LAB #3 –Using Inheritance and Polymorphism

Student: _____

Due Date: **Week 7**.

Purpose: The purpose of this Lab assignment is to:

- Practice the use of Inheritance
- Practice the use of Polymorphism.

References: Read the Lecture Notes #5, 6.

- This material provides the necessary information you need to complete the exercises.

Be sure to read the following general instructions carefully:

This lab should be completed individually by all the students. You will have to demonstrate your solution in a scheduled lab session and submitting the project **through the dropbox link on D2L**.

You must name your Eclipse project according to the following rule:

YourFullName_COMP228Labnumber
Example: **JohSmith_COMP228Lab3**

Each exercise should be placed in a separate project named *exercise1*, *exercise2*, etc.

Submit your assignment in a **zip file** that is named according to the following rule:
YourLastName_COMP228Labnumber.zip
Example: **JohSmith_COMP228Lab3.zip**

Apply the naming conventions for variables, methods, classes, and packages:
- *variable names* start with a *lowercase* character
- *classes* start with an *uppercase* character
- **packages** use only *lowercase* characters
- *methods* start with a *lowercase* character

Exercise 1

Write a Java application that implements different types of insurance policies for employees of an organization.

Let **Insurance** be an abstract superclass and **Health** and **Life** two of its subclasses that describe respectively health insurance and life insurance.

The **Insurance** class defines an instance variable of type **String** to describe the **type of insurance** and an instance variable of type **double** to hold the **monthly cost** of that insurance.

Implement the **get** methods for both variables of class **Insurance**. Declare also two **abstract** methods named **setInsuranceCost()** and **displayInfo()** for this class.

The **Life** and **Health** class should implement **setInsuranceCost** and **display** methods by setting the appropriate monthly fee and display the information for each insurance type.

Write a driver class to test this hierarchy. This application should ask the user to enter the type of insurance and its monthly fee. Then, will create the appropriate object (Life or Health) and display the insurance information.

As you create each insurance object, place an **Insurance** reference to each new **Insurance** object into an array. Each class has its own **setInsuranceCost** method. Write **a polymorphic** screen manager that walks through the array sending **setInsuranceCost** messages to each object in the array and displaying this information on the screen.

(3 marks)

Exercise #2:

Create an abstract class called GameTester. The GameTester class includes a name for the game tester and a boolean value representing the status (full-time, part-time).

Include an abstract method to determine the salary, with full-time game testers getting a base salary of $3000 and part-time game testers getting $20 per hour.

Create two subclasses called FullTimeGameTester, PartTimeGameTester. Create a console application that demonstrates how to create objects of both subclasses. Allow the user to choose game tester type and enter the number of hours for the part-time testers.

(3 marks)

Exercise #3:

CityToronto bank provides mortgages for individuals and businesses up to $300,000. Write a Java application that keeps track of mortgages and computes the total amount owed at any time (mortgage amount + interest).

Design the following classes to implement your application:

Mortgage – an abstract class that implements the *MortgageConstants* interface. A Mortgage includes a mortgage number, customer name, amount of mortgage, interest rate, and term.

Don't allow mortgage amounts over $300,000. Force any mortgage term that is not defined in the *MortgageConstants* interface to a short-term, one year loan. Create a *getMortgageInfo* method to display all the mortgage data.

MortgageConstants – includes constant values for *short-term* (one year), *medium-term* (three years) and *long-term* (5 years) mortgages. It also contains constants for bank name and the maximum mortgage amount.
BusinessMortgage – extends Mortgage. Its constructor sets the interest rate to 1% over the current prime rate.
PersonalMortgage - extends Mortgage. Its constructor sets the interest rate to 2% over the current prime rate.
ProcessMortgage – a main class that create an array of 3 mortgages. Prompt the user for the current interest rate. Then in a loop prompts the user for a mortgage type and all relevant information for that mortgage. Store the created Mortgage objects in the array. When data entry is complete, display all mortgages.

(4 marks)

Evaluation:

Functionality	
Correct implementation of classes (instance variable declarations, constructors, getter and setter methods, etc.)	40%
Correct implementation of driver classes (declaring and creating objects, calling their methods, interacting with user, displaying results)	40%
Comments, correct naming of variables, methods, classes, etc.	5%
Friendly input/output	15%
Total	100%