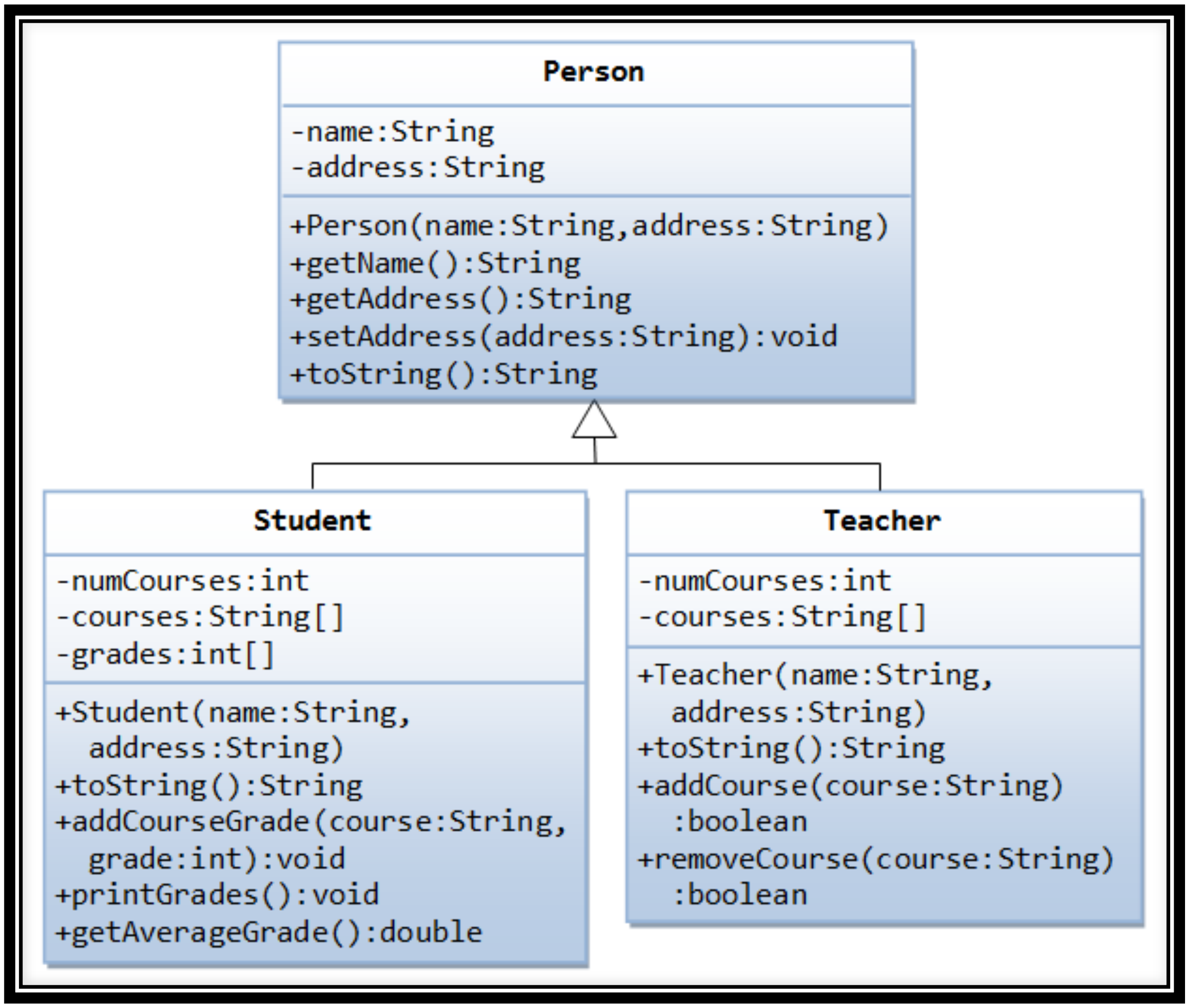
Inheritance

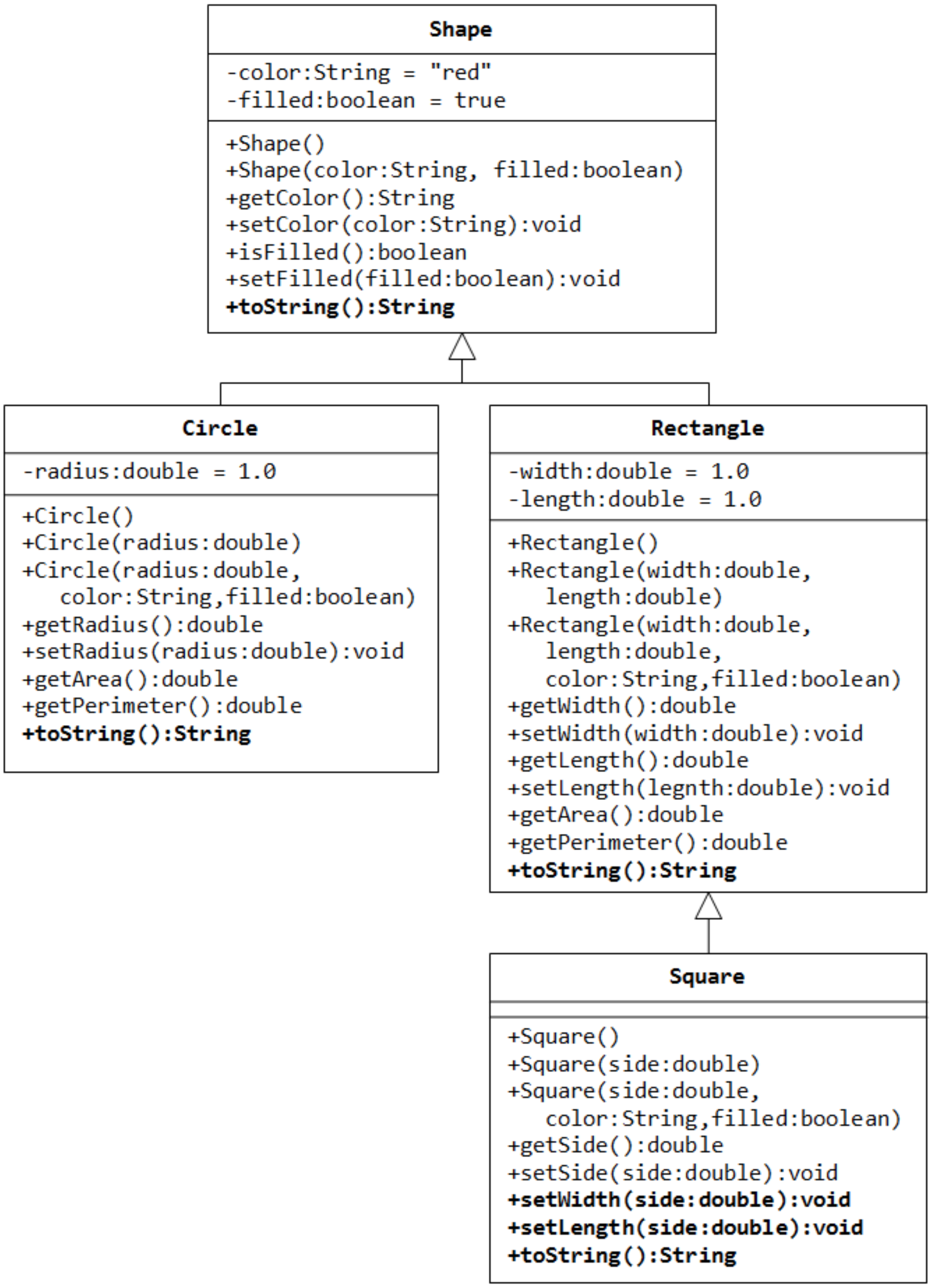
Suppose that we are required to **model students** and **teachers** in our application.
We can define a **superclass called Person** to store common properties such as name and address, and **subclasses Student and Teacher** for their specific properties.

Specifications:
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. 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. A teacher teaches not more than 5 courses concurrently.

We design the classes as follows.


Person
-name:String
-address:String
+Person(name:String,address:String)
+getName():String
+getAddress():String
+setAddress(address:String):void
+toString():String
Student
-numCourses:int
-courses:String[]
-grades:int[]
+Student(name:String, address:String)
+toString():String
+addCourseGrade(course:String, grade:int):void
+printGrades():void
+getAverageGrade():double
Teacher
-numCourses:int
-courses:String[]
+Teacher(name:String, address:String)
+toString():String
+addCourse(course:String) :boolean
+removeCourse(course:String) :boolean

ASSIGNMENT


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
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
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
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

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-arg (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`. Convince yourself that `Square` can be modeled 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.
- Do you need to override the `getArea()` and `getPerimeter()`? Try them out.
- Override the `setLength()` and `setWidth()` to change both the `width` and `length`, so as to maintain the square geometry.

Additional Requirements

Create a client code that has the following components:

1. Create objects from the different shapes classes.
2. Convert shapes class into an abstract class
3. Create a polymorphic arraylist with a bunch of different shape objects
4. Create a method that will traverse the arraylist and will find the shape with the largest area
5. Implement the comparable interface