COSC 211: Programming Data Structure
Fall 2013

Lab #4: Polymorphism and Abstract Classes
Laboratory Exercises for Monday, October 14
(Source: Bloss, A and Ingram, N.J., Lab Manual Java Software Solutions, Pearson, 2009)

The following laboratory exercises should be completed by the end of today's laboratory session. Upon completion of each exercise, you are expected to have the lab instructor or the lab assistant check the correctness of your program by demonstrating its execution. Make sure you have a printout of the source codes ready before demonstrating your program. At that time, the lab instructor (or the lab assistant) will assign a grade for the completed program. If you have not completed both exercises by the end of the laboratory session, you still have to make sure you submit a printout of any partially working program to get partial credit for your efforts**. Failure to do so will result in a grade of ZERO for the lab exercise**.

Exercise #1: Another Type of Employee

Download the files *Firm.java, Staff.java, StaffMember.java, Volunteer.java, Employee.java, Executive.java,* and *Hourly.java* from my.emich. The program illustrates inheritance and polymorphism. In this exercise you will add one more employee type to the class hierarchy. The employee will be one that is an hourly employee but also earns a commission on sales. Hence the class, which we'll name *Commission*, will be derived from the *Hourly* class.

Write a class named *Commission* with the following features:

1. It extends the *Hourly* class.
2. It has two instance variables (in addition to those inherited):
   - One is the total sales the employee has made (type double) and
   - the second is the commission rate for the employee (the commission rate will be type double and will represent the percent (in decimal form) commission the employee earns on sales (so .2 would mean the employee earns 20% commission on sales)).
3. The constructor takes 6 parameters:
   - The first 5 are the same as for *Hourly* (name, address, phone number, social security number, hourly pay rate) and the 6th is the commission rate for the employee.
   - The constructor should call the constructor of the parent class with the first 5 parameters then use the 6th to set the commission rate.
4. One additional method is needed:

   public void addSales (double totalSales)

   that adds the parameter to the instance variable representing total sales.
5. The *pay* method must call the pay method of the parent class to compute the pay for hours worked then add to that the pay from commission on sales. (See the pay method in the *Executive* class.) The total sales should be set back to 0 (note: you don't need to set the hoursWorked back to 0—why not?).
6. The *toString* method needs to call the *toString* method of the parent class then add the total sales to that. To test your class, update *Staff.java* as follows:
7. Increase the size of the array to 8.

8. Add two commissioned employees to the *staffList*—make up your own names, addresses, phone numbers and social security numbers. Have one of the employees earn $6.25 per hour and 20% commission and the other one earn $9.75 per hour and 15% commission.
9. For the first additional employee you added, put the hours worked at 35 and the total sales $400; for the second, put the hours at 40 and the sales at $950.
10. Compile and run the program. Make sure it is working properly. Print a copy of your source code for the *Commission* and *Staff* classes only and demo your program to the instructor or the lab assistant.

Exercise #2: Painting Shapes

In this lab exercise you will develop a class hierarchy of shapes and write a program that computes the amount of paint needed to paint different objects. The hierarchy will consist of a parent class *Shape* with three derived classes - *Sphere, Rectangle,* and *Cylinder*. For the purposes of this exercise, the only attribute a shape will have is a name and the method of interest will be one that computes the area of the shape (surface area in the case of three-dimensional shapes). Do the following.

1. Write an abstract class *Shape* with the following properties:
   - An instance variable *shapeName* of type String
   - An abstract method *area()*
   - A t*oString()* method that returns the name of the shape

2. The file *Sphere.java* contains a class for a sphere which is a descendant of Shape.
   - A sphere has a radius and its area (surface area) is given by the formula *4*PI*radius^2*.
   - Define similar classes for a rectangle and a cylinder. Both the *Rectangle* class and the *Cylinder* class are descendants of the *Shape* class.
   - A rectangle is defined by its length and width and its area is *length*width*.
   - A cylinder is defined by a radius and height and its area (surface area) is *PI*radius^2*height*.
   - Define the *toString()* method in a way similar to that for the *Sphere* class.

3. The file *Paint.java* contains a class for a type of paint (which has a "coverage" and a method to compute the amount of paint needed to paint a shape). Correct the *return* statement in the *amount()* method so the correct amount will be returned. Use the fact that the amount of paint needed is the area of the shape divided by the coverage for the paint. (NOTE: Leave the print statement - it is there for illustration purposes, so you can see the method operating on different types of *Shape* objects.)

4. The file *PaintThings.java* contains a program that computes the amount of paint needed to paint various shapes. A paint object has been instantiated. Add the following to complete the program:
   - Instantiate the three shape objects:
     - deck to be a 20 by 35 foot rectangle,
     - bigBall to be a sphere of radius 15, and
     - tank to be a cylinder of radius 10 and height 30.
   - Make the appropriate method calls to assign the correct values to the three amount variables.
   - Run the program and test it. You should see polymorphism in action as the amount method computes the amount of paint for various shapes.
5. Print a copy of your source code for the *Shape, Sphere, Paint* and *PaintThings* classes only and demo your program to the instructor or the lab assistant.