General Instructions:

1. Follow the instructions and especially the naming standards.
2. Put only the **source files** in a folder named {**YOURID}_{FIRSTNAME}_{LASTNAME}_AS4** and compress it**. (**Use **.zip** format**)**
3. **Note:** replace your info, like **123456_ALI_ALILI_AS4.zip** be careful with the order(id, first name, last name).
4. You will lose points for not following the naming standard.
5. Be as clear and neat as possible when you write codes. Use naming conventions and indentations properly.
6. **Neither plagiarism nor any type of cheating will be tolerated!**

Assignment 4

Now that you have already learned OOP principles and file input/output in Java. It is time to apply them together.

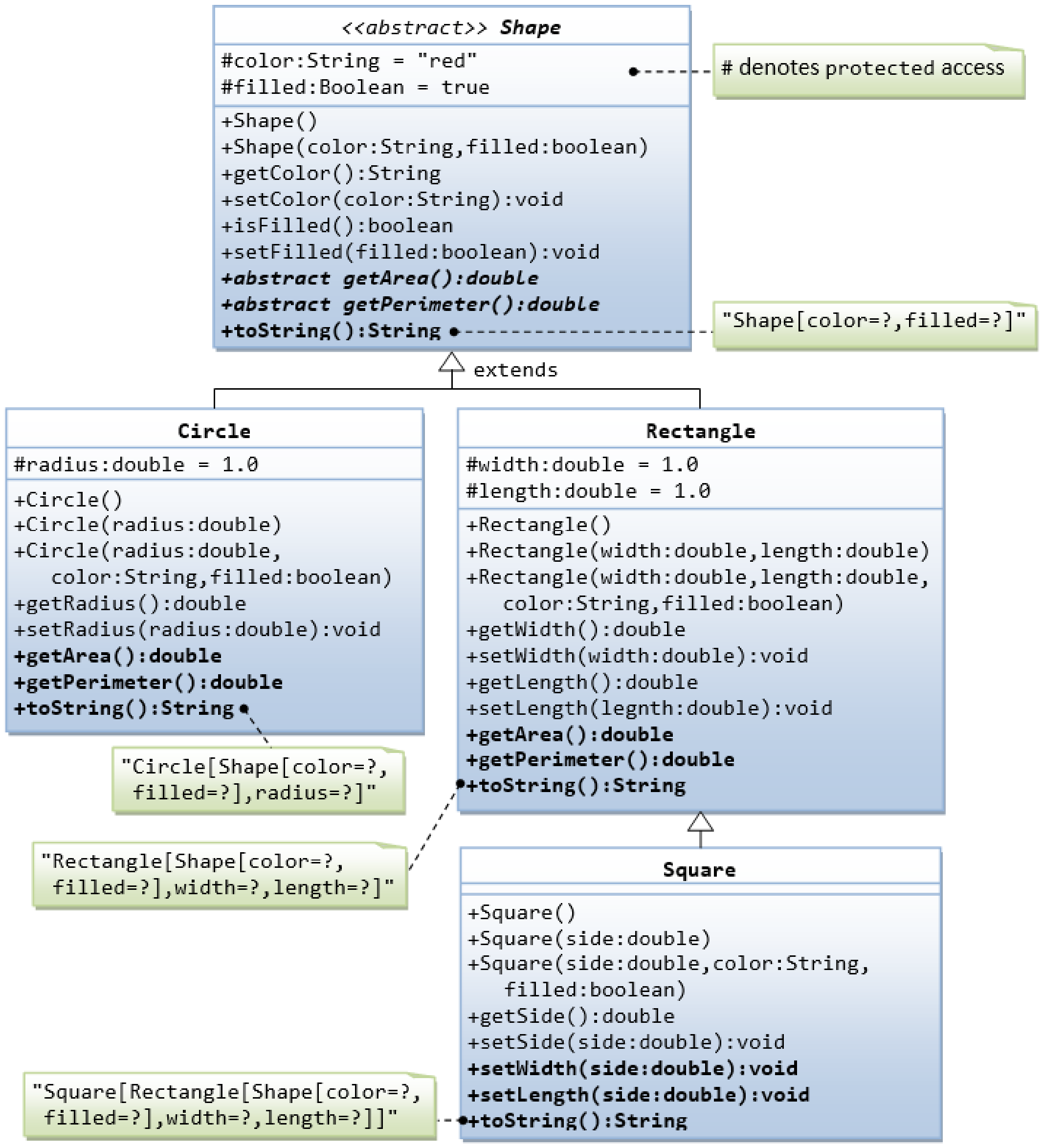
Part 1

Consider the following UML diagram and try to develop classes and check if they are working as expected from the Main class.
Recall that this UML is exactly from *Exercise6 Problem 10*. So you might have already designed and developed this.

Note: The classes designed in UML are already developed and shared with you. However, it is highly recommended to **try to develop on your own**. Feel free to refer to the source codes shared with you.

Note: In Main class, there are several questions for you to answer. They are not to submit but for you to understand if you know what exactly is happening. Fix errors, and run Main to see the output.

UML class diagram:



<<abstract>> Shape
#color:String = "red"
#filled:Boolean = true
+Shape()
+Shape(color:String,filled:boolean)
+getColor():String
+setColor(color:String):void
+isFilled():boolean
+setFilled(filled:boolean):void
+abstract getArea():double
+abstract getPerimeter():double
+toString():String
# denotes protected access
"Shape[color=?,filled=?]"
extends
Circle
#radius:double = 1.0
+Circle()
+Circle(radius:double)
+Circle(radius:double, color:String,filled:boolean)
+getRadius():double
+setRadius(radius:double):void
+getArea():double
+getPerimeter():double
+toString():String
"Circle[Shape[color=?, filled=?],radius=?]"
Rectangle
#width:double = 1.0
#length:double = 1.0
+Rectangle()
+Rectangle(width:double,length:double)
+Rectangle(width:double,length:double, color:String,filled:boolean)
+getWidth():double
+setWidth(width:double):void
+getLength():double
+setLength(legnth:double):void
+getArea():double
+getPerimeter():double
+toString():String
"Rectangle[Shape[color=?, filled=?],width=?,length=?]"
Square
+Square()
+Square(side:double)
+Square(side:double,color:String, filled:boolean)
+getSide():double
+setSide(side:double):void
+setWidth(side:double):void
+setLength(side:double):void
+toString():String
"Square[Rectangle[Shape[color=?, filled=?],width=?,length=?]]"

Source: https://www3.ntu.edu.sg/home/ehchua/programming/java/J3f_OOPExercises.html

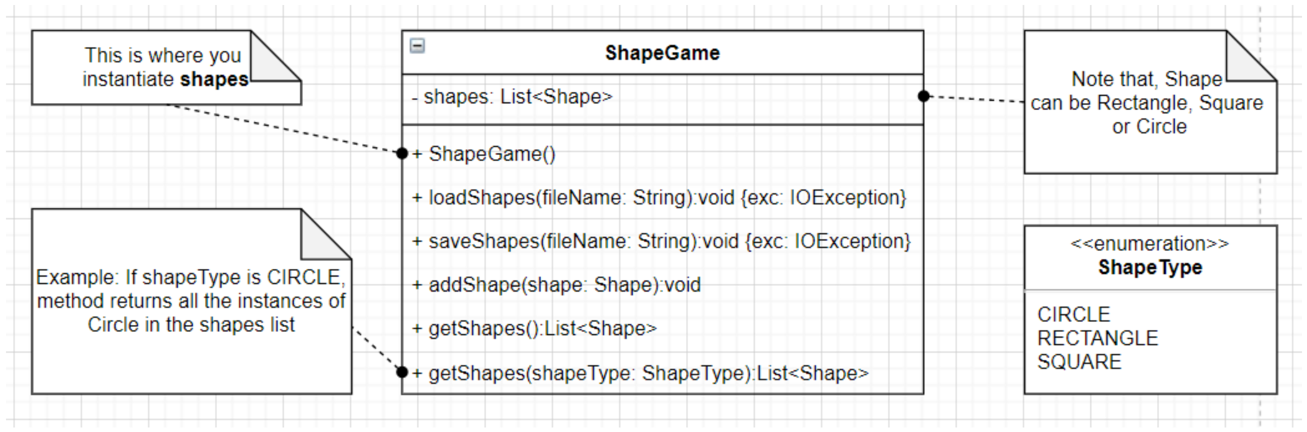
Note: You might want to use the main class example in the link above.

Part 2

1. Define a new interface called *ParsableEntity*. Add the following methods to it:
    a. **String parseToSaveString()**
    b. **void parseFromSaveString(String line)**
2. Make *Shape* implement *ParsableEntity* interface without implementing its abstract methods.
3. Each class (Circle, Rectangle and Square) can be parsed to and from string differently. So see the following examples:
    a. *Circle string:* Circle;blue;false;5.0
    b. *Rectangle string:* Rectangle;black;true;10.0;20.0
    c. *Square string:* Square;black;false;10.0
4. Implement and define the **String parseToSaveString()** method so that it can return one of the above strings depending on which class you define it within.
5. Similarly, define the **void parseFromSaveString(String line)** method so that it can take one of the strings above (you will need to decide later which string will be sent to which class instance).
    a. Example: if you have a Square class instance **sq**, calling the **sq.parseFromSaveString("**Square;black;false;10.0**")** will end up setting the sq fields as following:
        i. color <- "black"
        ii. filled <- false (be careful with the data type: Boolean not String)
        iii. side <- 10.0 (since there is no field named **side** set width and **length** of the **parent** OR call **setSide()** method).
6. In the Main, test if these functions of all three classes work properly. Remember, they are instance methods.

Part 3

1. Define a class named **ShapeGame**.


This is where you instantiate **shapes**
ShapeGame
- shapes: List<Shape>
+ ShapeGame()
+ loadShapes(fileName: String):void {exc: IOException}
+ saveShapes(fileName: String):void {exc: IOException}
+ addShape(shape: Shape):void
+ getShapes():List<Shape>
+ getShapes(shapeType: ShapeType):List<Shape>
Note that, Shape can be Rectangle, Square or Circle
Example: If shapeType is CIRCLE, method returns all the instances of Circle in the shapes list
<<enumeration>>
ShapeType
CIRCLE
RECTANGLE
SQUARE

2. Loading and saving is basic file input and output operations. Take a file name (or path) as input and
    a. either save the *shapes* (List of shapes of any type: Circle, Square, Rectangle) into a file
    b. or read the file and load each line as a new shape into the *shapes* list. (See **shapes.txt** for an example)
3. **addShape(shape:Shape)** method is to add any shape to the list of *shapes*.
4. **getShapes()** is a basic getter method.
5. **getShapes(shapeType: ShapeType)** is overloaded method which will return a subset of shapes list:
    a. *ShapeType* is enum, so if, ShapeType.Circle is the input argument, the method returns a list of shapes which are all Circles (ignoring all the instances of Squares and Rectangles)

Finally, the *ShapeGameDemo* class is shared with you, which has some statements. Execute it and you should have similar to the following output (next page).
Please, also note that, it will produce a file named "**shapes_modified.txt**" with the info about modified shapes.

Download and uncompress the file. Rename the folder as required and work on it.
Once completed, submit only the **source files** in the required format. No need to submit .txt files.

//HERE GOES THE EXPECTED OUTPUT in the console

```
Square [Rectangle [Shape [color=red, filled=true], width=3.0, length=3.0]]
Rectangle [Shape [color=black, filled=false], width=10.0, length=20.0]
Circle [Shape [color=blue, filled=false], radius=5.0]
Square [Rectangle [Shape [color=black, filled=false], width=10.0, length=10.0]]
Circle [Shape [color=yellow, filled=true], radius=3.0]
Square [Rectangle [Shape [color=purple, filled=true], width=5.0, length=5.0]]
Rectangle [Shape [color=cyan, filled=true], width=5.0, length=5.0]
----------------------------------------

Circle [Shape [color=blue, filled=false], radius=5.0], Area: 78.53981633974483, Per: 31.41592653589793
Circle [Shape [color=yellow, filled=true], radius=3.0], Area: 28.274333882308138, Per: 18.84955592153876
----------------------------------------

Square [Rectangle [Shape [color=red, filled=true], width=3.0, length=3.0]], Area: 9.0, Per: 12.0
Square [Rectangle [Shape [color=black, filled=false], width=10.0, length=10.0]], Area: 100.0, Per: 40.0
Square [Rectangle [Shape [color=purple, filled=true], width=5.0, length=5.0]], Area: 25.0, Per: 20.0
----------------------------------------

Square [Rectangle [Shape [color=red, filled=true], width=3.0, length=3.0]], Area: 9.0, Per: 12.0
Rectangle [Shape [color=black, filled=false], width=10.0, length=20.0], Area: 200.0, Per: 60.0
Square [Rectangle [Shape [color=black, filled=false], width=10.0, length=10.0]], Area: 100.0, Per: 40.0
Square [Rectangle [Shape [color=purple, filled=true], width=5.0, length=5.0]], Area: 25.0, Per: 20.0
Rectangle [Shape [color=cyan, filled=true], width=5.0, length=5.0], Area: 25.0, Per: 20.0
----------------------------------------
```