General Instructions:

1. Follow the instructions and especially the naming standards.
2. In this assignment you will need to create multiple source file. When you are done, in order to submit, put them (ONLY SOURCE FILES) in a folder named {**YOURID}_{FIRSTNAME}_{LASTNAME}_AS2.zip**
3. **Note:** replace your info, like **123456_ALI_ALILI_AS2.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 2

In this assignment you are to implement OOP principles you learned in a single mini project.

1. Define an interface **Measurable**. It has a single abstract method: **double getArea()**;
2. Define an abstract class **Shape** which has two fields: **type** and **name** both of type String.
    a. Provide constructor with a single input parameter: **name**.
    b. Provide accessor methods for both fields.
    c. Provide setter method for **name**.
    d. Provide **toString()** method that simply returns the "**{type}: {name}**"
    e. **Shape** class implements **Measurable** interface without implementing its only abstract method.
3. Define a concrete class **Rectangle** which extends the **Shape** class.
    a. It has **width** and **height** fields both of type double.
    b. Provide single constructor that takes three input parameters: **name**, **width** and **height** resp.
    c. Provide accessor methods for both fields.
    d. Implement the unimplemented method **getArea()** so that it returns the area of the rectangle object.
    e. Override toString() method so that it returns the following information:
        i. "**{type}: {name} – [{width}, {height}]**" e.g., **rectangle: my_rect_1 – [3.0, 5.0]**
4. Define a concrete class **Sphere** which extends the **Shape** class.
    a. It has **radius** field of type double.
    b. Provide single constructor that takes two input parameters: **name and radius** resp.
    c. Provide the accessor method.
    d. Implement the unimplemented method **getArea()** so that it returns the surface area of the sphere object.
    e. Override toString() method so that it returns the following information:
        i. "**{type}: {name} – [{radius}]**" e.g., **sphere: my_sphere_1 – [3.0]**

5. Now imagine you are given an array of different shapes and are required to sort them based on the area.

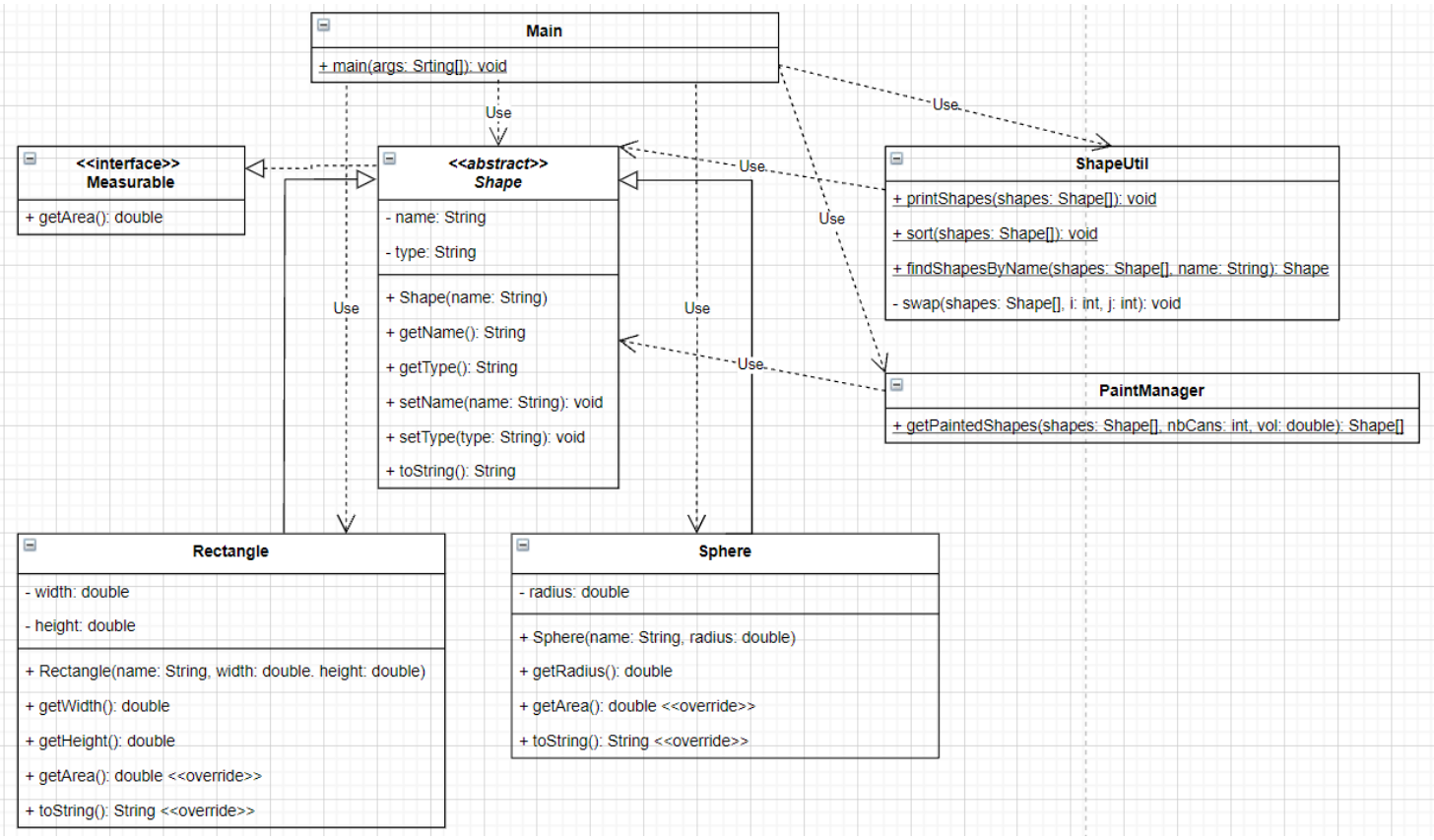
a. Define a static method **sort** in the class **ShapeUtil** that will sort the array of different Shape objects. (You may use any sorting algorithm, just make sure you implement it yourself)
b. Define another static method **printShapes** in the class **ShapeUtil** that prints an array of Shape objects in a single row. Consider using toString() methods of those.

6. The customer is willing to paint all the Shape objects. If he provides us with the number of liters of paint per can and the number of cans he bought, could you define a method [static method **getPaintedShape** in class **PaintManager**] that takes
   a. the array of shapes
   b. the volume of a single can
   c. the number of cans

   and return the shapes that will be completely painted.
   d. If the amount of paint is not sufficient, he desires to paint as many shapes as possible.

7. Customer also need you to provide her with the following functionality:
   a. As there are many objects and she likes to name each of them, sometimes she forgets about some. For simplicity, assume that the names are unique (there are no two objects with the same name)
   b. Define a method [**findShapeByName** in the class **ShapeUtil** which will take an **array** of Shape objects and a String **name**, which returns
      i. the Shape **object** with that name if exists or
      ii. **null** if there are no object with that name.

UML Class diagram



Main.java

```java
public class Main {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle("r1", 3, 5);
        Rectangle r2 = new Rectangle("r2", 2, 7);
        Rectangle r3 = new Rectangle("my_rect", 5, 5);

        Sphere s1 = new Sphere("s1", 3);
        Sphere s2 = new Sphere("s2", 6);
        Sphere s3 = new Sphere("ball", 5);

        Shape[] shapes = new Shape[] { s2, r1, s1, r2, s3, r3 };

        System.out.println("***Printing all the shapes***");
        ShapeUtil.printShapes(shapes);
        // TODO: uncomment the next two lines, test your sort method.
        // PaintManager.sort(shapes);
        // printShapes(shapes);
        // Comment them out then.

        System.out.println("***Printing the shapes that could be painted\n\tgiven the #cans and value of each can***");
        // TODO: Ask from user:
        // nbCans - first argument and volume of a single can - second argument
        ShapeUtil.printShapes(PaintManager.getPaintedShapes(shapes, 3, 100));

        System.out.println("***Finding and printing the shapes based on the given name***");
        // TODO: Ask from user the name to search for
        Shape res = ShapeUtil.findShapeByName(shapes, "r1");
        System.out.println("r1: " + res);

        System.out.println("ball: " + ShapeUtil.findShapeByName(shapes, "ball"));

        System.out.println("rect: " + ShapeUtil.findShapeByName(shapes, "rect"));
    }
}
```

Output: (if you execute Main)

```
***Printing all the shapes***
[sphere: s2[6.0]: 452.389, rectangle: r1[3.0, 5.0]: 15.000, sphere: s1[3.0]: 113.097, rectangle: r2[2.0,
7.0]: 14.000, sphere: ball[5.0]: 314.159, rectangle: my_rect[5.0, 5.0]: 25.000]

***Printing the shapes that could be painted
        given the #cans and value of each can***
[rectangle: r2[2.0, 7.0]: 14.000, rectangle: r1[3.0, 5.0]: 15.000, rectangle: my_rect[5.0, 5.0]: 25.000,
sphere: s1[3.0]: 113.097]

***Finding and printing the shapes based on the given name***
r1: rectangle: r1[3.0, 5.0]
ball: sphere: ball[5.0]
rect: null
```