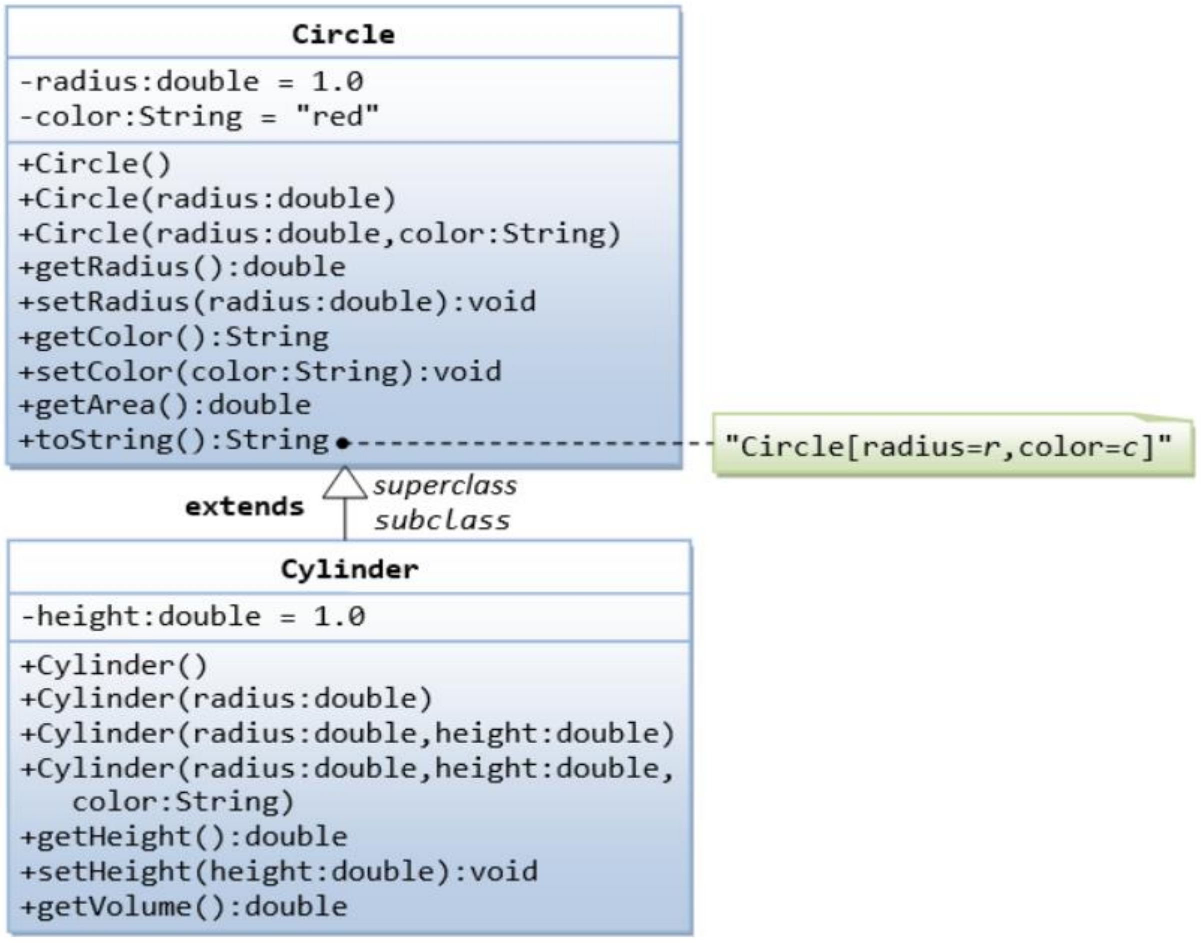
Assignment (BCAC391)

(Inheritance, super, abstract class, abstract method, downcast, upcast, array)

1.



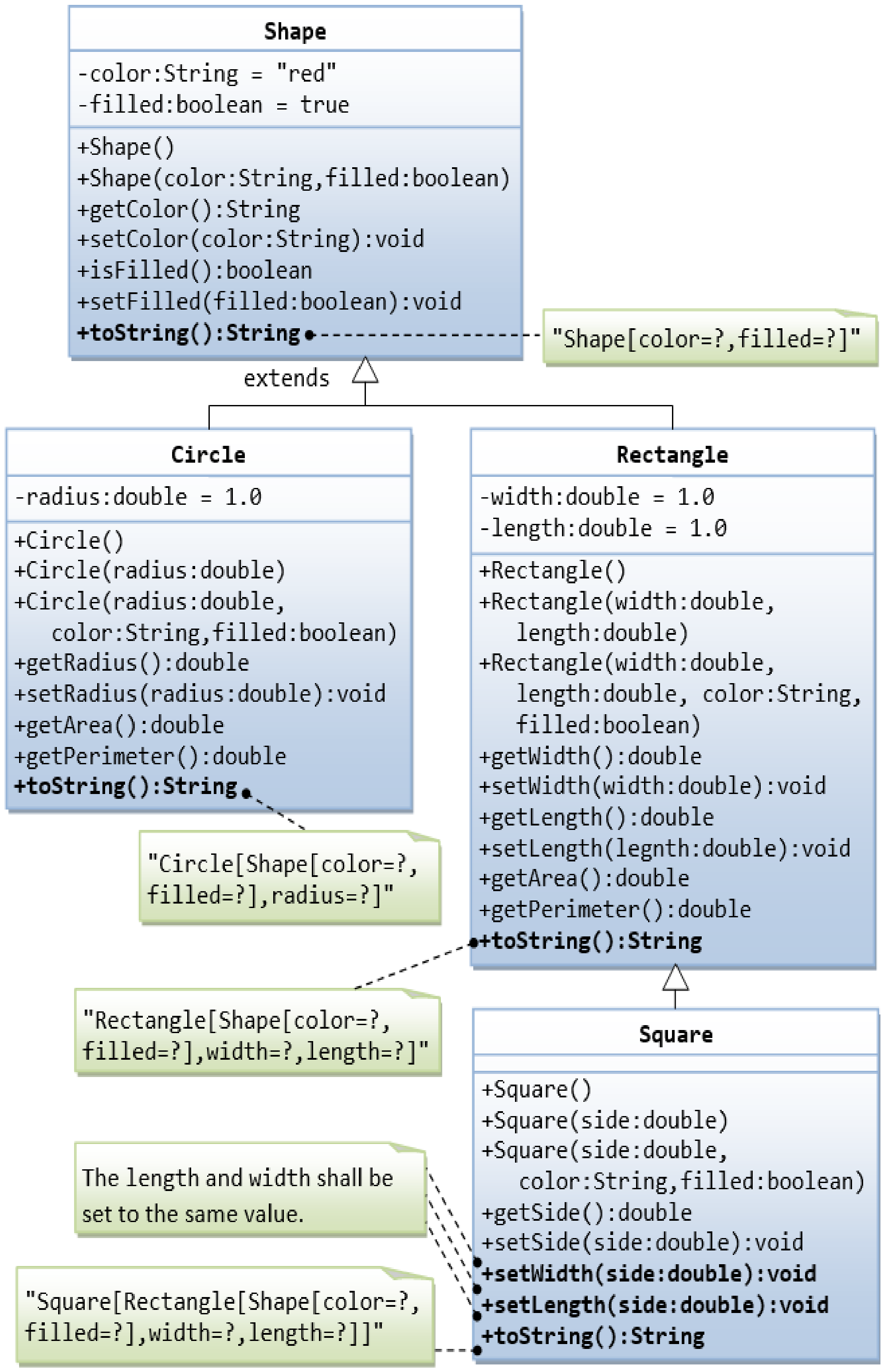
Write a test program (says `TestCylinder`) to test the **`Cylinder`** class created, as follow:

```
public class TestCylinder {  // save as "TestCylinder.java"
   public static void main (String[] args) {
      // Declare and allocate a new instance of cylinder
      //   with default color, radius, and height
      Cylinder c1 = new Cylinder();
      System.out.println("Cylinder:"
            + " radius=" + c1.getRadius()
            + " height=" + c1.getHeight()
            + " base area=" + c1.getArea()
            + " volume=" + c1.getVolume());

      // Declare and allocate a new instance of cylinder
      //   specifying height, with default color and radius
      Cylinder c2 = new Cylinder(10.0);
      System.out.println("Cylinder:"
            + " radius=" + c2.getRadius()
            + " height=" + c2.getHeight()
            + " base area=" + c2.getArea()
            + " volume=" + c2.getVolume());

      // Declare and allocate a new instance of cylinder
      //   specifying radius and height, with default color
      Cylinder c3 = new Cylinder(2.0, 10.0);
      System.out.println("Cylinder:"
            + " radius=" + c3.getRadius()
            + " height=" + c3.getHeight()
            + " base area=" + c3.getArea()
            + " volume=" + c3.getVolume());
   }
}
```

<u>Expected Output</u>:

```
Cylinder: radius=1.0 height=1.0 base area=3.141592653589793
volume=3.141592653589793

Cylinder: radius=10.0 height=1.0 base area=314.1592653589793
volume=314.1592653589793

Cylinder: radius=2.0 height=5.0 base area=12.566370614359172
volume=62.83185307179586
```

2.



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
+toString():String
"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
The length and width shall be set to the same value.
"Square[Rectangle[Shape[color=?, filled=?],width=?,length=?]]"

Write a test program (says **TestDriver**) to test the above class created, as follow:

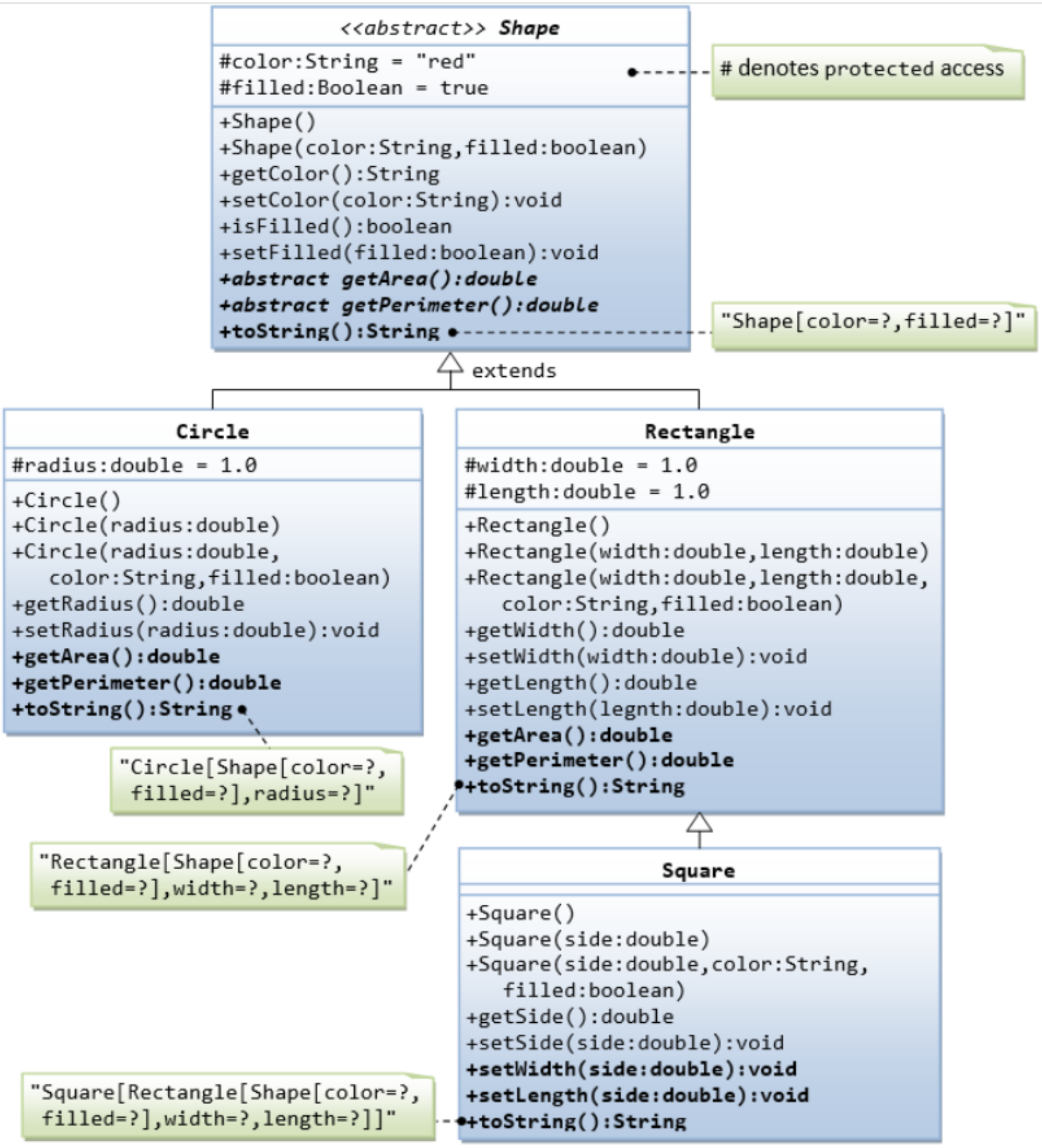
```
public class testDriver {
    public static void main(String[] args) {
        Shape s1=new Shape();
        System.out.println(s1);
        s1.setColor("black");
        s1.setFilled(false);
        System.out.println("Color:"+s1.getColor());
        System.out.println("Color:"+s1.isFilled());

        Circle c1=new Circle();
        System.out.println(c1);

        Circle c2=new Circle(10);
        System.out.println(c2);

        Circle c3=new Circle(11.0,"Red",false);
        System.out.println("Radius:"+c3.getRadius());
        c3.setRadius(12.0);
        System.out.println("Radius:"+c3.getRadius());
        System.out.println(c3);

        Rectangle r1=new Rectangle(10.0,15.0,"White",true);
        System.out.println(r1);
        r1.setColor("Blue");
        r1.setLength(20.0);
        System.out.println("Color:"+r1.getColor());
        System.out.println("Length:"+r1.getLength());

        Square sq1=new Square(10,"Black",true);
        System.out.println("Sides:"+sq1.getSide());
        sq1.setWidth(5);
        sq1.setLength(10);
        System.out.println(sq1);
    }
}
```

<u>Expected Output:</u>

```
Shape[color=red, filled=true]
Color:black
Color:false

Circle[Shape[color=red, filled=true], radius=1.0]

Circle[Shape[color=red, filled=true], radius=10.0]

Radius:11.0
Radius:12.0
Circle[Shape[color=Red, filled=false], radius=12.0]
Rectangle[Shape[color=White, filled=true],width=10.0,length=15.0]
Color:Blue
Length:20.0
```

```
Sides:10.0

Square[Rectangle[Shape[color=Black, filled=true],width=10.0,length=10.0]]
```

3.



<<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=?]]"

Write a test class to test these statements involving polymorphism and explain the outputs. Some statements may trigger compilation errors. Explain the errors, if any.

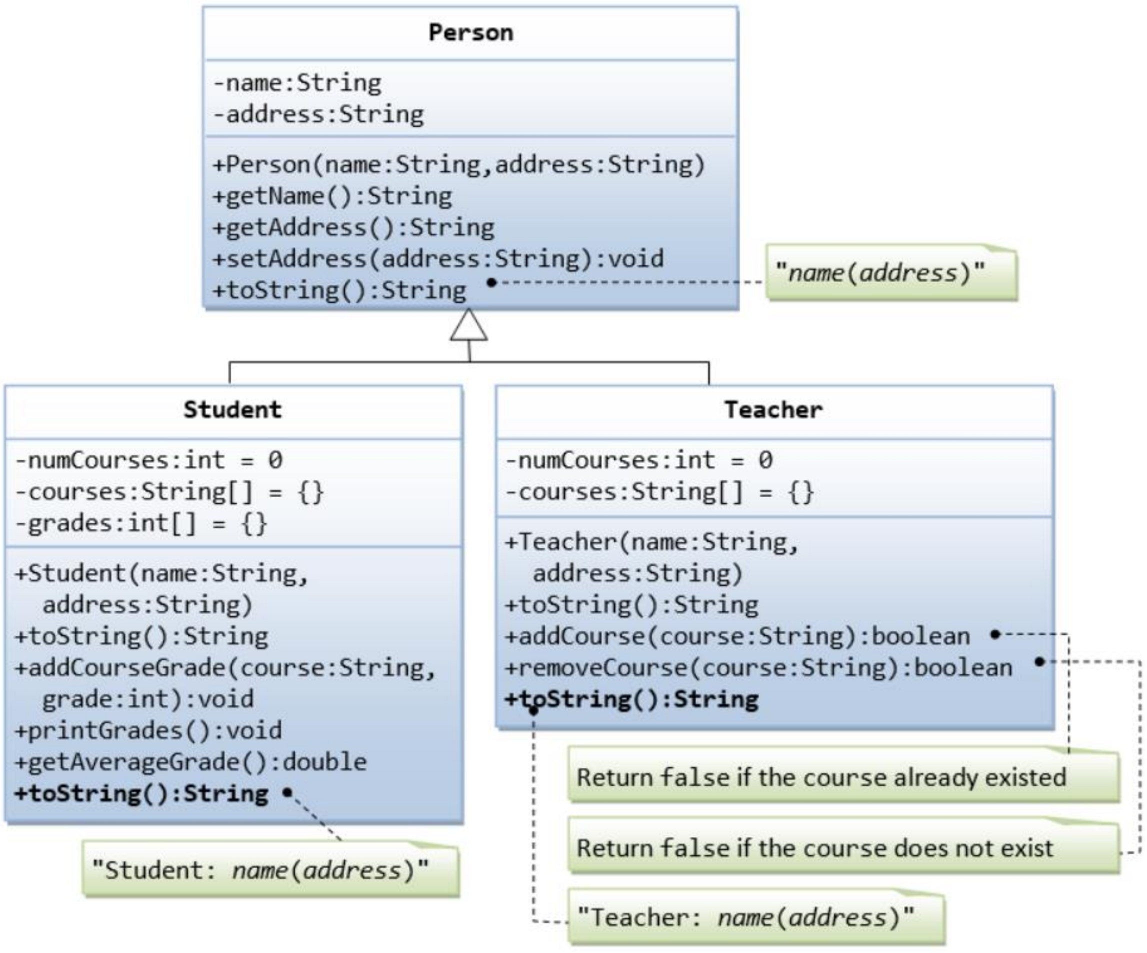
```
public class testDriver {
   public static void main(String[] args) {
      Shape s1 = new Circle(5.5, "red", false);  // Upcast Circle to Shape
       System.out.println(s1);                  // which version?
       System.out.println(s1.getArea());        // which version?
       System.out.println(s1.getPerimeter());     // which version?
       System.out.println(s1.getColor());
       System.out.println(s1.isFilled());
       //System.out.println(s1.getRadius());// we can't access this child class method

       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());// we can't access this child class method

       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());// we can't access this child class method

       // 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());
```

```
    }
}
```

Expected Output:

```
Circle[Shape[color=red,filled=false]],radius=5.5]
95.03317777109125
34.55751918948772
red
false
Circle[Shape[color=red,filled=false]],radius=5.5]
95.03317777109125
34.55751918948772
red
false
5.5
Rectangle[Shape[color=red,filled=false],width=1.0,length=2.0]
2.0
6.0
red
Rectangle[Shape[color=red,filled=false],width=1.0,length=2.0]
2.0
red
2.0
Square[Rectangle[Shape[color=red,filled=true],width=6.6,length=6.6]]
43.559999999999995
red
Square[Rectangle[Shape[color=red,filled=true],width=6.6,length=6.6]]
43.559999999999995
red
6.6
Square[Rectangle[Shape[color=red,filled=true],width=6.6,length=6.6]]
43.559999999999995
red
6.6
6.6
```

4.


Person
-name:String
-address:String
+Person(name:String,address:String)
+getName():String
+getAddress():String
+setAddress(address:String):void
+toString():String
"name(address)"
Student
-numCourses:int = 0
-courses:String[] = {}
-grades:int[] = {}
+Student(name:String, address:String)
+toString():String
+addCourseGrade(course:String, grade:int):void
+printGrades():void
+getAverageGrade():double
+toString():String
"Student: name(address)"
Teacher
-numCourses:int = 0
-courses:String[] = {}
+Teacher(name:String, address:String)
+toString():String
+addCourse(course:String):boolean
+removeCourse(course:String):boolean
+toString():String
Return false if the course already existed
Return false if the course does not exist
"Teacher: name(address)"

Suppose that we are required to model **students** and **teachers** in our application. We can define a superclass called **Person** to store common properties such as **name** and **address**, and subclasses **Student** and **Teacher** for their specific properties. For students, we need to maintain the courses taken and their respective grades; add a course with grade, print all courses taken and the average grade. Assume that a student takes no more than 30 courses for the entire program. For teachers, we need to maintain the courses taught currently, and able to add or remove a course taught. Assume that a teacher teaches not more than 5 courses concurrently.

Write the Test Driver(**testDriver.java**) code to check the above implementation:

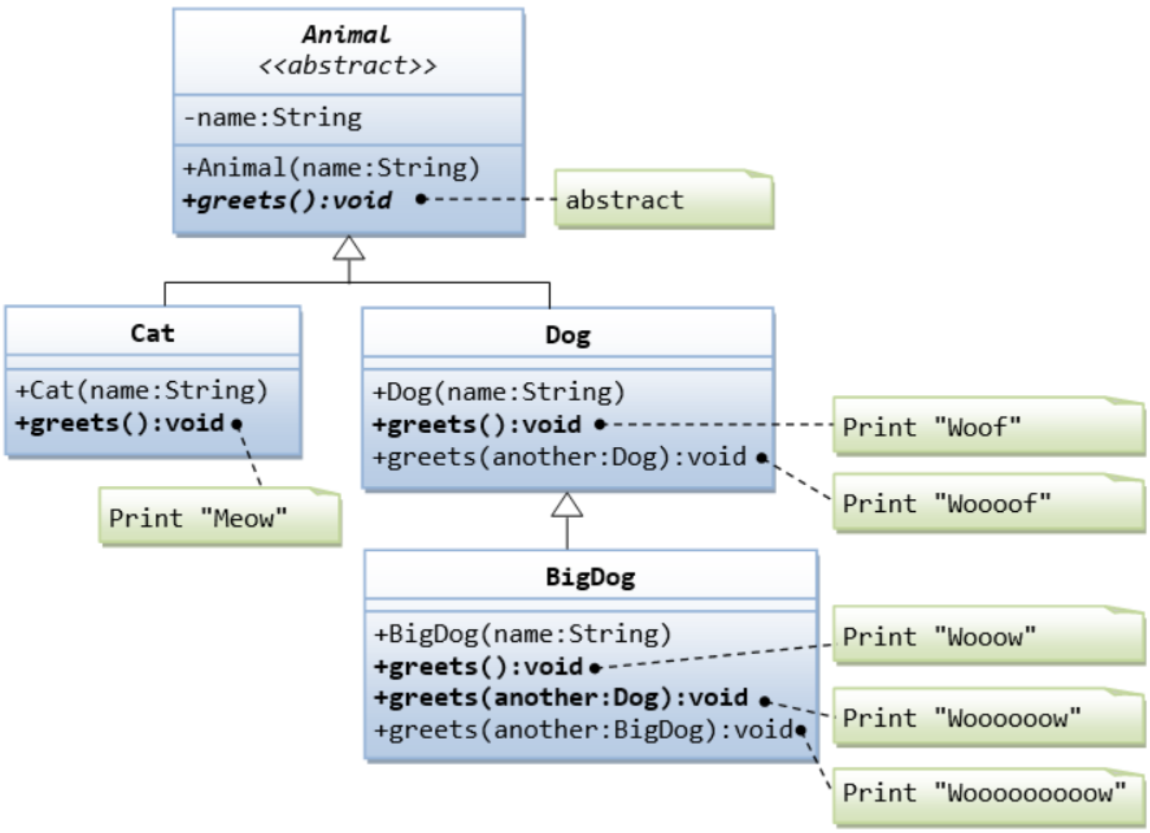
```
public class TestPerson {
  public static void main(String[] args) {

     Student s1 = new Student("Subhajit", "Kolkata");
     s1.addCourseGrade("B.tech(CSE)", 97);
     s1.addCourseGrade("M.tech(CSE)", 68);
     s1.printGrades();

     System.out.println("Average is " + s1.getAverageGrade());
```

```
      Teacher t1 = new Teacher("Susovan Kumar Pan", "Kolkata");
      System.out.println("\n"+t1);
      //Teacher: Paul Tan(8 sunset way)
      String[] courses = {"B.tech(CSE)", "B.tech(IT)", "B.tech(CSE)"};
      for (String course: courses) {
        if (t1.addCourse(course)) {
          System.out.println(course + " added");
        } else {
          System.out.println(course + "already in list,cannot be added");
        }
      }

      for (String course: courses) {
        if (t1.removeCourse(course)) {
          System.out.println(course + " removed");
        } else {
          System.out.println(course + "Not in list,cannot be removed");
        }
      }
    }
}
```

Expected Output:

```
Student: Subhajit(Kolkata)
B.tech(CSE):97
M.tech(CSE):68
Average is 82.5

Teacher: Susovan Kumar Pan(Kolkata)
B.tech(CSE) added
B.tech(IT) added
B.tech(CSE)already in list,cannot be added
B.tech(CSE) removed
B.tech(IT) removed
B.tech(CSE)Not in list,cannot be removed
```

5. Write the codes for all the classes shown in the class diagram. Mark all the overridden methods with annotation @Override.



Write the TestAnimal(**TestAnimal.java**) code to check the above implementation:

```
public class TestAnimal {
   public static void main(String[] args) {
      // TODO code application logic here
      Cat cat1=new Cat("Tom");
      cat1.greets();

      Dog dog1=new Dog("Pug");
      dog1.greets();
      dog1.greets(dog1);

      BigDog bigDog1=new BigDog("Big Pug");
      bigDog1.greets();
      bigDog1.greets(bigDog1);
   }
}
```

Expected Output:

```
Meow
Woof
Woooof
Wooow
Wooooooooooooow
```