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? Class A
- Which represents a derived class? Class B
- What attributes does the derived class have that the base class does not?
  Hold's Class A's value and then prints the added value
- 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.

1) printMe(objectA) = x
2) printMe(objectB) = y z
4) printMe(objectB) = b c
5) printMe(objectA) = d e
6) printMe(objectB) = f g

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?

Invalid, run time error. parent can't go into derived class. Compiler is being cast to class B so it is compiling.

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

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

Yes No Casting error

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.