Part 1

E26: Pros and Cons of each design

	Pros	Cons
Design 1	The user can choose between cartesian and polar coordinates. Has a flag to indicate whether the coordinates are cartesian or polar.	When requesting either cartesian or polar, the system always has to check and convert.
Design 2	Polar coordinates are stored so there is no need for conversion. Processing time for polar coordinates is reduced.	Cartesian coordinates are not stored, thus needing conversion.
Design 3	Cartesian coordinates are stored so there is no need for conversion. Processing time for cartesian coordinates is reduced.	Polar coordinates are not stored, thus needing conversion.
Design 5	Reduce redundant and duplicated code. Easy maintenance and error tracing.	More memory consumed. Harder to debug.

This table shows the pros and cons for each design

E30: Average execution time. 1_000_000 runs for each method.

	PointCP1	PointCP2	PointCP3	PointCP52	PointCP53
toPolar()	237	245	244	206	185
toCartesian()	194	191	207	317	200
getDistance()	279	231	250	246	275
rotatePoint()	230	320	261	300	320
average	235	246.75	240.5	267.25	245

This table shows the average execution time for each method of each class

Test Procedure

- Create two arrays, one stores names of designs, one stores names of methods.
- A for loop is created to loop through every class and every method within that class.
- Each method runs for a total number of 1,000,000 times.
- After a run is completed, its execution time is added towards the variable called totalExecutionTime.
- Finally, we divide totalExecutionTime by 1,000,000 and get the average execution time.
- The reason why milliseconds was not used is due to the fact that the execution time in milliseconds is too small compared to the number of runs, which leads to the average execution time being 0ms for some methods.

Part 2: Arrays

	Array (populate)	Array (sum)	ArrayList (populate)	ArrayList (sum)	Vector (populate)	Vector (sum)
1	1091	33	3197	77	5552	2017
2	1131	35	3284	86	5266	1875
3	1486	40	5166	98	7631	2373
4	1757	47	5346	134	7380	2202
5	1455	49	4174	99	7829	2564
Min	1091	33	3197	77	5266	1875
Max	1757	49	5346	134	7829	2564
Median	1455	40	4174	98	7380	2202

This table compares the execution time in ms between Array, ArrayList, and Vector. (Test size is 99,000,000)

My laptop is going to explode if I conduct more tests, which is why the execution time keeps increasing over time.

Report

- Array yields the lowest execution time for both populating and calculating sum.
- ArrayList has the second lowest execution time for both populating and calculating sum.
- Vector has the highest execution time for both populating and calculating sum.
- Vector's lowest execution time still exceeds Array's highest execution time.
- Consistency:
    - Array (populating) (max - min) = 1757 - 1091 = 666 ms
    - Array (sum) (max - min) = 49 - 33 = 16 ms
    -
    - ArrayList (populating) (max - min) = 5346 - 3197 = 2149 ms
    - ArrayList (sum) (max - min) = 134 - 77 = 57 ms
    -
    - Vector (populating) (max - min) = 7829 - 5266 = 2563 ms
    - Vector (sum) (max - min) = 2564 - 1875 = 689 ms

Based on the data presented in the table above, it is recommended that Array should be used to reduce the execution time. However, when dealing with more complex data structures, ArrayList and Vector will be more dominant. My suggestion is that, unless the code is improved with the implementation of ArrayList and Vector, it is wise to just stick with Array.