SEG 2105 Assignment 1

Peter Bou-Farah – 300295193
Adam Barefoot – 300311556

Oct 4, 2023

E26 - PROS AND CONS OF DESIGNS

This section discusses the pros and cons of the five designs for the PointCP class.

Table 1. Pros and Cons of Different Designs

Design	Summary	Pros	Cons
Design 1	Least efficient overall.	- Simple. - Just a pair of coordinates and a flag, so instances don't take a lot of resources to create. - Memory is saved by storing only one pair of coordinates.	- Not great at converting coordinates - Not good for frequently switching between different types of coordinates when trying to get coordinates that aren't stored as they then have to be computed.
Design 2	Very efficient when working only with polar coordinates.	- Focuses on polar coordinates so isn't as long. - Only initializes for polar coordinates, making initialization more efficient.	- Only stores polar coordinates - Not good at computing cartesian coordinates
Design 3	Very efficient when working with cartesian coordinates	- Essentially same as above but for cartesian coordinates. - Only initializes cartesian coordinates, which makes its initialization process quicker.	- Only stores cartesian coordinates - Not good at computing polar coordinates
Design 4	Best for retrieving data but worst for altering it.	- Can initialize both types of coordinates - Good for switching between coordinate forms as it has direct access to both (doesn't need to compute them every time). - Doesn't need conversion method as it stores both types of coordinates	- Modifications are slow as it needs to keep both forms up to date (if it changes one and not the other, the pairs won't match anymore) - Uses lots of memory
Design 5	Most	- Code is separated into	- Conversions need more

	modular and able for expansion	more classes making it more modular. - Has potential to implement different coordinate systems.	calculations than other designs. - Doesn't have runtime measurements. - Subclasses are required to provide implementations for the abstract methods. - Inconsistent implementation of conversion methods in subclasses

E28 - DESIGN PERFORMANCE ANALYSIS

In this section, we analyzed the performances of all the designs. The same table is shown below in E30 as Table 3.

	Runtimes (ms) for 100 M Iterations			
Operation	Design 1 starting as type 'P'	Design 1 starting as type 'C'	Design 2	Design 3
convertStorageToCartesian()	49.5	16.5	197.0	6.3
convertStorageToPolar()	4.1	106.2	4.0	226.1
getX()	51.0	11.0	47.4	10.9
getY()	43.0	5.3	37.9	5.6
getTheta()	3.8	201.1	8.3	198.5
getRho()	14.9	7.8	13.0	8.0
getDistance(point)	159.0	17.6	94.6	23.1
rotatePoint(50)	86.8	34.4	97.3	24.5

Function	1st	2nd	3rd	4th
convertStorageToCartesian()	D3	DC	DP	D2
ConvertStorageToPolar()	D2 = DP		DC	D3
getX()	DC = D3		D2	DP
getY()	DC = D3		D2	DP
getTheta()	DP	D2	D3	DP
getRho()	D3	DC	D2	DP
getDistance(point)	DC	D3	D2	DP
rotatePoint(50)	D3	DC	DP	D2

*Design 1 P = DP, Design 1 C = DC, Design 2 = D2, Design 3 = D3

E29 - PERFORMANCE ANALYSIS: COMPARING DESIGN

We compared the designs and received the following data seen in Table 2.

Table 2. Median, Minimum, and Maximum Runtimes for Different Designs

Design	Median (ms)	Min (ms)	Max (ms)
Design 1 Polar Runtime	500	300	50302100
Design 1 Cartesian Runtime	500	300	6588500
Design 2 Runtime	600	200	7832200
Design 3 Runtime	800	600	12522000

The analysis shows that polar and cartesian designs had similar median and min runtimes, but very different max times.

E30 - SUMMARY OF ANALYSIS OF DESIGNS

Table 3. Performance of Various Functions with Different Designs

	Runtimes (ms) for 100 M Iterations			
Operation	Design 1 starting as type ‘P’	Design 1 starting as type ‘C’	Design 2	Design 3
convertStorageToCartesian()	49.5	16.5	197.0	6.3
convertStorageToPolar()	4.1	106.2	4.0	226.1
getX()	51.0	11.0	47.4	10.9
getY()	43.0	5.3	37.9	5.6
getTheta()	3.8	201.1	8.3	198.5
getRho()	14.9	7.8	13.0	8.0
getDistance(point)	159.0	17.6	94.6	23.1
rotatePoint(50)	86.8	34.4	97.3	24.5

DISCUSSION OF TESTING PROCEDURE:

Firstly, we ran a performance analysis test for designs 1, 2 and 3. In this performance analysis, we our methods in this specific order: : convertStorageToCartesian(), convertStorageToPolar(), convertStorageToCartesian() and finally, convertStorageToPolar(). Doing this made sure that design 1 included the runtime for the conversion between cartesian and polar coordinates.

The data table above summarizes the runtime performance of design 1 and design 3 for a multitude of operations. The findings show that the decision between these designs is based on the particular usage and operation needs.

- convertStorageToCartesian: design 3 (D3) is the most efficient
- convertStorageToPolar: design 2 and design 1P are the most efficient in polar coordinate operations

- getX **AND** getY: design 1C and design 3 perform equally well
- getTheta and getRho: design 1P excels in retrieving polar coordinates
- getDistance(point): design 1C is the most efficient in calculating the distance between points
- rotatePoint(50): design 3 is the most efficient for the rotating points

The strengths and weaknesses of each design discovered in this experiment provide an understanding of what each design excels at.

These results provide valuable insights into the strengths and weaknesses of each design for different use cases, assisting in informed decision-making during software development.

PART 2:

Table 4. Performance of Various Data Types with Several Trial Sizes

Trial Size:	15000000	10000000	500000
ArrayList	37149200 ns	26411900 ns	14977100 ns
Vector	104143000 ns	117469100 ns	19011600 ns
Array	13788000 ns	9346000 ns	8361200 ns

This shows that ArrayList is usually the slowest, and that Array tends to be the quickest. However, from a programmer's perspective, vectors and ArrayLists still have their advantages. They are easier to extend and make them a more flexible data type; it is quite easy to extend the size of an ArrayList or a Vector to accommodate extra elements. However, the array likely outperformed here because we didn't evaluate it on its ability to handle size changes, only it's ability to access values.