E26

|  | Simplicity | Efficiency w/ Creating instances | Efficiency when doing computations that require both | Memory Used | notes |
| --- | --- | --- | --- | --- | --- |
| Design 2 | Both are rather simple, a flag determines if computation is needed which is triggered upon returning as a string. | Very efficient, uses minimum (2) variables for coordinates + 1 variable for the flag that determines output. No initial computation needed to “translate” inputs. | 2 Computations needed whenever transitioning between coordinate formats. Not terrible but not ideal. | Both require only 4 (3 in original implementation) variables for the majority of the code, memory used is minimal. | Both are nearly identical, the stored value is simply mirrored. Thus, the 4 columns in my opinion should be about the same, as getting one value from the other is about the same. That said, Design 3 has 1 less mathematical computation when getting theta/rho (3 of them vs. 4) so it is slightly faster overall. |
| Design 3 |
| Design 5 | A little more complicated, uses 2 subclasses for C and P coordinates | Slightly more inefficient, essentially creates 2(Subclasses)\*4(variables) variables to work with | No computations required - it’s already there to be accessed. | Requires more memory due to the created subclasses |  |

|  |  | PointCP2 | PointCP3 | PointCP5 |
| --- | --- | --- | --- | --- |
| Generate Point |  |  |  |  |
|  | Collected Numbers | 219657+286353+293184+237983+225657 | 212190+262628+283382+291573+266275 | 236120+291107+281277+244661+300133 |
|  | Median | 252566.8 | 263209.6 | 270659.6 |
|  | Avrg Runtime(ms) | 25.25668 | 26.32096 | 27.06596 |
|  | Highest # generated | 293184 | 291573 | 300133 |
|  | Lowest # generated | 219657 | 212190 | 236120 |
| Convert To Cartesian |  |  |  |  |
|  | Collected Numbers | 175839+18  2815+176083+183296+163592 | 168015+178659+188599+173650+207791 | 254337+223791+252225+249718+230330 |
|  | Median | 176325 | 183342.8 | 242080.2 |
|  | Avrg Runtime(ms) | 17.6325 | 18.33428 | 24.20802 |
|  | Highest # generated | 183296 | 207791 | 252225 |
|  | Lowest # generated | 163592 | 168015 | 253791 |
| Convert To Polar |  |  |  |  |
|  | Collected Numbers | 185106+193011+176879+173639+174979 | 170455+178981+163865+193248+168032 | 202944+224696+188318+242959+209748 |
|  | Median | 180722.8 | 174916.2 | 213733 |
|  | Avrg Runtime(ms) | 18.07228 | 17.49162 | 21.3733 |
|  | Highest # generated | 193011 | 193248 | 242959 |
|  | Lowest # generated | 173639 | 163865 | 202944 |
| getDistance |  |  |  |  |
|  | Collected Numbers | 163924+151464+188173+197637+177661 | 164559+177211+176920+149520+150093 | 221945+214922+200384+240258+208726 |
|  | Median | 175771.8 | 163660.6 | 217247 |
|  | Avrg Runtime(ms) | 17.57718 | 16.36606 | 21.7247 |
|  | Highest # generated | 197637 | 177211 | 240258 |
|  | Lowest # generated | 151464 | 149520 | 200384 |
| rotatePoint |  |  |  |  |
|  | Collected Numbers | 164761+176818+163749+197981+191927 | 201546+188303+178030+150034+162944 | 225609+243862+245700+203541+247603 |
|  | Median | 179047.2 | 176171.4 | 233263 |
|  | Avrg Runtime(ms) | 17.90472 | 17.61714 | 23.3263 |
|  | Highest # generated | 197981 | 201546 | 247603 |
|  | Lowest # generated | 163749 | 150034 | 203541 |

All given 10 seconds to generate and test as many points as possible.

Each number is # of times the indicated activity was completed (alongside initial point generation)

Each pointCP class + method was tested 5 times.   
  
Note: Due to unknown reasons, processing efficiency of java seemed to improve right after testing of PointCP3, meaning that PointCP5’s results seemed extremely boosted. They are still kept in consideration for general trends, and have been adjusted by approximately the difference in performance of PointCP2 and 3 after the change.  
  
These results seem to verify my initial hypothesis about the comparable performance between CP2 and 3,while each is quicker when converting to their own type for evident reasons. Surprisingly, CP5 performed much quicker in the startup efficiency than expected, however all of the relative performances in relation to the startup numbers were as predicted - a higher number of points were able to be generated and a method was able to be tested much quicker than the other two classes. This may be due to my description of efficiency of computations column of CP5.