Assignment 1

Ex1 Q5: Textbook work

E25. Answer the following questions with respect to the above designs of the PointCP class:

a) Discuss why it might be useful to allow users of class PointCP (Design 1) to explicitly change the internal storage format, using convertStorageToCartesian or convertStorageToPolar.

Being able to explicitly change the internal storage format is useful for the class user in that it allows the user to quickly change geometric coordinate representations with the use of only 1 method. It allows the user to efficiently maintain their own coding readability in quickly figuring out what internal storage format is being used.

b) What might be a potential hidden weakness of these methods? Hint: what could happen if one is called, then the other, and this process is repeated multiple times.

The number will be rounded slowly by a small value, however after even one loop back and forwards, the float amount will change. Due to this rounding, this method will slowly change the variables for the point so that the numbers aren’t near perfect after many loops.

c) Write a short program to test whether the weakness you discussed in part b is, in fact, real.

<https://pastebin.com/JDbYjsAg>

import java.lang.\*;

public class Example {

public static void main(String[] args) {

double a = 10.9876543210;

double copy = a;

for (int i = 0; i < 1000; i++) {

copy = Math.sqrt(copy);

copy = copy \* copy;

}

System.out.println("Original Version: " + a + "\nChanged Version: " + copy); // from 10.9876543210 to // 10.987654321000003

}

}

E26. Create a table describing the various advantages (pros) and disadvantages (cons) of each of the five design alternatives. Some of the factors to consider are: simplicity of code, efficiency when creating instances, efficiency when doing computations that require both coordinate systems, and amount of memory used.

|  |  |  |
| --- | --- | --- |
| **Design** | **Pros** | **Cons** |
| Design 1 | Simplicity of code, very simple to change coordinate systems.  Saves on memory however since only 1 type is stored at a time, but much slower. | Inefficient for the CPU as you have to constantly recalculate your coordinate systems and store the type.  Not very efficient for creating instances, since calculations have to be made each time it’s called. |
| Design 2 | Code is very efficient when doing computations since it’s already assumed what the type is, no need to input and use another variable / memory for it. | Less robust than design 1, since you can only input cartesian coordinates. Heavier on the CPU in that computations have to be made.  Not very efficient for creating instances, since calculations have to be made each time it’s called.. |
| Design 3 | Code is very efficient when doing computations since it’s already assumed what the type is, no need to input and use another variable / memory for it. | Less robust than design 1, since you can only input polar coordinates. Heavier on the CPU in that computations have to be made.  Not very efficient for creating instances, since calculations have to be made each time it’s called. |
| Design 4 | Much faster and more efficient on the CPU. Very efficient when creating instances. | Requires more memory since both types are stored |
| Design 5 | Much faster than design 1 due to no flag being saved, so things to keep track of and more memory saved.  Instances are extremely easy and efficient to use. | More memory being used to store the information, as all information is directly stored. |

E28. (Manual low volume, higher volume test given below for E29/E30.)

Simple non efficient test of 6 manual inputs for each design.

|  |  |
| --- | --- |
| **DESIGN**  **1** | |
| Input | Time Taken (millisecond) |
| C 3 3 | 14 |
| C 8 5 | 16 |
| C 9 15 | 13 |
| P 3 3 | 11 |
| P 8 5 | 10 |
| P 9 15 | 17 |

|  |  |
| --- | --- |
| **DESIGN**  **5** | |
| Input | Time Taken (millisecond) |
| C 3 3 | 7 |
| C 8 5 | 3 |
| C 9 15 | 5 |
| P 3 3 | 5 |
| P 8 5 | 3 |
| P 9 15 | 5 |

Median for design 1: 13.5 milliseconds

Median for design 2: 4.667 milliseconds

Mean for design 1: 13.5 milliseconds

Mean for design 2: 5 milliseconds

13.5/5 = 2.7 therefore design 5 is 2.7 times more time efficient than design 1 on average, hypothesis from E26 is true, Design 5 is faster.

**E29//E30.**

Based on testing programs made:

PointCPEfficiencyTest.java and PointCP5EfficiencyTest.java

Both tests entering in equal number of points for polar and cartesian, testing for all degrees up to 360 in the second input, the first input rho/x going up to 20, to a total of 14400 tests.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PointCPEffiencyTest.java (testing design 1) | | | | PointCP5EfficiencyTest.java (testing design 5) | | | |
| Mean Test Time (nano) | Smallest Test Time (nano) | Largest Test Time (nanosecond) | Median time for tests (nano) | Smallest Test Time (nano) | Largest Test Time (nanosecond) | Median time for tests (nano) | Mean Test Time (nano) |
| 13 | 0 | 2,940,100 | 100 | 0 | 3,031,400 | 100 | 7 |
| 10 | 0 | 2,968,900 | 100 | 0 | 2,907,500 | 100 | 7 |
| 11 | 0 | 3,003,600 | 100 | 0 | 2,950,900 | 100 | 8 |
| 10 | 0 | 3,025,600 | 100 | 0 | 3,152,700 | 100 | 8 |
| 12 | 0 | 2,881,900 | 100 | 0 | 2,892,300 | 100 | 8 |

Mean of design1: 11.2 Mean of design5: 7.6

From the mean data given, we can assume that design 5 is 147% more efficient. The reason for this is because Design 5 is an abstract class. It has two subclasses that exclusively store polar or cartesian coordinates, and doesn’t include a flag to indicate the type. This means that design 1 *knows* without a flag if its polar or cartesian, and automatically gets the other. While design 1 requires a flag, hence more memory being used. Design 5 also has less methods to call while design 1 has to go back and forth converting and internally keeping track of whether or not it is polar or cartesian. Something interesting to note is that although design 5 had a smaller mean time than design 1, it had the largest test time. This is due to the fact that design 5 does store all the information at once, so it takes a heavier initial load memory wise. However in due time it is the faster option.

As one can tell our timing units changed from our small, manual entry tests to the large scale automated one. This is due to something weird that happened when we extended the test. For information up to ~20 tests, our timing was normal and could be represented in milliseconds. However, as our number of tests went up to a large degree to fill the 10 second quota given, it appeared as though our code became more optimized. One can see the inconsistency in timings if they printout our whole arraylist in any of the efficiency test files. For this reason, our median would have been 0 ms, so the choice to switch over to nanoseconds was taken.