Lab 12: Inheritance and Polymorphism

Let's get started!

Today we'd like you to:

1. **Open Eclipse**. Remember to keep all your projects in the same workspace. Think of a workspace as a shelf, and each project is a binder of stuff on the shelf. If you need some help organizing your workspace, we'd be happy to help!
2. **Create a new Java project**. Call it something like COMP 1510 Lab 12.
3. **Complete the following tasks**. Remember you can right-click a Java project's src package in the Eclipse Package Explorer pane (Window > Show View > Package Explorer) to quickly create a new Java class, enum, interface, etc.
4. When you have completed the exercises, **show them to your lab instructor**. Be prepared to answer some questions about your code and the choices you made.
5. **Remember that for full marks your code must be properly indented, fully commented, and free of Checkstyle complaints**. Remember to activate Checkstyle by right-clicking your project in the Package Explorer pane and selecting Checkstyle > Activate Checkstyle.

What will you DO in this lab?

In this lab, you will:

1. employ inheritance in a program design
2. explore and modify sorting algorithms
3. gain more experience with the Comparable interface
4. use an abstract class and override its methods
5. study how polymorphism works with some simple class hierarchies.

Table of Contents

1. Painting shapes (using polymorphism!)

Let's dive right into polymorphism and develop a class hierarchy of shapes. We will write a program that computes the amount of paint needed to paint different objects. We'll start with an abstract superclass called Shape, and develop three concrete subclasses:

1. **Create an abstract Java class called Shape**. The class should contain the following members:
    i. private instance variable of type String to store the name of the Shape
    ii. abstract method called surfaceArea that returns a double
    iii. toString method that returns the name of the Shape.
2. **Create a Java class called Sphere**. Sphere extends Shape. Sphere **IS-A** Shape. Sphere should contain the following members:
    i. private instance variable of type double to store the radius of the Sphere
    ii. constructor that accepts a parameter of type double and assigns it to the instance variable if the parameter is greater than 0. If it is zero or negative, throw a new IllegalArgumentException.
    iii. implementation of the Shape's surfaceArea method that returns the surface area of the Sphere based on the radius.
    iv. toString method that returns the name of the Shape, and its state.
3. **Create a Java class called Cube**. Cube extends Shape. Cube **IS-A** Shape. Cube should contain the following members:
    i. private instance variable of type double to store the side length of the Cube
    ii. constructor that accepts a parameter of type double and assigns it to the instance variable if the parameter is greater than 0. If it is zero or negative, throw a new IllegalArgumentException.
    iii. implementation of the Shape's surfaceArea method that returns the surface area of the Cube based on the side length.
    iv. toString method that returns the name of the Shape, and its state.
4. **Create a Java class called Cylinder**. Cylinder extends Shape. Cylinder **IS-A** Shape. Cylinder should contain the following members:
    i. private instance variables of type double to store the radius of the Cylinder and the height of the Cylinder
    ii. constructor that accepts two parameters of type double. If either parameter is equal to zero or negative, throw a new IllegalArgumentException, otherwise assign the parameters to the instance variables.
    iii. implementation of the Shape's surfaceArea method that returns the surface area of the Cylinder based on the radius and height.
    iv. toString method that returns the name of the Shape, and its state.
5. **Create a Java class called PaintCan**. A PaintCan is not a Shape. It stores the amount of coverage in a can of paint, i.e., how much surface area it will cover. A PaintCan should contain:
    i. private instance variables of type double to store coverage
    ii. constructor that accepts a positive double and assigns it to coverage

iii. a method called public double amount(Shape shape) which computes the number of cans of paint needed to paint the surface area of the Shape passed as a parameter.

6. Finally, let's create a driver class called **PaintThings**. PaintThings should:
    a. Contain a **main** method that drives the program.
    b. Use a Scanner to ask the user for how much coverage a can of their favourite paint will provide (I used 250 when experimenting with this)
    c. Use this value to instantiate a PaintCan with the specified coverage.
    d. Create a Cube, a Sphere, and a Cylinder. Ask the user for the dimensions for each Shape.
    e. How many cans of paint are required to paint the surface area of each Shape? Report the total to the user.

2. Sorting (using polymorphism!)

The file Sorting.java (available on D2L with this lab document) implements both the Selection Sort and the Insertion Sort algorithms. Copy the file to your project. Take a moment to look at the source code. Note that:

1. Both sorting methods are static
2. Both sorting methods accept a parameter of type T[], where T[] is an array of some sort of class that implements Comparable. Remember that a String implements Comparable<String>, for example.
3. Can you tell by looking at the code whether the algorithms sort into ascending or descending order?

The file Numbers.java (also available on D2L with this lab document) uses the Insertion Sort to sort some numbers. There is a small compiler error in this file which prevents it from executing. Move your mouse over the compiler error and read the message. It's a little cryptic, but it should help you decide how to fix the error. (Hint: think about the difference between an object and a primitive.)

4. Correct the compiler error in Numbers.java and execute the file.
5. **Create a Java class called Strings**. This file should be identical to Numbers.java except it should ask the user for some Strings and then sort them. We can do this because String implements Comparable<String>.
6. Modify the algorithms so that they now sort in descending rather than ascending order.
7. Modify Numbers.java and Strings.java so they use the Selection Sort as well. Confirm both algorithms work correctly.

Let's create another class that implement Comparable and let's try sorting it:

8. Create **a Java class called Salesperson**:

   a. A Salesperson has a first name and a last name (Strings) and a total number of sales (int). We'll pretend we work in one of those shoppes where we ignore coins and deal in round integer values only.
   b. Create a **constructor** that accepts parameters for all three instance variables. If either of the names are null or empty String, throw an IllegalArgumentException.
   a. Create **accessors** for all three instance variables.
   b. We will not create mutators, so the instance variables should be **final**. We are creating an immutable class.
   c. Salesperson should implement Comparable<Salesperson>. This means the Salesperson must have a method called compareTo that accepts another Salesperson as a parameter and returns an int. Salespeople who sell more come first.
2. Finally, let's create a driver class called **WeeklySales**. WeeklySales should:
   a. Contain a **main** method that drives the program.
   b. Create an array of 10 Salespeople.
   c. Add 10 Salespeople to the array (you can hard-code a variety of values)
   d. Invoke the static sorting method on the array.
   e. Print out the ranked sales for the week. Make it easy to read!

3. Managing employees (with inheritance and polymorphism!)

This is it. The final lab exercise in your final COMP 1510 lab. Take a deep breath and think about how much you've learned in the last three months. Wow! Let's get started.

1. The classes **Firm**.java. **Staff**.java, **StaffMember**.java, **Volunteer**.java, **Employee**.java, **Executive**.java, and **Hourly**.java have been provided to you. Examine the code in each class in the order listed above.
2. Can you **draw** a simple diagram of the class hierarchy here?
3. Can you **explain** how this program works to your neighbour?
4. Let's create an additional class to make things interesting. Let's create an employee who gets paid hourly and who also earns commission.
5. Create **a Java class called Commission**:
   a. Commission **extends** the Hourly class because it IS-A type of Hourly employee
   b. Add two **instance variables**:
      i. a double to store total sales
      ii. a double to store the commission rate or percentage (a value between 0 and 1.0). For example, a commission rate of 0.2 means the Commission employee earns 20% commission on sales.
   c. The **constructor** should accept 6 parameters:
      i. Name
      ii. Address
      iii. Phone number
      iv. SIN number
      v. Hourly pay rate
      vi. Commission rate

   d. Remember to pass most of these to the superclass. Only commission rate is stored in a Commission object.
   e. Create a method called **addSales** which accepts a double and adds the parameter to the instance variable representing total sales.
   f. The **pay** method must call the pay method of the superclass to compute the pay for hours worked, and then add to that the pay from commissioned sales. For a hint, check out the pay method in the Executive class. Make sure to reset totalSales to zero before returning the pay.
   g. Create a **toString** that appends the total sales to the toString returned from the superclass.
6. Finally, let's modify the driver to test the Commission class. In the Staff class:
   a. Increase the size of the array to 8.
   b. Add 2 commissioned employees to the staff list:
      i. One makes $10 per hour and 20% commission
      ii. The other makes $30 per hour and 5% commission
   c. The employee who earns $10 per hour worked 35 hours and had sales of $500.
   d. The employee who earns $30 per hour worked 20 hours and had sales of $2000.
   e. Compile and run the program. Does it work correctly?

4. You're done! Show your lab instructor your work.