Learning is not attained by chance, it must be sought for with ardor and attended to with diligence.

~ Abigail Adams

CS1010J Programming Methodology

Tutorial 8: Basic OOP Part II

I. Manual Tracing

1. The following class **Point** represents a point in a 2-dimensional plane.

```
class Point {

  private double x, y;  // x- and y- coordinates

  // Construct a point of coordinates (inX, inY)
  public Point(double inX, double inY) {
    /* fill in your code below */
    this.x = inX;
    this.y = inY;



  }

  // Return the x-coordinate of a point
  public double getX() {
    /* fill in your code below */
    return this.x;




  }

  // Return the y-coordinate of a point
  public double getY() {
    /* fill in your code below */
    return this.y;





  }

  // Return the string representation of a point
  // in the format "[x, y]"
  public String toString() {
    /* fill in your code below */
```

```
      return "[" + this.x + ", " this.y+"]";

   }
}
```

(a) Fill in the **Point** class above for the parts marked with /* fill in your code below */.

(b) Given the **Point** class above, what's the output of the following **user method**?

```
class T8Q1 { // user (driver) class

  // Other code omitted for brevity

  public static void test1() {

    Point p1 = new Point(0, 0);
    Point p2 = new Point(4.5, 5.2);

    System.out.println(p1);
    System.out.println(p2);
  }
}
```

[0.0] [4.5, 5.2]

(c) Suppose we want to compute the distance between two points. One way is to write a **user method** `distanceBetween()` that takes two points as parameters and computes their distance.

Complete the user method `distanceBetween()` below.

```
// Compute the distance between two points
public static double distanceBetween(Point p1, Point p2) {

  /* fill in your code below */
   return(
   Math.sqrt(
   Math.pow((p1.getX() - p2.getX(), 2.0)
   + Math.pow((p1.getY() - p2.getY(), 2.0))
   )

}
```

(d) An alternative way to compute the distance between two points is to define a **member method** **`distanceTo()`** in the **Point** class. This method takes 'another' point as parameter and compute the distance between 'this' point (i.e. the point object who invokes the **`distanceTo()`** method) and 'another' point.

Complete the member method **`distanceTo()`** below.

```
class Point {

  private double x, y;  // x- and y- coordinates

  // Other code omitted for brevity

  // Compute the distance between 'this' point
  // and 'another' point
  public double distanceTo(Point another) {

    /* fill in your code below */

    return(
    Math.sqrt(
    Math.pow((this.x - another.getX(), 2.0)
    + Math.pow((this.y - another.getY(), 2.0))
    )

    //another.x is an alternative instead of another.getX() as they have the
    same class Point

  }
}
```

(e) What’s the output of the following **user method**?

```
public static void test2() {

  Point p1 = new Point(0, 0);
  Point p2 = new Point(3, 4);

  System.out.println(distanceBetween(p1, p2));

  System.out.println(p1.distanceTo(p2));

  System.out.println(p2.distanceTo(p1));
}
```

5

(f) Now add a **member method** `equals()` to the **Point** class which takes 'another' point as parameter and checks if 'this' point equals to 'another'.

Two points are considered equal if they are close enough (see method description below for details).

```
class Point {

  private double x, y;  // x- and y- coordinates

  // Other code omitted for brevity

  // Return true if 'this' point is close enough
  // to 'another', or false otherwise.
  // Two points are close enough if their
  // distance is no more than 0.000001.
  public boolean equals(Point another) {

    /* fill in your code below */
    return this.distanceTo(another) < 0.000001





  }
}
```

(g) Write another **member method** `getMidPoint()` that takes 'another' point and returns a third point that lays middle in the line having 'this' (the point that invokes the `getMidPoint()` method) and 'another' points (the parameter point) as endpoints.

For example, given two points `(2, 4)` and `(2, 8)`, the middle point is `(2, 6)`.

```
class Point {

  private double x, y;  // x- and y- coordinates

  // Other code omitted for brevity

  // Return a third point that lays middle
  // in the line having 'this' and 'another'
  // points as endpoints.
  public Point getMidPoint(Point another) {

    /* fill in your code below */
```

```
      int midPtX = (this.x+another.getX) / 2;
      int midPtY = (this.y+another.getY) / 2;

      return new Point(midPtX, midPtY);




   }
}
```

(h) Finally, what's the output of the following **user method**?

```
public static void test3() {

  Point p1 = new Point(0, 0);
  Point p2 = new Point(8, 8);

  Point p3 = p1.getMidPoint(p2);
  System.out.println(p3);    [4,4]

  for (int i = 0; i < 3; i++) {
    p2 = p1.getMidPoint(p2);
  }
  System.out.println(p2);         [1.0,1.0]
}
```

To students:

The complete program of this question is provided for you to have a holistic view.

II. Programming

2. **[Problem Set 4 Exercise #01] My Circle**

3. **[Problem Set 4 Exercise #04] Voucher**

4. **[Problem Set 4 Exercise #06] Car**