Part 1:

E26:

DESIGN TYPE	Advantages	Disadvantages
1	● Versatile: Allows for computations or storage of both types + decision making	● Complex design: Involves decision making from input ● Requires memory + processing power: Storing AND calculating both coordinates ● Very inefficient instantiation
2	● Simpler design: Only polar coordinates can be entered ● More efficient: Only calculating for the cartesian coordinate + simply returning the polar coordinates	● Limited decision making: Inputs MUST be in polar form ● Instantiation required: Memory must be allocated
3	● Simpler design: Only cartesian coordinates can be entered ● More efficient: Only calculating for the polar coordinate while simply returning the cartesian coordinates	● Limited decision making: Inputs MUST be in cartesian form ● Instantiation required: Memory must be allocated
4	● Simple design: Simply storing both inputs ● Efficient processing: No need to compute coordinates	● Higher memory use: Requires the storage of both types of coordinates
5	● Versatile: Allows for decision making based on concrete class used ● No instantiation needed ● Most efficient: Computing/storage dependent on client needs using abstract classes ● No instantiation required: Saves memory and processing power	● Complex design: Use of abstract classes to implement the design

Sample Outputs for E28-E30:

	Design 2	Design 3	Design 5
Trial 1 time (ms)	8943	8974	1933
Trial 2 time (ms)	7044	8227	3490
Trial 3 time (ms)	8784	8809	2768
Trial 4 time (ms)	9249	9008	3237
Trial 5 time (ms)	8898	8943	3311
Median time (definitive result) (ms)	8898	8943	3237
Min time (ms)	7044	8227	1933
Max time (ms)	9249	9008	3490

*11001 random cases used each trial

Methods:

The performance tests were conducted by tracking the time required for the designs to calculate all of our randomized inputs.

A for loop was used to create a static number of instances that had a randomized coordinate type and randomized X/Y inputs. The time elapsed was calculated by subtracting the start time (after random inputs are generated, before calculations start) from the end time (after all calculations/outputs are done). This elapsed time was recorded into our table for each of our trials.

Discussion:

Our results have a strong correlation to our predictions table in E26. Using abstract classes in design 5 yielded the fastest definitive result of 3237 ms while designs 2 and 3 had definitive results of around 8900 ms.

The tests demonstrate the advantage of abstract classes and show that the lack of instantiation can greatly improve efficiency by saving memory and processing power in comparison to concrete classes.

Part 2:

	Array List	**Vector**	**Array**
Creation (10 Runs with 671,000,000 elements)	Run 1: **7.196960917s** Run 2: **6.936198375s** Run 3: **8.983641875s** Run 4: **6.769467583s** Run 5: **8.942093833s** Run 6: **7.645587125s** Run 7: **7.089619708s** Run 8: **7.309762083s** Run 9: **7.310750792s** Run 10: **7.286580417s**	Run 1: **8.637935167s** Run 2: **8.47299425s** Run 3: **6.868520625s** Run 4: **6.117909625s** Run 5: **6.854957583s** Run 6: **6.354540625s** Run 7: **6.12001975s** Run 8: **6.107750958s** Run 9: **6.36168025s** Run 10: **6.3495805s**	Run 1: **76.506127209s** Run 2: **77.880181458s** Run 3: **84.7087315s** Run 4: **84.962491958s** Run 5: **67.763356s** Run 6: **78.818063833s** Run 7: **80.157386s** Run 8: **80.14143075s** Run 9: **77.91921175s** Run 10: **80.635623417s**
Creation (Avg over 10 runs)	**7.5470662708s**	**6.824588933299999s**	**78.9492603875s**
Sum (250,000,000 elements)	Array List Sum: 1000027423 Sum Time: **0.133769041s**	Vector Sum: 999978207 Sum Time: **2.354658458s**	Array Sum: 1000038279 Sum Time: **0.085614209s**

From the table above, we can see that the longest creation time on average comes from the array, while the shortest creation time comes from a vector. When summing elements, the fastest time comes from an array, while the longest time comes from a vector.

For a recommendation to designers, it would seem that it depends on the task that needs to be performed. If the goal is to store many elements, then it would seem that a Vector works best. If the goal is to sum elements, it would seem that the use of an Array would be best. However, if the goal is to do both things and more, it would seem that the best overall recommendation is the use of an Array List. This is because the Array List has the best average of the 2 methods.