E.26:

Design number	Pros	Cons
1	● Efficient when creating instances and doing computations requiring both coordinate systems	● Complexity of code ● Large memory consumed.
2	● Efficient when creating instances and computing only polar coordinates. ● Simplicity of code ● Less memory used	● Inefficient in computing coordinates with the cartesian system
3	● Efficient when creating cartesian instances and its computations. ● Less memory used ● Simplicity of code	● Inefficient in computing coordinates with the polar system
5	● Avoiding incomplete inherited classes. ● Efficient when computing both coordinates systems ● Simplicity of code	● Lots of backtracking sorting in memory. ● Backward callings resulting in a longer runtime.

E.27: Design 5 has been implemented and we added a PointCP5Test.java file to test it out which can be found on our github repository.

E.28-29:

- We compared all 4 classes by creating a test class for each one of them and calculating the run time to have a performance analysis. After calling each method many thousands of times, we were able to find the elapsed time in milliseconds for a fixed number of iterations.
    a. Design 1 was about 913 milliseconds.
    b. Design 2 was about 790 milliseconds.
    c. Design 3 was almost the same as design 2 at about 781 milliseconds.
    d. Design 5 was about 7895 milliseconds.

After analyzing each and every design implemented, our hypotheses cited in E.26 have been confirmed.

E30:

a.

	Design2	**Design3**	**Design 5**
convertStrorageToPolar ()	141 ns	2573 ns	1363 ns
getDistance(PointCp pointB)	2900 ns	1563 ns	3058 ns
rotatePoint(double rotation)	2022 ns	2154 ns	2456 ns

b. The above table contains the average computation speed over 6 consecutive tests of each method in each design. The results indicate that design 2 is the most efficient when creating instances and computing polar coordinates. In contrast, design 3 is the most inefficient for polar coordinates, while design 5 is an average computation between the times of design 2 and 3. This is all in accordance with the previously made predictions in E26. Design 3 is the most efficient design for computing the distance, as compared to design 2 and design 5 which is the most inefficient out of all three designs. When the design 2 getDistance() method is called, the getX() and getY() methods return values that are converted within the getters, whereas in design 3, the getters contain the values of x and y set. This may result in a faster running time for design 3, as compared to design 2 and design 5. Finally, rotatePoint has a more closely related computation time. This means that each method is only slightly more efficient than the next. According to the average times, design 2 is most efficient, followed by design 3 and then finally design 5. In this case, where the getter methods helped design 3 have a more efficient getDistance() method, the getters of design 2 provide a more efficient rotatePoint() method, because the conversions are made when the getter is called, and then implemented within the rest of the rotate method, whereas in design 3, the conversions are not previously conducted.

** In design 3 the getX() method returns only the instantiated value of x. Similarly, getY() returns only the instantiated variable y. Design 2 for the same methods returns “(Math.cos(Math.toRadians(theta)) * rho);” for getX() and “(Math.sin(Math.toRadians(theta)) * rho); “ for getY(). The implementation of the getDistance() method however is the same in both classes. Design 2 is inefficient, because when getDistance() is called, and the method calls getX() and getY(), these methods have to then call on the java math library, and perform additional calculations in the method, which increases its operational time.

However, design 3 only returns the instantiated variables, reducing its overall time for that method.