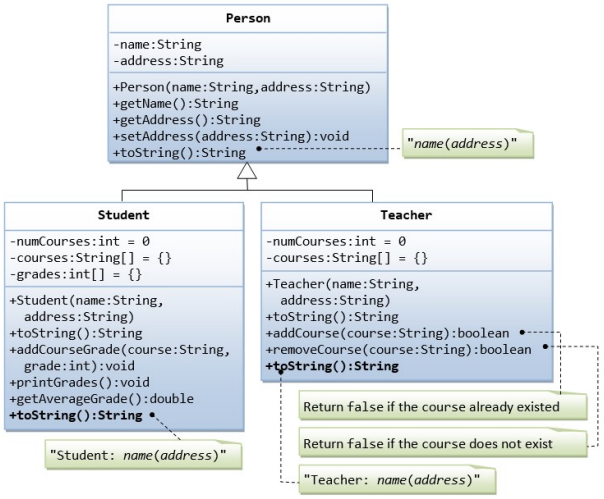
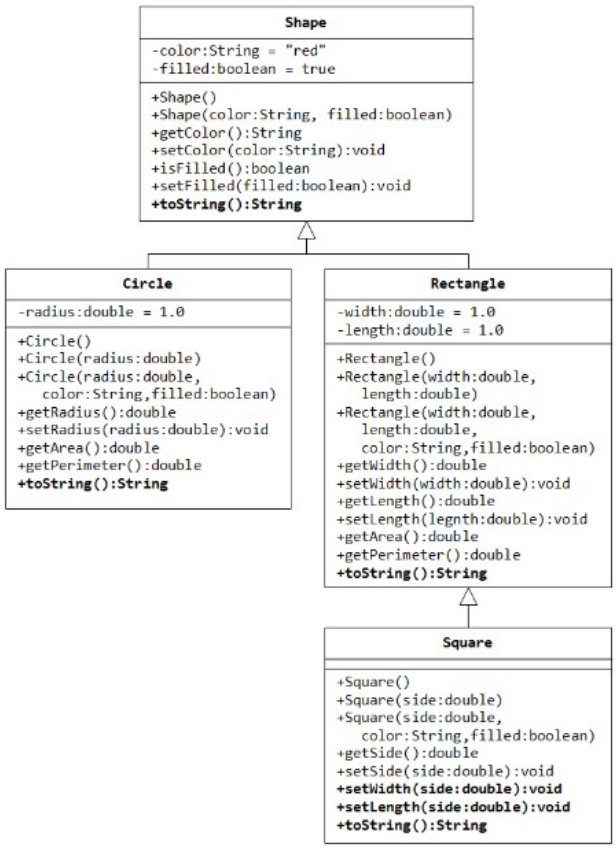
Assignment-7

We are required to model students and teachers in an 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 **6** 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.Test the methods of both the derived classes by creating objects of the derived classes in the main method of another class.





2.
Write a superclass called **Shape** (as shown in the class diagram), which contains:

- Two instance variables **color** (String) and **filled** (boolean).
- Two constructors: a no-argument constructor that initializes the color to "green" and filled to true, and a constructor that initializes the color and filled to the given values.
- Getter and setter for all the instance variables. By convention, the getter for a boolean variable xxx is called isXXX() (instead of getXxx() for all the other types).
- A toString() method that returns "A Shape with color of xxx and filled/Not filled".

Write a test program to test all the methods defined in Shape.
Write two subclasses of Shape called **Circle** and **Rectangle**, as shown in the class diagram.
The **Circle** class contains:

- An instance variable **radius** (double).
- Three constructors as shown. The no-arg constructor initializes the radius to 1.0.
- Getter and setter for the instance variable radius.
- Methods **getArea**() and **getPerimeter**().
- Override the **toString**() method inherited, to return "A Circle with radius=xxx, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

The **Rectangle** class contains:

- Two instance variables **width** (double) and **length** (double).
- Three constructors as shown. The no-arg constructor initializes the width and length to 1.0.
- Getter and setter for all the instance variables.
- Methods **getArea**() and **getPerimeter**().
- Override the toString() method inherited, to return "A Rectangle with width=xxx and length=zzz, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

Write a class called **Square**, as a subclass of Rectangle. Square has no instance variable, but inherits the instance variables width and length from its superclass Rectangle.

- Provide the appropriate constructors (as shown in the class diagram).
  Hint: public Square(double side) {
  super(side, side); // Call superclass Rectangle(double, double)
  }
- Override the **toString**() method to return "A Square with side=xxx, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

Override the **setLength**() and **setWidth**() to change both the width and length, so as to maintain the square geometry.

3.

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

```
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());
```

```
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());
```