uOttawa

L'Université canadienne
Canada's university

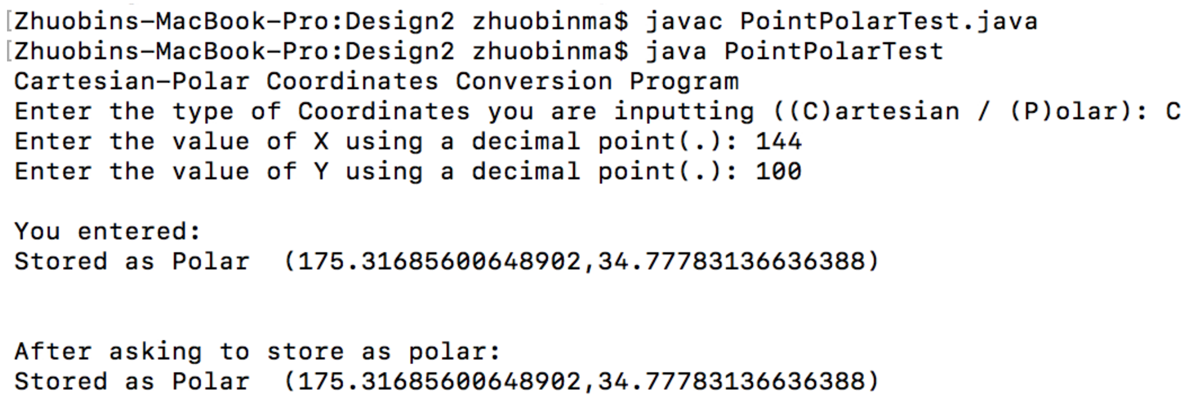
SEG2105
Assignment 1- Report

Name :	Student # :
Frédérick Gaudet	8035208
Zhuobin Ma	7469161

Laboratory date: 19 septembre, 2018
Rapport deadline: 30 septembre, 2018

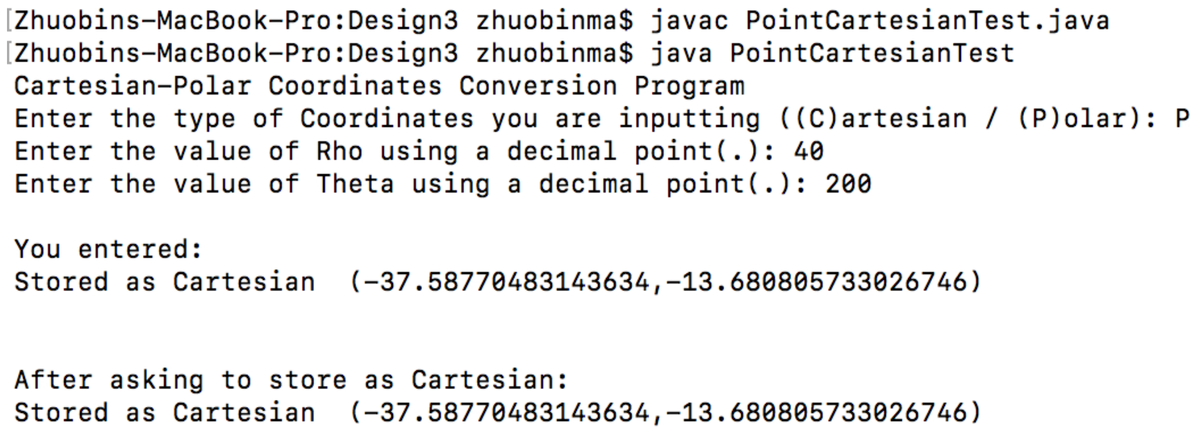
Point CP

Sample Test for Design 2



```
[Zhuobins-MacBook-Pro:Design2 zhuobinma$ javac PointPolarTest.java
[Zhuobins-MacBook-Pro:Design2 zhuobinma$ java PointPolarTest
 Cartesian-Polar Coordinates Conversion Program
 Enter the type of Coordinates you are inputting ((C)artesian / (P)olar): C
 Enter the value of X using a decimal point(.): 144
 Enter the value of Y using a decimal point(.): 100

 You entered:
 Stored as Polar  (175.31685600648902,34.77783136636388)


 After asking to store as polar:
 Stored as Polar  (175.31685600648902,34.77783136636388)
```

Sample Test for Design 3

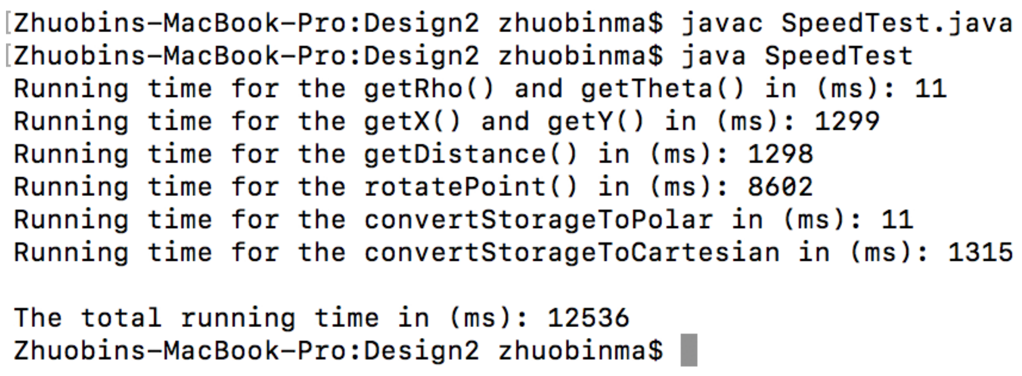


```
[Zhuobins-MacBook-Pro:Design3 zhuobinma$ javac PointCartesianTest.java
[Zhuobins-MacBook-Pro:Design3 zhuobinma$ java PointCartesianTest
 Cartesian-Polar Coordinates Conversion Program
 Enter the type of Coordinates you are inputting ((C)artesian / (P)olar): P
 Enter the value of Rho using a decimal point(.): 40
 Enter the value of Theta using a decimal point(.): 200

 You entered:
 Stored as Cartesian  (-37.58770483143634,-13.680805733026746)


 After asking to store as Cartesian:
 Stored as Cartesian  (-37.58770483143634,-13.680805733026746)
```

<u>Table for E26</u>

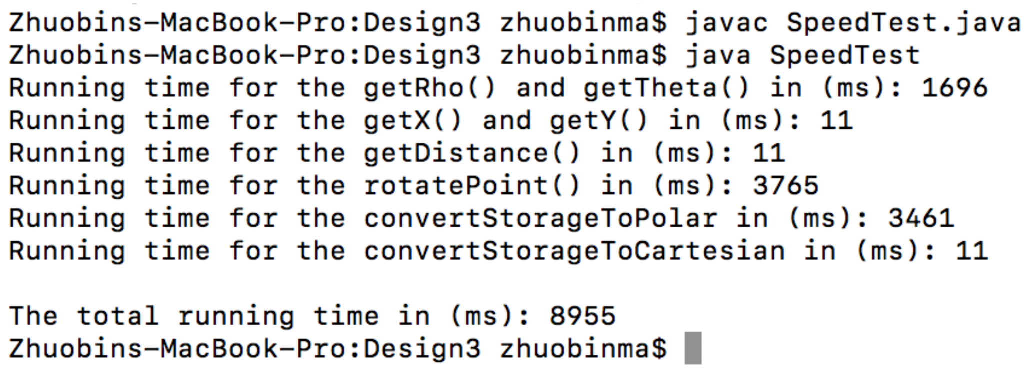
Design Type	**Advantage**	**Disadvantage**
Design 2 Store polar coordinate only	Only need to deal ***Char*** type in constructor, in most instance method only deals with two ***double*** variables. So, the code for some methods are simpler. Stored all data points in ***polar*** make any method that use polar type coordinate in Rho and Theta with less running time	Take more running time for methods that use cartesian type coordinate in X and Y.
Design 3 Store cartesian coordinate only	Only need to deal ***Char*** type in constructor, in most instance method only deals with two ***double*** variables. So, the code for some methods are simpler. Stored all data points in ***cartesian*** make any method that use cartesian type coordinate in X and Y with less running time	Take more running time for methods that use cartesian type coordinate in Rho and Theta.
Design 6 Interface with design 2 and 3 as class implementing it	Only need to deal ***Char*** type in constructor, in most instance method only deals with two ***double*** variables. So, the code for some methods are simpler. With help of interface, the running time for some methods will become faster than before with optimized code.	When deal with single type data in cartesian or polar will took longer time than both design 2 and design 3.

Sample SpeedTest for Design 2



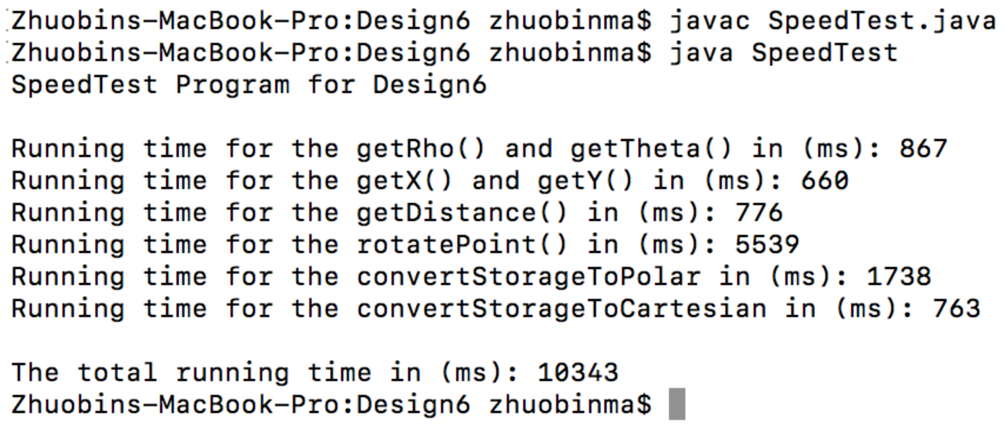
```
Zhuobins-MacBook-Pro:Design2 zhuobinma$ javac SpeedTest.java
Zhuobins-MacBook-Pro:Design2 zhuobinma$ java SpeedTest
Running time for the getRho() and getTheta() in (ms): 11
Running time for the getX() and getY() in (ms): 1299
Running time for the getDistance() in (ms): 1298
Running time for the rotatePoint() in (ms): 8602
Running time for the convertStorageToPolar in (ms): 11
Running time for the convertStorageToCartesian in (ms): 1315

The total running time in (ms): 12536
Zhuobins-MacBook-Pro:Design2 zhuobinma$
```

Sample SpeedTest for Design 3



```
Zhuobins-MacBook-Pro:Design3 zhuobinma$ javac SpeedTest.java
Zhuobins-MacBook-Pro:Design3 zhuobinma$ java SpeedTest
Running time for the getRho() and getTheta() in (ms): 1696
Running time for the getX() and getY() in (ms): 11
Running time for the getDistance() in (ms): 11
Running time for the rotatePoint() in (ms): 3765
Running time for the convertStorageToPolar in (ms): 3461
Running time for the convertStorageToCartesian in (ms): 11

The total running time in (ms): 8955
Zhuobins-MacBook-Pro:Design3 zhuobinma$
```

Sample SpeedTest for Design 6



```
Zhuobins-MacBook-Pro:Design6 zhuobinma$ javac SpeedTest.java
Zhuobins-MacBook-Pro:Design6 zhuobinma$ java SpeedTest
SpeedTest Program for Design6

Running time for the getRho() and getTheta() in (ms): 867
Running time for the getX() and getY() in (ms): 660
Running time for the getDistance() in (ms): 776
Running time for the rotatePoint() in (ms): 5539
Running time for the convertStorageToPolar in (ms): 1738
Running time for the convertStorageToCartesian in (ms): 763

The total running time in (ms): 10343
Zhuobins-MacBook-Pro:Design6 zhuobinma$
```

Description of SpeedTest

For each SpeedTest there are 20000000 data points in total. These data points are mixed with Polar and Cartesian coordinate. To make sure all three designs have consistent data points, the amount of Polar and Cartesian coordinate is in half-and-half. All data points are generating by **for loop** using the Math.random() and stored in **Array**.

After generated all the data points will use **for loop** to test each method. By setting start time and end time with System.currentTimeMillis() to get running time for each method in million second.

Answer for E28

The following comparison shows the efficiency for each design under different methods:
getRho() and getTheta(): Design 2 > Design 6 > Design 3
getX() and getY(): Design 3 > Design 6 > Design 2
getDistance(): Design 2 > Design 6 > Design 3
rotatePoint(): Design 2 > Design 6 > Design 3
convertStorageToPolar(): Design 3 > Design 6 > Design 2
convertStorageToCartesian(): Design 2 > Design 6 > Design 3
Total running time: Design 2 > Design 6 > Design 3

Table for E29

Sample running time for design 2:

	Trail 1(ms)	Trail 2 (ms)	Trail 3 (ms)	Trail 4 (ms)	Trail 5 (ms)
getRho() and getTheta()	11	10	11	10	10
getX() and getY()	1299	1336	1324	1382	1316
getDistance()	1298	1377	1377	1403	1368
rotatePoint()	8602	9012	8911	8872	8897
converStorageToPolar()	11	12	11	11	11
convertStorageToCartesian()	1315	1408	1399	1370	1337

Sample running time for design 3:

	Trail 1(ms)	Trail 2 (ms)	Trail 3 (ms)	Trail 4 (ms)	Trail 5 (ms)
getX() and getY()	1696	1719	1737	1738	1722
getRho() and getTheta()	11	12	12	11	13
getDistance()	11	11	13	12	13
rotatePoint()	3765	3810	3917	3741	3681
converStorageToPolar()	3461	3500	3510	3514	3383
convertStorageToCartesian()	11	10	10	12	16

Sample running time for design 6:

	Trail 1(ms)	Trail 2 (ms)	Trail 3 (ms)	Trail 4 (ms)	Trail 5 (ms)
getX() and getY()	867	954	857	838	839
getRho() and getTheta()	660	672	636	616	617
getDistance()	776	778	717	702	707
rotatePoint()	5539	5489	5488	5462	5428
converStorageToPolar()	1738	1724	1725	1707	1692
convertStorageToCartesian()	763	733	707	655	757

Table for E30

The following running time are average of 5 trails from tables above

	Design 2 (ms)	Design 3 (ms)	Design 6 (ms)
getRho() and getTheta()	10.4	1722.4	871
getX() and getY()	1331.4	11.8	640.2
getDistance()	1364.6	12	736
rotatePoint()	8750.8	3596.4	5481.2
converStorageToPolar()	11.2	3473.6	1717.2
convertStorageToCartesian()	1365.8	11.8	723
Total running time	123634.2	8828	101686

Conclusion

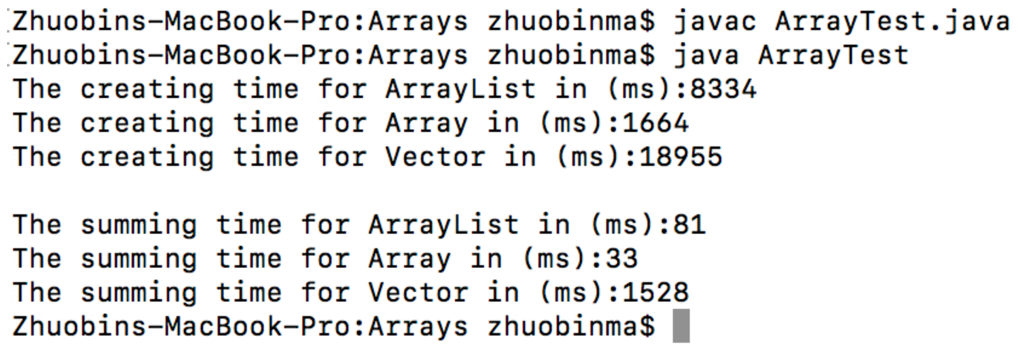
From average running time above, we can see **design6** don't have fastest total running time in 10.17s. But it balanced the running time between different methods with interface, the running time for all methods in design6 are second fast compare to other two designs.

Because the most methods are using cartesian coordinate data points directly, these make **design2** have the slowest total running time in 12.36s. The **design2** only have advantage in getRho(), getTheta() and covertStorageToPolar(). So **design3** have the fastest total running time in 8.83s, and design3 have advantage in getX(), getY(), getDistance(), rotatePoint() and convertStorageToCartesian().

Therefore, the result of SpeedTest is consistent with the table E26.

Array

Sample ArrayTest for ArrayList, Array and Vector



```
Zhuobins-MacBook-Pro:Arrays zhuobinma$ javac ArrayTest.java
Zhuobins-MacBook-Pro:Arrays zhuobinma$ java ArrayTest
The creating time for ArrayList in (ms):8334
The creating time for Array in (ms):1664
The creating time for Vector in (ms):18955

The summing time for ArrayList in (ms):81
The summing time for Array in (ms):33
The summing time for Vector in (ms):1528
Zhuobins-MacBook-Pro:Arrays zhuobinma$
```

Sample running time for ArrayList

	Trail 1 (ms)	**Trail 2 (ms)**	**Trail 3 (ms)**	**Trail 4 (ms)**	**Trail 5 (ms)**
Generate time	8834	8499	9017	8442	8548
Adding time	81	81	78	72	75

Sample running time for Array

	Trail 1 (ms)	**Trail 2 (ms)**	**Trail 3 (ms)**	**Trail 4 (ms)**	**Trail 5 (ms)**
Generate time	1664	1614	1671	1611	1675
Summing time	33	55	52	31	35

Sample running time for Vector

	Trail 1 (ms)	**Trail 2 (ms)**	**Trail 3 (ms)**	**Trail 4 (ms)**	**Trail 5 (ms)**
Generate time	18955	18778	19743	18373	18933
Summing time	1528	1597	1565	1519	1586

Average running time for ArrayList, Array and Vector

	ArrayList	Array	Vector
Generate time	8668	1647	189564
Summing time	77.4	41.2	1559

<u>Conclusion</u>

From the average running time table above in both generating and summing time the Array is the fastest. So under fixed data size the Array is best choice to store the data. Without the fixed data size, the ArrayList will be better choice with second fastest in both generating and summing time.