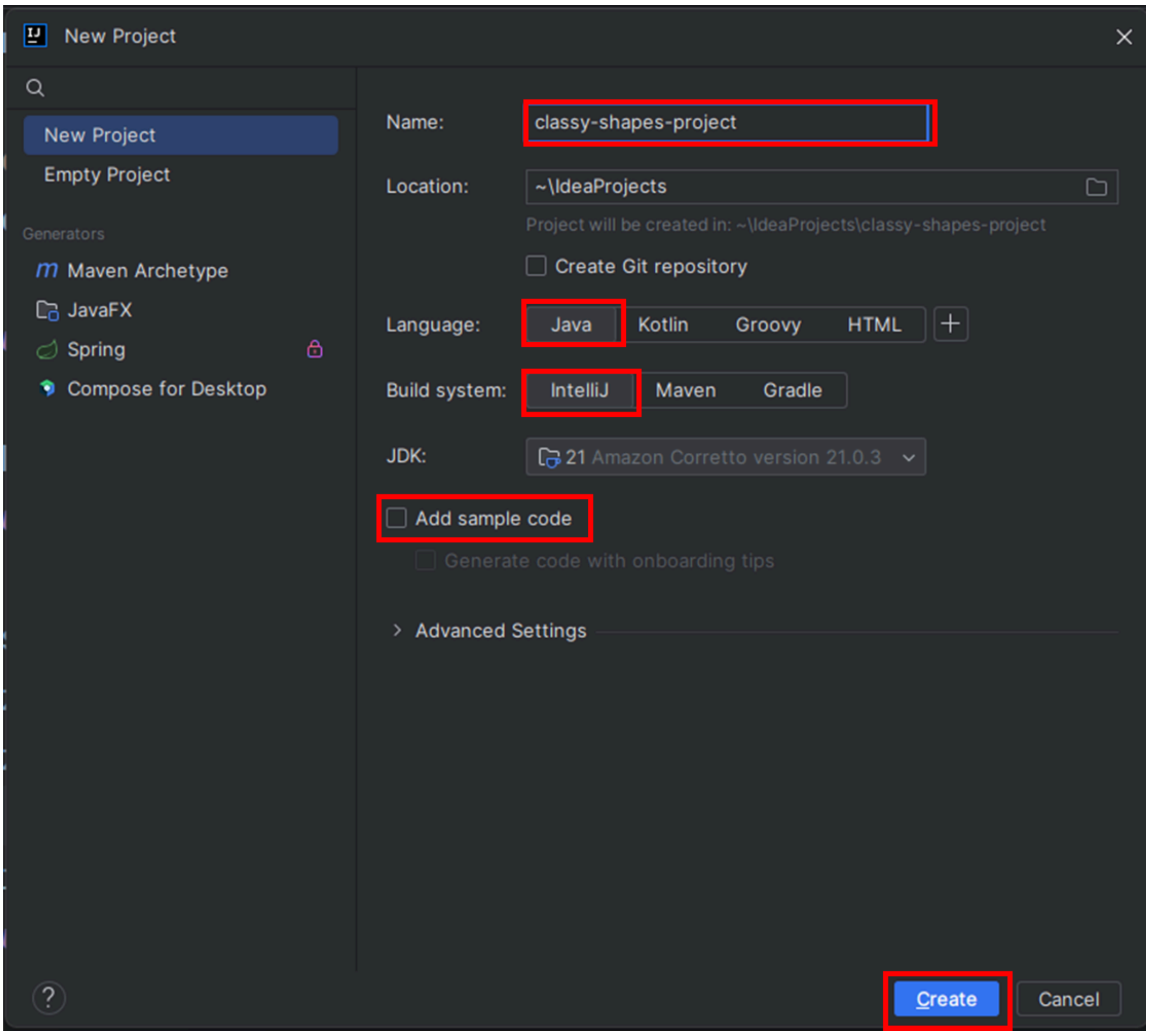
🖥️ It's time to get coding!

Step 1: Create a new project

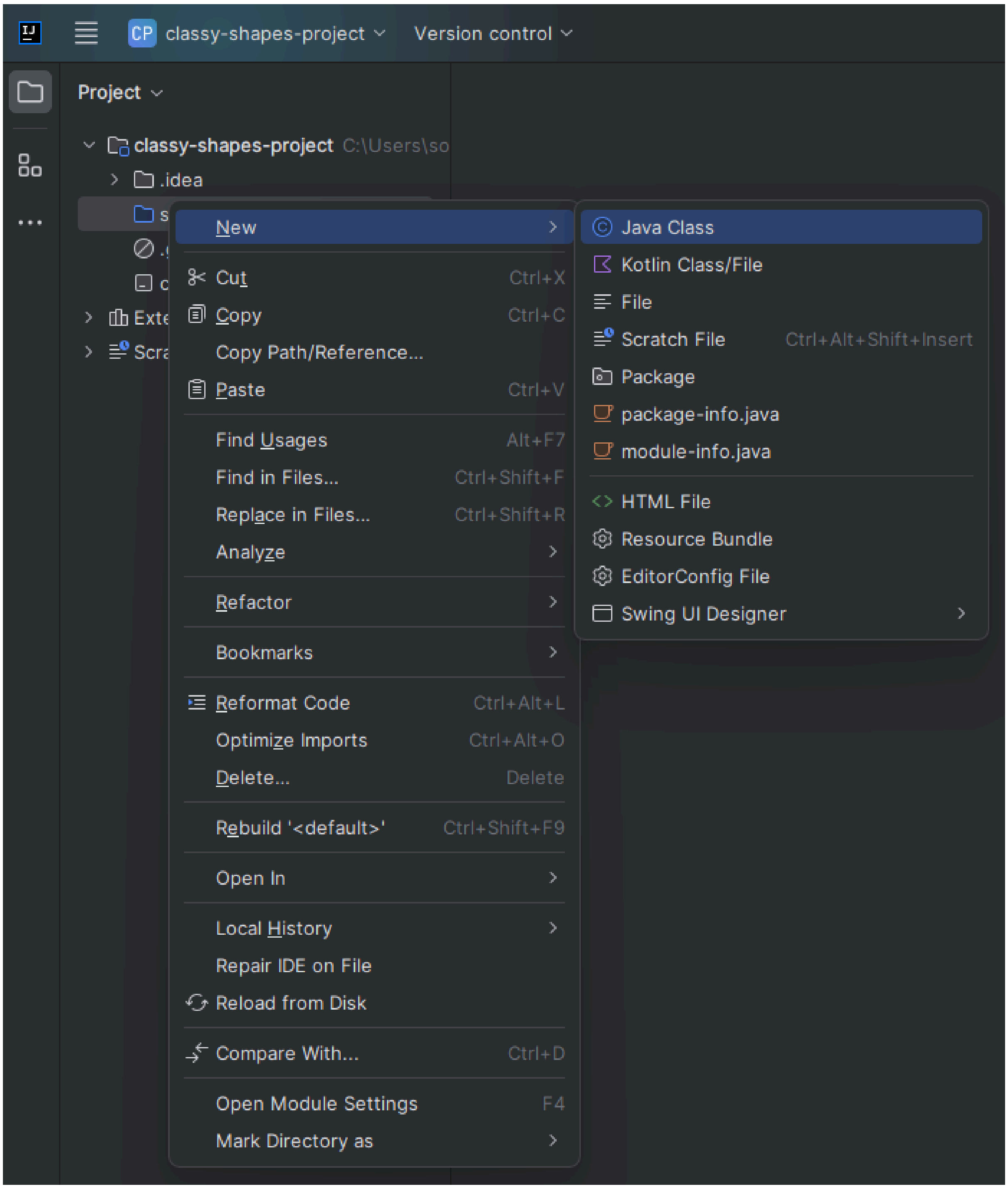
- 🖥️ TODO 1: Open IntelliJ and create a new Java Project.
  - Name your project as *classy-shapes-project*.
  - Select the Language as Java, and Build system as IntelliJ.
  - Untick Add sample code and click on Create.


New Project

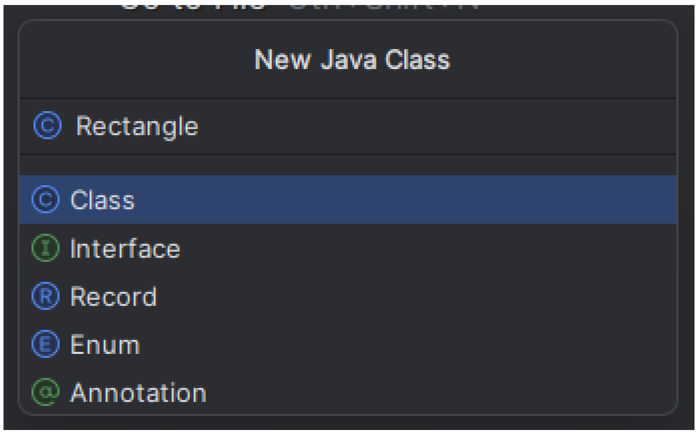
Name: classy-shapes-project
Location: ~\IdeaProjects
Project will be created in: ~\IdeaProjects\classy-shapes-project
[ ] Create Git repository
Language: [✓ Java] Kotlin Groovy HTML
Build system: [✓ IntelliJ] Maven Gradle
JDK: 21 Amazon Corretto version 21.0.3
[ ] Add sample code
[ ] Generate code with onboarding tips
Advanced Settings

Step 2: Create a class named Rectangle

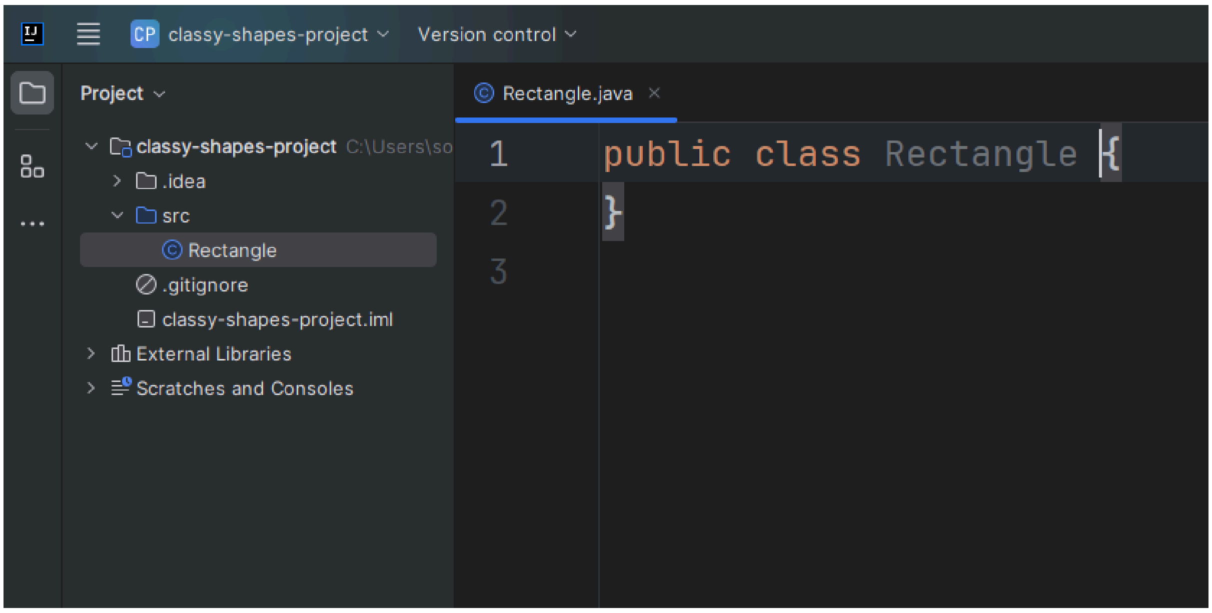
- 🖥️ TODO 2: When the project is created, right-click on the *src* folder, select New, and click on Java Class.



- 🖥️ TODO 3: In the pop-up window, notice that the Class option is preselected. You must only enter the name of your class, which is `Rectangle`.



- 🖥️ TODO 4: Press the Enter key, and a file named *Rectangle.java* is created containing the class.



```java
public class Rectangle {
}
```

Step 3: Add properties to the Rectangle class

The **`Rectangle`** class represents a rectangular shape; therefore, it will store information such as the length and width of the rectangle.

- 🖥️ TODO 5: Declare two member variables of **`double`** type:
  - **`length`**: Stores the length of the rectangle.
  - **`width`**: Stores the width of the rectangle.

```java
public class Rectangle {

    // member variables to store the length and width of a rectangle

    double length;

    double width;
}
```

Step 4: Add constructors to the Rectangle class

Remember, constructors help you to initialize the state of your objects, therefore you will define two constructors for creating the **`Rectangle`** objects.

- 🖥️ TODO 6: Firstly, a No-argument constructor that enables you to create a rectangle object without specifying an initial value for its properties. You can assign default values (for example, 1.0), within the constructor if needed.

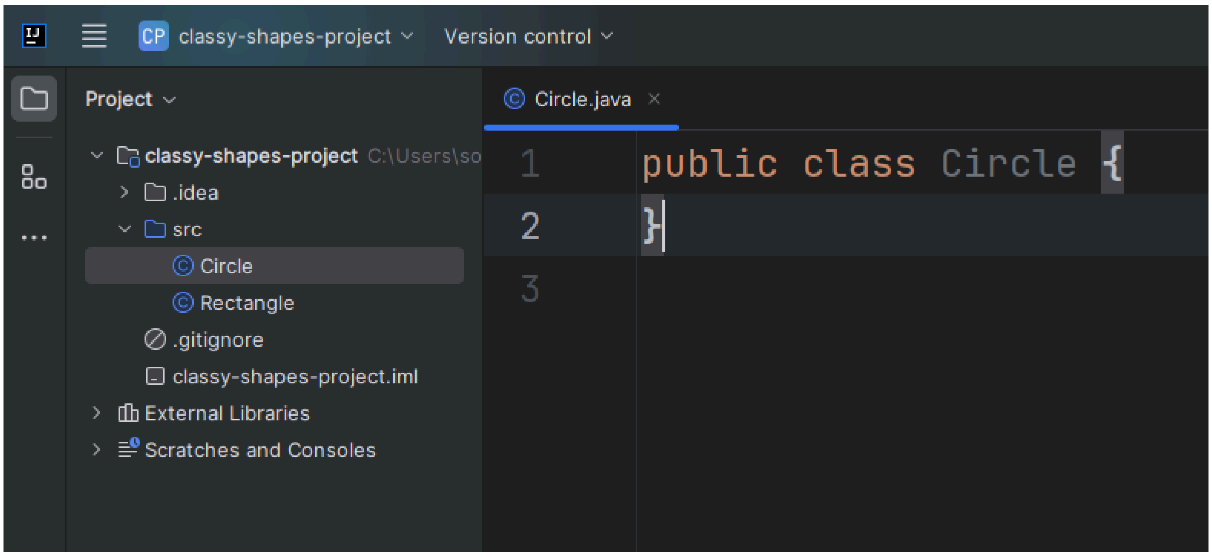
```
// No-argument constructor with default values

    public Rectangle() {

        this.length = 1.0;

        this.width = 1.0;

    }
```

- 🖥️ TODO 7: Secondly, a Parameterized constructor that helps you have control over the initial state of your objects. You can ensure that rectangles have valid dimensions when they are created. Place this one after your No-argument constructor.

```
// Parameterized constructor

    public Rectangle(double length, double width) {

        this.length = length;

        this.width = width;

    }
```

Step 5: Create one more class named Circle

- 🖥️ TODO 8: Create a New Java Class (as discussed above) named **`Circle`** to represent a circular shape.



```java
public class Circle {
}
```

Step 6: Add properties to the Circle class

- 🖥️ TODO 9: The Circle class represents a circular shape; therefore, it will store information such as the radius of the circle. Declare a member variable named radius of double type.

```java
public class Circle {

    // member variable

    // Your code here..

}
```

Step 7: Add constructors to the Circle class

- 🖥️ TODO 10: Define a No-argument constructor inside the **`Circle`** class like the **`Rectangle`** class, and assign a default value to **`radius`** (for example, 1.0)

```java
    public Circle() {

        // Your code here..

    }
```

- 🖥️ TODO 11: Now, define a Parameterized constructor.

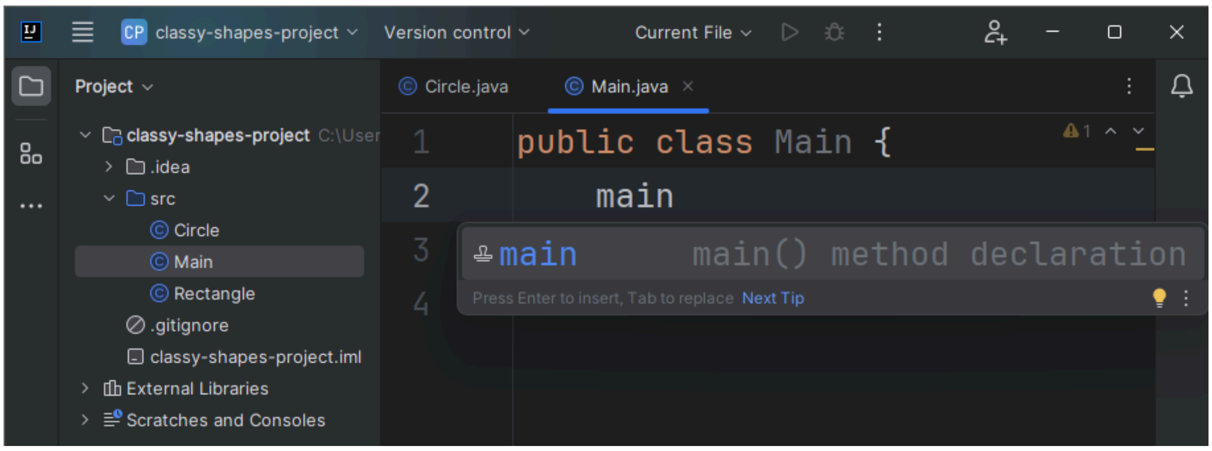
```java
Parameterized constructor

    public Circle(double radius) {

        this.radius = radius;
    }
```

Create Rectangle and Circle objects inside your main program to represent these shapes and provide values to their properties using their parameterized constructors, so you can leverage the flexibility of deciding the state of your object while it is created.

🖥️ It's time to get coding!

Step 1: Create the Main class and main method

- 🖥️ TODO 12: Create a New Java Class named **`Main`**.
  - Inside the **`Main`** class, type **`main`**, and IntelliJ will suggest completing the method signature.
  - Press Enter.



```
public class Main {
    main
```

main   main() method declaration

Press Enter to insert, Tab to replace Next Tip

Step 2: Create Rectangle and Circle objects

- 🖥️ TODO 13: Inside the **`main`** method, declare a Rectangle reference variable, named **`rectangle`**, and a Circle reference variable named **`circle`**.
  - Initialize these reference variables using the new keyword, calling their respective parameterized constructors, and passing specific values for their properties.

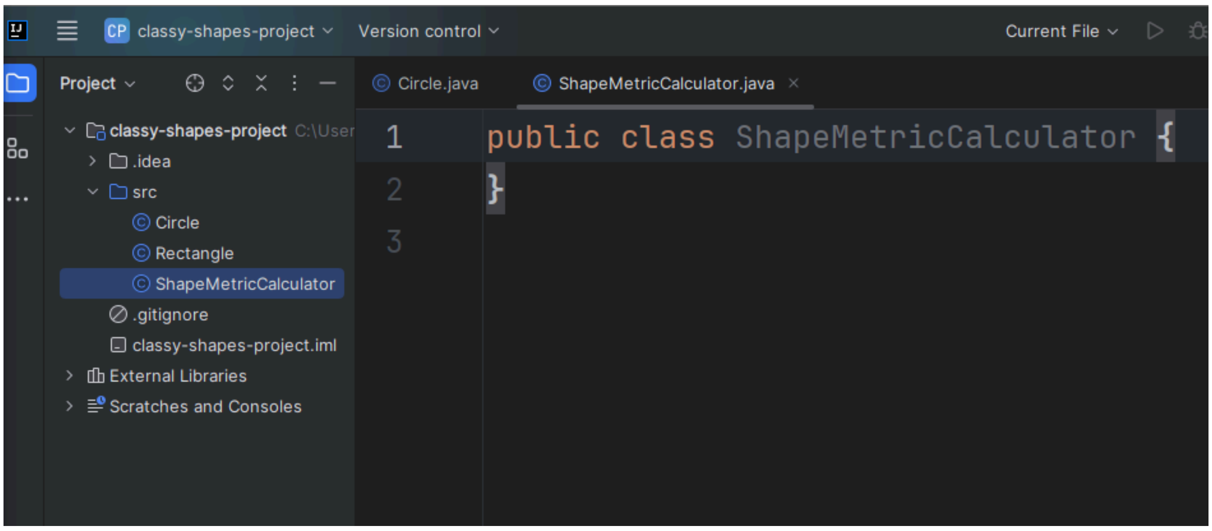
```
public class Main {

    public static void main(String[] args) {

        // Create a Rectangle object with specified length and width

        Rectangle rectangle = new Rectangle(5.0, 3.0);



        // Create a Circle object with specified radius
        // You code here..

    }

}
```

Creating a utility class in this lab will help you group all your calculation logic related to metrics of a shape (area or perimeter of a shape) under one roof. You can reuse the calculation logic for different shapes by creating a dedicated class.

🖥️ It's time to get coding!

Step 1: Create a class named ShapeMetricCalculator

- 🖥️ TODO 14: Create a New Java Class (as discussed above) and name it **`ShapeMetricCalculator`**. This class will contain methods for calculating shape properties like area in this lab (and maybe perimeter in the future).



```
public class ShapeMetricCalculator {
}
```

Step 2: Define the Area Calculation method for Rectangle objects

- 🖥️ TODO 15: Inside the **`ShapeMetricCalculator`** class, define a method named **`calculateRectangleArea`**.
    - Make this method **`static`** so it can be used by any class dealing with **`Rectangle`** objects to calculate their area.
    - The method's return type will be double as it will work with double values.
    - Now, because it is supposed to calculate the area of a rectangle, why not pass a **`Rectangle`** type of argument to it?

```
// method to calculate the area of a rectangle object

public static double calculateRectangleArea(Rectangle rectangle) {
```

```
            return rectangle.length * rectangle.width;

    }
```

Just like a method accepts arguments (variables) of type int, double, or String, it can also take a class type of argument. After all, the **`rectangle`** in the above code is a variable of type **`Rectangle`**, a class.

So, is the **`rectangle`** a reference variable? Yes!

Observe that the properties of the rectangle (length and width) are easily accessible through the **`rectangle`** variable.

Step 2: Define the Area Calculation method for Circle objects

- 🖥️ TODO 16: Inside the **`ShapeMetricCalculator`** class, define another **`static`** method named **`calculateCircleArea`**.
    - The method's return type will also be double as it will work with double values.
    - Now, because it is supposed to calculate the area of a circle, you must pass a **`Circle`** type of argument to it.

```
// method to calculate the area of a circle object

    public static double calculateCircleArea(Circle circle) {

        // Write your logic for calculating the area of a circle

      // return the area of the circle

    }
```

Your method already contains objects of **`Rectangle`** and **`Circle`** classes. You only need to call the appropriate method inside the main to calculate their area. Recall that both the methods inside your **`ShapeMetricCalculator`** class are **`static`**.

Now, **`static`** methods have a specialty associated with them, they do not require an object of the class to call them. They can be called using the **`dot`** operator with the name of the class that defines them. So, you need not create the object of **`ShapeMetricCalculator`**.

🖥️ It's time to get coding!

- 🖥️ TODO 17: Within the **`main`** method, call the **`calculateRectangleArea`** method, pass the **`rectangle`** reference as an argument, and store the result in a double type of variable.

```
public class Main {

    public static void main(String[] args) {

        // … (Rectangle and Circle objects defined as before)

        //Calculate the area of the rectangle, and store the result in a variable

        double areaOfRectangle =
ShapeMetricCalculator.calculateRectangleArea(rectangle);

    }

}
```

- 🖥️ TODO 18: Now, call the **`calculateCircleArea`** method, in the same way, and store the result in a double type of variable.

```
public class Main {

    public static void main(String[] args) {

        // … (Rectangle and Circle objects defined as before)

        // Call the method to calculate the area of the rectangle, and store the
result in a variable

        double areaOfRectangle =
ShapeMetricCalculator.calculateRectangleArea(rectangle);

        //Call the method to calculate the area of the circle, and store the result
in a variable

      // Your code here..

    }

}
```

🖥️ It's time to get coding!

- 🖥️ TODO 19: Print the respective areas of **`Rectangle`** and **`Circle`** objects using the print statements.

```
public class Main {

    public static void main(String[] args) {
        // … (Rectangle and Circle objects defined as before)

        // … (area calculation as before)

      // Print the results

        // Your code here..

    }

}
```

In this lab, you took a deep dive into the concepts of classes, objects, and methods in Java. You explored defining more than one class (**`Rectangle`**, and **`Circle`**), including a utility class (**`ShapeMetricCalculator`**), that focused on calculating areas of various shapes.

Your key learnings from this lab are as follows:

- You learned how to define classes with member variables and constructors.
- You explored creating objects from these classes and assigning values to their properties using parameterized constructors.
- You learned how to define common or shared functionality in a utility class using static methods.
- You practiced using methods to calculate the area of a **`Rectangle`** and **`Circle`** object.

This lab focused on **`Rectangle`** and **`Circle`** objects. For further exploration, you can extend it by creating a **`Triangle`** class and adding a respective method in the **`ShapeMetricCalculator`** to calculate its area.

By continuing to practice and explore, you will gain a deeper understanding of these concepts and their power in building a well-structured reusable code. Happy learning!
Tovább a következő utasításra
E