Assignment 1

SEG2105 A

Langlois, Matthew: 7731813

Yaraskavitch, Matthew: 6301664

E-26

Design	Pros	Cons
Default	• Stores coordinates in either Polar or Cartesian format • Uses less memory • Quick to initialize (lower memory usage) • No conversion on initialization	• Requires further computation to convert between formats upon request
Design 2	• No conversion necessary if working in mainly polar coordinates	• Slower to initialize if input is cartesian • Must perform conversion when Cartesian coordinates are requested back
Design 4	• Accepts either Cartesian or Polar coordinates for initialization • Returns both formats readily and quickly without any further conversion	• Slower to initialize for any input since a conversion between Cartesian/Polar must be performed based upon input format • Uses more memory since both formats are stored

E-28, E-29, E-30

Fig 2 – Average time to complete specified operation for each design. Operations were run on loop for 60 seconds and the average time determined. Maximum and Minimum values are in brackets.

Operation	Default	Design 2	Design 4
Initialize w/ Cartesian	52 ns (55ns /51ns)	106 ns (112ns /100ns)	105 ns (110ns /101ns)
Initialize w/ Polar	50 ns (52ns /52ns)	51 ns (53ns /50ns)	51 ns (52ns /50ns)
Get Cartesian from Initialized Cartesian	5 ns (6ns /5ns)	20 ns (23ns /19ns)	5 ns (5ns /5ns)
Get Cartesian from Initialized Polar	5 ns (5ns /5ns)	5 ns (5ns /5ns)	5 ns (5ns /5ns)
Get Polar from Initialized Cartesian	33 ns (35ns /32ns)	5 ns (5ns /5ns)	5 ns (6ns /5ns)
Get Polar from Initialized Polar	5 ns (6ns /5ns)	5 ns (5ns /5ns)	5 ns (5ns /5ns)

In order to test the implemented designs, the PointCPTest class was adapted for each new design (PointCPD2Test and PointCPD4Test). Once these test classes were shown to be functioning, another class called EfficienyTest was developed to create a large number of instances of each design class and time their initialization as well as various operations to compare efficiency.

Discussion

In general, algorithms either utilize more compute time or more memory space. The goal of this lab was to illustrate this difference by exploring three different implementations of essentially the same class interface. The design originally provided in PointCP.java contained an implementation that would accept coordinates in either Cartesian or Polar. The class then would keep a record of the format of the storage type and perform storage conversions when needed and return conversions from the stored value when requested. This implementation is seen as the reference for this laboratory. As expected, an instance of PointCP takes essentially the same amount of time to initialize regardless of input type (52ns for Cartesian versus 50ns for polar). Also as expected, getting polar coordinates from instances initialized as Cartesian took significantly longer (33ns vs 5ns) since the expensive conversion to polar coordinates must be performed.

Design 2 is different in that it only stores coordinates in polar format and performs conversions when requested. Thusly, the operations to convert between storage formats do nothing and are left as returning null. Initialization with Cartesian coordinates takes roughly twice as long as the default implementation since the conversion to polar must be performed. Likewise, it takes much longer to return Cartesian coordinates than the polar form (20ns vs 5ns) since a conversion must be performed. Otherwise, the other times for accessing variables have remained unchanged.

Design 4 accepts coordinates in either Cartesian or Polar format and then stores them in both formats. Again, the storage conversion methods are redundant and thus return null. This implementation takes a different approach in that it performs all possibly required conversions at initialization but then simply needs to return either format when requested, without further computation. Naturally, the initialization time with a Cartesian set of coordinates is the same as Design 2 since a conversion must be performed to Polar. However, the conversion from polar to Cartesian is nearly indistinguishable from the Default design where no conversion is performed during initialization. Also as expected, the time to fetch any of the variables after

initialization is the nearly constant value of 5ns.

In general, a few main conclusions can be made. Clearly, the implementation of the Cartesian to Polar coordinates is more computationally difficult than going from polar to Cartesian in this Java implementation. Also, the reference implementation seems the best candidate for scenarios in which a large number of coordinates are being initialized in a roughly even mix of Cartesian and polar forms. When working with a smaller number of coordinates but accessing their values much more frequently, Design 4 would be the best implementation. Despite its longer initialization times, it maintains very low access times for coordinates in either format.