Lab 3

Objectives

- Introduction to interfaces
- Introduction to abstract data types

Part I - Interfaces

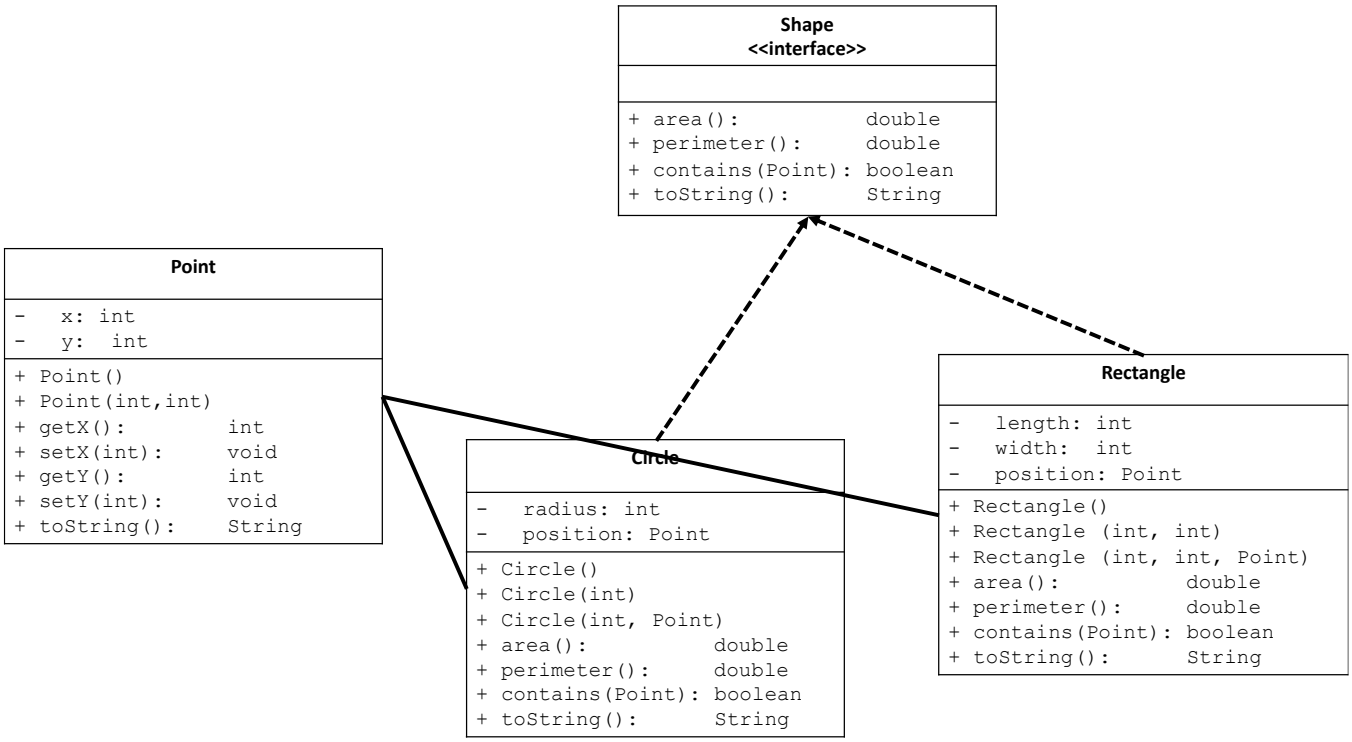
1. Download all of the provided lab files to your Lab3 working directory.

We are going to use an interface to specify what a Shape class should look like. As depicted in the UML diagram below, a Shape should have methods for calculating area, calculating perimeter and determining whether a given Point lies with in a Shape. An interface does not include the implementation of these methods, only their signatures and documentation. The implementation details of these method is dependent on the type of Shape and therefore will be defined in the classes that implement the Shape interface (ie. Circle, Rectangle)

2. Start by opening Shape.java to familiarize yourself with the method documentation.

The Circle and Rectangle classes *implement* the Shape interface (depicted by the **dashed arrow** between them in the UML diagram below)
Circle and Rectangle both have an attribute of type Point (depicted by the **solid line with no arrow** between them in the UML diagram below).

3. Open Circle.java and you will see a full implementation of the Circle class. Notice, it contains implementations of all methods listed in Shape.java



4. Begin the implementation of Rectangle.java
    a. Create a new file that will be named Rectangle.java
    b. Make the Rectangle class implement the Shape method. Look at the beginning of Circle.java if you are unsure how to do this.
    c. Try to compile Rectangle.java – You will see compile error something like this:

    error: Rectangle is not abstract and does not override abstract method contains(Point) in Shape

    Fix this compile error by introducing stubs for each of the methods Rectangle must implement:

    d. Copy the documentation and method signatures from Shape.java to Rectangle.java
    e. For each method signature:
        i. remove the **";"** from the end of the signature
        ii. add an **"{"** and **"}"** to make it a method
        iii. make the method public
        iv. if the method that has a non-void return type, add a dummy return statement

    For example, for the area method the stub would look like this:

```
public double area() {
        return 0;
}
```

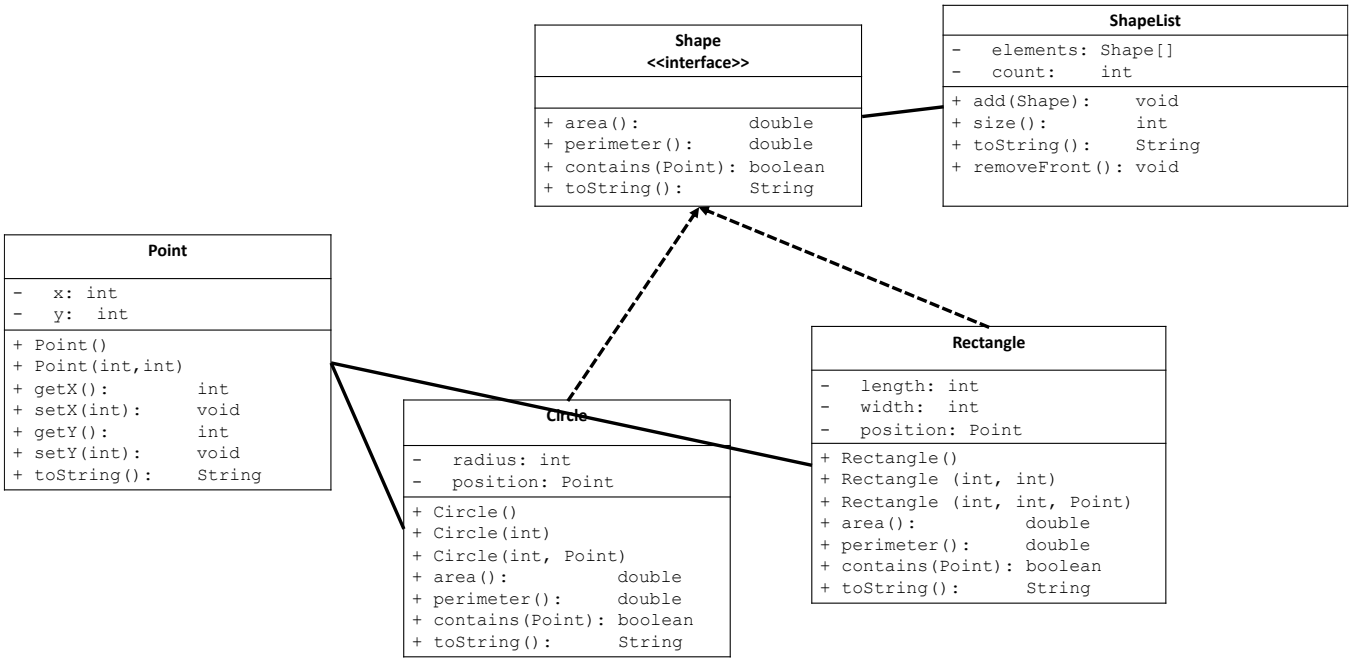
    f. In the UML diagram on the previous page, notice that the Rectangle class has three fields: length (which is type int), width (also type int), and position (type Point). Add these fields to the Rectangle class.
    g. The UML diagram on the previous page specifies three constructors for the Rectangle class. Write the three constructors. **Note: For the Rectangle class, we will assume the point is in the bottom-left corner of the rectangle.** This is important for the contains method you will do later in the lab.

CHECKPOINT (Ungraded) – Now might be a good time to check-in with the TA if you are unclear of how to proceed, or if you are unable to fix the compile errors in your program.

5. Complete the implementation of Rectangle.java
    a. Open Lab3Tester.java and compile and run it. You will see tests failing.
    b. Implement one method at a time, recompiling and rerunning the tester after each one

Part II – Using classes to define a data structure

We are now going to implement a class called ShapeList that will hold a collection of Shapes. The addition of this class is depicted in our UML diagram:



1. Start by opening ShapeList.java Notice we have added the attributes, a blank constructor and method stubs for you
    a. Uncomment the call to testShapeList() in the main of Lab3Tester.java and compile/run it
    b. Implement the constructor and each method one at a time.
2. Add tests to testShapeList() to ensure your add method has the correct behaviour when more than 2 elements are added to the list.

SUBMISSION (Graded) – Submit the **ShapeList.java** and **Rectangle.java** files into the Lab3 submission page on ConneX.