

# **CST8132: Object-Oriented Programming (in Java)**

## **Midterm Test: Part B**

**Friday April 11, 2014**

**Course Professor: Rex Woollard**

1. This midterm test will be conducted during the regular 2-hour lecture.
2. You will have no aids in completing this test.
3. The midterm is split into two parts: A and B.
4. Part A is a mix of multiple choice and short answer. All answers are to be entered on a the test paper. Part A will be worth 21 marks.
5. Part B involves writing Java program code to solve a problem. Part B will be worth 24 marks: program code 20; and the memory map 4. Write your answer for Part B on separate blank sheets of paper. Your solution will be stapled to this test paper when complete.
6. You will begin with Part A. When you have completed Part A, submit it to me. At that point, you can choose to take an unsupervised break before receiving Part B.
7. You may separate the pages in the test. They will be re-stapled when you submit the test.

**Name:** \_\_\_\_\_

## The Program (4 marks)

You will write a working Java program to manage basic geometric shapes. Because of time constraints, you will only handle *circles* and *rectangles*.

Your program will ask a user to how many shapes to create. This will result in the creation of a suitably sized collection, either an *array* or an **ArrayList** (your choice). A loop will create that number of shapes. On each iteration through the creation loop, the user will be asked which shape to create: *circle* or *rectangle*. The user will then be asked to input valid data for those shapes. Of course, a *circle's* dimensioning is based on the *radius*. A *rectangle's* dimensioning is based on the *length* and *width*. These differences are a bit like the differences of *Hobbit* and *Wizard*, so you will need a method such as **inputAllFields()** to manage input.

A second loop will display all data regarding the shapes, including (and this is the important part) a calculation of the area of the shape. Of course, the calculation of a *circle's* area is different than the calculation for a *rectangle*:

- *circle*: The area calculation is  $PI * radius^2$  (and you can access the constant PI using **Math.PI**)
- *rectangle*: The area calculation is  $length * width$ .

A sample run of the program is shown at the top of the page. I placed boxed labels to show which part is *user input*. The user asked for 3 shapes, then created a *circle* followed by two *rectangles* (of course, a user could have specified any number of shapes and any sequence of *circle* and *rectangle* objects. Notice that the input prompts alert the user to the acceptable range of values. The display of these range values is automatically generated by the input methods in my **Input** class (documented on the next page).

Following the shape creation, the information about each of the shapes is output. Notice that the output differs depending on the type of the shape. This is similar to the way you have been managing your *Actor-Hobbit-Wizard* classes.

Finally, here are some general comments about your code style:

- You *do not need* to add any import statements.
- You *do not need* to add any comments.
- Your variable names, class names and method names should *adhere to the Java standard*.

## Output From Program Execution

```
Input number shapes: (1-10): 3 User Input

Choose 1:Circle 2:Rectangle (1-2): 1 User Input
Enter x: (0.0-1920.0): 100 User Input
Enter y: (0.0-1080.0): 200 User Input
Enter radius: (1.0-1000.0): 30 User Input

Choose 1:Circle 2:Rectangle (1-2): 2 User Input
Enter x: (0.0-1920.0): 52.2 User Input
Enter y: (0.0-1080.0): 49.3 User Input
Enter length: (1.0-1000.0): 30 User Input
Enter width: (1.0-1000.0): 6 User Input

Choose 1:Circle 2:Rectangle (1-2): 2 User Input
Enter x: (0.0-1920.0): 92.3 User Input
Enter y: (0.0-1080.0): 44.3 User Input
Enter length: (1.0-1000.0): 4 User Input
Enter width: (1.0-1000.0): 5 User Input

Calculating Area of Shapes
Circle: Location:[100.0:200.0]: Area:2827.43 radius:30.0
Rectangle: Location:[52.2:49.3]: Area:180.00 length:30.0 width:6.0
Rectangle: Location:[92.3:44.3]: Area:20.00 length:4.0 width:5.0
```

## Class Details: Shape

This is to be an **abstract** superclass.

It will have two **double** instance fields to manage the *x* and *y* location coordinates. This is because all Shape objects need to track their location.

You will need an **inputAllFields()** method to gather the user input, and a **toString()** method to support the display of output.

You will need an **abstract** method called **calcArea()**. This method is **abstract** because the **Shape** class cannot be used to define the method for calculating the area. The actual calculation must be left to the subclasses (since calculating the area of a *circle* is different from calculating the area of a *rectangle*). The **calcArea()** method will return the result of the calculation as a **double**.

## Class Details: Circle

The **Circle** class is a subclass of **Shape**. It will add **radius** as an instance field. It must also have suitable overridden methods for **inputAllFields()**, **toString()** and **calcArea()**.

## Class Details: Rectangle

The **Rectangle** class is a subclass of **Shape**. It will add **length** and **width** as instance fields. It too must have suitable overridden methods for **inputAllFields()**, **toString()** and **calcArea()**.

## Class Details: Test

You will use your **Test** class to test the behaviour of your **Shape**, **Circle** and **Rectangle** classes. When your program executes, it must generate the output that you see at the top of the page.

Notice that the user chooses the type of **Shape** at run time (either **Circle** or **Rectangle**)

You will need some kind of container to hold the variable number of **Shape** objects. You can use either an *array* or an **ArrayList**. The choice is yours. If you choose an array, you must ensure that you stay within the bounds of the array.

## Using my Input Class

You can assume that my **Input** class is already available. I have included the JavaDoc documentation for my **Input** class.

## Drawing the Memory Map (4 marks)

On a separate sheet of paper, draw a representative memory map that shows the organization of objects in memory. Your memory map must show the sample data that was entered during the sample program execution shown on the preceding page.

### Methods in Input Class

#### Field Summary

##### Fields

Modifier and Type	Field and Description
static Input	<b>instance</b> Keyword <i>static</i> makes this a <i>class</i> oriented variable, rather than <i>object</i> oriented variable; only one instance of an <i>Input</i> object will ever exist, and the instance is tracked by this reference variable.
static int	<b>MAX_ATTEMPTS</b> Each input method allows for a number failed attempts before throwing an exception.

#### Method Summary

##### Methods

Modifier and Type	Method and Description
boolean	<b>getBoolean</b> (java.lang.String prompt) Presents a prompt to the user and retrieves a <i>boolean</i> value.
double	<b>getDouble</b> (java.lang.String prompt) Presents a prompt to the user and retrieves an <i>double</i> value.
double	<b>getDouble</b> (java.lang.String prompt, double low, double high) Presents a prompt to the user and retrieves a <i>double</i> value which is within the range of <i>low</i> to <i>high</i> (inclusive).
int	<b>getInt</b> (java.lang.String prompt) Presents a prompt to the user and retrieves an <i>int</i> value.
int	<b>getInt</b> (java.lang.String prompt, int low, int high) Presents a prompt to the user and retrieves an <i>int</i> value which is within the range of <i>low</i> to <i>high</i> (inclusive).
java.lang.String	<b>getString</b> (java.lang.String prompt) Presents a prompt to the user and retrieves a <i>reference-to-String</i> .