Assignment 1
Classes for Representing Geometric Points

SEG2105B - Introduction to Software Engineering
Fall 2022

University of Ottawa

Professor:	**Wassim El Ahmar**
Student 1:	**Noah Ndahirwa (300152285)**
Student 2:	**Steven Wilson (300220675)**

Group 55

Submission Date: Oct. 9, 2022

Analysis of Various Design Alternatives

Design	**How cartesian coordinates are computed**	**How polar coordinates are computed**
Design 1: Store one type of coordinates using a single pair of instance variables, with a flag indicating which type is stored	Simply returned if Cartesian is the storage format, otherwise computed	Simply returned if polar is the storage format, otherwise computed
Design 2: Store polar coordinates only	Computed on demand, but not stored	Simply returned
Design 3: Store cartesian coordinates only	Simply returned	Computed on demand, but not stored
Design 5: Abstract class with designs 2 and 3 as subclasses	Depends on the concrete class used	Depends on the concrete class used

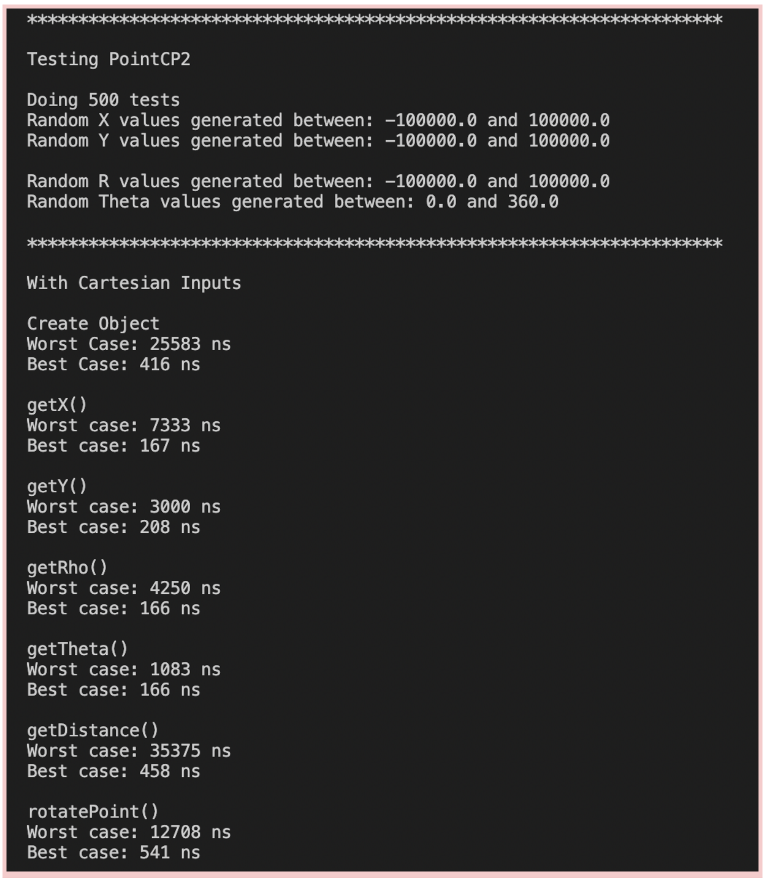
Table: Alternative designs for the PointCP class

Testing

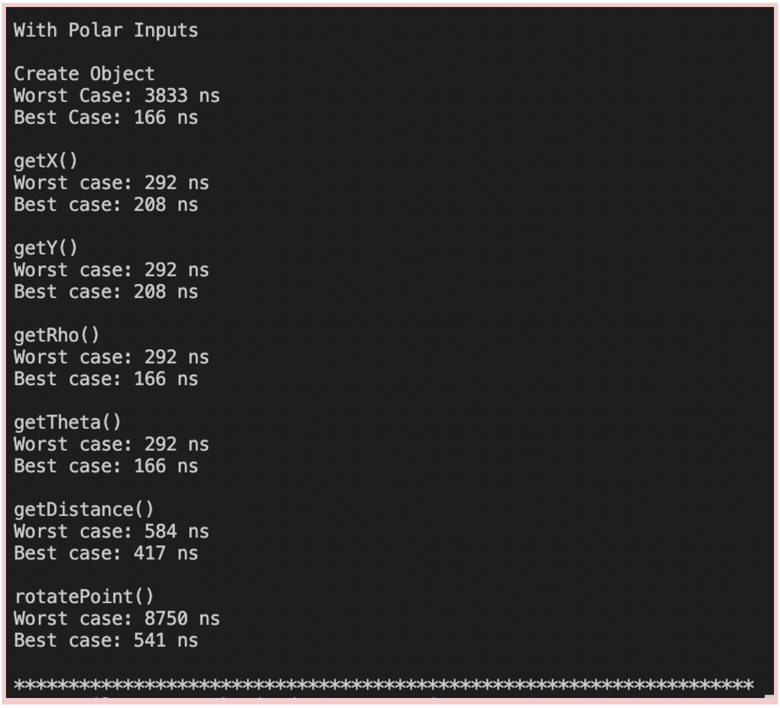
The design2 and design3 were tested using 500 iterations, on each iteration random x, y, rho, and theta values were created. The values for x and y were randomly generated and were in the range of -100,000 to 100,000. The random generation of the rho value was between -100,000 to 100,000 and theta uses a random value between 0 to 360 degrees.

On each test the minimum and maximum runtime values were measured for each method. All methods were tested using cartesian and polar coordinates. This would help to analyze how each class would perform depending on the type storage that it was implemented with.

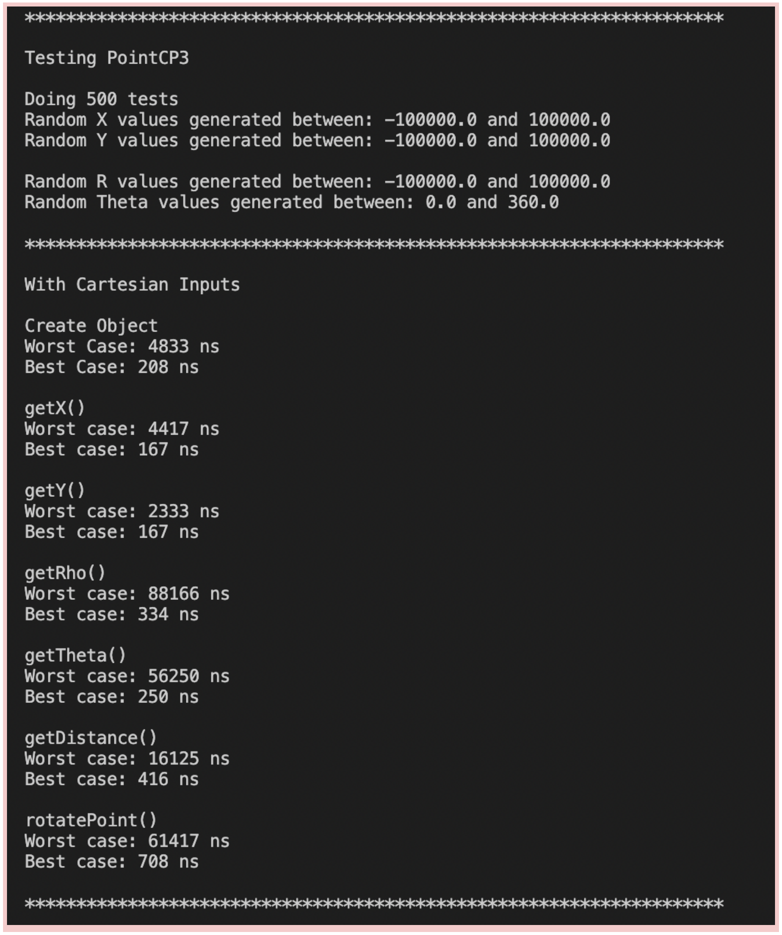
Sample Outputs



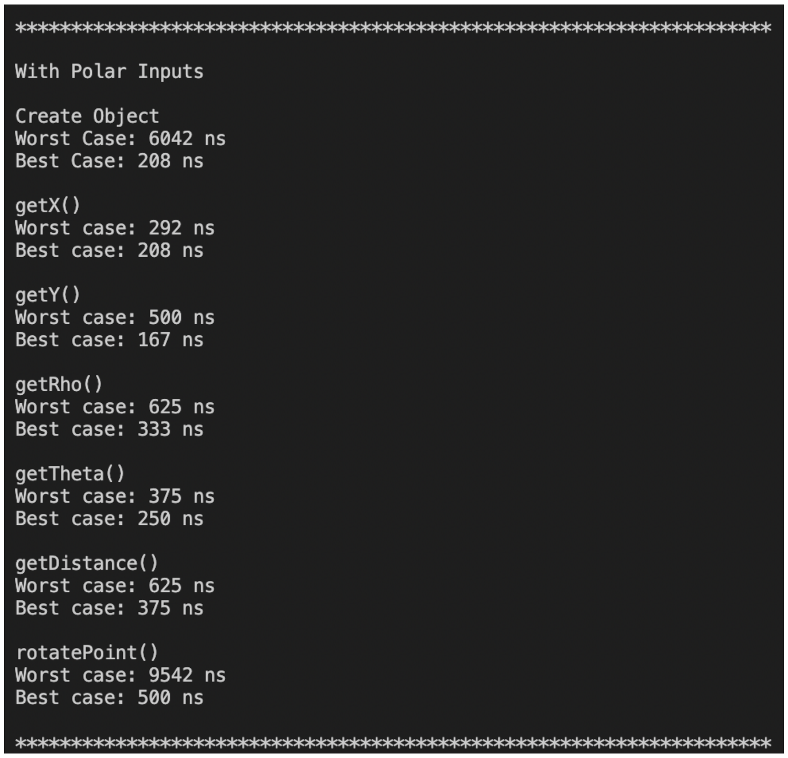
```
********************************************************************

Testing PointCP2

Doing 500 tests
Random X values generated between: -100000.0 and 100000.0
Random Y values generated between: -100000.0 and 100000.0

Random R values generated between: -100000.0 and 100000.0
Random Theta values generated between: 0.0 and 360.0

********************************************************************

With Cartesian Inputs

Create Object
Worst Case: 25583 ns
Best Case: 416 ns

getX()
Worst case: 7333 ns
Best case: 167 ns

getY()
Worst case: 3000 ns
Best case: 208 ns

getRho()
Worst case: 4250 ns
Best case: 166 ns

getTheta()
Worst case: 1083 ns
Best case: 166 ns

getDistance()
Worst case: 35375 ns
Best case: 458 ns

rotatePoint()
Worst case: 12708 ns
Best case: 541 ns
```



```
With Polar Inputs

Create Object
Worst Case: 3833 ns
Best Case: 166 ns

getX()
Worst case: 292 ns
Best case: 208 ns

getY()
Worst case: 292 ns
Best case: 208 ns

getRho()
Worst case: 292 ns
Best case: 166 ns

getTheta()
Worst case: 292 ns
Best case: 166 ns

getDistance()
Worst case: 584 ns
Best case: 417 ns

rotatePoint()
Worst case: 8750 ns
Best case: 541 ns

********************************************************************
```



```
********************************************************************

Testing PointCP3

Doing 500 tests
Random X values generated between: -100000.0 and 100000.0
Random Y values generated between: -100000.0 and 100000.0

Random R values generated between: -100000.0 and 100000.0
Random Theta values generated between: 0.0 and 360.0

********************************************************************

With Cartesian Inputs

Create Object
Worst Case: 4833 ns
Best Case: 208 ns

getX()
Worst case: 4417 ns
Best case: 167 ns

getY()
Worst case: 2333 ns
Best case: 167 ns

getRho()
Worst case: 88166 ns
Best case: 334 ns

getTheta()
Worst case: 56250 ns
Best case: 250 ns

getDistance()
Worst case: 16125 ns
Best case: 416 ns

rotatePoint()
Worst case: 61417 ns
Best case: 708 ns

********************************************************************
```

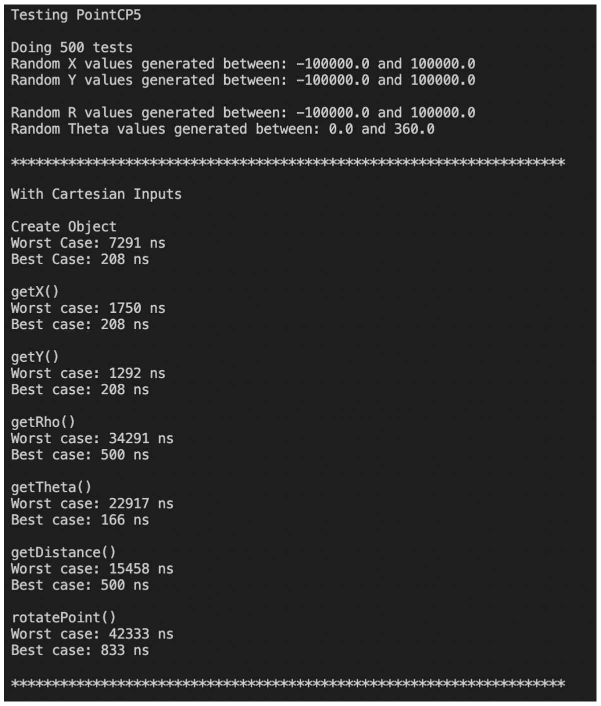


```
********************************************************************

With Polar Inputs

Create Object
Worst Case: 6042 ns
Best Case: 208 ns

getX()
Worst case: 292 ns
Best case: 208 ns

getY()
Worst case: 500 ns
Best case: 167 ns

getRho()
Worst case: 625 ns
Best case: 333 ns

getTheta()
Worst case: 375 ns
Best case: 250 ns

getDistance()
Worst case: 625 ns
Best case: 375 ns

rotatePoint()
Worst case: 9542 ns
Best case: 500 ns

********************************************************************
```

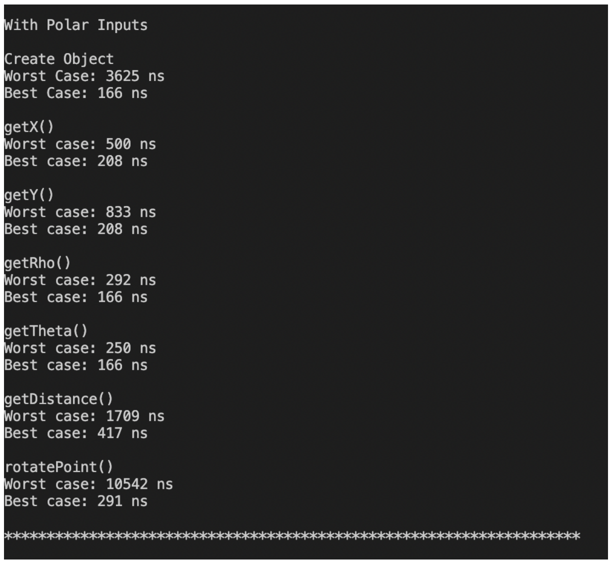
E26: Table Describing Advantages and Disadvantages of Design Alternatives

Design	**Simplicity of Code**	**Efficiency: creating instances**	**Efficiency: requiring both coordinate systems**	**Type of Storage**
Design 2: Store polar coordinates only	The code is easy to understand. Polar values are stored, if the input is in cartesian it is converted to polar. The functions getX() and getY() convert the stored polar coordinates on demand.	This design is most efficient when creating instances that use polar coordinates. The code will be less efficient when dealing with cartesian values.	This design is less efficient when there are calls to getX() and getY() since the values are stored in polar coordinates and it is necessary to make a conversion at runtime.	Polar coordinates are stored as rho and theta.
Design 3: Store cartesian coordinates only	The code is easy to understand. The design stores cartesian values, if the input is in polar coordinates it is converted to cartesian. The functions getRho() and getTheta() convert the stored cartesian coordinates to polar on demand.	This design is most efficient when creating instances that use cartesian coordinates. The code will be less efficient when dealing with polar values.	This design is less efficient when the methods getRho() and getTheta() are called. The values are stored in cartesian coordinates and when these functions are called it is necessary to make a conversion to polar coordinates on demand.	Cartesian coordinates are stored as x, y.
Design 5: Abstract class with designs 2 and 3 as subclasses	Design2 and Design3 are subclasses of Design5. The abstract class PointCP5 implements all the abstract methods necessary to run methods with stored cartesian or stored polar coordinates.	The design is efficient if the correct constructor is selected, the constructor used needs to be the one appropriate for the type. For example, if the user intends to use a cartesian input the user will have the most efficiency if the design that uses cartesian storage is selected.	The design is efficient depending on how the user makes the calls and the coordinate system the user wants. For example, it will be efficient if the user intends to use cartesian values and uses the cartesian implementation of the abstract class.	This design can store cartesian or polar coordinates.

E28: Run a performance analysis in which you compare the performance of Design 5, as you implemented it in the previous exercise, with Design 1. Determine the magnitude of the differences in efficiency, and verify the hypotheses you developed in E26.



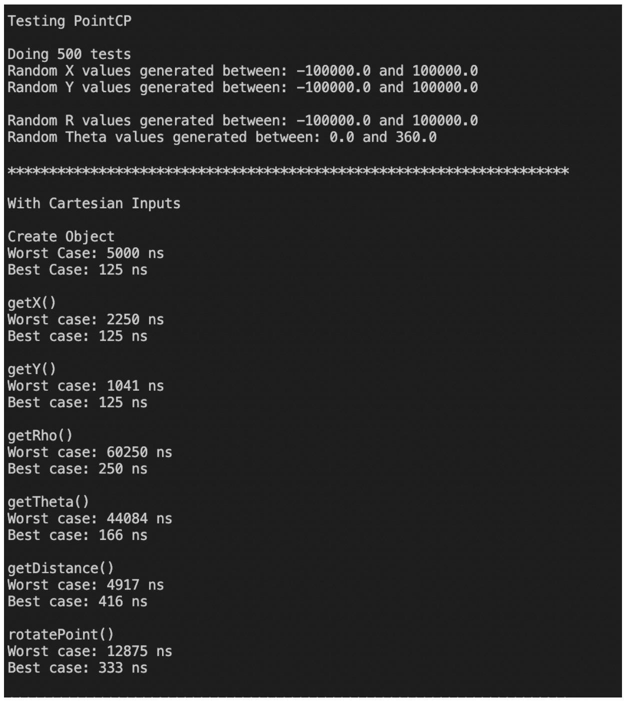
```
Testing PointCP5

Doing 500 tests
Random X values generated between: -100000.0 and 100000.0
Random Y values generated between: -100000.0 and 100000.0

Random R values generated between: -100000.0 and 100000.0
Random Theta values generated between: 0.0 and 360.0

******************************************************************

With Cartesian Inputs

Create Object
Worst Case: 7291 ns
Best Case: 208 ns

getX()
Worst case: 1750 ns
Best case: 208 ns

getY()
Worst case: 1292 ns
Best case: 208 ns

getRho()
Worst case: 34291 ns
Best case: 500 ns

getTheta()
Worst case: 22917 ns
Best case: 166 ns

getDistance()
Worst case: 15458 ns
Best case: 500 ns

rotatePoint()
Worst case: 42333 ns
Best case: 833 ns

******************************************************************
```



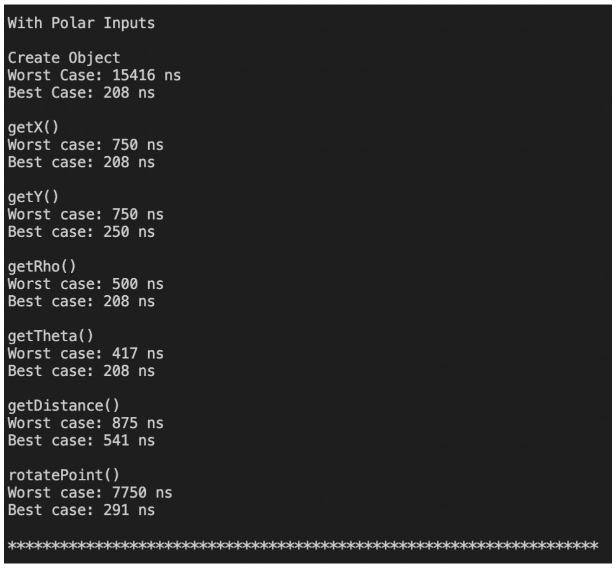
```
With Polar Inputs

Create Object
Worst Case: 3625 ns
Best Case: 166 ns

getX()
Worst case: 500 ns
Best case: 208 ns

getY()
Worst case: 833 ns
Best case: 208 ns

getRho()
Worst case: 292 ns
Best case: 166 ns

getTheta()
Worst case: 250 ns
Best case: 166 ns

getDistance()
Worst case: 1709 ns
Best case: 417 ns

rotatePoint()
Worst case: 10542 ns
Best case: 291 ns

******************************************************************
```

After testing the performance of design 5 and comparing it to the implementation of design 1 we can see that there is no overall outperformer by a big margin. The implementation of Design 5 performs better but not by a large margin.

E29: To run a performance analysis, you will have to create a new test class that randomly generates large numbers of instances of PointCP, and performs operations on them, such as retrieving polar and Cartesian coordinates. You should then run this test class with the two versions of PointCP – Design 1 and Design 5.



```
Testing PointCP

Doing 500 tests
Random X values generated between: -100000.0 and 100000.0
Random Y values generated between: -100000.0 and 100000.0

Random R values generated between: -100000.0 and 100000.0
Random Theta values generated between: 0.0 and 360.0

*********************************************************************

With Cartesian Inputs

Create Object
Worst Case: 5000 ns
Best Case: 125 ns

getX()
Worst case: 2250 ns
Best case: 125 ns

getY()
Worst case: 1041 ns
Best case: 125 ns

getRho()
Worst case: 60250 ns
Best case: 250 ns

getTheta()
Worst case: 44084 ns
Best case: 166 ns

getDistance()
Worst case: 4917 ns
Best case: 416 ns

rotatePoint()
Worst case: 12875 ns
Best case: 333 ns
```



```
With Polar Inputs

Create Object
Worst Case: 15416 ns
Best Case: 208 ns

getX()
Worst case: 750 ns
Best case: 208 ns

getY()
Worst case: 750 ns
Best case: 250 ns

getRho()
Worst case: 500 ns
Best case: 208 ns

getTheta()
Worst case: 417 ns
Best case: 208 ns

getDistance()
Worst case: 875 ns
Best case: 541 ns

rotatePoint()
Worst case: 7750 ns
Best case: 291 ns

*********************************************************************
```

We compare the tests run with design 1 and the tests that were previously run on design 5, if the tests are run for design 5 with the correct type and method used it will outperform design 1. However, if the type and method storage is not the optimal it will run slower than design1.

E30: Summarize your results in a table: the columns of the table would be the two designs; the rows of the table would be the operations. The values reported in the table would be the average computation speed. Make sure you explain your results.

Cartesian Inputs

Design	Create Object	getX()	getY()	getRho()	getTheta()	getDistance()	rotatePoint()
Design 2	25583	7333	3000	4250	1083	35375	12708
Design 3	4833	4417	2333	88166	56250	16125	614417

Polar Inputs

Design	Create Object	getX()	getY()	getRho()	getTheta()	getDistance()	rotatePoint()
Design 2	3833	292	292	292	292	584	8750
Design 3	6042	292	500	625	375	625	9542