaried employee has an annual salary, while each hourly paid employee has a basic hourly rate, a number of contracted hours per month, and a number of hours actually worked (in the current month). Permanent hourly paid workers are paid 1.5 times their normal rate

for any hours above their contracted number, but temporary employees are always paid at their basic rate.

Construct a suitable class hierarchy to represent this situation, with use of one or more abstract classes if appropriate. All employee classes should implement (either directly or indirectly) the Waged interface. Your classes should contain the appropriate set of methods – constructors, accessors, mutators, and so on, as well as those indicated in the previous paragraph. You should ensure that the `toString` method returns, among other things, the status (i.e., salaried or hourly paid, and if the latter, temporary or permanent) of the employee concerned.

The UniversityPayroll class provides a `main` method that creates a number of employee objects, tests the key methods that have been implemented, and outputs appropriate values. Some of your code should illustrate the concepts of **polymorphism** and **dynamic binding**, and you should include comments to indicate the statements that rely on these features.

Output from `main`:

```
Anton Muscatelli (salaried, #1, since 2009) pay: 20833.33
Albus Dumbledore (salaried, #2, since 1881) pay: 4166.67
Jeremy Singer (salaried, #3, since 2010) pay: 2916.67
Bob Builder (permanent, #4, since 2013) pay: 975.00
Freda Bloggs (temp, #5, since 2010) pay: 757.20
JP2Lab Tutor (temp, #6, since 2014) pay: 600.00
----------
total amount to pay: 30248.87
```

Observation

This question is primarily an exercise in object-oriented design, exploiting inheritance, polymorphism and abstract classes. Focus on the appropriate classes and the relationships between them. In terms of lines of code, a complete description of all of the classes and methods is fairly lengthy, and it will be quite time-consuming to generate all of this code. It is not crucial to complete all of the details of all of the methods of each class, but deciding which methods should be overridden is an important issue.

There are no JUnit test classes this week – so you will need to run the main method (and understand it!) If you want to follow example JUnit source code from previous weeks and write your own test cases, please go ahead!

Hint

Eclipse has many features that can assist the Java programmer. For example, for a chosen collection of instance variables in a class, accessors and mutators can be generated automatically by selecting Generate Getters and Setters from the Source menu. Try it – but be aware that these automatically generated methods may not use deep copies if objects are involved.

Please make use of the `StringBuilder` library class to construct compound `String`s in your `toString` methods.

Submission

You should submit your work before the deadline no matter whether the programs are fully working or not.

When you are ready to submit, go to the JOOSE2 moodle site. Click on Laboratory 7 Submission. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code …\Laboratory7\Submission7_1\ and drag the **five** Java files Employee.java, SalariedEmployee.java, HourlyEmployee.java, PermanentEmployee.java and TemporaryEmployee.java into the drag-n-drop area on the moodle submission page. **Your markers want to read your java files, not your class files.** Also locate your class diagram image file (JPEG or PDF) and drag this to the submission area. Then click the blue save changes button. Check the five .java files and the graphics file are uploaded to the system. Then click submit assignment and fill in the non-plagiarism declaration. Your tutor will inspect your file and return feedback to you via moodle.

Outline Markscheme

A: code compiles. correct behaviour. neat formatting. good documentation. correct UML class diagram.

B: code compiles. mostly correct behaviour. fairly neat formatting. some documentation. correct UML diagram.

C: code compiles. sensible (if not entirely correct) behaviour. readable formatting. attempt at documentation. mostly correct UML diagram.

D: code may not compile. Missing methods. readable formatting. attempt at documentation. attempt at UML diagram.

E: missing classes. poor attempt. no UML diagram.