

Homework 7: Computing Areas with Object-Oriented Methods

Due: March 8 at 11:55pm

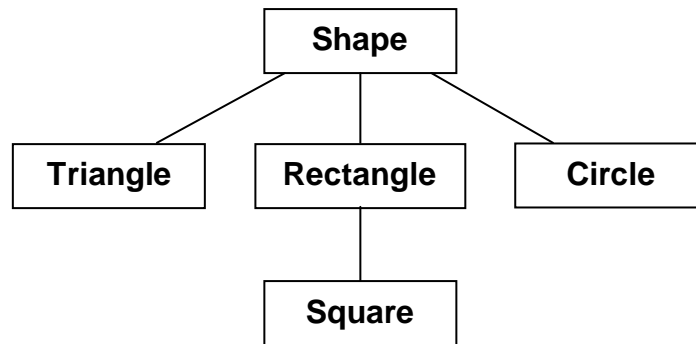
This program revisits an earlier assignment, Homework 2, in which you computed the areas of various shapes. In this assignment, you will investigate how this problem could be solved using object-oriented methods.

As in Homework 2, your program will read descriptions of some shapes from `System.in` and will compute their areas. The program will then print the sum of the areas of the shapes, and a description of the shape that has the largest area. If more than one shape is tied for largest area, you should print the description of the first of these largest shapes that is encountered in the input.

Your program should consist of a class named `HW7`. Its `main` method should read input lines from `System.in`. Each input describes one shape which is a colored triangle, square, rectangle, or circle, except the last line, which contains the single word `end`. The descriptions of the various shapes are in the same format that was used in Homework 2, and are repeated here for your convenience:

- A triangle description consists of the word `triangle` followed by a string specifying a color and two doubles representing the base and altitude of the triangle. The area of a triangle can be computed by $0.5 * b * a$ where `b` is the base and `a` is the altitude.
- A square description consists of the word `square` followed by a string specifying a color and one double representing the side of the square. The area of a square can be computed by $s * s$ where `s` is the side of the square.
- A rectangle description consists of the word `rectangle` followed by a string specifying a color and two doubles representing the length and width of the rectangle. The area of a rectangle is given by $len * wid$ where `len` is the length and `wid` is the width.
- A circle description consists of the word `circle` followed by a string specifying a color and one double representing the radius of the circle. The area of a circle can be computed by $Math.PI * r * r$ where `r` is the radius of the circle and `Math.PI` is a constant defined in the `java.lang.Math` class.

In addition to the `HW7` class, your program should contain a class `Shape` that represents a general shape, and more specialized classes named `Triangle`, `Rectangle`, `Square`, and `Circle`. The type hierarchy among these classes should be as shown below. Examine this figure and apply the "is a" test to confirm that the type hierarchy is correct (e.g., a square "is a" rectangle, so `Square` is a subclass of `Rectangle`.)



Represent each shape property in the highest applicable class in the type hierarchy. For example, the `color` property should be represented as an instance variable in the `Shape` class because every `Shape` has a color. The `Square` class should have no instance variables because a square is simply a rectangle whose height and width happen to be the same, so the side of the square can be represented by the `height` and `width` instance variables in the `Rectangle` class.

Write a constructor for each of your classes. Each constructor should take as many inputs as needed to construct an instance of the class. Each constructor should call the constructor of its superclass to initialize the properties of the superclass. The signatures of the constructors should be as follows:

```

Shape(String color)
Triangle(String color, double base, double altitude)
Rectangle(String color, double length, double width)
Square(String color, double side)
Circle(String color, double radius)
  
```

`Shape` should be an abstract class containing abstract methods named `area` and `toString` that take no parameters. The `area` method returns a `double` and the `toString` method returns a `String`. All the other classes in the class hierarchy should contain implementations of the `area` and `toString` methods. Therefore, it will not be possible to instantiate a `Shape`, but it will be possible to instantiate any of the other classes in the hierarchy. None of the classes in the `Shape` hierarchy should have a `main` method.

The `toString` methods of the `Triangle`, `Rectangle`, `Square`, and `Circle` classes should each return a `String` consisting of a color and shape. For example, the `toString` method of the `Square` class might return the `String` `red square` for an instance whose color is `red`. The `toString` method must be declared `public` in every class where it appears.

As in Homework 2, your program must read all the shapes from `System.in`, terminated by the word `end`, and then print the total area (sum of areas of all the shapes)

and a description of the largest shape. Do not prompt the user for input. Assume that `System.in` will be redirected to read from a file. As you read the description of each shape, call the constructor for that shape to construct an instance of the appropriate class.

In your main method, declare a variable named `shapes` to contain an `ArrayList` of shapes, as follows:

```
ArrayList<Shape> shapes = new ArrayList<Shape>();
```

Add each of your constructed objects to `shapes`, using the `add` method of the `ArrayList`. Note that, although `shapes` was declared as an `ArrayList` of `Shape` objects, each of its contents actually has a more specific type such as `Triangle` or `Square`.

After you have added all the input shapes to your `ArrayList`, iterate over the `ArrayList` to compute the areas of the various shapes and add them up. Print the sum of the areas of the shapes, and print a description of the largest shape using its `toString` method.

Your iteration should look like this:

```
// initialize variables as needed before the iteration
for (Shape s: shapes) {
    // your code goes here
}
```

The code inside your for-loop should not contain any reference to a specific class such as `Triangle` or `Square`. Therefore, if a new shape such as `Parallelogram` is added in the future, no change will be needed to this part of your program.

This program illustrates a technique called *polymorphism*, which is often used in object-oriented programming.

The following shows an example input and the output that your program should print when running on this input. Your output should conform to the format shown in this example. This is the same example input and output that was used in Homework 2. Your submitted program will be run on a different input stream to check its correctness. Your program must be able to accept any number of shapes, in any order.

Example input:

```
rectangle red 2.0 3.0
circle blue 1.5
square green 6.8
triangle yellow 4.0 2.5
rectangle orange 9.3 4.1
end
```

Expected output for this example input:

```
Total area = 102.43858347057704
Biggest shape is a green square
```

Submit one file named `HW7.java` to Moodle before the assignment deadline. You do not need to submit the output of your program. Each partner should submit a file, following Pair Programming Guidelines.