Jason Phung (300311115)
Ben Miller (300297574)

PointCP

E26 Pros/cons of each of the five design types

	Pros	Cons
Design 1: Store one type of coordinates using a single pair of instance variables, with a flag indicating which type is stored	- The user is able to pick between storing Cartesian and polar coordinates - Less memory usage. only a pair of instance variables and a flag will need to be stored	- Any use of the value in the non-selected type of coordinates will require conversion
Design 2: Store polar coordinates only	- Polar coordinates are ready for use without further computation - Less memory usage, no flag needs to be stored	- Cartesian coordinates cannot be stored and need to be computed for each use - Instance creation is slower if the input is Cartesian as conversion will be necessary
Design 3: Store Cartesian coordinates only	- Cartesian coordinates are ready for use without further computation - Less memory usage, no flag needs to be stored	- Polar coordinates cannot be stored and need to be computed for each use - Instance creation is slower if the input is polar as conversion will be necessary
Design 4: Store both types of coordinates, using four instance variables	- Both types of coordinates are ready for use without further computation	- Higher memory usage as both types of coordinates will need to be stored
Design 5: Abstract superclass with designs 2 and 3 as subclasses	- Abstraction allows easier maintenance	- Higher memory usage as separate files will be required for designs 2 and 3

E30 (average computation speeds) Over 100 000 000 runs, all time is in nanoseconds (ns)

Operations	Design 2	Design 3	Design 5-2	Design 5-3
convertStorageToPolar()	31.35	35.185	33.253	34.245
convertStorageToCartesian()	32.2	35.752	34.959	36.021
getDistance()	164.776	43.418	52.163	70.162
rotatePoint()	143.515	126.898	121.366	123.683

Test Analysis

The tests were conducted by using one ‘testRunTime()’ method. This method would individually run each method for each class *n* times (for my tests I used n=100 000 000). It would keep track of how much time each method call lasted, and after it would display which method was run, and the average runtime (in nanoseconds). This method was used on all four tested classes a total of ten times (for a total of 40 calls).

Test Example

Here is an example of the test called on class PointCP2
POINTCP2 TEST

// Run 0
convertStorageToPolar() -> The average runtime was : 40.788853 ns
convertStorageToCartesian() -> The average runtime was : 42.376557 ns
getDistance(PointCP2 pointB) -> The average runtime was : 235.495285 ns
rotatePoint(double rotation) -> The average runtime was : 197.885481 ns
// Run 1
convertStorageToPolar() -> The average runtime was : 35.798362 ns
convertStorageToCartesian() -> The average runtime was : 40.1306 ns
getDistance(PointCP2 pointB) -> The average runtime was : 240.606789 ns
rotatePoint(double rotation) -> The average runtime was : 181.740023 ns
// Run 2
convertStorageToPolar() -> The average runtime was : 31.008422 ns
convertStorageToCartesian() -> The average runtime was : 32.147166 ns
getDistance(PointCP2 pointB) -> The average runtime was : 224.229455 ns
rotatePoint(double rotation) -> The average runtime was : 155.939757 ns
// Run 3
convertStorageToPolar() -> The average runtime was : 29.7713 ns
convertStorageToCartesian() -> The average runtime was : 30.77322 ns
getDistance(PointCP2 pointB) -> The average runtime was : 168.793983 ns
rotatePoint(double rotation) -> The average runtime was : 142.661382 ns
// Run 4
convertStorageToPolar() -> The average runtime was : 30.600831 ns
convertStorageToCartesian() -> The average runtime was : 31.614624 ns
getDistance(PointCP2 pointB) -> The average runtime was : 161.933488 ns
rotatePoint(double rotation) -> The average runtime was : 134.180384 ns
// Run 5
convertStorageToPolar() -> The average runtime was : 29.173779 ns
convertStorageToCartesian() -> The average runtime was : 32.046028 ns
getDistance(PointCP2 pointB) -> The average runtime was : 165.510874 ns
rotatePoint(double rotation) -> The average runtime was : 127.807905 ns

// Run 6
convertStorageToPolar() -> The average runtime was : 31.827843 ns
convertStorageToCartesian() -> The average runtime was : 33.551687 ns
getDistance(PointCP2 pointB) -> The average runtime was : 171.215546 ns
rotatePoint(double rotation) -> The average runtime was : 126.781035 ns
// Run 7
convertStorageToPolar() -> The average runtime was : 28.666091 ns
convertStorageToCartesian() -> The average runtime was : 32.660152 ns
getDistance(PointCP2 pointB) -> The average runtime was : 159.52247 ns
rotatePoint(double rotation) -> The average runtime was : 126.980937 ns
// Run 8
convertStorageToPolar() -> The average runtime was : 30.018557 ns
convertStorageToCartesian() -> The average runtime was : 31.559838 ns
getDistance(PointCP2 pointB) -> The average runtime was : 160.77176 ns
rotatePoint(double rotation) -> The average runtime was : 131.555704 ns
// Run 9
convertStorageToPolar() -> The average runtime was : 29.878533 ns
convertStorageToCartesian() -> The average runtime was : 32.165643 ns
getDistance(PointCP2 pointB) -> The average runtime was : 159.308066 ns
rotatePoint(double rotation) -> The average runtime was : 125.051415 ns

Results

ConvertStorageToPolar:
Every class was very fast with this method. The fastest was PointCP2 at 31.35ns on average. PointCP3, PointCP3-5, and PointCP2-5 were 4ns, 2ns, and 3ns slower respectively.

ConvertStorageToCartesian:
The results were very similar again, PointCP2 was the fastest again, however with this method, PointCP3-5 was the slowest at 4ns slower than PointCP2 on average

getDistance:
PointCP2 was by far the slowest for this method, at an average of 165ns. Design 3 was the fastest here with an average of 40ns. The subclasses of PointCP5 were in the middle, at 52ns and 70ns.

rotatePoint:
PointCP2 was the slowest here again at 143ns. The other three classes were very close to their runtime, at about 120ns each. PointCP2-5 was the fastest with this method at 121ns.

Part 2: Arrays

	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Min. run time	Max. run time	Median run time
Array (population)	2.644	2.634	2.666	2.649	2.626	2.673	2.709	2.69	2.649	2.66	2.626	2.709	2.6545
Array (iteration)	0.141	0.142	0.142	0.141	0.142	0.144	0.142	0.141	0.141	0.141	0.141	0.144	0.1415
ArrayList (population)	15.581	12.511	12.549	13.215	12.442	13.255	13.034	12.499	12.981	12.276	12.276	15.581	12.765
ArrayList (iteration)	0.532	0.526	0.543	0.533	0.53	0.529	0.532	0.528	0.53	0.533	0.526	0.543	0.531
Vector (population)	4.221	4.267	4.406	4.333	4.249	4.274	4.25	4.258	4.263	4.257	4.221	4.406	4.2605
Vector (iteration)	1.369	1.537	1.321	1.326	1.323	1.322	1.321	1.325	1.326	1.315	1.315	1.537	1.324

Performance comparison of Arrays, ArrayLists, and Vectors, with a test size of 600000000 (measured in seconds)

Results

Using an Array always resulted in the lowest run time for both populating and iterating through, with a median populating run time of 2.6545 seconds and a median iterating run time of 0.1415 seconds.

Using ArrayLists, the median populating run time for ArrayLists was 12.765 seconds and the median iterating run time was 0.531 seconds.

Using Vectors, the median populating run time for Vectors was 4.2605 seconds and the median iterating run time was 1.324 seconds.

Comparing Arrays and ArrayLists, Arrays were ≈79.2% faster in populating and ≈73.3% faster in iterating.

Comparing Arrays and Vectors, Arrays were ≈37.6% faster in populating and ≈89.3% faster in iterating.

Comparing ArrayLists and Vectors, Vectors were ≈66.6% faster in populating and ArrayLists were ≈149.3% faster in iterating.

Overall, using Arrays was the fastest in both populating and iterating, with very significant increases in speed compared to using ArrayLists or Vectors. Using Vectors rather than ArrayLists would be faster in populating but significantly slower in iterating.

My recommendations based on the data above would be to use Arrays when possible as they have the fastest times in both populating and iterating, significantly faster than the other 2 types. However, when using Arrays is not possible and both ArrayLists and Vectors are suitable options, designers need to weigh their decision based on whether they will be populating or iterating through the data types more, as ArrayLists will be faster in iterating and Vectors will be faster in populating.