Object-Oriented Programming

Lab session #3

-

I. References
- Oracle Java Documentation: https://docs.oracle.com/javase/tutorial/java/IandI/index.html
- Inheritance in Java Tutorial: https://www.tutorialspoint.com/java/java_inheritance.htm
- Method overriding in Java Tutorial: https://www.tutorialspoint.com/java/java_overriding.htm
- Polymorphism in Java Tutorial: https://www.tutorialspoint.com/java/java_polymorphism.htm
- Abstraction in Java Tutorial: https://www.tutorialspoint.com/java/java_abstraction.htm
- Interface in Java Tutorial: https://www.tutorialspoint.com/java/java_interfaces.htm
- Set attribute default value: https://stackoverflow.com/questions/43509987/how-do-i-set-default-values-for-instance-variables

II. Exercises
You are required to implement the following design as well as a main() method in an another class to test your implementation:

Question 1B: Abstract Superclass Animal and its Implementation Subclasses (15 marks)
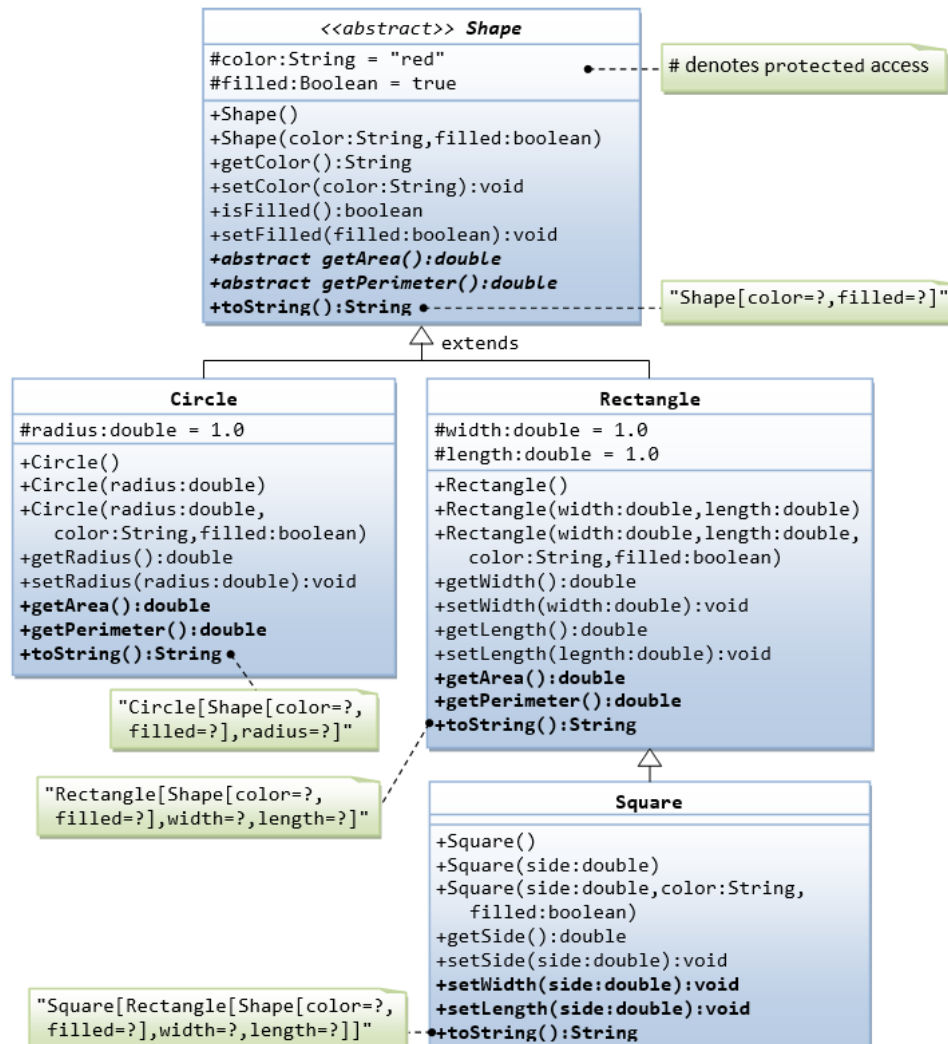
Write the codes for all the classes shown in the class diagram. Mark all the overridden methods with annotation @Override.
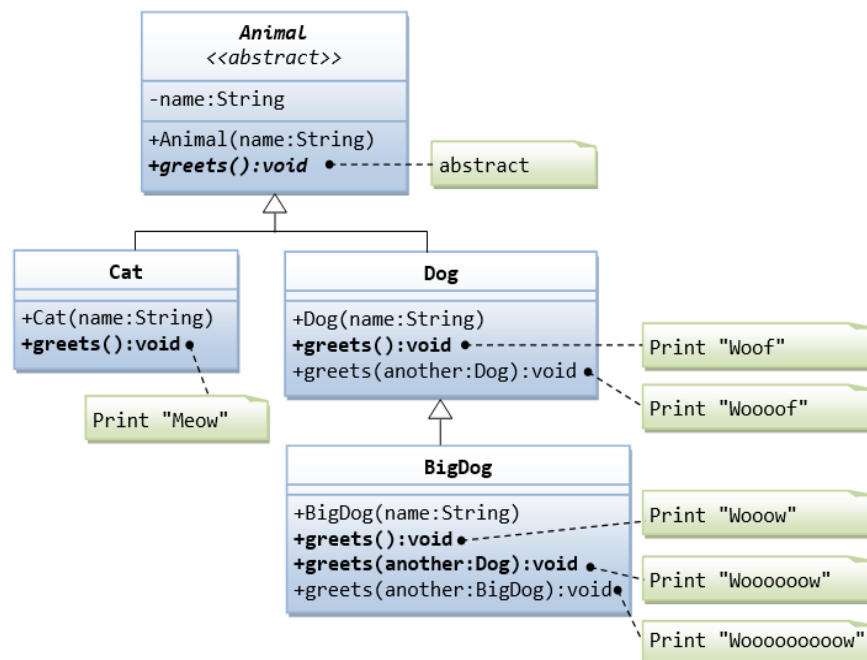
Question 2B: Abstract Superclass Shape and Its Concrete Subclasses (15 marks)

Rewrite the superclass **Shape** and its subclasses **Circle**, **Rectangle** and **Square**, as shown in the class diagram.

Shape is an **abstract** class containing 2 **abstract** methods: **getArea()** and **getPerimeter()**, where its concrete subclasses must provide its implementation. All instance variables shall have **protected** access, i.e., accessible by its subclasses and classes in the same package. Mark all the overridden methods with annotation @**Override.**

```
                    <<abstract>> Shape
        #color:String = "red"              •------ # denotes protected access
        #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 •------------------ "Shape[color=?,filled=?]"

                        extends
```

```
            Circle                              Rectangle
#radius:double = 1.0                #width:double = 1.0
                                   #length:double = 1.0
+Circle()
+Circle(radius:double)             +Rectangle()
+Circle(radius:double,             +Rectangle(width:double,length:double)
   color:String,filled:boolean)    +Rectangle(width:double,length:double,
+getRadius():double                    color:String,filled:boolean)
+setRadius(radius:double):void     +getWidth():double
+getArea():double                  +setWidth(width:double):void
+getPerimeter():double             +getLength():double
+toString():String •               +setLength(legnth:double):void
                                   +getArea():double
                                   +getPerimeter():double
       "Circle[Shape[color=?,      •+toString():String
        filled=?],radius=?]"

    "Rectangle[Shape[color=?,                Square
     filled=?],width=?,length=?]"
                                   +Square()
                                   +Square(side:double)
                                   +Square(side:double,color:String,
                                       filled:boolean)
                                   +getSide():double
                                   +setSide(side:double):void
                                   +setWidth(side:double):void
    "Square[Rectangle[Shape[color=?,   +setLength(side:double):void
     filled=?],width=?,length=?]]"  --•+toString():String
```

In this exercise, **Shape** shall be defined as an **abstract** class, which contains:

- Two **protected** instance variables **color(String)** and **filled(boolean)**. The **protected** variables can be accessed by its subclasses and classes in the same package. They are denoted with a **'#'** sign in the class diagram.
- Getter and setter for all the instance variables, and **toString()**.
- Two **abstract** methods **getArea()** and **getPerimeter()** (shown in italics in the class diagram).

The subclasses **Circle** and **Rectangle** shall *override* the **abstract** methods **getArea()** and **getPerimeter()** and provide the proper implementation. They also *override* the **toString()**.

Write a test class (main class) to test these statements involving polymorphism and explain the outputs. Some statements may trigger compilation errors. Explain the errors, if any. Your main method should could look like this:

```
Shape s1 = new Circle(5.5, "red", false);  // Upcast Circle to Shape
System.out.println(s1);                     // which version?
System.out.println(s1.getArea());           // which version?
```

```java
System.out.println(s1.getPerimeter());    // which version?
System.out.println(s1.getColor());
System.out.println(s1.isFilled());
System.out.println(s1.getRadius());

Circle c1 = (Circle)s1;              // Downcast back to Circle
System.out.println(c1);
System.out.println(c1.getArea());
System.out.println(c1.getPerimeter());
System.out.println(c1.getColor());
System.out.println(c1.isFilled());
System.out.println(c1.getRadius());

Shape s2 = new Shape();

Shape s3 = new Rectangle(1.0, 2.0, "red", false);   // Upcast
System.out.println(s3);
System.out.println(s3.getArea());
System.out.println(s3.getPerimeter());
System.out.println(s3.getColor());
System.out.println(s3.getLength());

Rectangle r1 = (Rectangle)s3;   // downcast
System.out.println(r1);
System.out.println(r1.getArea());
System.out.println(r1.getColor());
System.out.println(r1.getLength());

Shape s4 = new Square(6.6);     // Upcast
System.out.println(s4);
System.out.println(s4.getArea());
System.out.println(s4.getColor());
System.out.println(s4.getSide());

// Take note that we downcast Shape s4 to Rectangle,
//   which is a superclass of Square, instead of Square
Rectangle r2 = (Rectangle)s4;
System.out.println(r2);
System.out.println(r2.getArea());
System.out.println(r2.getColor());
System.out.println(r2.getSide());
System.out.println(r2.getLength());

// Downcast Rectangle r2 to Square
Square sq1 = (Square)r2;
System.out.println(sq1);
System.out.println(sq1.getArea());
System.out.println(sq1.getColor());
System.out.println(sq1.getSide());

System.out.println(sq1.getLength());
```
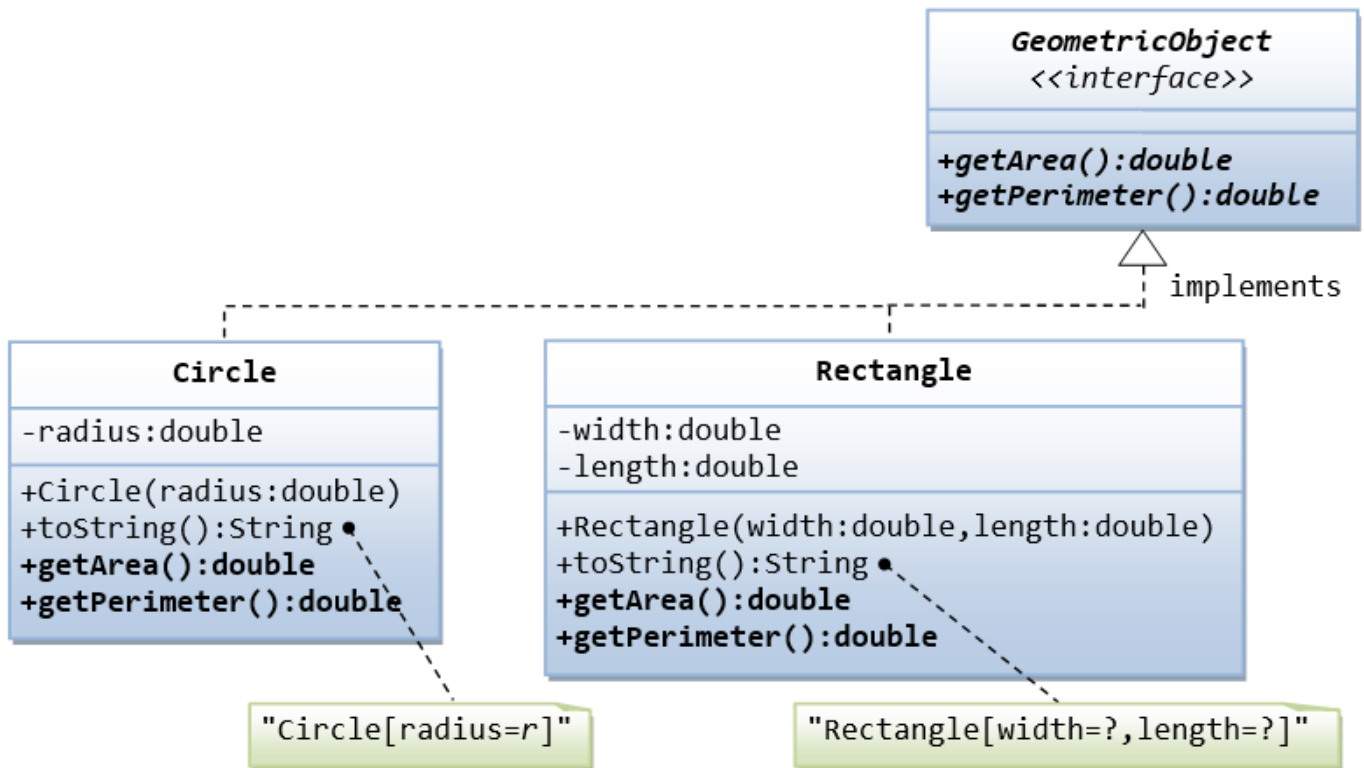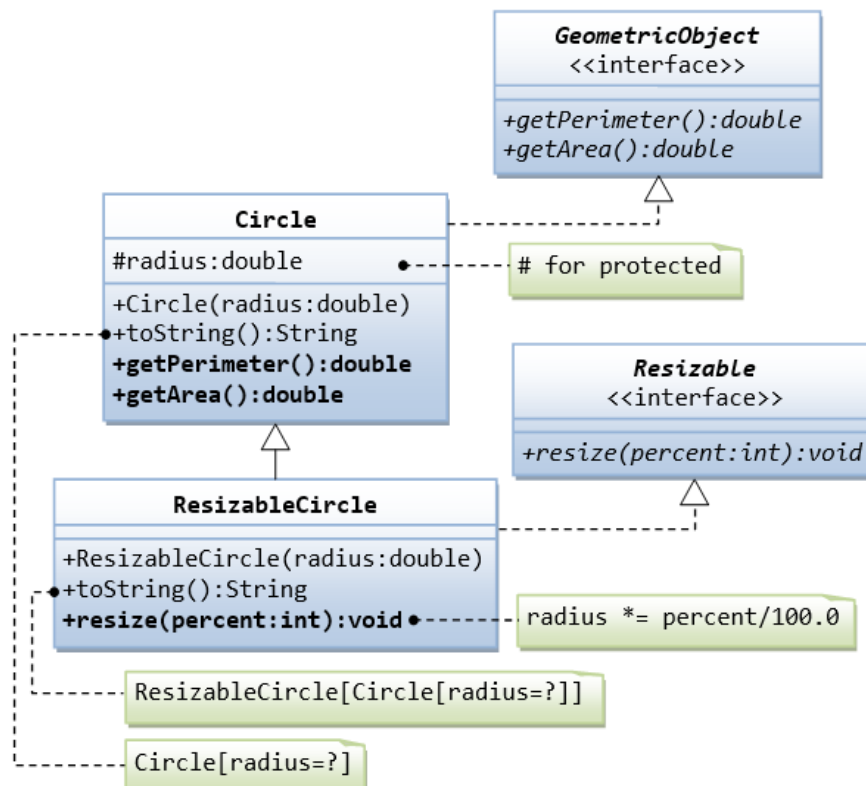
What is the usage of the **abstract** method and **abstract** class?

Question 3A: GeometricObject Interface and its Implementation Classes Circle and Rectangle (15 marks)

Write an **interface** called `GeometricObject`, which contains 2 **abstract** methods: `getArea()` and `getPerimeter()`, as shown in the class diagram. Also write an implementation class called `Circle`. Mark all the overridden methods with annotation @`Override`.

```
        GeometricObject
         <<interface>>

    +getArea():double
    +getPerimeter():double
```
                                    △ implements

```
        Circle
-radius:double

+Circle(radius:double)
+toString():String •
+getArea():double
+getPerimeter():double
```
"Circle[radius=r]"

```
                    Rectangle
-width:double
-length:double

+Rectangle(width:double,length:double)
+toString():String •
+getArea():double
+getPerimeter():double
```
"Rectangle[width=?,length=?]"

Question 3B: Interfaces Resizable and GeometricObject (15 marks)

```
        GeometricObject
         <<interface>>

    +getPerimeter():double
    +getArea():double
```
                                △

```
        Circle
#radius:double          •--------- # for protected

+Circle(radius:double)
+toString():String
+getPerimeter():double
+getArea():double
```
                    △

```
        Resizable
        <<interface>>

+resize(percent:int):void
```
                    △

```
        ResizableCircle

+ResizableCircle(radius:double)
+toString():String
+resize(percent:int):void •------ radius *= percent/100.0
```

ResizableCircle[Circle[radius=?]]

Circle[radius=?]

Hints (step by step):

1. Write the **interface** called **GeometricObject**, which declares two **abstract** methods: **getParameter()** and **getArea()**, as specified in the class diagram.

```
public interface GeometricObject {
   public double getPerimeter();
   ......
}
```

2. Write the implementation class **Circle**, with a protected variable **radius**, which implements the interface **GeometricObject**.

```
public class Circle implements GeometricObject {
   // Private variable
   ......

   // Constructor
   ......

   // Implement methods defined in the interface GeometricObject
   @Override
   public double getPerimeter() { ...... }

   ......
}
```

3. Write a test program called **TestCircle** to test the methods defined in **Circle**.
4. The class **ResizableCircle** is defined as a subclass of the class **Circle**, which also implements an interface called **Resizable**, as shown in class diagram. The interface **Resizable** declares an **abstract** method **resize()**, which modifies the dimension (such as **radius**) by the given percentage. Write the interface **Resizable** and the class **ResizableCircle**.
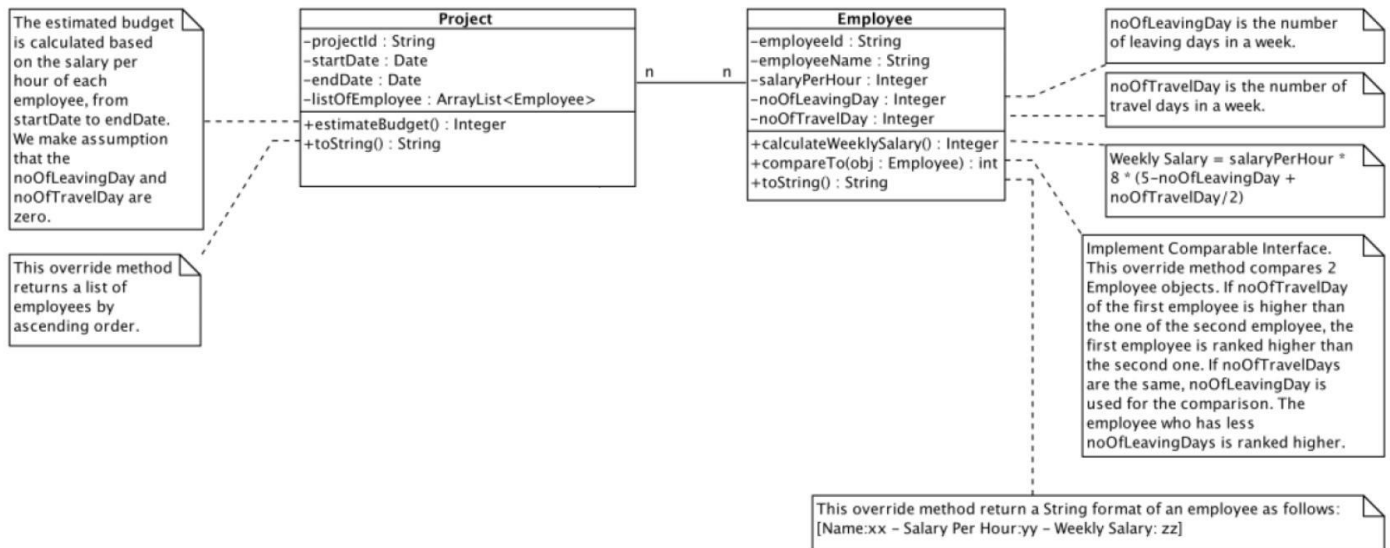
```
public interface Resizable {
   public double resize(...);
}
```

```
public class ResizableCircle extends Circle implements Resizeable {

   // Constructor
   public ResizableCircle(double radius) {
      super(...);
   }

   // Implement methods defined in the interface Resizable
   @Override
   public double resize(int percent) { ...... }
}
```

5. Write a test program called **TestResizableCircle (with main method)** to test the methods defined in **ResizableCircle**.

Question 4 (Bonus): Projects and Employees (30 marks)

You are required to implement the following design as well as an AppTest class to test your implementation. You must add appropriate constructors, getter and setter for each class. You can add private methods into the classes to support your work.

The estimated budget is calculated based on the salary per hour of each employee, from startDate to endDate. We make assumption that the noOfLeavingDay and noOfTravelDay are zero.

This override method returns a list of employees by ascending order.

Project
-projectId : String
-startDate : Date
-endDate : Date
-listOfEmployee : ArrayList<Employee>
+estimateBudget() : Integer
+toString() : String

Employee
-employeeId : String
-employeeName : String
-salaryPerHour : Integer
-noOfLeavingDay : Integer
-noOfTravelDay : Integer
+calculateWeeklySalary() : Integer
+compareTo(obj : Employee) : int
+toString() : String

noOfLeavingDay is the number of leaving days in a week.

noOfTravelDay is the number of travel days in a week.

Weekly Salary = salaryPerHour * 8 * (5−noOfLeavingDay + noOfTravelDay/2)

Implement Comparable Interface. This override method compares 2 Employee objects. If noOfTravelDay of the first employee is higher than the one of the second employee, the first employee is ranked higher than the second one. If noOfTravelDays are the same, noOfLeavingDay is used for the comparison. The employee who has less noOfLeavingDays is ranked higher.

This override method return a String format of an employee as follows:
[Name:xx – Salary Per Hour:yy – Weekly Salary: zz]

Hints:
To use Date datatype, remember to *import java.util.Date;*

References:
- Comparable Interface tutorial: https://www.javatpoint.com/Comparable-interface-in-collection-framework
- The reason why we want to implement comparable interface so that we can sort ArrayList of Employee to print it out in order in method toString() of Project like so:
https://beginnersbook.com/2013/12/java-arraylist-of-object-sort-example-comparable-and-comparator/