SEG2105 – Assignment 1 Written Answers

Yahya Saleh, 300063517
Yichen Liu, 300191840

E26 – Advantages & Disadvantages of Each Design

Design	Advantages	Disadvantages
PointCP	Efficient instance creation (simple variable assignments). Flexibility for users to change how coordinates are stored; useful when users wish to perform a series of operations which are more efficient in one coordinate system compared to the other (e.g., rotation is more efficient using the polar system).	Longer code and more complex logic required to handle storing coordinates in either system; harder to read and maintain. Any given coordinate retrieval (i.e., getter methods) may require computation. More memory use (`typeCoord` instance variable required).
PointCP2	Shorter and simpler code compared to PointCP; easier to read and maintain. Efficient retrieval of polar coordinates (simply returned). *Potentially* more efficient `rotatePoint()` method (if rotation angle is just added to existing angle). Less memory use (only two instance variables).	Inefficient instance creation when provided cartesian coordinates (conversion required). Inefficient retrieval of cartesian coordinates (always computed). Less efficient `getDistance()` method (cartesian coordinates must always be computed).
PointCP3	Shorter and simpler code compared to PointCP; easier to read and maintain. Efficient retrieval of cartesian coordinates (simply returned). Efficient `getDistance()` method (cartesian coordinates are efficiently retrieved). Less memory use (only two instance variables).	Inefficient instance creation when provided polar coordinates (conversion required). Inefficient retrieval of polar coordinates (always must be computed). *Potentially* less efficient `rotatePoint()` method (more computation required).
PointCP5	Minimal code duplication (`getDistance()`, `toString()` and logic for `rotatePoint()` collected in superclass); easier to maintain. Subclasses are the simplest and shortest of all designs; easier to read and maintain.	Potentially inefficient instance creation depending on how coordinates are provided (conversion may be required).

Note that, in terms of efficiency and memory-use, design 5's subclasses share all the same advantages and disadvantages of designs 2 and 3. Additionally, the advantage associated with design 2's `rotatePoint()` method is hypothetical (i.e., this method could be implemented more efficiently in this class); however, the code for all the classes submitted contain the same implementation for `rotatePoint()`, and so this advantage was not actually realized.

E28-30 – Performance Analysis of Each Design

The performance of the various designs was tested by first instantiating five point objects (corresponding to each of the designs) using the same random cartesian coordinates. For each object, each of its methods (excluding the storage conversion methods in `PointCP`) was separately called 5,000,000 times, and the elapsed time was measured. This process was repeated 10 times for each method. The maximum, minimum and average runtime (ms) to call each method 5,000,000 times is shown in the following table.

Note: some of the average runtimes are reported as 0 since long division truncates towards 0. Green cells indicate the best performing class at a given method.

Runtime (ms)	getX()	getY()	getRho()	getTheta()	getDistance()	rotatePoint()	toString()
PointCP							
Maximum	5	4	6	247	31	342	936
Minimum	0	0	0	171	29	264	844
Average	0	0	0	197	29	277	873
PointCP2							
Maximum	73	74	24	21	157	632	2044
Minimum	61	58	4	4	142	575	1961
Average	63	60	6	6	147	589	1992
PointCP3							
Maximum	26	27	26	182	30	278	1899
Minimum	14	14	19	171	25	261	1843
Average	16	17	20	176	26	269	1867
PointCP5 (design 2)							
Maximum	82	77	18	23	150	579	2143
Minimum	71	69	15	20	143	566	1960
Average	74	71	15	21	144	570	1988
PointCP5 (design 3)							
Maximum	40	29	30	225	44	276	1898
Minimum	25	27	24	170	36	263	1845
Average	33	27	26	182	28	268	1871

Some of the obtained results subvert expectations while others match them. Firstly, it was not expected that design 1 would be able to perform 5,000,000 cartesian retrievals in less than a millisecond (on average), leading to the belief that later trials were somehow optimized by the JVM. More iterations could have been performed to drive up the runtime, allowing for higher resolution performance data; however; the same number of iterations were used to test each method and so a greater number of iterations would cause the machine on which tests were performed to overheat. Design 1 was also the most performant at retrieving rho, despite the instance tested having its data stored in cartesian form.

As expected, design 2 had relatively slow cartesian retrieval but fast polar retrieval. Its retrieval of theta was the fastest amongst all designs; which makes sense as it was simply returned. Additionally, the designs that only stored polar coordinates were the worst performing at `getDistance()` and `rotatePoint()`, since the implementation of both methods required the retrieval of cartesian coordinates.

Design 3 had relatively fast cartesian retrieval, but slow polar retrieval, as expected. It was the most performant at `getDistance()`, marginally beating out design 1. Once again, this makes sense as this method relies on cartesian retrieval.

Finally, the design 5 subclasses were the least performant across the board with the exception of `rotatePoint()` for design 5-3. It appears that the added abstraction introduces a small amount of overhead to method execution, mildly decreasing efficiency in most cases.

Sample Output of Performance Test

```
*** PointCP method runtimes (ms) ***
getX():
Max: 5
Min: 0
Avg: 0

getY():
Max: 4
Min: 0
Avg: 0

getRho():
Max: 6
Min: 0
Avg: 0

getTheta():
Max: 247
Min: 171
Avg: 197

getDistance():
Max: 31
Min: 29
Avg: 29

rotatePoint():
Max: 342
Min: 264
Avg: 277

toString():
Max: 936
Min: 844
Avg: 873


*** PointCP2 method runtimes (ms) ***
getX():
Max: 73
Min: 61
Avg: 63
```

```
getY():
Max: 74
Min: 58
Avg: 60

getRho():
Max: 24
Min: 4
Avg: 6

getTheta():
Max: 21
Min: 4
Avg: 6

getDistance():
Max: 157
Min: 142
Avg: 147

rotatePoint():
Max: 632
Min: 575
Avg: 589

toString():
Max: 2044
Min: 1961
Avg: 1992


*** PointCP3 method runtimes (ms) ***
getX():
Max: 26
Min: 14
Avg: 16

getY():
Max: 27
Min: 14
Avg: 17

getRho():
Max: 26
```

```
Min: 19
Avg: 20

getTheta():
Max: 182
Min: 171
Avg: 176

getDistance():
Max: 30
Min: 25
Avg: 26

rotatePoint():
Max: 278
Min: 261
Avg: 269

toString():
Max: 1899
Min: 1843
Avg: 1867


*** PointCP5 (design 2) method runtimes (ms) ***
getX():
Max: 82
Min: 71
Avg: 74

getY():
Max: 77
Min: 69
Avg: 71

getRho():
Max: 18
Min: 15
Avg: 15

getTheta():
Max: 23
Min: 20
Avg: 21
```

```
getDistance():
Max: 150
Min: 143
Avg: 144

rotatePoint():
Max: 579
Min: 566
Avg: 570

toString():
Max: 2143
Min: 1960
Avg: 1988


*** PointCP5 (design 3) method runtimes (ms) ***
getX():
Max: 40
Min: 25
Avg: 33

getY():
Max: 29
Min: 27
Avg: 27

getRho():
Max: 30
Min: 24
Avg: 26

getTheta():
Max: 225
Min: 170
Avg: 182

getDistance():
Max: 44
Min: 36
Avg: 38

rotatePoint():
Max: 276
Min: 263
```

```
Avg: 268

toString():
Max: 1898
Min: 1845
Avg: 1871
```