Exercise 08

Inheritance: Person, Student and Staff

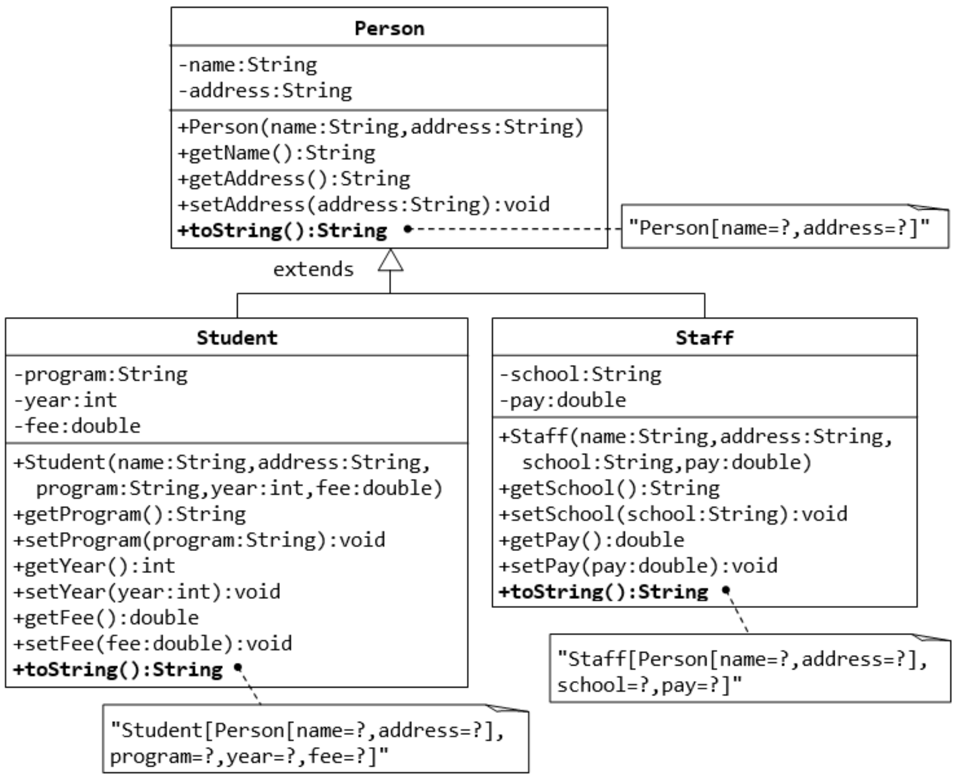
Create a class **Person**.
The class has two data fields (String **name** and String **address**)
Create a **constructor** for Person which takes as input name and address.
Write **getter** methods for name and address.
Write a **setter** method for address (we do not want to have one for name, it is read only).
Write a **toString** method (for the format see class diagram).

Implement **two child classes** for the **Person** class. One is called **Student** the other one **Staff**.

Student has three additional data fields. String program, integer year and double fee.
Create also a **constructor** for student (name, address, program, year and fee). Use super to reuse the constructor of Person (constructor chaining).
Write **getter** and **setter** methods for program, year and fee.
Override the toString method and extend it with the information from Student (for the format see class diagram).

Staff has two additional data fields, String school and double pay.
Write a **constructor** for the Staff class (name, address, school, pay). Try also here to utilize super.
Write **getter** and **setter** methods for school and pay.
Override the toString method and extend it with the information about the school (for the format see class diagram)

Write a **test class** called **TestPerson**. In the TestPerson class create 5 persons. 3 students and 2 staff members. Test **all methods** for the specific classes. Compare the toString between Student and Staff.


Person
-name:String
-address:String
+Person(name:String,address:String)
+getName():String
+getAddress():String
+setAddress(address:String):void
+toString():String
"Person[name=?,address=?]"
extends
Student
-program:String
-year:int
-fee:double
+Student(name:String,address:String, program:String,year:int,fee:double)
+getProgram():String
+setProgram(program:String):void
+getYear():int
+setYear(year:int):void
+getFee():double
+setFee(fee:double):void
+toString():String
"Student[Person[name=?,address=?], program=?,year=?,fee=?]"
Staff
-school:String
-pay:double
+Staff(name:String,address:String, school:String,pay:double)
+getSchool():String
+setSchool(school:String):void
+getPay():double
+setPay(pay:double):void
+toString():String
"Staff[Person[name=?,address=?], school=?,pay=?]"

Composition example – compare composition to inheritance

Let us begin with *composition* with the statement "a line composes of two points".
Complete the **code** of the following two classes: Point and Line. Add your code at the parts marked with // ADD CODE The class Line composes 2 instances of class Point, representing the beginning and ending points of the line. Also write test classes for Pointand Line (says TestPoint and TestLine).

```
public class Point {
   // Private variables
   private int x;    // x co-ordinate
   private int y;    // y co-ordinate

     // Constructor
   public Point (int x, int y) {//ADD CODE}

   // Public methods
   public String toString() {
      return "Point: (" + x + "," + y + ")";
   }

   public int getX() {//ADD CODE }
   public int getY() {//ADD CODE }
   public void setX(int x) {//ADD CODE }
   public void setY(int y) {//ADD CODE }
   public void setXY(int x, int y) {//ADD CODE}
}
```

That was the Point class. Lets **write a test class** for it.

```
public class TestPoint {
   public static void main(String[] args) {
      Point p1 = new Point(10, 20);   // Construct a Point
      System.out.println(p1);
      //ADD CODE Try setting p1 to (100, 10).

   }
}
```

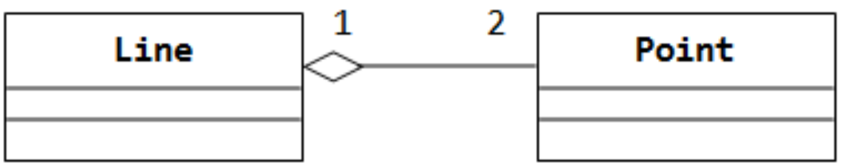
The point class is tested and works as indented. Now lets **create the Line class**.

```
public class Line {
   // A line composes of two points (as instance variables)
   private Point begin;    // beginning point
   private Point end;      // ending point

   // Constructors
   public Line (Point begin, Point end) {  // caller to construct the Points
      this.begin = begin;
      this.end = end;
   }
   public Line (int beginX, int beginY, int endX, int endY) {
      begin = new Point(beginX, beginY);   // construct the Points here
      end = new Point(endX,endY);
   }

   // Public methods
   public String toString() { //ADD CODE }

   public Point getBegin() { //ADD CODE }
   public Point getEnd() { //ADD CODE }
   public void setBegin(//ADD CODE) { //ADD CODE }
   public void setEnd(//ADD CODE) { //ADD CODE }

   public int getBeginX() { //ADD CODE }
   public int getBeginY() { //ADD CODE }
   public int getEndX() { //ADD CODE }
   public int getEndY() { //ADD CODE }

   public void setBeginX(//ADD CODE) { //ADD CODE }
   public void setBeginY(//ADD CODE) { //ADD CODE }
   public void setBeginXY//ADD CODE { //ADD CODE }
   public void setEndX//ADD CODE { //ADD CODE }
   public void setEndY(//ADD CODE) { //ADD CODE }
   public void setEndXY(//ADD CODE) { //ADD CODE }

   public int getLength() { //ADD CODE } // Length of the line
                                  // Math.sqrt(xDiff*xDiff + yDiff*yDiff)
   public double getGradient() { //ADD CODE } // Gradient in radians
                                       // Math.atan2(yDiff, xDiff)
}
```

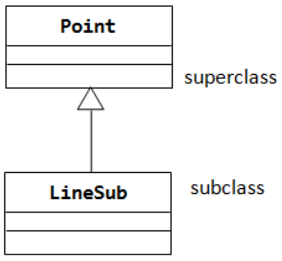
Now we test our line class. **Write for that a class TestLine**.

```
public class TestLine {
   public static void main(String[] args) {
      Line l1 = new Line(0, 0, 3, 4);
      System.out.println(l1);

      Point p1 = new Point(...);
      Point p2 = new Point(...);
      Line l2 = new Line(p1, p2);
      System.out.println(l2);

   }
}
```

The simple class diagram for *composition* looks as follows (a diamond-hollow-head arrow pointing to its constituents) **Note:** We left out the details to save space and because the final task of this exercise is to draw the class diagram yourself.



Instead of *composition*, we can also design a Line class using inheritance. Instead of "a line composes of two points", we can say that "a line is a point extended by another point", as shown in the following class diagram:



Let's **re-design the Line class** (called LineSub) as a subclass of class Point. LineSub inherits the starting point from its superclass Point and adds an ending point. **Complete** the class by adding missing code. Write a **testing class** called TestLineSub to test LineSub. Again, where you are supposed to add your code is marked with //ADD CODE.

```
public class LineSub extends Point {
   // A line needs two points: begin and end.
   // The begin point is inherited from its superclass Point.
   // Private variables
   Point end;               // Ending point

   // Constructors
   public LineSub (int beginX, int beginY, int endX, int endY) {
      super(beginX, beginY);             // construct the begin Point
      this.end = new Point(endX, endY);  // construct the end Point
   }
   public LineSub (Point begin, Point end) {  // caller to construct the Points
      super(begin.getX(), begin.getY());      // need to reconstruct the begin Point
      this.end = end;
   }

   // Public methods
   // Inherits methods getX() and getY() from superclass Point
   public String toString() { //ADD CODE }

   public Point getBegin() { //ADD CODE }
   public Point getEnd() { //ADD CODE }
   public void setBegin//ADD CODE { //ADD CODE }
   public void setEnd//ADD CODE { //ADD CODE }

   public int getBeginX() { //ADD CODE }
   public int getBeginY() { //ADD CODE }
   public int getEndX() { //ADD CODE }
   public int getEndY() { //ADD CODE }
```

```
    public void setBeginX(//ADD CODE) { //ADD CODE }
    public void setBeginY(//ADD CODE) { //ADD CODE }
    public void setBeginXY(//ADD CODE) { //ADD CODE }
    public void setEndX(//ADD CODE) { //ADD CODE }
    public void setEndY(//ADD CODE) { //ADD CODE }
    public void setEndXY(//ADD CODE) { //ADD CODE }

    public int getLength() { //ADD CODE }       // Length of the line
    public double getGradient() { //ADD CODE }  // Gradient in radians
}
```

Write a new **test class** for the inheritance version of the lines that does the same tests.

Summary: There are two approaches that you can design a line, `composition` or `inheritance`. "A line composes two points" or "A line is a point extended with another point"”. Compare the `Line` and `LineSub`designs: `Line` uses *composition* and `LineSub` uses *inheritance*. **What do you think, which design is better?**

Finally, try to **draw** the complete **class diagram yourself for both cases (composition and inheritance)**.