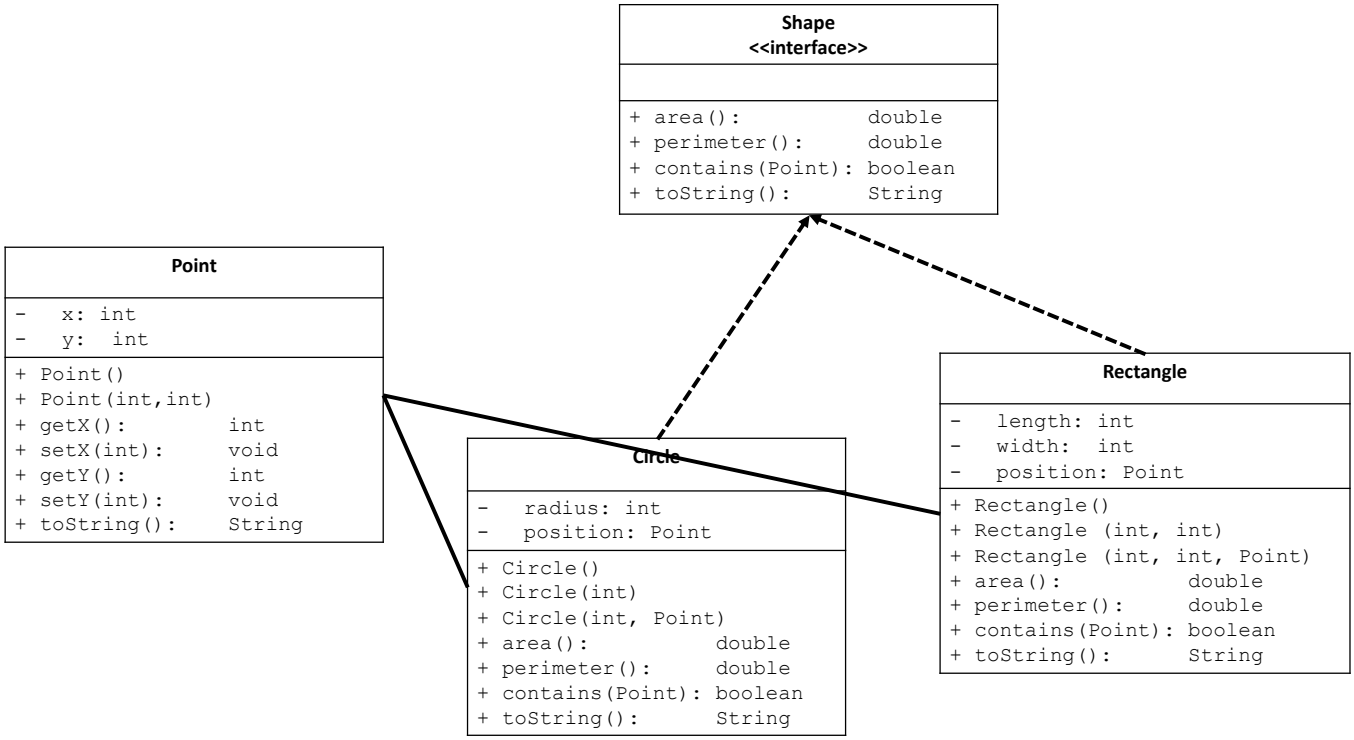
Lab 7

Objectives

- Introduction to inheritance and abstract classes

Part I - Inheritance

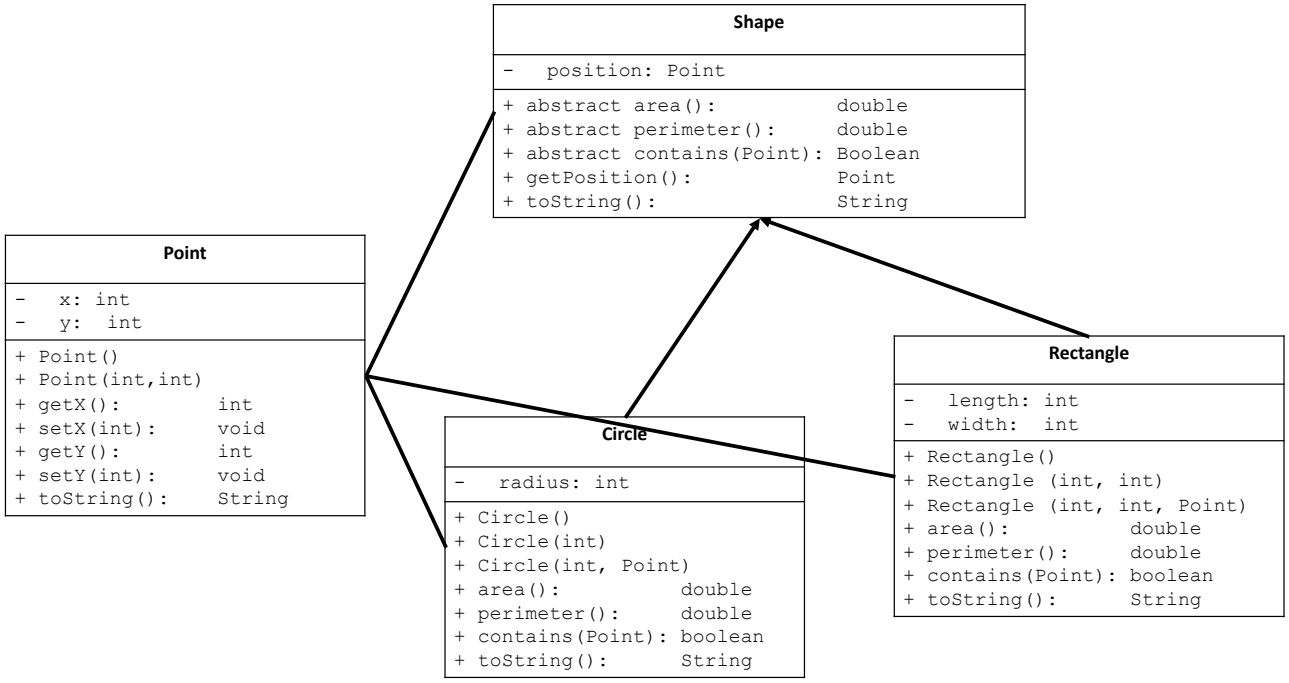
1. Download Shape.java, Rectangle.java, Circle.java and Point.java to your Lab7 working directory. What we have provided you with is the solution to Lab3, which is the implementation of the classes and interface depicted in the following UML diagram along with a tester.



2. You will be updating the files given to change the Shape interface to be an abstract class where we move the **private** point field from the Circle and Rectangle class to the abstract Shape class. This update will require you to make other changes to the Shape, Circle and Rectangle classes. The updated UML diagram is provided on the following page. You can attempt to make these updates based on the UML diagram. We have although provided some notes with suggested steps if you are stuck.
   TIP: You can compile a single class at a time to allow yourself to focus on one module's set of errors at a time. ie. `javac Rectangle.java`

   Suggested steps:
   - Make `Shape` an `abstract class` and having `Circle` and `Rectangle` `extend Shape`
   - Make the specified methods in `Shape` abstract (see the UML)
     - RECALL – this is very similar to the method prototype in an interface as there is no implementation – the implementation will be in the extending classes
   - Remove the `position` field from `Circle` and `Rectangle` and add it to `Shape`
     - Add the specified constructors (see the UML). The constructor with no arguments should initialize the position to a new `Point` at (0,0). The constructor with an argument should initialize the position to the given position

- You will no longer be able to access the `position` directly in the `Rectangle` and `Circle`
    - update the constructors in `Rectangle` and `Circle` so that the correct constructor in `Shape` is invoked when a new `Circle` or `Shape` is created
      Recall: constructor chaining forces the constructor of the super class with no arguments to execute before the current constructor executes
      to override this, use `super(…)` passing the required arguments to force the invocation of the super classes constructor with those arguments.
    - Add a `toString` method to `Shape` and edit the `toString` methods in `Circle` and `Rectangle` to ***refine*** that method
      Recall: Refinement means you use the keyword `super` to invoke the method in the super class
      ie. `super.foo()` would call a method named `foo` in the class this class is extending
    - You still cannot access the `position` in `Shape` and the `contains` method needs it.
        - Add a getter method for `position` to `Shape`
        - Update the `contains` method implementations in `Circle` and `Rectangle` to use the `position` getter method



CHECK POINT – get your lab TA to check off after you have completed this. They will want to see you compile and run `Lab7Tester.java` They will also want to see that you have not accessed the position field directly in Circle or Rectangle and that you are making use of super to invoke methods in the Shape class where needed.

3. Add a method called `moveBy` to the Shape class according to the following documentation. Make sure you test it!

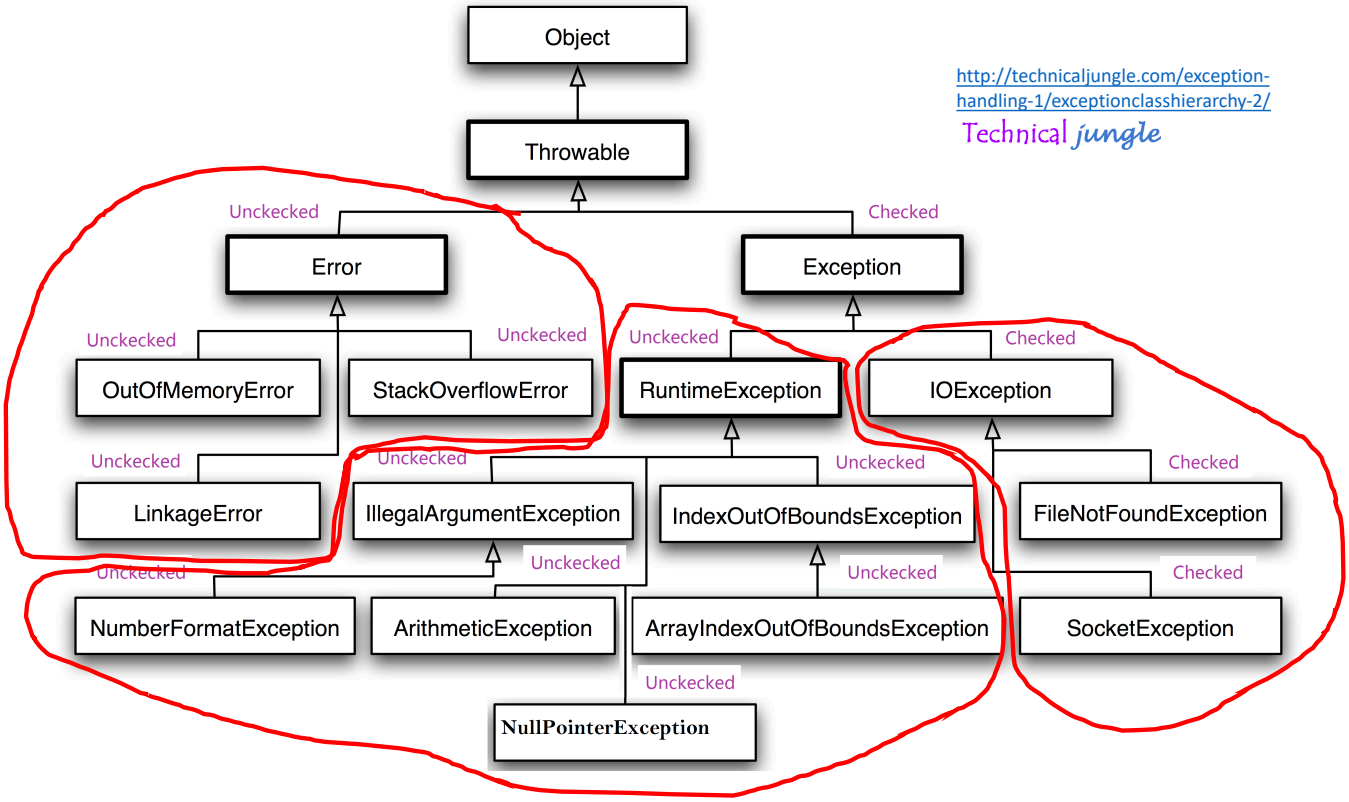
```
/*
 * Purpose: move the position of this Shape by
 *   xdir units in the x direction and ydir units in the y direction
 * Parameters: int xdir, int ydir - units to move in x and y directions
 * Returns: Nothing
 */
```

CHECK POINT – get your lab TA to check off after you have completed this. They will want to see the test you added for this method in `Lab7Tester.java`, see you run the tester and see your method implementation.

Part II – Exceptions

Now that you understand inheritance, we can look at the following class hierarchy diagram which illustrates the inheritance structure of Exceptions in Java.
Notice, the `Error` class along with any of its subclasses and the `RuntimeException` and any of its subclasses are ***unchecked*** exceptions.
All other classes that extend `Exception` are ***checked*** exceptions (those shown in the diagram and any you write that `extend Exception`)
Recall:

- Checked exception– the compiler checks to ensure the code calling a method that throws an exception is handling that exception
    - Declare the method will throw on the exception (lets the exception be thrown on)
    - Wraps the calling code in a try/catch block (catches the exception and deals with it)
- Unchecked exception – the compiler does not check to ensure the code handles the exception



1. Download `RuntimeExceptionExercise.java` to your Lab7 working directory.
   Work through the 5 questions in this file labeled Q1, Q2, etc.

2. Download `CheckedExceptionExercise.java`, `TooSmallException.java` and `TooBigException.java` to your Lab7 working directory.
   Work through the 3 questions in this file labeled Q1, Q2, etc.

CHECK POINT – get your lab TA to check off after you have completed this. They will want to see your hand traces on paper and that you have correctly introduced exception handling into `CheckedExceptionExercise`