Object-Orientation & Inheritance

Table of Contents

1. Session 2 (lab): Object-Orientation & Inheritance (I)

1.1. Points and Geometric Figures (I)

A point at the plane can be represented by a pair of coordinates x and y, both with real values. In Java, we can represent a point at the plane as an instance of the following class:

```
public class Point {
    private double x;
    private double y;
}
```

Section 1: Point class constructor

Program a constructor for the `Point` class. It must receive both point coordinates as input parameters.

Section 2: Method to get a text representation of a point.

The `String toString()` method has a special meaning for objects in Java. This method is used to get a text representation of the object as a string.

Program the `String toString()` method of the `Point` class so that it returns a string with the representation of the point like in the following example: "(23, -3)". The first number represents the *x* coordinate of the point, and the second one the *y* coordinate.

Section 3: The `main` method

Now you have to create a class to test the previous code. Program a class called `TestPoint` which has a `main` method. This method must receive, as command line arguments, the *x* and *y* coordinates, then create a new `Point` object with those coordinates and print a text representation of the object to the standard output.

The program must check that the number of command line arguments received is correct.

Take into account that the `parseDouble` method of the `Double` class transforms a string of characters to a `double` primitive type.

Section 4: The access methods

It is common that object's attributes are declared as `private` to avoid uncontrolled accesses from other parts of the program. To provide such access from outside code, there are special methods. Those methods usually begin with the following strings: "`get`" (read access) and "`set`" (write access).

Program the following access methods that return the coordintate values *x* and *y* in the `Point` class:

```
public double getX() {
    /* ... */
}

public double getY() {
```

```
    /* ... */
}
```

Modify the code of your `TestPoint` class to check that its behaviour is correct.

Section 5: Computing distances

Program the following method of the `Point` class, which returns the distance from the point to the origin of the coordinates:

```
public double distance() {
  /* ... */
}
```

Overload the previous method with the following one, which receives another object of the `Point` class and returns the distance between the point to which the method belongs and the point received as parameter:

```
public double distance(Point anotherPoint) {
    /* ... */
}
```

Modify the code of your `TestPoint` class to check that its behaviour is correct.

Section 6: Quadrant calculation.

Program a method of the `Point` class that returns the quadrant in which the point is located as an `int` value:

- It returns 0 if it is placed at the origin of coordinates or on any of the axis.
- It returns 1 if it is placed at the first quadrant (both x and y positives).
- It returns 2 if it is placed at the second quadrant (x negative and y positive).
- It returns 3 if it is placed at the third quadrant (both x and y negatives).
- It returns 4 if it is placed at the fourth quadrant (x positive and y negative).

The prototype of the method is shown below:

```
public int quadrant()  {
  /* ... */
}
```

Modify the code of your `TestPoint` class to check that its behaviour is correct.

Section 7: Computing the nearest point.

Program a method of the `Point` class that receives, as input parameter, an array of `Point` objects and returns a reference to the object of the array that represents the nearest point to the current object (this). The prototype of the method is as follows:

```
public Point nearest(Point[] otherPoints) {
  /* ... */
}
```

Modify the code of your `TestPoint` class to check that its behaviour is correct.

Section 8: The `Triangle` class.

A triangle is fully defined by three of its vertices. These vertexes can be represented as objects of the `Point` class.

Program the `Triangle` class with its attributes and a constrctor that receives three points (vertices) as input parameters.

Section 9: Computing side lengths.

Program the `sideLengths()` method of the `Triangle` class. It must return an array of three decimals (`double`) that represents, respectively, the length of each side of the triangle. The prototype of the method is as follows:

```
public double[] sideLengths() {
    /* ... */
}
```

Program a class to test the `Triangle` class.

2. Homework

2.1. Rational numbers

A rational number is any number that can be expressed as the quotient of two integers. In this exercise you must program a class called `Rational` that represents rational numbers and allow operating with them.

We recommend testing the class while you are coding it, instead of testing it when it is finished. You should program a test class and use it to test the `Rational` class as you program it.

Section 1

Program the basic structure of the class: its attributes and two constructors. The first constructor has two integers as its arguments (numerator and denominator). The second constructor will receive only one argument, an integer, an will create the rational representation of that ordinary integer, this is, having 1 as the denominator. This second constructor must be implemented using the first one.

Add a `String toString()` method to the class. It must return a textual representation of rational numbers as shown in this examples: "-5 / 2", "1 / 3", "6", "-7". Notice that only the numerator is shown when the denominator is 1.

Section 2

Modify the first constructor to create uniform objects according to this set of rules:

1. If the denominator is negative, multiply numerator and denominator by -1.
2. Simplify the quotient by dividing numerator and denominator by its maximum common divisor.

We recommend using a private method to implement this operations, which should be invoqued by the constructor.

You should be able to calculate the maximum common divisor of two integers by using the following code:

```
private int gcd() {
    int a = /* put here the attribute that represents the numerator */
    int b = /* put here the attribute that represents the denominator */
    while (b != 0) {
        int tmp = b;
        b = a % b;
        a = tmp;
    }
    if (a < 0) {
        a = -a;
    }
    return a;
}
```

Section 3

Add another method that receives an object of the class `Rational` and returns **a new object** of that same class representing the sum of the current object (this) and the object passed as an argument.

```
public Rational sum(Rational other) {
    (...)
}
```

Section 4

Repeat the previous section, but returning the product of the objects.

```
public Rational multiply(Rational other) {
    (...)
}
```

Section 5

Overload the methods from the two previous sections to receive the numerator and denominator of the operand instead of its `Rational` representation. You must reuse your previous code by calling the original `sum` and `multiply` methods from the overloaded ones.

```
public Rational sum(int otherNumerator, int otherDenominator) {
    (...)
}
```

```
    public Rational multiply(int otherNumerator, int otherDenominator) {
        (...)
    }
```