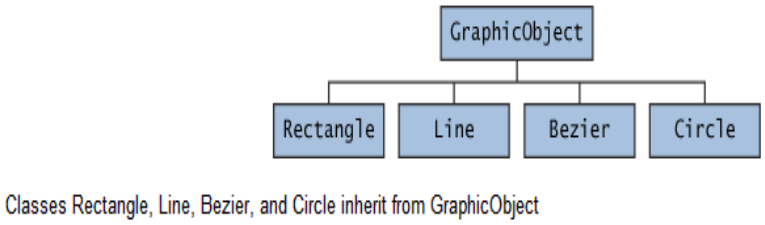
Indian Institute of Information Technology, Allahabad
Object Oriented Methodology (OOM) -2015
Lab Assignment-04

TAs: Bharat Singh, Nidhi Kushwaha, Monika Rani, Balasaheb, Diksha, Daisy Monika, Yogesh Kumar, Saurabh, Ravi shankar, Rakesh, Sunakshi, Fahim Altaf

Date of assignment: 19/08/2015 **Due Date: 02/09/2015** **Instructor: Prof. O. P. Vyas & Dr. Ranjana Vyas**

Note: 1) All Assignments should be done independently.
2) Assignment should be evaluated within the deadline is essential.
3) Write simple code in java for printing appropriate value for class instances.

Q1. In an object-oriented drawing application, you can draw circles, rectangles, lines, Bezier curves, and many other graphic objects. These objects all have certain states (for example: position, orientation, line color, fill color) and behaviors (for example: moveTo, rotate, resize, draw) in common. Some of these states and behaviors are the same for all graphic objects—for example: position, fill color, and moveTo. Others require different implementations—for example, resize or draw. All GraphicObjects must know how to draw or resize themselves; they just differ in how they do it. This is a perfect situation for an abstract superclass. You can take advantage of the similarities and declare all the graphic objects to inherit from the same abstract parent object—for example, GraphicObject, as shown in the following figure.



Classes Rectangle, Line, Bezier, and Circle inherit from GraphicObject

Fig1.

First, you declare an abstract class, GraphicObject, to provide member variables and methods that are wholly shared by all subclasses, such as the current position and the moveTo method. GraphicObject also declares abstract methods for methods, such as draw or resize, that need to be implemented by all subclasses but must be implemented in different ways. The GraphicObject class can look something like this:

```
abstract class GraphicObject {
int ----
void moveTo(int newX, int newY) {
…..  }
abstract void draw ();
abstract void resize ();
}
```

Implements the abstract methods and print it with necessary output.

Q2. Old Lady had a farm and several types of animals. Every animal shared certain characteristics. They each had a type (such as Deer, Cat, and Chick.) and each made a sound (moo, mew, cluck). Follow the steps to help old lady to maintain her farm house.

1. Define a class Animal with the methods required to be an animal on the farm like: getSound() and getType().**[Animal.java]**

2. Create an Interface that defines the things required to be a Vegetarian on the farm.

[Vegetarian.java]

3. Other Interface that defines the properties of the things. Create methods getBehavior(),getThingType().**[Object.java]**

4. Implement all these method for each of the classes for the Deer, Cat and Chick using the interface Animal and Object. **[Deer.java, Cat.java, Chick.java]**

5. Create a complete farm to test all animals' classes and named it as Farm.java. Also complete the test program to verify your work so far. **[Hint :]**

class farm extends Animal implements Object, Vegetarian.

Q3. Write a program to represent geometric shapes and some operations that can be performed on them. The idea here is that shapes in higher dimensions inherit data from lower dimensional shapes. For example a cube is a three dimensional square. A sphere is a three dimensional circle and a glome is a four dimensional circle. A cylinder is another kind of three dimensional circle. The circle, sphere, cylinder, and glome all share the attribute radius. The square and cube share the attribute side length. There are various ways to use inheritance to relate these shapes but please follow the inheritance described in the table below.
All shapes inherit getName() from the superclass Shape.
Specification:

Your program will consist of the following classes: Shape, Circle, Square, Cube, Sphere, Cylinder, and Glome and two interfaces Area and Volume (Area.java and Volume.java are given below).

Your classes may only have the class variable specified in the table below and the methods defined in the two interfaces Area and Volume. You will implement the methods specified in the Area and Volume interfaces and have them return the appropriate value for each shape. Class Shape will have a single public method called getName that returns a string.

Class	Class Variable	Constructor	Extends	Implements
Shape	String name	Shape()		
Circle	double radius	Circle(double r, String n)	Shape	Area
Square	double side	Square(double s, String n)	Shape	Area
Cylinder	double height	Cylinder(double h, double r, String n)	Circle	Volume
Sphere	None	Sphere(double r, String n)	Circle	Volume
Cube	None	Cube(double s, String n)	Square	Volume
Glome	None	Glome(double r, String n)	Sphere	Volume

Q4. Develop a polymorphic banking program using the Account hierarchy (based on your understanding). Create a vector of Account pointers to Savings_Account and Checking_Account objects. For each Account in the vector, allow the user to specify an amount of money to withdraw from the Account using member function debit and an amount of money to deposit into the Account using member function credit. As you process each Account, determine its type. If an Account is a SavingsAccount, calculate the amount of interest owed to the Account using member function calculateInterest, then add the interest to the account balance using member function credit. After processing an Account, print the updated account balance obtained by invoking base class member function getBalance. Use **method overloading** in the prescribed place in your code.

Q5. In **overriding** the new method should essentially perform t he same functionality as the method that it is replacing however by changing the functionality we can improve the method and make its function more appropriate to a specific subclass.

To see its use Lets Imagine a special category of Magazine which has a disc attached to each copy-we can call this a DiscMag and we would create a subclass of Magazine to deal with DiscMags. When a new issue of a DiscMag arrives not only do we want to update the current stock but we want to check that discs are correctly attached. Therefore we want some additional functionality in the recvNewIssue() method to remind us to do this. We achieve this by redefining recvNewIssue() in the DiscMag subclass. Note that when a new issue of Magazine arrives, as these do not have a disc we want to invoke the original recNewIssue() method defined in the Magazine class.

When we call the recvNewIssue() method on a DiscMag object Java automatically selects the new overriding version-the caller doesn't need to specify this, or even know that it is an overridden method at all. When we call the recvNewIssue() method on a Magazine it is the method in the super class that is invoked.

To implement DiscMag we must create a subclass od Magazine using extends. No additional instance variables or methods are required though it is possible to create some if there was a need. The constructor for DiscMag simply passes ALL its parameters directly on to the super class and a version of a newIssue() is defined in discMag to overrides the one inherited from Magazine.

Implement the program having the functionality as described above and also have a look on Fig.2,3 for simplicity.

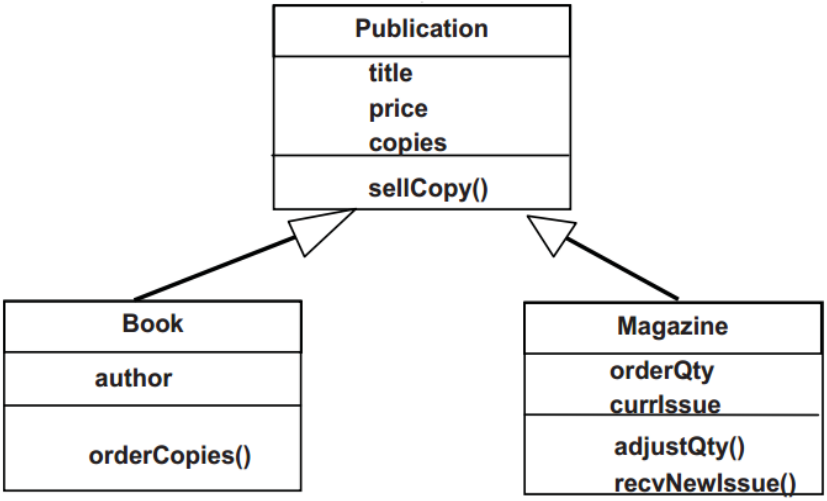
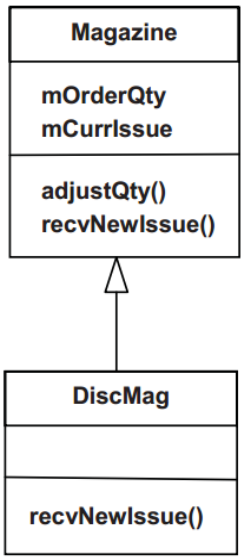


Fig.2



- The definition of **recvNewIssue()** in DiscMag overrides the inherited one.
- Magazine is not affected – it retains its original definition of **recvNewIssue()**
- By showing **recvNewIssue()** in DiscMag we are stating that the inherited method is being overridden (ie. replaced) as we do not show in inherited methods in subclasses.

Fig.3

Q6. Write a program using Inheritance having following functionalities:

1. Write a class called Shape that only contains a method named draw(), with no parameters and a void return value. The draw method should print out "Drawing a Shape".
2. Save and compile the Shape class.
3. Write a class named Rectangle that extends Shape. Add two int fields for the width and height of the rectangle, and add a constructor with two int parameters that are used to initialize these fields. Use encapsulation to ensure that the value of width and height are between 1 and 15.
4. In your Rectangle class, override the draw() method in Shape. Using nested for loops and asterisks (*), print out the rectangle using its proper width and height. For example, if width is 7 and height is 3,the draw() method should display:

```
*******
* *
*******
```

5. Save and compile the Rectangle class.
6. Write a class named RightTriangle that extends Shape. Add two int fields for the base and height of the triangle, and add a constructor with two int parameters that are used to initialize these fields. Use encapsulation to ensure that the value of base and height are between 1 and 20.
7. In your RightTriangle class, override the draw() method in Shape. Using nested for loops and asterisks, print out the triangle similarly to the way you printed out the Rectangle. For example, if the base is 8 and the height is 4, the output should look similar to:

```
*
***
******
********
```

8. Save and compile the RightTriangle class.
9. Write a class named Artist that contains a method named draw- Shape(). The method has one parameter of type Shape and returns void. The drawShape() method should invoke the draw() method on whatever Shape object is passed in.
10. Save and compile the Artist class.
11. Write a class named ArtistDemo that contains main(). Within main(), instantiate a Shape, a Rectangle, and a RightTriangle object.
12. Within main(), also instantiate an Artist object. Using your Artist object, invoke drawShape() three times, once for each of the three shapes instantiated in the previous step.
13. Save, compile, and run the Artist class.

Q7. The purpose of this lab is to become familiar with writing abstract classes and methods. This lab is a continuation of the work you did in previous question.

1. Within your Shape class from the previous question, remove the body of the draw() method, and declare the method abstract.
2. Save and compile your Shape class.
3. Write a class named Ladder that extends Shape. Add a field of type int to represent the number of rungs in the ladder, and add a constructor that initializes this field. Use encapsulation to ensure the number of rungs is between 1 and 10.
4. Your Ladder class needs to override draw(). Using asterisks, draw a ladder with the appropriate number of rungs. For example, a fiverung ladder should look similar to:

```
*************************
* * * * *
* * * * *
*************************
```

5. Save and compile the Ladder class.

6. Modify your ArtistDemo program so that it also instantiates and draws a Ladder object.

Q8. Write a program for following scenario**:-**

1. Creating Abstract Superclass Employee
2. Creating Concrete Subclass SalariedEmployee
3. Creating Concrete Subclass HourlyEmployee
4. Creating Concrete Subclass CommissionEmployee
5. Creating Indirect Concrete Subclass BasePlusCommissionEmployee
6. Demonstrating Polymorphic Processing, Operator instanceof and Downcasting
7. Summary of the Allowed Assignments Between Superclass and Subclass Variables.