## Assignment 2 specification – Inheritance, Polymorphism and Abstraction

This overall assignment is worth 55% for the Programming Fundamentals II module. The interview to assess your understanding and ability to explain of the code is very important. The interview marking will be used as a multiplier to determine your final mark.

**Phase 1 – Inheritance and Polymorphism :**

Create the following inheritance hierarchy in Java that defines the following types of Employee (i.e. its subclasses):

- Manager – for employees that automatically get a managerial bonus on top of their regular salary
- SalesWorker – for employees that get a sales performance bonus on top of their regular salary
- TempWorker – for employees earning their salary on an hourly basis (no bonuses)

The following rules should be applied to this hierarchy:

- The fields common for all type of Employee include first name, last name and hourly rate. The hourly rate must be greater than or equal to zero.
- In addition, there will be a CONSTANT field that will store the number of hours considered a normal working week (NORMAL_WORKWEEK). The code for creating this constant field is as follows:

  final static double NORMAL_WORKWEEK = 37.5;
- The Manager subclass should have a field for the manager bonus and a collection of Employees working under them (i.e. in their department). The manager bonus must be greater than or equal to zero, otherwise set the value to zero. If a value is already stored in the manager bonus, do not overwrite it with a zero.
- The SalesWorker subclass should have a field to store their sales performance bonus percentage. This bonus percentage must be greater than or equal to zero and not greater than 20%, otherwise set the value to zero.
- Where required, subclasses of Employee should override the superclass method double calculateSalary(double numHours), in order to calculate the salary based on the different rules defined for each type of employee (see below).
- The Employee class should have a String toString() method that formats the printing of the object state and returns it. The subclasses of Employee should override the superclass method String toString() so that they can report on the new fields defined in these subclasses.
- Each class in the hierarchy should define a constructor that initialises each instance variable based on user input data.
- Each class should also define accessors and mutators for each instance field.

The rules for calculating the weekly salary ( double calculateSalary(double numHours) )of each Employee are:

- The numHours passed in as a parameter must be greater than or equal to zero. If the numHours is not greater than or equal to zero, return 0.0 .
- Employee – An employee is paid for each hour they have worked in the current week, based on a fixed rate per hour.
- Manager – A manager is paid by the hour at a fixed hourly rate. In addition, the manager always receives a fixed manager bonus. This figure is added directly to their salary.
- SalesWorker – A Sales worker is also paid by the hour at a fixed hourly rate. However, their bonus is calculated differently to the managers. The sales worker may or may not qualify for a bonus. If they do not qualify for a bonus, their performance bonus percentage is zero. If they do qualify for a bonus their performance bonus percentage must be greater than zero but less than 20. The bonus is calculated on their salary and added to it.
- TempWorker – A temp worker is paid for each hour they have worked in the current week, based on a fixed rate per hour.

- NOTE: The rules for calculating the overtime are exactly the same for each Employee; for each hour worked overtime during the week, the hourly rate is doubled. (Overtime is any time worked over the value contained in NORMAL_WORKWEEK). The overtime calculation must be done in a private method in the Employee class, double calculateOvertime (double numHours) and should be called in the double calculateSalary method(double numHours) method.

Use abstract classes and interfaces if and when when appropriate.

**Phase 2 – Testing & Additional functionality ()**

1. Create a driver class called Driver that uses console I/O to interact with the user. The Driver class should create an ArrayList of Employees, this would be populated with different instances of the Employee class, for example some managers, some salesemployees and some temp employees. The Driver class should allow the user, through a series of menus, to do the following:

- Add a new employee of any type to the arraylist.
- Add an existing employee to a manager's department.
- Calculate salaries for each employee or the total for all employees.
- Print employee details.

Note: Aside from the above minimum requirements, the design of the menu system and the contents you include is left open to you. This is an important area of the assignment where you can demonstrate your programming skills. Also, robustness of this menu system is an important factor (i.e. exception handling).

2. JUnit test classes will be provided for all but one class (these test classes automate the testing of your solution). Having reviewed the test classes provided, create an appropriate test class for the class that is missing a JUnit test class. Test methods in this class should include:

- test method for getters and setters

- test method for constructor/s.
- test methods for each other method in the class.

Ensure that you test for both valid and invalid data entry. Ensure that you use the setup() method to help you.

**Note on validations:**

When you wish to update fields though methods (though constructors or setters) , and the input data/new value is not valid according to the validation rules, you deal with this differently in the two cases. In consructors, you should give the field a default value,( 0 if this is allowed by the rules of validation, some other sensible value otherwise) . In the setters, you should leave the field unchanged if the new value is invalid according to the validation rules.

# The following are now compulsory i.e. not doing these may lead to you failing this assignment:

**Naming conventions:**

Class names begin with capital letter – lowercase after that., e.g. **Employee**

Field names / local variables begin with lowercase, use uppercase to separate words in name, e,g, **numRounds**

For instances of collections they should follow the conventions for field names and should be plural, e.g. **dvds.**

**Comments etc.**

Comments should be written for each method using both Javadoc and code comments.

Your name should be at the top of each source file.

**Submission**

Before you submit, save your project name as your name (i.e. first character of firstname followed by second name, e.g mmeagher for Mairead Meagher's assignment). Then zip it and submit it.