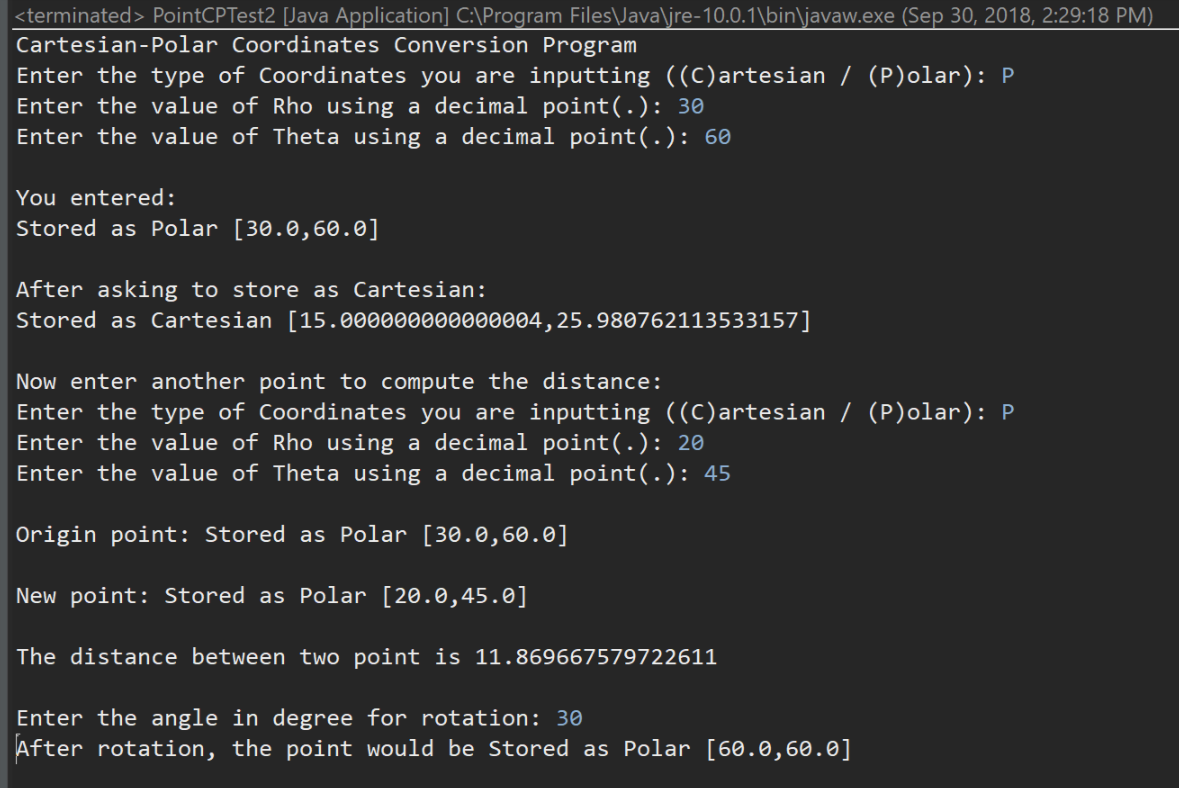
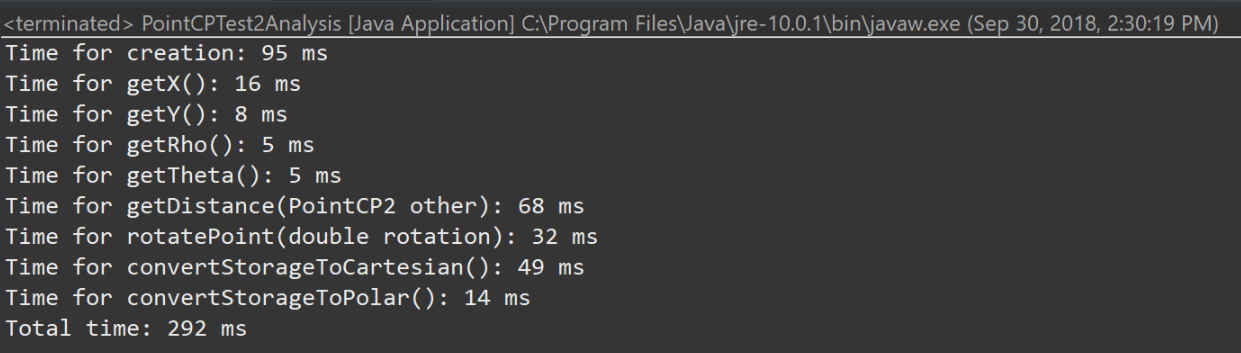
Test Cases and Test Analysis

Design 2:

Test Case:

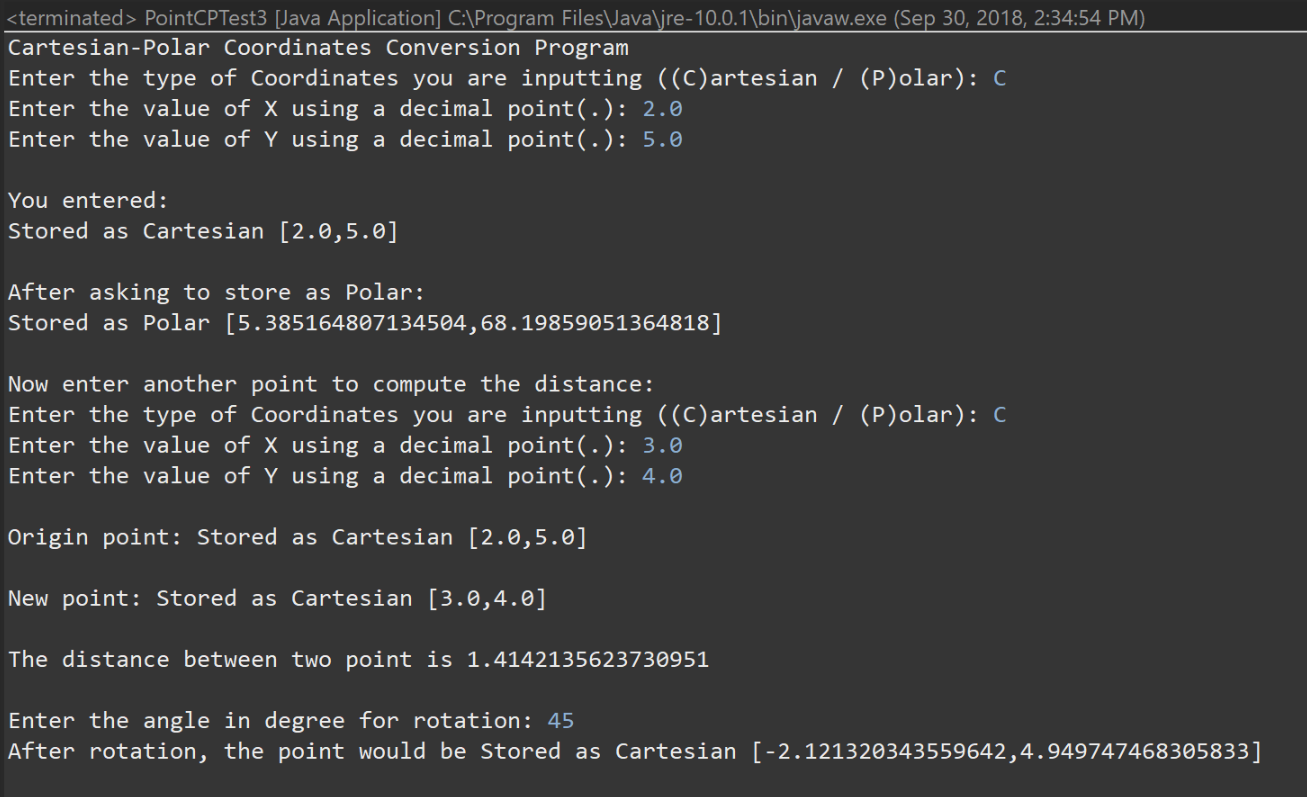


Test Analysis:

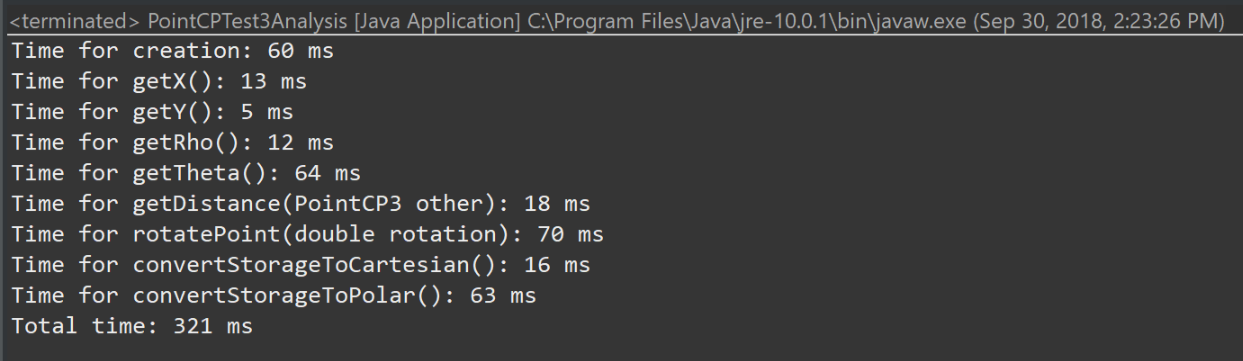


Design 3:

Test Case:

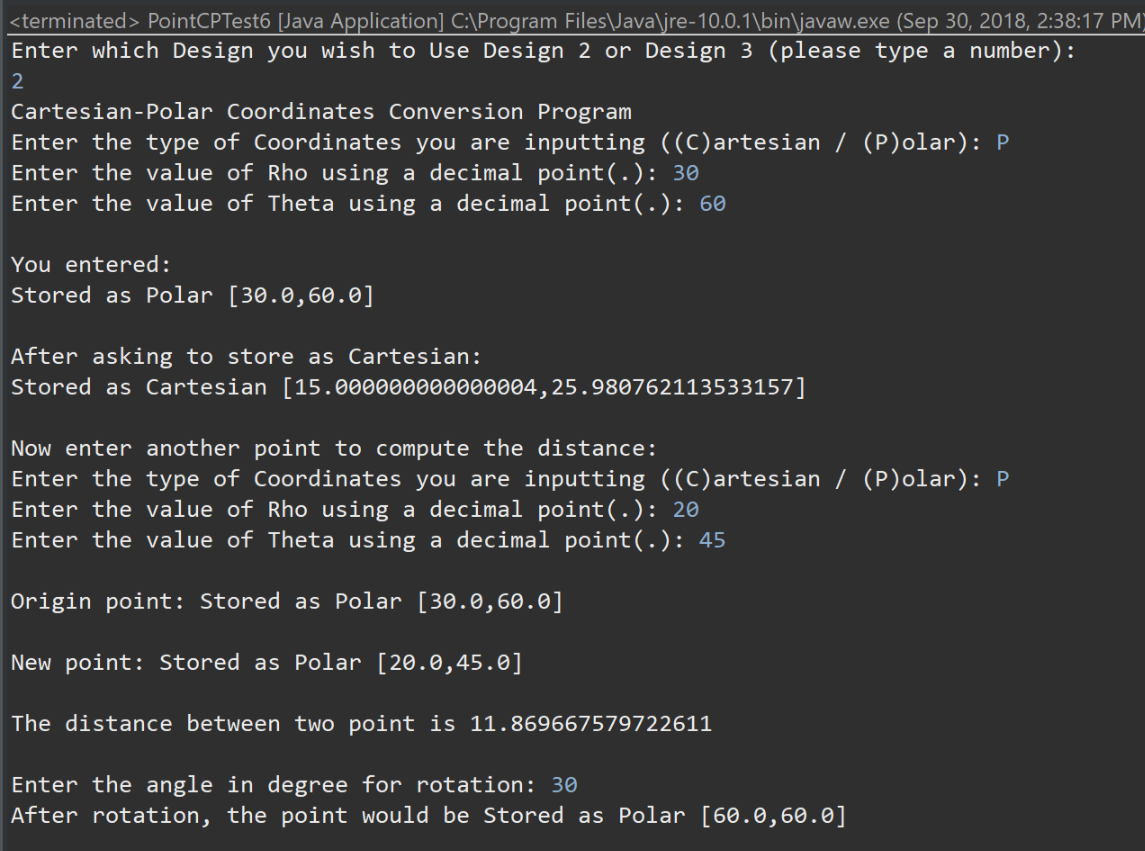


Test Analysis:

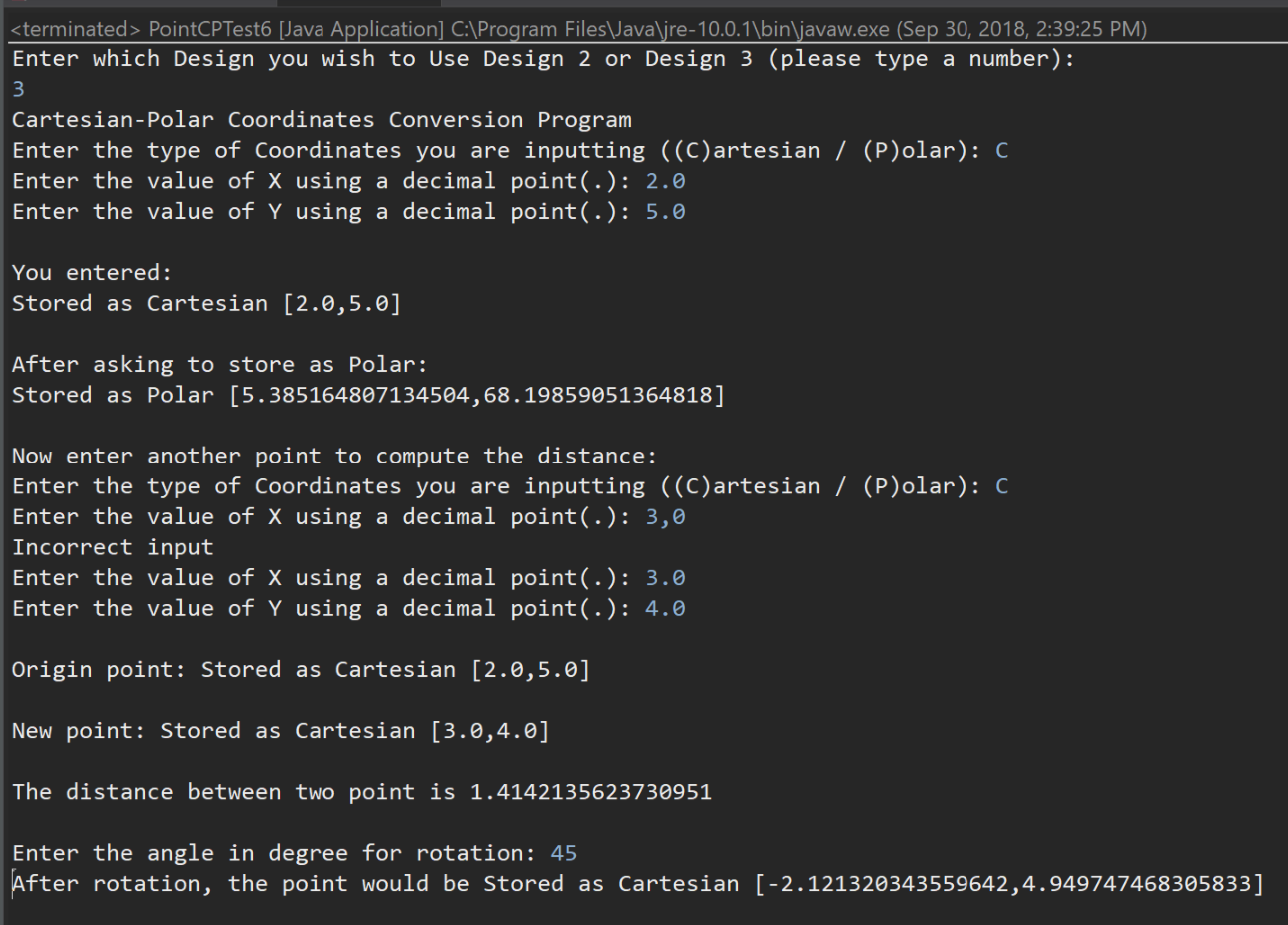


Design 6:

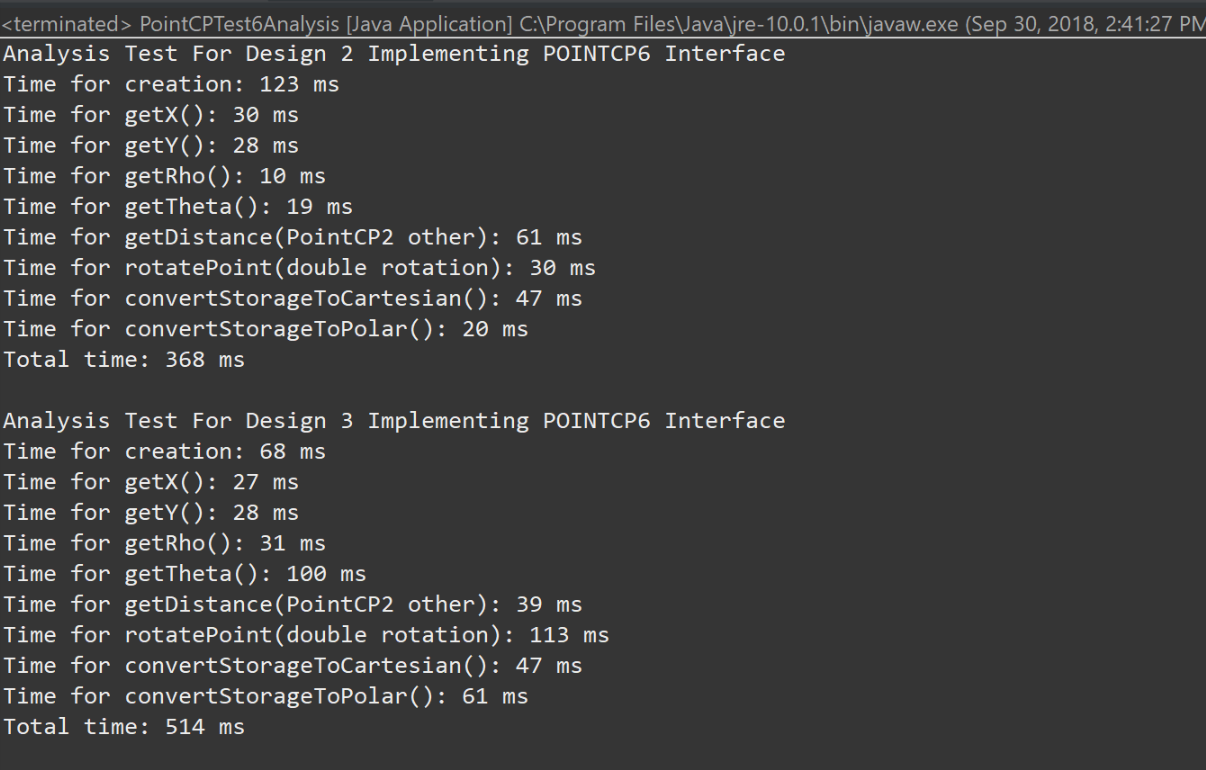
Test Case 1 Design 2:



Test Case 2 Design 3:



Test Analysis:



E26: Pros and Cons of Different PointCP Designs.

|  |  |  |
| --- | --- | --- |
|  | Pros | Cons |
| Design 2 | * Uses less memory to store instances compared to design 1 where you would have to Store one type of coordinates using a single pair of instance variables, with a flag indicating which type is stored * rotatePoint method is much efficient * efficient creation of point when using polar arguments | * worse performance when getting and computing cartesian coordinates * Creating points using cartesian coordinates is less efficient * getDistance method requires more calculations and takes more time processor power |
| Design 3 | * Uses less memory to store instances compared to design 1 where you would have to Store one type of coordinates using a single pair of instance variables, with a flag indicating which type is stored * Efficient creation of points when using cartesian arguments * GetDistance method is more efficient | * rotatePoint method requires more calculations and takes more time and processor power * performance is worse when getting/computing polar coordinates |
| Design 6 | * Good Software design idea | * Code is complex because of the interface and concrete classes |

E28:

Run a performance analysis in which you compare the performance of Design 5, as you implemented it in the previous exercise, with Design 1. Determine the magnitude of the differences in efficiency, and verify the hypotheses you developed in E26.

Answer:

Creation: Design 3 > Design 6(3) > Design 6(2) >Design 2

getX(): Design 3 > Design 6(2) > Design 2 > Design 6(3)

getY(): Design 3 > Design 6(2) > Design 2 > Design 6(3)

getRho(): Design 6(2) > Design 2 > Design 3 > Design 6(3)

getTheta(): Design 2 > Design 6(2) >Design 3 > Design 6(3)

getDistance(PointCP other): Design 3 > Design 6(3) > Design 6(2) > Design 2

rotatePoint(double rotation): Design 2 > Design 6(2) > Design 3 > Design 6(3)

convertStorageToCartesian(): Design 3 > Design 2 > Design 6(2) > Design 6(3)

convertStorageToPolar(): Design 6(2) > Design 2 > Design 3 > Design 6(3)

The results for performance testing are basically same as what we expected in E26 pros and cons. We only discovered slight differences caused by the java execution speed.

E29:

To run a performance analysis, you will have to create a new test class that randomly generates large numbers of instances of PointCP, and performs operations on them, such as retrieving polar and Cartesian coordinates. You should then run this test class with the two versions of PointCP – Design 1 and Design 5.

Answer:

We have run tests for each design and calculated the maximum, minimum and the average of them implementation time.

Raw data Design 2:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Time (ms) | | | | | | | |
| Methods | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | **Maximum** | **Minimum** | **Average** |
| Creation | 95 | 99 | 97 | 95 | 95 | **99** | **95** | **96.2** |
| getX() | 20 | 27 | 25 | 22 | 21 | **27** | **20** | **23** |
| getY() | 20 | 22 | 23 | 20 | 25 | **25** | **20** | **22** |
| getRho() | 17 | 16 | 13 | 19 | 17 | **19** | **13** | **16.4** |
| getTheta() | 8 | 8 | 7 | 6 | 8 | **8** | **6** | **7.4** |
| getDistance  (PointCP other) | 86 | 72 | 68 | 72 | 69 | **86** | **68** | **73.4** |
| rotatePoint (double rotation) | 183 | 187 | 187 | 185 | 186 | **187** | **185** | **185.6** |
| convertStorageToCartesian() | 50 | 44 | 44 | 41 | 48 | **50** | **41** | **45.4** |
| convertStorageToPolar() | 21 | 19 | 18 | 21 | 33 | **33** | **18** | **22.4** |
| Total Time | 500 | 494 | 482 | 481 | 502 | **502** | **481** | **491.8** |

Raw data Design 3:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Time (ms) | | | | | | | |
| Methods | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | **Maximum** | **Minimum** | **Average** |
| Creation | 57 | 60 | 61 | 60 | 58 | **57** | **61** | **59.2** |
| getX() | 14 | 15 | 16 | 16 | 14 | **14** | **16** | **15** |
| getY() | 4 | 17 | 8 | 4 | 7 | **17** | **4** | **40** |
| getRho() | 19 | 5 | 18 | 17 | 14 | **19** | **5** | **14.6** |
| getTheta() | 70 | 71 | 67 | 65 | 70 | **71** | **65** | **68.6** |
| getDistance  (PointCP other) | 14 | 13 | 14 | 17 | 13 | **17** | **13** | **14.2** |
| rotatePoint (double rotation) | 85 | 74 | 77 | 72 | 78 | **85** | **72** | **77.2** |
| convertStorageToCartesian() | 18 | 22 | 27 | 29 | 20 | **29** | **18** | **23.2** |
| convertStorageToPolar() | 73 | 71 | 67 | 68 | 75 | **75** | **67** | **70.8** |
| Total Time | 354 | 348 | 355 | 348 | 349 | **355** | **348** | **350.8** |

Design 6(2):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Time (ms) | | | | | | | |
| Methods | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | **Maximum** | **Minimum** | **Average** |
| Creation | 92 | 103 | 100 | 98 | 104 | **104** | **92** | **99.4** |
| getX() | 16 | 22 | 19 | 17 | 22 | **22** | **16** | **19.2** |
| getY() | 8 | 9 | 13 | 10 | 8 | **13** | **8** | **9.6** |
| getRho() | 8 | 9 | 22 | 10 | 8 | **22** | **8** | **11.4** |
| getTheta() | 17 | 16 | 8 | 17 | 16 | **17** | **8** | **14.8** |
| getDistance  (PointCP other) | 60 | 86 | 83 | 65 | 60 | **86** | **60** | **70.8** |
| rotatePoint (double rotation) | 31 | 37 | 33 | 33 | 33 | **37** | **31** | **33.4** |
| convertStorageToCartesian() | 48 | 69 | 54 | 51 | 53 | **69** | **48** | **55** |
| convertStorageToPolar() | 17 | 36 | 29 | 19 | 20 | **36** | **17** | **24.6** |
| Total Time | 297 | 387 | 361 | 320 | 324 | **387** | **297** | **337.8** |

Design 6(3):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Time (ms) | | | | | | | |
| Methods | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | **Maximum** | **Minimum** | **Average** |
| Creation | 61 | 95 | 65 | 66 | 65 | **95** | **61** | **70.4** |
| getX() | 26 | 43 | 36 | 37 | 39 | **43** | **26** | **36.2** |
| getY() | 28 | 44 | 31 | 38 | 33 | **44** | **28** | **34.8** |
| getRho() | 27 | 51 | 35 | 32 | 31 | **51** | **27** | **35.2** |
| getTheta() | 88 | 107 | 105 | 88 | 89 | **107** | **88** | **95.4** |
| getDistance  (PointCP other) | 30 | 47 | 46 | 34 | 37 | **47** | **30** | **38.8** |
| rotatePoint (double rotation) | 139 | 133 | 120 | 129 | 124 | **139** | **120** | **129** |
| convertStorageToCartesian() | 56 | 41 | 42 | 44 | 40 | **56** | **40** | **44.6** |
| convertStorageToPolar() | 62 | 62 | 59 | 63 | 64 | **64** | **59** | **62** |
| Total Time | 517 | 623 | 539 | 531 | 522 | **623** | **517** | **546.4** |

E30:

Summarize your results in a table: the columns of the table would be the two designs; the rows of the table would be the operations. The values reported in the table would be the average computation speed. Make sure you explain your results

Answer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Average Time (ms) | | | |
| Operation | Design 2 | Design 3 | Design 6(2) | Design 6(3) |
| Creation | 96.2 | 59.2 | 99.4 | 70.4 |
| getX() | 23 | 15 | 19.2 | 36.2 |
| getY() | 22 | 40 | 9.6 | 34.8 |
| getRho() | 16.4 | 14.6 | 11.4 | 35.2 |
| getTheta() | 7.4 | 68.6 | 14.8 | 95.4 |
| getDistance  (PointCP other) | 73.4 | 14.2 | 70.8 | 38.8 |
| rotatePoint (double rotation) | 185.6 | 77.2 | 33.4 | 129 |
| convertStorageToCartesian() | 45.4 | 23.2 | 55 | 44.6 |
| convertStorageToPolar() | 22.4 | 70.8 | 24.6 | 62 |
| Total Time | 491.8 | 350.8 | 337.8 | 546.4 |

Discussion:

The values in the table are the average time over many iterations for each operation, in milliseconds. The fastest methods were the getters in Design 2 and Design 3, which is expected because the values were simply returned from memory. Since Java’s trigonometry functions are slower than the square root and power functions, the getters for polar coordinates in Design 3 were slower than the getters for Cartesian coordinates in Design 2. Overall the most efficient design is Design 2.