

E26.

	Advantages	Disadvantages
PointCP1	Both cartesian coordinates can be converted to polar coordinates and vice versa	Because both cartesian and polar coordinates were calculated in the same class, and there had to be if statement check which type of coordinate was being inputted, the storage had to be converted.
PointCP2	Polar coordinates can be converted to cartesian coordinates	Cartesian coordinates cannot be converted to polar coordinates
PointCP3	Cartesian coordinates can be converted to polar coordinates	Polar coordinates cannot be converted to cartesian coordinates
PointCP5	Both cartesian coordinates can be converted to polar coordinates and vice versa. Each coordinate type was stored in their respective class.	Each coordinate type had to be instantiated separately, the implementation of checking each coordinate type and reading the coordinates is more redundant.

E 30.

Both Design1 and Design5 each had 2000 trial runs, here is the total runtime:

	Average Runtime(ms)	Shortest Runtime(ms)	Longest Runtime(ms)
Design1	1055.7783	988.2345	1543.9762
Design5	5122.6321	414.6234	734.2346

PART 2

The chosen size of the data structures was 100580000. Any larger produced an OutOfMemory error in the constructions. Furthermore, the average construction time for an arraylist of size=100580000 was around 10s.

	Trial	Array	ArrayList	Vector
Construction time (s)	1	2.268559600	11.581081500	3.732627300
	2	2.172125700	5.363177900	3.532759600
	3	2.476271900	13.252801400	3.280881900
Iteration time (s)	1	0.106452400	0.404177600	1.287172500
	2	0.133485100	3.407768900	0.747613600
	3	0.115067800	0.307810600	3.723729800

Figure 2: Iteration and construction runtimes of the array, arraylist and vector for three trials.

The construction time averaged around 10s for the arraylist, with some outlier times reaching 5s, as shown in Trial 2 of the arraylist's construction time in Figure 2. Both the array and the vector consistently took 2s and 3s, respectively.

Note that both the array and vector were initialized with a set index. While it is inevitable for the arrays, it was done due to technical limitations for the vector. OutOfMemory errors occur whenever the vector is not initialized with a set index, and any lower size value will reduce the amount of time arraylist used to construct.

From the data, it is clear that the array had the lowest construction and iteration times. Though its iteration times were usually the same as the arraylist's, the array also had lower construction compared to the arraylists, which averaged at around 10s as opposed to the array's 2s.

As a recommendation, arrays work well if the number of entries has an easily determinable maximum. If not, the fixed nature of the array can cause issues. For this experiment, the size was fixed, so the cons of the array were negated. Because all of the array's index was available without the need to iterate through the whole collection, the array proved to be the most efficient.

Arraylists and vectors are better for more dynamic situations. Because the summation only required a single iteration through the elements, the pros of both of these data structures was not explored thoroughly.

Thus, if the problem has a fixed size, arrays are the best choice, whereas if the situation has a potentially unlimited number of variables, the more dynamic data structures would be

preferable. Vectors took less time to construct and approximately the same amount of time to iterate, however, this is due to a technical limitation of the experiment. The vectors were initialized with a set index, similar to the arrays, which allowed for quicker construction. This was because OutOfMemory errors occurred whenever the vector was initialized without a set index.

If the vectors had not been initialized with a predetermined index size and had to populate their collection similarly to the arraylists, it is hypothesized that the construction times would be identical to the arraylists.