Classes and Objects I

1 `Rectangle.java` and `RectangleApp.java`

1.1 Download, compile and run

Download `RectangleApp.java` and `Rectangle.java` from the ELE. Save them in this workshop's folder.

First, compile it with command:

>> `javac RectangleApp.java` **RETURN**

If successful, you will see two `class` files (`Rectangle.class` and `RectangleApp.class`) are generated, which means you don't need compile the `Rectangle.java` manually.

Then, run it with command:

>> `java RectangleApp` **RETURN**

Read through the codes. *Make sure you understand how the program is working.*

1.2 The `this` keyword

- In the `Rectangle` class, change the second constructor's arguments as follows:

  ```
  public Rectangle(double width, double height)
  ```

  What changes should you make for the constructor's body to set the local object attributes.

Hints

Use the `this` keyword to resolve the conflict between instance attributes and constructors' arguments.

- In the third constructor's body, currently the attributes (`width` and `height`) are initialized with two assignment statements:

```
width = w;
height = h;
```

Could you replace the above statements with one statement which calls the second constructor?

Hints

```
this(w,h);//call the second constructor
```

Recompile it and make sure it is compiled successfully.

1.3 More methods

- Add two methods, both are for zooming the rectangle, therefore, they share the same method name (`zoom`) but with different arguments, which is known as *method overloading*.
  - The rectangle's width and height are zoomed with the same factor, i.e., only one argument for the `zoom()` method.
  - The rectangle's width and height are zoomed with different factors, i.e., two arguments are needed for the `zoom()` method.

  In the `RectangleApp` class's `main()` method, create one more rectangle (name it `rect1`) by calling one of the three constructors with any initial values. Then call the two methods you have created, test if the two methods are correct or not.

- Add one method for determining if two rectangles are overlapped or not:

```
public boolean isOverlappedWith(Rectangle r){...}
```

As discussed in the lecture, this must be an instance method, as we need create one rectangle instance to call this method whose argument is another rectangle instance. In the `RectangleApp` class's `main()` method, test if the method is correct or not by using the two existing rectangle objects (`myRect` and `rect1`). You may either

```
boolean b1 = myRect.isOverlappedWith(rect1);
```

Or

```
boolean b2 = rect1.isOverlappedWith(myRect);
```

The output values (`b1` and `b2`) for the above two statements must always equal, otherwise, your method must be wrong.

- Add one method `calcRatio()` for calculating the ratio of width to height.
- Add one method `isSquare()` for determining if the rectangle is square or not.

  Note: use the correct way of testing floating-point numbers for equality, as practiced in the first workshop.
- Can you think about any other behaviors/methods a rectangle has?

Hints

For example, compare two rectangles' size, move the rectangle horizontally or vertically, compute the diagonal, permute the width and height, etc.

Instead of implementing all these methods now, let's move on to the next task.

1.4 The access modifier

As mentioned in the lecture, a *well-encapsulated* class always hide their attributes to avoid the object's state (or attributes) been directly changed outside of the class. To achieve this principle, we need set all the instance attributes `private`. Then provide `public` *setter* and *getter* methods to modify and view the attributes.

Now change the `Rectangle` class to a *well-encapsulated* class. Compile the `Rectangle.java` file first, make sure it is all correct.

Hints

```
private double width; // private attribute
public double getWidth(){ // getter method
  return width;
}
public void setWidth(double width){// setter method
  this.width = width;
}
// do similar changes for the other attributes
...
```

Now if you compile the `RectanleApp.java`, you must get a few compiling errors because of accessing private attributes. You need make corresponding changes to solve the issue.

Hints

For example, change the `myRect.originX` to `myRect.getOrginX()`.

Recompile and rerun the program.

1.5 Object reference

Now create three rectangles in the `main()` method as follows.

```
Rectangle rect2 = new Rectangle(10.0,5.0);
Rectangle rect3 = new Rectangle(10.0,5.0);
Rectangle rect4 = rect3;
```

Then use `println()` to print the three rectangle's 'value'.

```
System.out.println("rect2: " + rect2);
System.out.println("rect3: " + rect3);
System.out.println("rect4: " + rect4);
```

When run the program, you will see the output like this: `Rectangle@7852e922`, which is the class name together with the hashcode of the object in hexadecimal (you may understand it as the memory address for each object). You may see the hashcode of `rect2` and `rect3` are different, because whenever you `new` an object, the compiler will allocate a new block of memory for it.

While you must have seen that `rect3` and `rect4` have the same hash code, because these two objects point to the same memory address. This means that whenever you change any attribute's value for one object, the other object's attribute value also changes. In another word, you may treat them as one object, but with two names (`rect3` and `rect4`). Try the following statements yourself.

```
rect3.zoom(0.5);
System.out.println("rect3's width: " + rect3.getWidth());
System.out.println("rect4's width: " + rect4.getWidth());
```

Re-run it, you must see both `rect3` and `rect4` have been zoomed into half size.

1.6 `toString()` method

Every well-designed Java class should contain a `public` method called `tostring()` that returns a short description of the instance (in a return type of `String`). The `toString()` method can be called explicitly (via `objectName.toString()`) just like any other method; or implicitly through `println()` or `print()`. If an instance is passed to the `println(objectName)` method, the `toString()` method of that instance will be invoked implicitly. Now, add the following `toString()` method to the `Rectangle` class:

```
// Return a description of a rectangle in the form of
// Rectangle[x=*,y=*,w=*,h=*]
public String toString(){
  return
       "Rectangle[x="+originX+",y="+originY+",w="+width+",h="+height+"]";
}
```

Re-compile and re-run `RectangleApp.java`, you must have seen the outputs of the following statements are different.

```
System.out.println("rect2: " + rect2);
System.out.println("rect3: " + rect3);
System.out.println("rect4: " + rect4);
```

This is because if a class doesn't have the `toString()` method, the compiler will output the default "className@hashcode"[1]. If a class has defined the `toString()` method, the compiler will output the form which you customize.

As metioned above, you may call `toString()` method explicitly, just like any other methods:

```
System.out.println("rect2: " + rect2.toString());
```

2 `Circle.java` and `CircleApp.java`

We have learnt how to calculate areas and circumference of a circle by putting everything in the `main()` method in the `CircleComputation.java` file. Now let's re-implement it in the object-oriented way. Similar to what you have done with the rectangles, you need first define a `Circle` class to encapsulate all relevant attributes and methods about a circle. Then add a test class `CircleApp` with a `main()` method to create circles and call all circles' methods.

You may consider the following attributes, methods and constructors in the `Circle` class.

- Attributes: (all set to be '`private`')
  - radius
  - originX
  - originY
- Constructors:
  - no-arg constructor
  - a constructor with one argument (radius)
  - a constructor with three arguments (radius and originX, originY)
- Methods:
  - Set the radius
  - Set the origin
  - Get the radius
  - Get the originX

[1]For detailed reasons, we will explain it later in this module when we learn inheritance.

- Get the originY
- Compute the area [2]
- Compute the circumference.
- Move the circle.
- Describe the basic information of a circle by defining the `toString()` method.
- Zoom the circle by a factor.
- Determine if a circle is overlapped with another circle. There are two options. One is to define an instance method:

  ```
  public boolean isOverlappedWith(Circle c)
  ```

  The other is to define a static method:

  ```
  public static boolean isOverlapped(Circle c1, Circle c2)
  ```

  Since you have practiced the instance method in `Rectangle` class, here choose the static method to implement. If any problem, ask the assistant for help.
- Determine if one circle is inside another circle. This method could also be implemented with either an instance method or a static method. Choose one you prefer to implement.

Then compile it with command:

>> `javac Circle.java` **RETURN**

If successful, you will see a `class` files (`Circle.class`). At the moment the byte codes is not executable, as there is no `main()` method.

Now create an application (with `main()` method) `CircleApp.java` to test your `Circle` class.

In the `main()` method, create a few circles with different attributes. Test all the methods you have created. Make sure all work as they are supposed to before moving to the next task.

3 ShapeApp.java

Create a new application with name `ShapeApp.java`. This application has one character input argument. The execution command is like this:

>> `java ShapeApp c` **RETURN**

You need first check if there is one and only one arguement as follows in the `main()` method.

[2] You may need to use `PI`, instead of defining yourself, use the Java provided static attribute `Math.PI` instead.

```
if (args.length!=1){
  System.out.println("One and only one argument needed!");
  System.exit(0);
}
```

To convert the String arguement into a character, use the follow statement:

```
char c = args[0].charAt(0);
```

This application works like this:

- If the input is 'C' or 'c', create two circles. One circle shrinks with a random factor[3] (between 0.5 and 1), the other circle expands with a random factor (between 1.0 and 2.0). This process repeats until one circle is inside the other.
- If the input is 'R' or 'r', create two rectangles. One rectangle moves randomly (the maximum distance it moves on x or y axis is 5.0), the other zooms randomly (the zoom factor for each axis is between 0.5 and 2.0). When zooming the rectangle, also need make sure the ratio of width to height is within [1/4,4]. This process repeats until the two rectangles overlap.
- Otherwise, print on the screen a message: 'Only support four characters ('r','R','c','C').'.

For each iteration, print out on the screen the two objects' information, which could help you to check if the program runs correct or not.

Hints

As `nextDouble()` method in the `Random` class only generates a random number between 0 and 1. To generate a random number between `rangeMin` and `rangeMax`, see the following:

```
import java.util.Random;//import it at the beginning of the program
...
Random r = new Random();
double x = rangeMin + (rangeMax - rangeMin) * r.nextDouble();
```

[3]You may use either the `Random` class's `nextDouble()` method or the `Math.random()` to generate random numbers.