



Joe Del Rocco
jdelrocco [at] stetson [dot] edu
Assistant Professor of Practice, Computer Science
Stetson University
421 N Woodland Blvd, DeLand, FL, 32723
www.stetson.edu

CSCI 141
(all sections)
Fall 2021
Assignment

Assignment 4

Due: Monday 10/25/2021 11:59pm

Contents

Program	2
(a) Review the provided source code	2
(b) Rectangle class	2
(c) Circle class	2
(d) fillShapes() method	2
(e) printShapes() method	3
(f) Finish up main()	3
Submission	3
Rubric	3
Example Output	4

Program

This program is meant to give you practice with basic Object-Oriented Programming (OOP). You will write 2 of your own classes, each of which represent a 2D shape and inherit some methods from a base class. You will then tie the program together with a `Main` class which has been partially written and provided to you. See the [Example Output](#) for examples of program runs.

Here is the link to the GitHub Classroom assignment:

<https://classroom.github.com/a/R3YyaSmB>

(a) Review the provided source code

Take a look at the 2 files that have been provided already, `Main.java` and `Shape.java`. The `Main` class has some comments that will guide you through coding that file. The `Shape` class will serve as an abstract base class for the 2 new classes you will create, `Rectangle` and `Circle`. **Do not make any changes to the file `Shape.java`.** In `Main.java`, notice that an array of `Shape` objects has already been provided, so you do not need to change that either. However, `Shape` itself is an abstract class (meaning we cannot make instance objects of it). This is an indication that you will need to create your own classes that extend (inherit from) `Shape`, and then you will make instance objects of those classes and put them into the `Shape` array provided. For example, when you create a class called `Rectangle` that extends `Shape`, then you can make instance objects of type `Rectangle` and put them in the `Shape` array because objects of type `Rectangle` are also of type `Shape`. As you've learned in lecture and in the book, this is called *polymorphism*.

(b) Rectangle class

Start by creating your `Rectangle` class in its own file, `Rectangle.java`. Your class should inherit from `Shape`. It should contain 4 private member variables, all of type `double`, representing: the top-left coordinate x value, the top-left coordinate y value, the width of the rectangle, and the height of the rectangle. It should also have a single constructor that takes 4 parameters of type `double` that should be assigned to the appropriate private member variables. If the width or the height passed in to the constructor is less than 1, then make it 1, because users of this class shouldn't be able to create `Rectangle` objects with a width or height of less than 1. By doing this, you are encapsulating the private member variables and protecting their values. We have also learned about accessor and mutator methods when talking about encapsulation, so go ahead and make accessor methods for each of the 4 private member variables. You do not need to provide mutator methods. Next, you must override the methods `area()` and `perimeter()` so they compute and return the proper computation per shape object. By overriding these base class methods from `Shape` in the child class `Rectangle`, you ensure that objects of this child class provide specific functionality per type (another advantage of polymorphism). See `Shape` for the method signatures. Finally, override the `toString()` method so it returns a `String` that describes the `Rectangle`, such as the word "Rectangle" plus the values of the top-left coordinate and the width and height. If you do not override `toString()`, then when you print a `Rectangle` object from `Main`, the output will not look like what you expect.

(c) Circle class

Create your `Circle` class in the same fashion as the `Rectangle` class, except storing the information needed for a circle (x, y, and radius).

(d) fillShapes() method

Once the classes `Rectangle` and `Circle` are defined properly, you can work on `Main`. Complete the `fillShapes()` method and call it from `main()` passing the array of shapes already created. This method should fill in the array that was passed to it. By fill in, we mean, if the user specified they wanted 3

shapes (code already provided in `main()`), then you need to actually create 3 object instances and put them in the array at indices 0, 1, and 2. Since the `fillShapes()` method is technically a procedure, you will fill the array in the method, not in `main()`. Iterate through the array parameter passed into the method, and for each index of the array, ask the user if it should be a `Circle` or not. If they want a circle, then ask them for the x, y and radius and create a `Circle` object and assign it into the array at the current index. If not, then do the same but with a `Rectangle` object. See the [Example Output](#) for an example of the input/output.

(e) `printShapes()` method

Complete the `printShapes()` method and call it from `main()` passing the array of shapes. This method simply iterates through the array passed in and prints out each index followed by the `Shape` object itself. If you have overridden the `toString()` method for each class properly, then the objects should print out as you expect. Use a definite loop.

(f) Finish up `main()`

Now finish the rest of the code in `Main`. Call the `menu()` method already provided (once). Then complete the code in the `while` loop. Each loop iteration, ask the user their menu selection, read in their choice, and handle it appropriately. If they indicate to EXIT, then break out of the loop. Otherwise ask the user which shape they are interested in, and read in the index as they specify. The user's input here is an integer representing the index of the shape they are interested in. Use a `switch` statement to handle the remaining choices, Area or Perimeter, each of which should call the `area()` or `perimeter()` methods of the shape at the specified index, respectively, and print out the result.

Submission

You will commit and push your changes to your specific GitHub Classroom repository for this assignment. Please follow the directions in this assignment to install all software. Commit and push your code changes early and often any time before the due date. Please see the advice below; it is important for grading purposes. **Failure to follow these directions will result in a loss of points.**

Always make sure to:

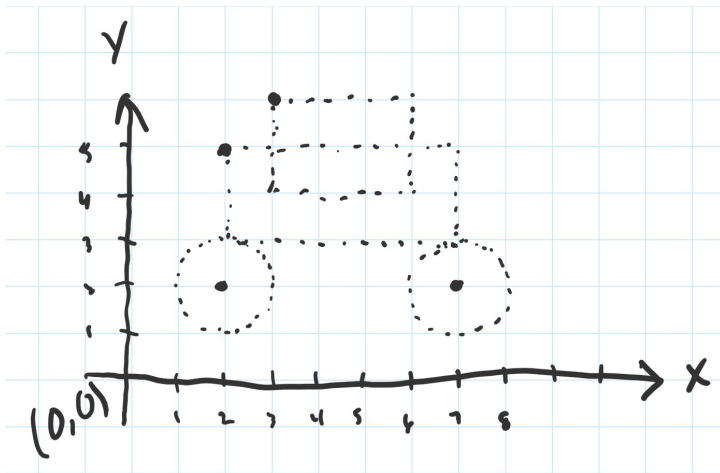
- Keep all source files in the folder called `src`, which is one directory in from the root of your repo
- Do not commit multiple copies of the same named source file; modify the ones provided to you. In other words, do not make an old and new version of the same file
- The main starting source file should always be called `Main`
- When loading resources, do not use absolute paths to files on your drive; [use relative paths](#)
- Do not have the keyword `package` at the top of any files. Some IDEs add your files to a custom package by default. Please remove this line, as it complicates grading.

Rubric

Task	Percentage
Following submission instructions	10%
General attempt at solving the problem	40%
Program compiles and runs	10%
Properly created Rectangle and Circle classes	20%
Properly created and called fillShapes and printShapes methods	20%
Total	100%

Example Output

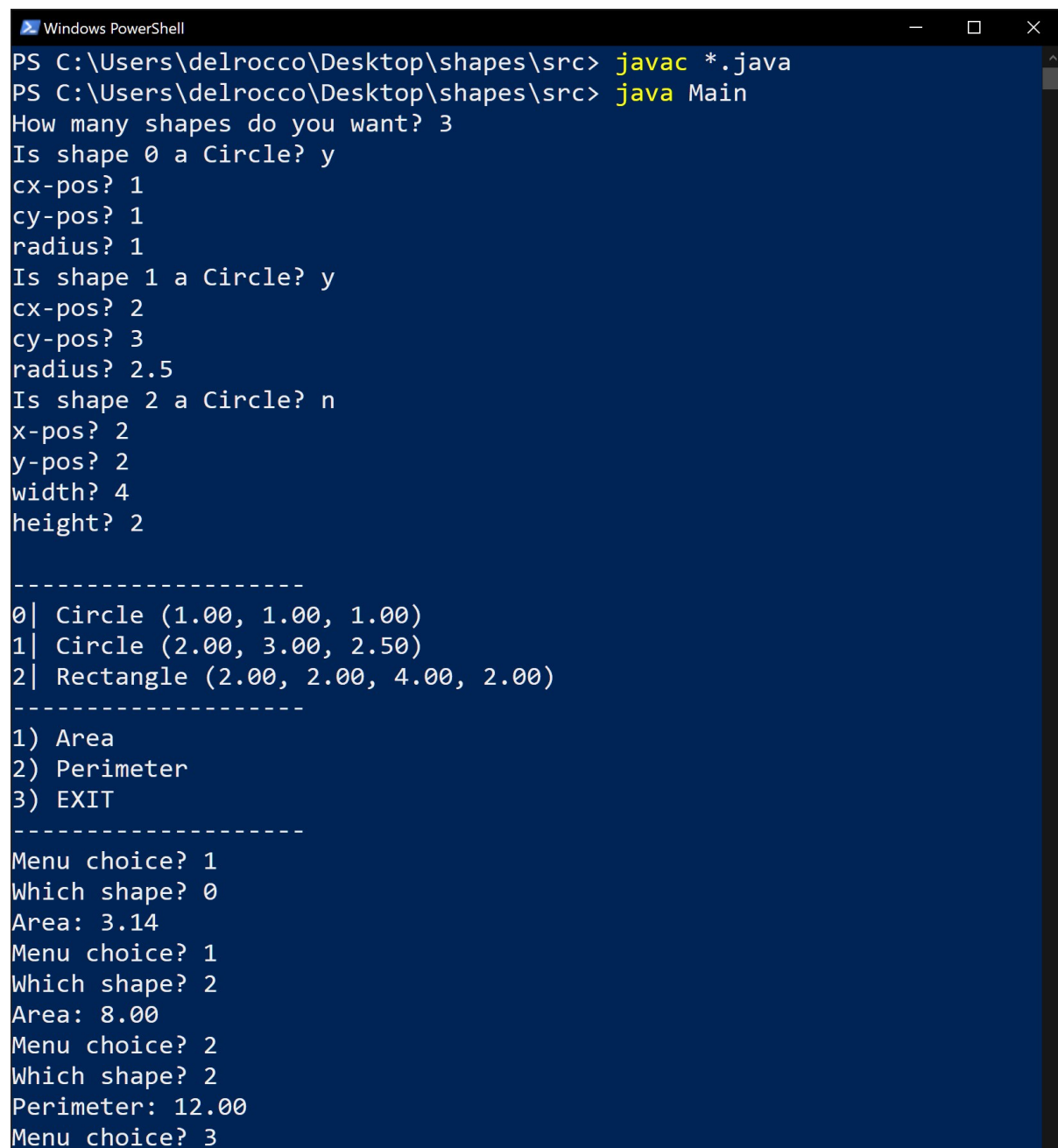
Here is an example of providing the following shapes located at the coordinates in this diagram. Notice that you cannot specify a negative radius for a `Circle` shape.



```
Windows PowerShell
PS C:\Users\delrocco\Desktop\shapes\src> java Main
How many shapes do you want? 4
Is shape 0 a Circle? y
cx-pos? 2
cy-pos? 2
radius? 1
Is shape 1 a Circle? y
cx-pos? 7
cy-pos? 2
radius? -5
Is shape 2 a Circle? n
x-pos? 2
y-pos? 5
width? 5
height? 2
Is shape 3 a Circle? n
x-pos? 3
y-pos? 6
width? 3
height? 2

-----
0| Circle (2.00, 2.00, 1.00)
1| Circle (7.00, 2.00, 1.00)
2| Rectangle (2.00, 5.00, 5.00, 2.00)
3| Rectangle (3.00, 6.00, 3.00, 2.00)
-----
1) Area
2) Perimeter
3) EXIT
-----
Menu choice? █
```

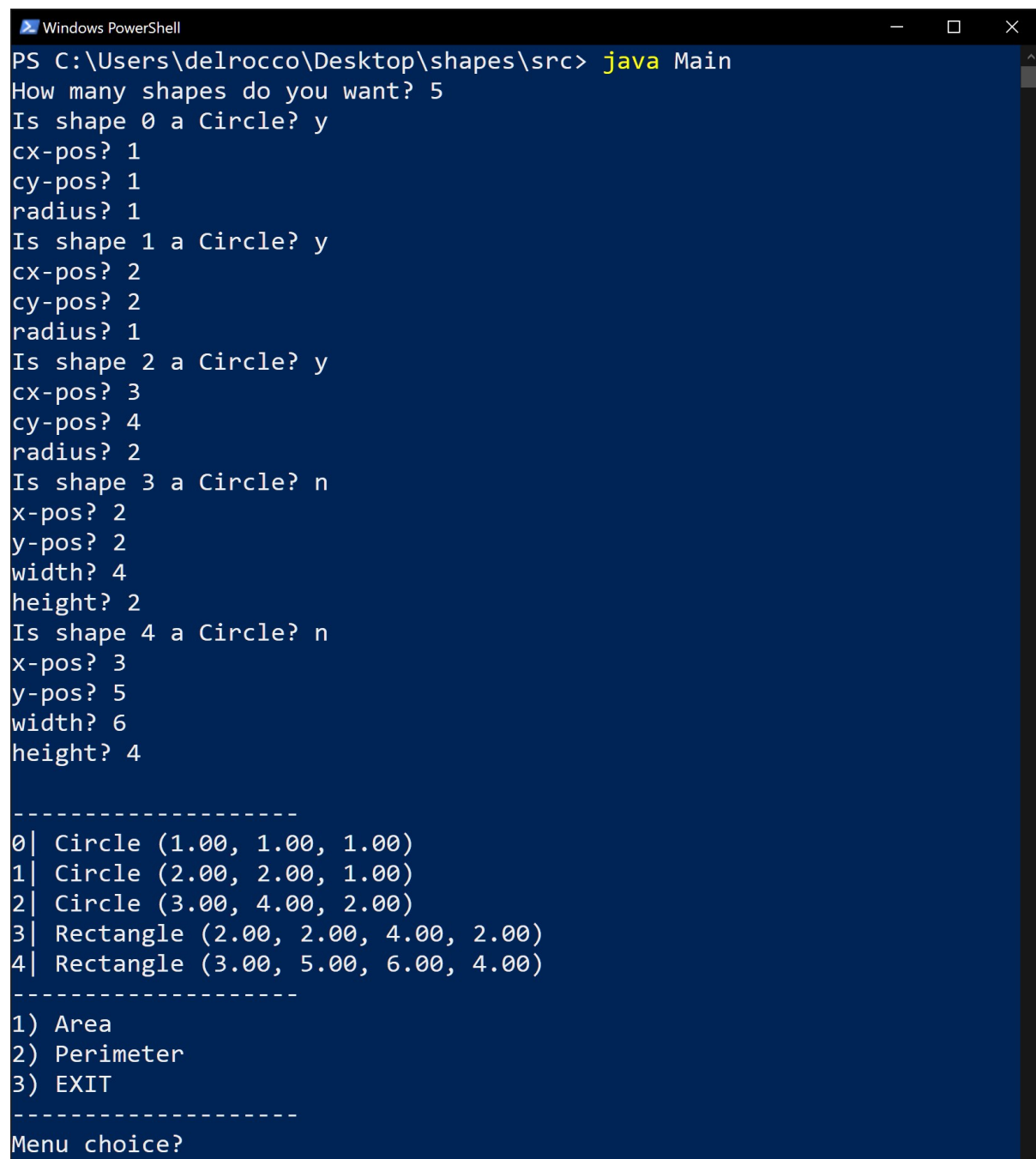
Another example of a run:



```
Windows PowerShell
PS C:\Users\delrocco\Desktop\shapes\src> javac *.java
PS C:\Users\delrocco\Desktop\shapes\src> java Main
How many shapes do you want? 3
Is shape 0 a Circle? y
cx-pos? 1
cy-pos? 1
radius? 1
Is shape 1 a Circle? y
cx-pos? 2
cy-pos? 3
radius? 2.5
Is shape 2 a Circle? n
x-pos? 2
y-pos? 2
width? 4
height? 2

-----
0| Circle (1.00, 1.00, 1.00)
1| Circle (2.00, 3.00, 2.50)
2| Rectangle (2.00, 2.00, 4.00, 2.00)
-----
1) Area
2) Perimeter
3) EXIT
-----
Menu choice? 1
Which shape? 0
Area: 3.14
Menu choice? 1
Which shape? 2
Area: 8.00
Menu choice? 2
Which shape? 2
Perimeter: 12.00
Menu choice? 3
```

Another example of a run:



```
Windows PowerShell
PS C:\Users\delrocco\Desktop\shapes\src> java Main
How many shapes do you want? 5
Is shape 0 a Circle? y
cx-pos? 1
cy-pos? 1
radius? 1
Is shape 1 a Circle? y
cx-pos? 2
cy-pos? 2
radius? 1
Is shape 2 a Circle? y
cx-pos? 3
cy-pos? 4
radius? 2
Is shape 3 a Circle? n
x-pos? 2
y-pos? 2
width? 4
height? 2
Is shape 4 a Circle? n
x-pos? 3
y-pos? 5
width? 6
height? 4

-----
0| Circle (1.00, 1.00, 1.00)
1| Circle (2.00, 2.00, 1.00)
2| Circle (3.00, 4.00, 2.00)
3| Rectangle (2.00, 2.00, 4.00, 2.00)
4| Rectangle (3.00, 5.00, 6.00, 4.00)
-----
1) Area
2) Perimeter
3) EXIT
-----
Menu choice?
```

continued below...

```
Windows PowerShell
-----
0| Circle (1.00, 1.00, 1.00)
1| Circle (2.00, 2.00, 1.00)
2| Circle (3.00, 4.00, 2.00)
3| Rectangle (2.00, 2.00, 4.00, 2.00)
4| Rectangle (3.00, 5.00, 6.00, 4.00)
-----
1) Area
2) Perimeter
3) EXIT
-----
Menu choice? 1
Which shape? 2
Area: 12.57
Menu choice? 2
Which shape? 2
Perimeter: 12.57
Menu choice? 2
Which shape? 3
Perimeter: 12.00
Menu choice? 2
Which shape? 4
Perimeter: 20.00
Menu choice? 1
Which shape? 4
Area: 24.00
Menu choice? 1
Which shape? 0
Area: 3.14
Menu choice? 2
Which shape? 0
Perimeter: 6.28
Menu choice? 3
```