# CSC 205 Lab 7 : Advanced Inheritance, Polymorphism, &

# Abstract Classes

## Goals

After completing this lab, you should be able to:

* Understand the role and capability of polymorphism in object-oriented programming.
* Be able to detect compiler errors and run time errors involving the assignment of base classes and derived classes.
* Be able to override methods of a derived class.
* Understand the use of the abstract identifier with both classes and methods, and be able to implement an abstract method of a base class.

## Lab Startup

Change into your Labs directory, and let's create and change into a Lab7 directory.

Now, let's copy over some files by typing the command below. Notice the **–rR** extension that will allow you to bring over a whole directory.

**cp –rR /pub/digh/CSC205/Lab7/\* .**

**Practice with Polymorphism**

Consider classes A, B, and Client on your reference sheet.

* Which represents a base class?
* Which represents a derived class?
* What attributes does the derived class have that the base class does not?
* Are any methods being overridden? If so, which ones?

Now, let’s trace through the Client program. What exactly will be output to the screen? Underline every point in this program where polymorphism occurs.

Now, suppose we add the following lines of code to this program.

objectA = new A('a');

objectB = (B) objectA;

System.out.print("\n3) printMe(objectB) = ");

printMe(objectB);

##### Are these lines valid or invalid? Why or why not?

Now, let’s uncomment them within our program in your current directory and be sure.

* Does the program compile and execute? Explain what happens.

## Adding Functionality to Shape and Its Subclasses

Now, let’s create a file name ShapeTest.java. Write a *test-driver* class named ShapeTest that contains a main method that tries to instantiate (declare) a Shape object using new.

* Can you compile or execute your program? Why not?

Now, let’s make the following changes to the Shape, Circle, and Rectangle classes.

1. Add an abstract method named scale to the Shape class and implement it in both the Circle and Rectangle classes. Add it in the area with the other abstract methods that have already been declared. An abstract method is a “dummy” method that has no body.

public abstract void scale(double factor);

scale will increase the size of a shape by a specified factor of type double (supplied as an parameter). So, this is a void method that merely multiplies the scaling factor times each of the private attributes of the Circle and Rectangle classes. Make sure you cast appropriately since your instance variables are integers and your parameter is a double.

2. Add an abstract method named area to the Shape class and implement it in both the Circle and Rectangle classes. area will have no parameters and will return the area of a shape as a double value. To find the area of a circle you can multiply pi times the diameter squared, and then divide this product by 4.

3. Now, let’s view the ShapeTest program. Notice we have a Color object to represent a red component of 10, a blue component of 30, and a green component of 20. Next, we have a Circle object myCircle at position (0,0) and a diameter of 5. Also, declare a Rectangle object myRectangle at position (0,0) and a width of 2, height of 3.

We then print out both objects. Then, scale both by a factor of 5.0, reprint, and print the area of the new scaled objects. Compile and run this program. You should have output similar to the following. If anything doesn’t compile or match the output below, something is wrong with your area or scale methods in the Circle and/or Rectangle classes.

myCircle = Position: (0, 0) Color: (r=10, b=30, g=20) Diameter: 5

myRectangle = Position: (0, 0) Color: (r=10, b=30, g=20) Width: 2 Height: 3

myCircle = Position: (0, 0) Color: (r=10, b=30, g=20) Diameter: 25

myRectangle = Position: (0, 0) Color: (r=10, b=30, g=20) Width: 10 Height: 15

myCircle's area = 490.8738521234052

myRectangle's area = 150.0

4. Now, let’s take a look at the Triangle class in your current directory. Since a triangle is a shape, we have inheritance! Now, let’s add a size attribute to this class of type integer, and include a four parameter constructor that takes the coordinates of the position, the color object, and a size. We’ll assume all of our triangle objects are equilateral triangles, so the Triangle class will only need this one instance variable which will represent the length of one of the triangle’s sides.

You’ll also need to add a scale and area method to the Triangle class so that it will not become an abstract class itself (what occurs when a derived class fails to implement an abstract method of its parent). You do not need a toString() method. Compile your Triangle class when you’re done. We’ll test it in the next section.

## The NervousShape Application

Look at the NervousShape application program. Notice the createShapes method to generate circles, rectangles, and triangles with equal probability. It generates circles with less than an 0.33 probability and rectangles with less than an 0.66 probability. Otherwise, generate triangles.

Now, let’s prepare to view our graphical creation. To do this, you’ll need to reboot in *Linux* mode.

Have a feeling about how these nervous little things are shaking!

**Lab #6 Reference Sheet**

**The A Class**

class A

{

private char c;

public A (char c)

{

this.c = c;

}

public void write()

{

System.out.print(c + " ");

}

}

**The B Class**

class B extends A

{

private char d;

public B(char ch1, char ch2)

{

super(ch1);

this.d = ch2;

}

public void write()

{

super.write();

System.out.print(d);

}

}

**The Client Class**

public class Client

{

public static void main(String[] args)

{

A objectA = new A('x');

B objectB = new B('y', 'z');

System.out.print("\n1) printMe(objectA) = ");

printMe(objectA);

System.out.print("\n2) printMe(objectB) = ");

printMe(objectB);

objectA = new B('b', 'c');

objectB = (B) objectA;

System.out.print("\n4) printMe(objectB) = ");

printMe(objectB);

objectB = new B('d', 'e');

objectA = (A) objectB;

System.out.print("\n5) printMe(objectA) = ");

printMe(objectA);

objectA = new B('f', 'g');

objectB = (B) objectA;

System.out.print("\n6) printMe(objectB) = ");

printMe(objectB);

System.out.println();

}

private static void printMe(A someObject)

{

someObject.write();

}

}