SEG 2105

Laboratory 2 (handed in as Assignment 1) – Object Oriented Concepts

PART I

POINTCP

Question 3

Please see attached classes PointCP.java, PointCPDesign2.java, PointCPDesign3.java and PointCPDesign6.java.

Question 4

Please see attached classe PointCPTestModified.java

Question 5 (E26, E28 – E30)

Table 1: Advantages and disadvantages

Classes	**Advantages**	**Disadvantages**
Design 1 (PointCP)	• A single class to contain both Cartesian and polar coordinates • A single instantiation form super/parent/calling classes for the use of either polar or Cartesian coordinate	• An extra memory allocation for typeOfCoordinate, compared compared to other classes • Code slightly more complex since required to use conditional if statements before computing internal public methods
Design 2 (PointCPDesign2)	• One less memory allocation since no need of variable typeOfCoordinate • Code simpler to read as there is no need of conditional if statement before computing internal public methods (type of coordianate is already known) • Internal public methods may run faster	• Multiple class instantiantion from super/parent/calling classes for the use of either polar or Cartesian coordinate
Design 3 (PointCPDesign2)	• Same as design 2	• Same as design 2
Design 6 (PointCPDesign6)	• An interface which allow uniformity in classes PointCPDesign2 and PointCPDesign2 which implement it	

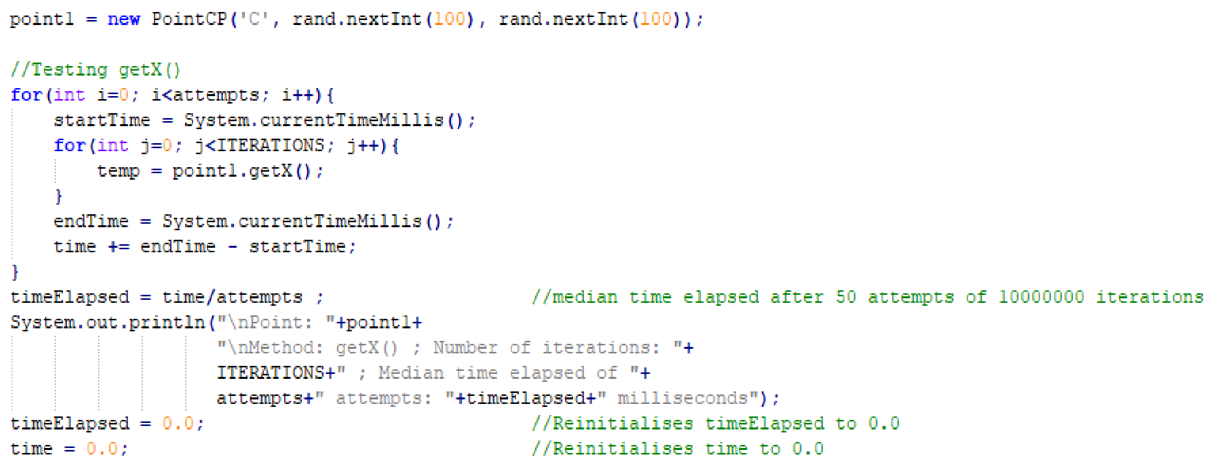
Please see attached class PerformanceAnalysis.java

Question 6

A similar methodology is used to test each public method contained in classes PointCP.java, PointCPDesign2.java and PointCPDesign3.java.

Two nested for-loops are used, the inner most loop calling the public method subjected to test for a constant number of iterations (10000000), and the outer most loop repeats the inner most loop a constant number of attempts (50), and stores the time difference taken before and after the execution of the inner most loop. The median time elapsed for a constant number of attempts is then generated, and the variables are reinitialised.

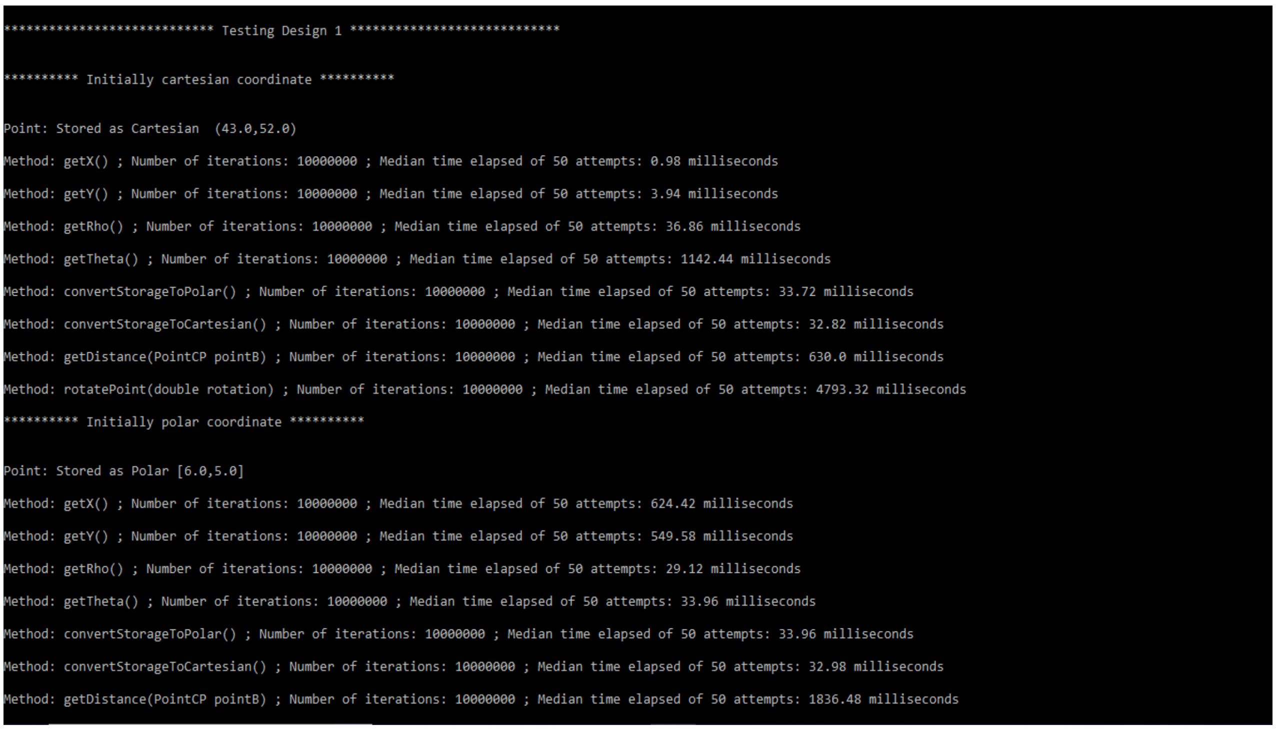
Figure 6.1 illustrate the above explanation, depicting method getX() being tested at inner most loop.



```java
point1 = new PointCP('C', rand.nextInt(100), rand.nextInt(100));

//Testing getX()
for(int i=0; i<attempts; i++){
    startTime = System.currentTimeMillis();
    for(int j=0; j<ITERATIONS; j++){
        temp = point1.getX();
    }
    endTime = System.currentTimeMillis();
    time += endTime - startTime;
}
timeElapsed = time/attempts ;                   //median time elapsed after 50 attempts of 10000000 iterations
System.out.println("\nPoint: "+point1+
                   "\nMethod: getX() ; Number of iterations: "+
                   ITERATIONS+" ; Median time elapsed of "+
                   attempts+" attempts: "+timeElapsed+" milliseconds");
timeElapsed = 0.0;                              //Reinitialises timeElapsed to 0.0
time = 0.0;                                     //Reinitialises time to 0.0
```

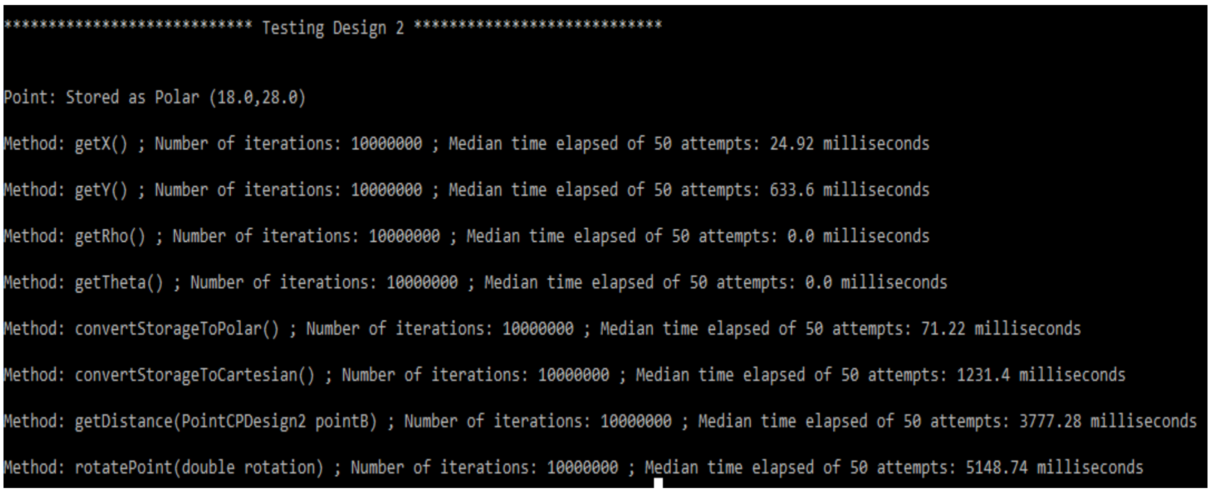
Figure 1: Two nested for-loop

Sample outputs from running the test are shown below



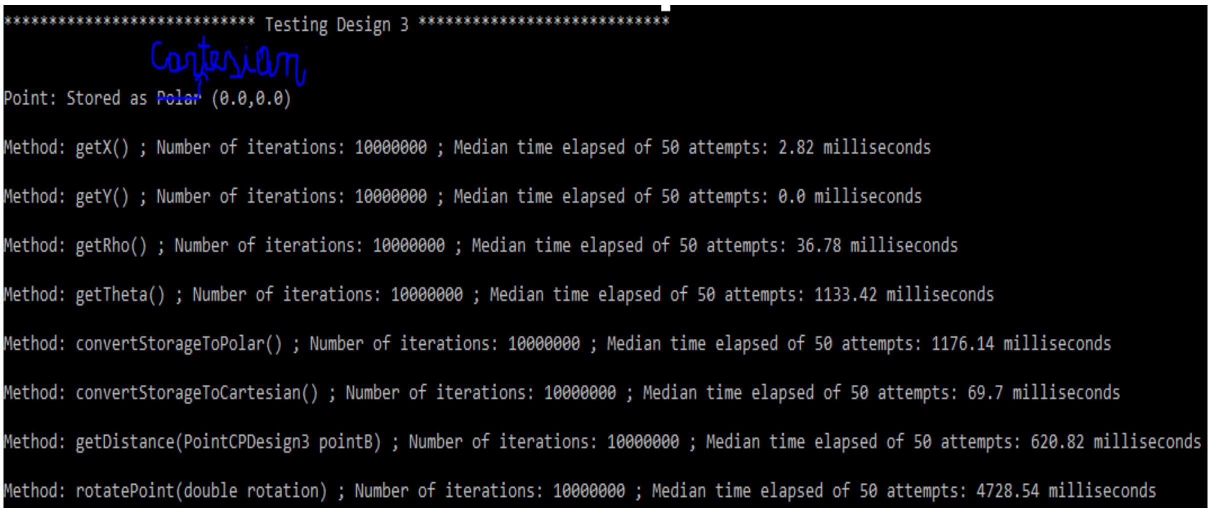
```
*************************** Testing Design 1 ***************************

********** Initially cartesian coordinate **********

Point: Stored as Cartesian  (43.0,52.0)

Method: getX() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 0.98 milliseconds

Method: getY() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 3.94 milliseconds

Method: getRho() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 36.86 milliseconds

Method: getTheta() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 1142.44 milliseconds

Method: convertStorageToPolar() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 33.72 milliseconds

Method: convertStorageToCartesian() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 32.82 milliseconds

Method: getDistance(PointCP pointB) ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 630.0 milliseconds

Method: rotatePoint(double rotation) ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 4793.32 milliseconds

********** Initially polar coordinate **********

Point: Stored as Polar [6.0,5.0]

Method: getX() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 624.42 milliseconds

Method: getY() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 549.58 milliseconds

Method: getRho() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 29.12 milliseconds

Method: getTheta() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 33.96 milliseconds

Method: convertStorageToPolar() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 33.96 milliseconds

Method: convertStorageToCartesian() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 32.98 milliseconds

Method: getDistance(PointCP pointB) ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 1836.48 milliseconds
```

Figure 2: Testing class PointCP.java



```
**************************** Testing Design 2 ****************************

Point: Stored as Polar (18.0,28.0)

Method: getX() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 24.92 milliseconds

Method: getY() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 633.6 milliseconds

Method: getRho() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 0.0 milliseconds

Method: getTheta() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 0.0 milliseconds

Method: convertStorageToPolar() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 71.22 milliseconds

Method: convertStorageToCartesian() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 1231.4 milliseconds

Method: getDistance(PointCPDesign2 pointB) ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 3777.28 milliseconds

Method: rotatePoint(double rotation) ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 5148.74 milliseconds
```

Figure 3: Testing PointCPDesign2.java



```
**************************** Testing Design 3 ****************************
                    Cartesian
Point: Stored as ~~Polar~~ (0.0,0.0)

Method: getX() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 2.82 milliseconds

Method: getY() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 0.0 milliseconds

Method: getRho() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 36.78 milliseconds

Method: getTheta() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 1133.42 milliseconds

Method: convertStorageToPolar() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 1176.14 milliseconds

Method: convertStorageToCartesian() ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 69.7 milliseconds

Method: getDistance(PointCPDesign3 pointB) ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 620.82 milliseconds

Method: rotatePoint(double rotation) ; Number of iterations: 10000000 ; Median time elapsed of 50 attempts: 4728.54 milliseconds
```

Figure 4: Testing PointCPDesign3.java

A table of the results

Table 2: Time in milliseconds

Method tested	**Time (milliseconds)**			
	Design 1 (PointCP)	Design 2 (PointCPDesign2)	Design 3 (PointCPDesign3)	Design 6 (PointCPDesign6)
getX()	0.98	24.92	2.82	N/A
getY()	3.94	633.6	0.0	N/A
getRho()	36.86	0.0	36.78	N/A
getTheta()	1142.44	0.0	1133.42	N/A
convertStorageToPolar()	33.72	71.22	1176.14	N/A
convertStorageToCartesian()	32.82	1231.4	69.7	N/A
getDistance(Point pointB)	630.0	3777.28	620.82	N/A
rotatePoint(double rotation)	4793.32	5148.74	4728.54	N/A

A discussion of the results (time in milliseconds).

Let compare Design 1 and Design 3 since both compute a Cartesian coordinate.

Although methods getX(), convertStorageToPolar() and convertStorageToCartesian() are faster for Design1, Design 3 is faster for all remaining methods, therefore validating the hypothesis made in question 5 (E26) concerning computation speed << Internal public methods may run faster>>.