Carleton University

Department of Systems and Computer Engineering

SYSC 2004 – Object Oriented Software Development – Winter 2017

Lab 3 – Object Oriented Programming

Objectives:

The objective of this lab is to give you a basic introduction to object oriented programming. At the end of this lab, you should be comfortable with overloading, static methods, static variables, copy constructor, math class and implementing equals method in Java.

Attendance/Demo:

To receive credit for this lab, you must demonstrate your lab work. When you have finished all the exercises, call a TA, who will grade the code you wrote. For those who do not finish during the lab time, a submission link will be provided on cuLearn to give you the chance to submit your solution. You will have a week to submit unfinished labs.

Note: You are required to go to your lab session. In case of your absence, your submission on cuLearn will not be considered. The TAs will collect attendance at the beginning of each lab.

Getting Started:

Step 1: Open Eclipse.
Step 2: Choose a workspace where you want to store your file (*Preferably on your USB*).
Step 3: Create a new Java project named "**Sysc2004Lab3**".

➢ Exercise 1: Create an Employee Class:

In this exercise, you will be implementing a simple program that will store information about an employee in a company.

Step 1: Create a class called **Employee** in the src folder of your created project.

Step 2: Add a *Javadoc* description to your class specifying the description of the class, your name, student number and today's date.

The basic attributes of an employee is *ID*, *name*, *age*, *title*, *salary* and *phone number*.

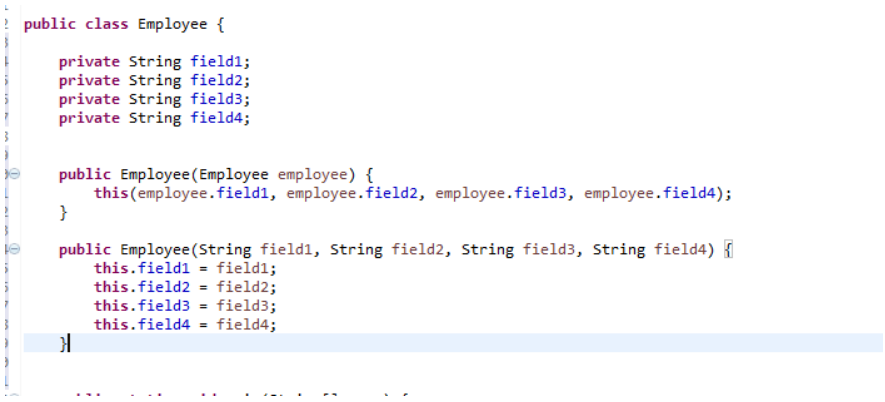
Step 3: Add the **fields** below to the *Employee class*. For each field, you will need to judge what Java type is appropriate i.e. String, Double, int …etc.

- id
- name
- age
- title
- salary
- phoneNumber

Step 4: Create a constructor for the Employee class that accepts all the fields you added in *Step 3* as parameters.

Step 5: Create a constructor for the Employee class that accepts an id, name and age as parameters. This constructor has to call the constructor you created in Step 4.

You will agree that the Employee class constructor in step 4 has too many parameters. There should be a way to simplify this. Fortunately, Java has a "**copy constructor"** strategy. A copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. Copy constructors are used when there is a complex object with many attributes.



```
public class Employee {

    private String field1;
    private String field2;
    private String field3;
    private String field4;


    public Employee(Employee employee) {
        this(employee.field1, employee.field2, employee.field3, employee.field4);
    }

    public Employee(String field1, String field2, String field3, String field4) {
        this.field1 = field1;
        this.field2 = field2;
        this.field3 = field3;
        this.field4 = field4;
    }
```

Step 6: Using the above picture as reference, create a copy constructor for the employee class. The constructor accepts an Employee class as parameters.

Step 7: Add the methods below to the Employee class and for each method, you will need to provide a return type (void, string etc.) and parameter type.

- setId ()
- setName ()
- setTitle ()
- setAge ()
- setSalary ()
- setPhoneNumber ()

- getId ()
- getName ()
- getTitle ()
- getAge ()
- getSalary ()
- getPhoneNumber ()

Overloading, toString and equals method in Java:
Now that we have an employee class, we need to add some behaviors to it. If an employee performs its job in an efficient fashion, the company tends to increase the **salary** and **title** of the employee. Therefore, we need to add a method for increasing salary.

Step 1: Create a method called **increaseSalary** that accepts **percentageOfIncrease** (int) as parameter, increases the salary by the percentage and returns the salary.

As previously stated, most company increase the salary of an employee and change the employee title consequently. The **increaseSalary** method increase the salary alone but does not change the title. Therefore, we should add a new method that increase the salary and change the title as well.

Instead of creating a new method with a different name, we will reuse the **increaseSalary** name. This is called method **overloading** in Java. Overloading is a feature that allows a class to have two or more methods have same name *if their argument lists are different.*

Step 2: Create a new method called **increaseSalary** that accepts **percentageOfIncrease** (int) and **newTitle** as parameters. The method increases the salary by the percentage, change the employee's title and returns the salary.

The company should be able to differentiate one employee from another. Therefore, we need to implement an equals method that compares two employees. In Java, every Object e.g. class has an equals method that does a shallow comparison. To have a deep comparison, the equals method has to be overridden. The correct way for overriding an equals method is shown below.

```
20    public boolean equals(Object obj) {
21        if (this == obj)
22            return true;
23
24        if (obj == null || obj.getClass() != this.getClass())
25            return false;
26
27        Employee employee = (Employee) obj;
28
29        return ((field1 == employee.field1) && (field2 == employee.field2) && (field3 == employee.field3)
30                && (field4 == employee.field4));
31    }
32
```

Step 3: Using the picture above as reference, create an **equals** method in the Employee class. This method would be used to compare 2 employees.

Every object in Java has a **toString** method. This method creates a string representation of the object. We will need to override this method to provide the company a brief overview of an employee.

Step 4: Write a **toString** method that returns a string containing the id, age, name, title and phoneNumber of an employee. Be creative with your string representation.

Static method and static variable in Java:

An employee should be able to know the total number of employees in the company. This information is not an attribute of an employee but it is an additional information for an employee and even a non-employee. For example, if you go for a job interview, the company can still reveal the total number of employees they have. Therefore, we need to implement this feature in our Employee class.

Fortunately, Java has static method and variables. A static variable is a variable that retains the same data throughout the execution of a program. The value of a static variable is the same for all object or instances of the same class. Below is how a static variable and methods are implemented.

```
public static String field5;

public static void concatenateField5 (String field6) {
    field5 += field6;
}
```

Step 1: Create a static variable called **employeeCounter**. You will have to determine its java type.

Step 2: Create a static method called **increaseEmployeeCounter**. This method will increase the **employeeCounter** by 1.

Step 3: Create a static method called **getEmployeeCounter**. This method will return the static employeeCounter value.

Step 4: Copy the **EmployeeTest** file to your src folder and run the Junit test to validate the code you have written. Refer to lab2 if you have forgotten how to run Junit tests.

➢ Exercise 2: Create a Point Class:

In this exercise, you will be implementing a point class that determines the position of an object using its x and y coordinate.

Step 1: Create a class called **Point** in the src folder of your created project.

Step 2: Add a Javadoc description to your class specifying the description of the class, your name, student number and today's date.

Step 3: From the description of a point, add the **fields** that are necessary for a point to your class.

Step 4: Create a **constructor** that accepts all the fields you just created and initializes them.

Step 5: Create a **default constructor** for your point class.

Step 6: Create a **copy constructor** for your point class.

Step 7: Add the necessary **getters** and **setters** for your Point class.

Step 8: Create a **toString** method that returns the values of all the **fields** in the point class as a String.

Step 9: Implement an **equals** method for your Point class.

Step 10: Implement a **distance** method that accepts **two** arguments (**x** and **y**) and returns the distance between **this** instance to the given x & y coordinate. You would have to figure out how to calculate the distance between two points.

Step 11: Implement another **distance** method that accepts a **Point class** as argument and returns the distance between **this** instance to the given point distance.

Step 12: Implement a third **distance** method that accepts no arguments but returns the distance between **this** instance and point **(0, 0).**

Step 13: Test all the methods you wrote in the **main** method of the Point class. This is what the TAs would use to judge if you have a working solution or not.

➢ Exercise 3: Create a Robot Class:

In this exercise, you will be implementing a robot class that models a moving robot. To move a robot, we are going to need **three** things.

- The position or point of the robot i.e. the x and y coordinate of the robot.

- The displacement or step in the x direction.
- The displacement or step in the y direction.

For example if a robot is at point (3,4), x-displacement is 5 and the y-displacement is 4. After moving, the robot would be at a point (8,8). If the robot moves again, it would be at a point (13, 12).

Step 1: Create a class called **Robot** in the src folder of your created project.

Step 2: Add a Javadoc description to your class specifying the description of the class, your name, student number and today's date.

Step 3: Add the following field to your robot class.

- **Point class** (*this is the same point you implemented in exercise 2*)
- **xDisplacement (double)**
- **yDisplacement (double)**

Step 4: Create a **constructor** that accepts a **Point class, xDisplacement and yDisplacement**.

Step 5: Create another constructor that accepts a **pointX, pointY, xDisplacement and yDisplacement.**

Step 6: Create a **default constructor** for your Robot class.

Step 7: Create a **copy constructor** for your Robot class.

Step 8: Add the necessary **getters** and **setters** for your Robot class.

Step 9: Create a **toString** method that returns the robot's **point, xDisplacement and yDisplacement** as a String. Below is an example.

- "Robot@(**x**,**y**), speed=(**xDispacement**, **yDisplacement**)".

Step 10: Implement an **equals** method for your Robot class. In your implementation, make sure you are comparing the robot's point using the **equals** method in the **Point class.**

Step 11: Implement a **move** method that accepts **no** arguments, moves the robot and returns **a Robot instance**.

Step 12: Implement a **move** method that accepts **two** arguments (**xDisplacement** and **yDisplacement**), *changes the direction of the Robot*, *moves the robot* and returns **a Robot instance**.

Step 13: Test all the methods you wrote in the **main** method of the Robot class. This is what the TAs would use to judge if you have a working solution or not.

➢ Exercise 4 (Bonus): Approximate the square root

There are several techniques for implementing the `sqrt` method in the Math class. One such technique is known as the **Babylonian method**. It approximates the square root of a number, `n`, by repeatedly performing a calculation using the following formula:

```
nextGuess = (lastGuess + n / lastGuess) / 2
```

When `nextGuess` and `lastGuess` are almost identical, `nextGuess` is the approximated square root. The initial guess can be any positive value (e.g., 1). This value will be the starting value for `lastGuess`. If the difference between `nextGuess` and `lastGuess` is less than a very small number, such as 0.0001, you can claim that `nextGuess` is the approximated square root of `n`. If not, `nextGuess` becomes `lastGuess` and the approximation process continues. In this question, you are required to implement the following method that returns the square root of `n`:

```
class SquareRoot {
// Find the square root of the value
public static double sqrt(double num) {
}
}
```