Iona Buchanan 6671041

Christopher Wong 8764900

Assignment 1

# Polar/Cartesian Points

## Advantages and Disadvantages of each design

|  |  |  |
| --- | --- | --- |
| Design | Pros | Cons |
| Design 1: Store one type of coordinates using a single pair of instance variables, with a flag indicating which type is stored | * Versatile * Everything can be done with this single class | * Extra memory to store flag * Most operations require checking the value of the flag |
| Design 2: Store polar coordinates only | * Returning polar coordinates is simply returning instance vars | * Retrieving cartesian coordinates requires calculation/conversion each time |
| Design 3: Store cartesian coordinates only | * Returning Cartesian coordinates is simply returning instance vars | * Retrieving polar coordinates requires calculation/conversion each time |
| Design 6: Interface with designs 2 and 3 as classes implementing it. | * Ties together designs 2 and 3, making the full design versatile like Design 1 | * Requires interface class file |

## Hypothesis of Running Times

The greatest factor in running time is whether the method requires calculating the conversion between and . Running time is also affected by the checking of coordinate type in the case of Design 1.

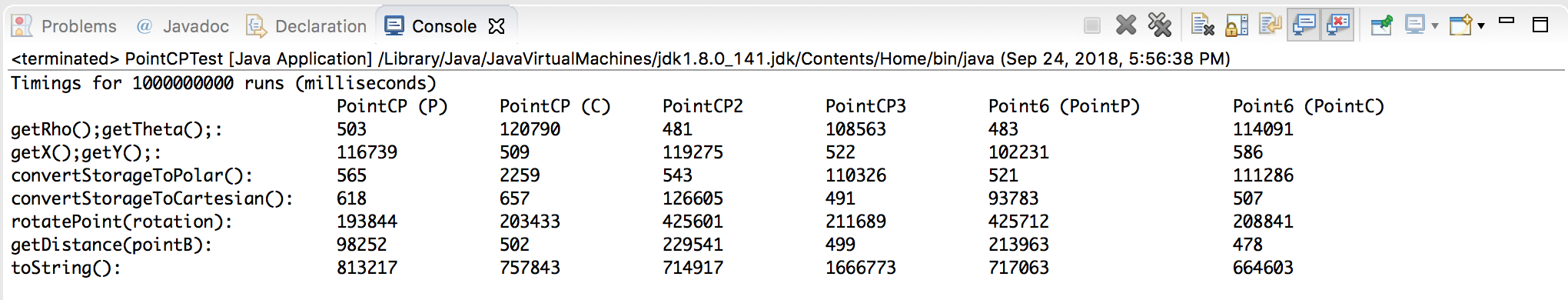
Estimated relative running times for each method:

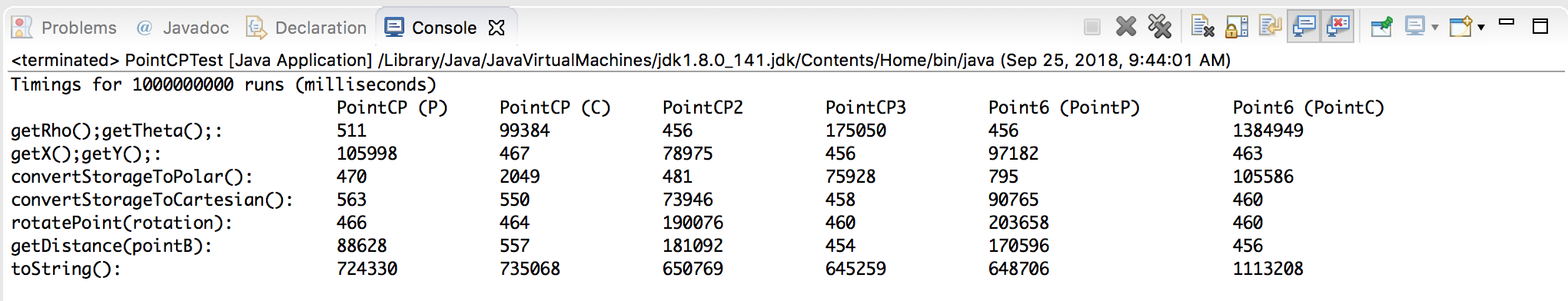
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | PointCP (P) | PointCP (C) | PointCP2 | PointCP3 | Point6 (PointP) | Point6 (PointC) |
| getRho();getTheta();: | fast | slow | fastest | slow | fastest | slow |
| getX();getY();: | slow | fast | slow | fastest | slow | fastest |
| convertStorageToPolar(): | fast | slower | fast | slow | fast | slow |
| convertStorageToCartesian(): | slower | fast | slow | fast | slow | fast |
| rotatePoint(rotation): | slow | faster | slow | faster | slow | faster |
| getDistance(pointB): | slow | fast | slow | fast | slow | fast |
| toString(): | slow | slow | slow | slow | slow | slow |

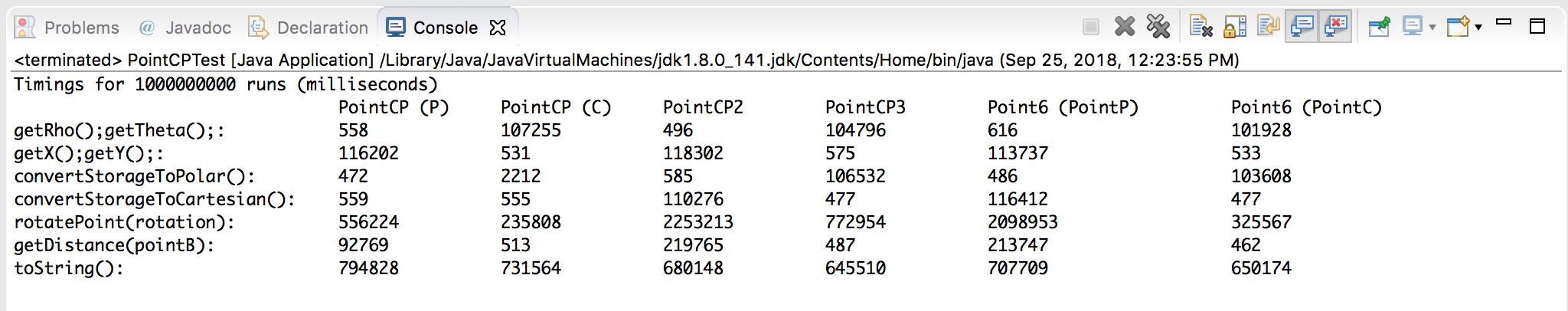
## Test Procedure

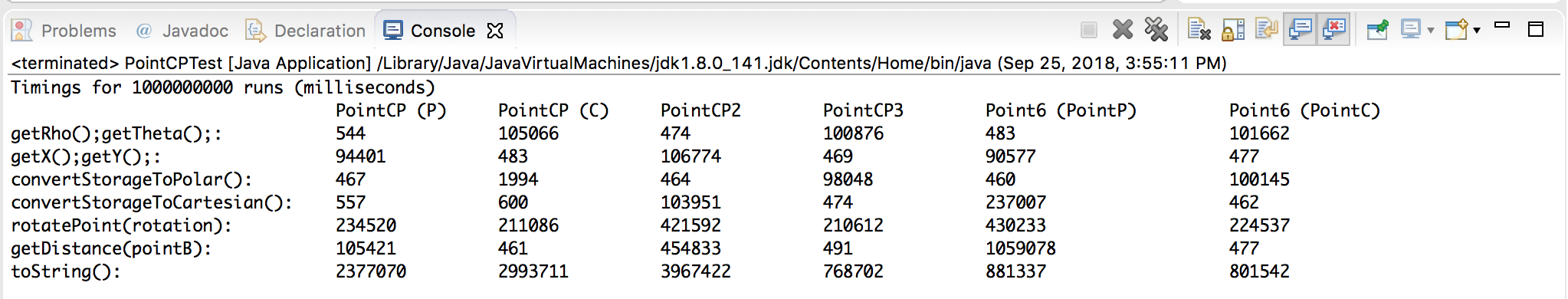
1. Create random instances of each design class (polar and cartesian).
2. For each instance, call each method (getRho(), getTheta(), convertStorageToPolar() etc), times and record time in milliseconds. Choose sufficiently large to get accurate timings.

## Results









## Discussion

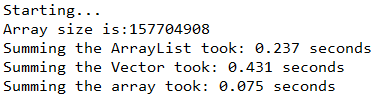
* getDistance() is slow for polar coordinates because it requires converting both sets of coordinates to Cartesian.
* It’s weird how converting PointCP from polar to Cartesian storage is so fast.
* getString() is slow due to the string concatenation/formatting operation.
* Designs 2, 3 and 6 are more specialized (they are fast depending on the type of coordinates being used. However, Design 1 is more versatile and better for dealing with both types of coordinates together.

# Arrays – Efficiency tests for summing large collections

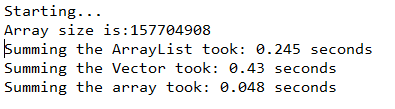
## Summary:

In these experiments, a very large collection of randomly generated numbers will be summed from Vectors, ArrayLists, and Arrays to see which structure is the most efficient. The size of the large collection is determined by adding to the ArrayList in a for-loop for approximately 10 seconds.

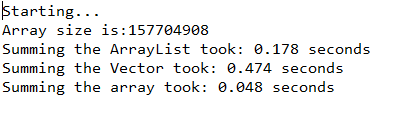
## Trial 1:



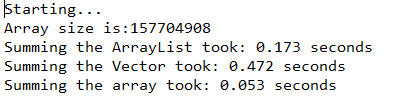
## Trial 2:



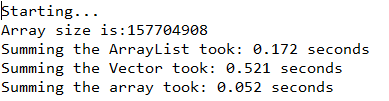
## Trial 3:



## Trial 4:



## Trial 5:



**Average time amongst all trials:**

ArrayList: 0.201 seconds

Vector: 0.466 seconds

Array: 0.055 seconds

**Variance between max and min times:**

ArrayList: 0.073 seconds

Vector: 0.091 seconds

Array: 0.023 seconds

## Conclusion

For reasons unknown, ArrayList sum time began to shorten as the trials went on but stabilized around 0.170 seconds. The Vector summing showed the most variance without a discernable pattern as to why, from 0.521 seconds to 0.43 seconds, around a 20% difference. The fastest structure to sum 157704908 numbers is the array by an order of magnitude, however it also showed the largest variance between min and max times, 0.023 seconds, which is a 48% increase in the fastest time.

Possible explanations for the variance in the ArrayLists and Vectors could be the fact that iterators were used for ArrayList and Vectors, or ArrayLists and Vectors are dynamically sized. The Array did not use an iterator, and summed through a for loop, by index.

Despite the large variance, I would still recommend the Array to sum large numbers.