PROGRAMMING ASSIGNMENT 3

Due: Thursday, February 12th at 11:59pm

This assignment will provide you experience working with

- Recursion
- Analyzing different algorithmic solutions for the same problem.
- Use of System.nanoTime() to compute the approximate algorithm time

Making Change

In this assignment you will write different programs to make change. Assume you are assigned a way to automate making change for a change machine similar to ones pictured:



Your job is to design and evaluate the algorithm for figuring out what coins to return where the goal is to return the fewest total coins. The company has told you that you have to be able to handle change from 1¢ to 1000 cents and you can assume that the number of coins available to return is limitless. You want to recommend the fastest algorithm that works correctly returning the fewest coins. In this assignment you will consider four different algorithms that can all make change in quarters, dimes, nickels and pennies.

1. Simple solution using loops.
    a. In this implementation, just use a loop and subtract the coin value until the amount left is less than the coin value.
2. Use of math operators
    a. In this implementation, you recall that you can take advantage of integer division to compute the coins of each value and the mod operator for the remaining change.
    b. For example, if the `change` amount is $0.84, you can figure out the number of quarters, and update the change left with the following:

```
int quarters = change/QUARTER;
change = change % QUARTER;
```

3. Recursion
    a. An alternative, similar to the loop solution, is to use recursion to count (or remove a coin). The recursive method should be general enough to handle all coin types. The base case should return when the change amount is less than the coin amount.
    b. Do not use tail recursion in this example.
4. Tail recursion
    a. Tail recursion will have a similar algorithm as 3, except the final executable command in your method is the recursive call. You are including this to gain insight into how the Java compiler deals with tail recursion.

You will test each algorithm to ensure it is correct via JUnit (no need to pass in) and to figure out which is, on average, the fastest.

Timing algorithms

Java provides a number of algorithms that return the elapsed time "from some arbitrary time (perhaps in the future, so values may be negative)." You can compute the nanoseconds via the following code:

```
startTime = System.nanoTime();
// Execute your selected algorithm for 1 cent to 1000
stopTime = System.nanoTime();
long elapsedTime = stopTime – startTime;
```

The elapsed time is not the time spent executing your algorithm since all of our computers are now multiprocessing tasks. As such, you will need to average your elapsed time over a number of runs to get a fair time to compare to the other algorithms. For a wonderful article on timing and the difference between currentTimeMillis and nanoTime read https://blogs.oracle.com/dholmes/entry/inside_the_hotspot_vm_clocks

Implementation

You will design and implement your classes to match the following UML class diagram. You may add new `private` methods and fields only.

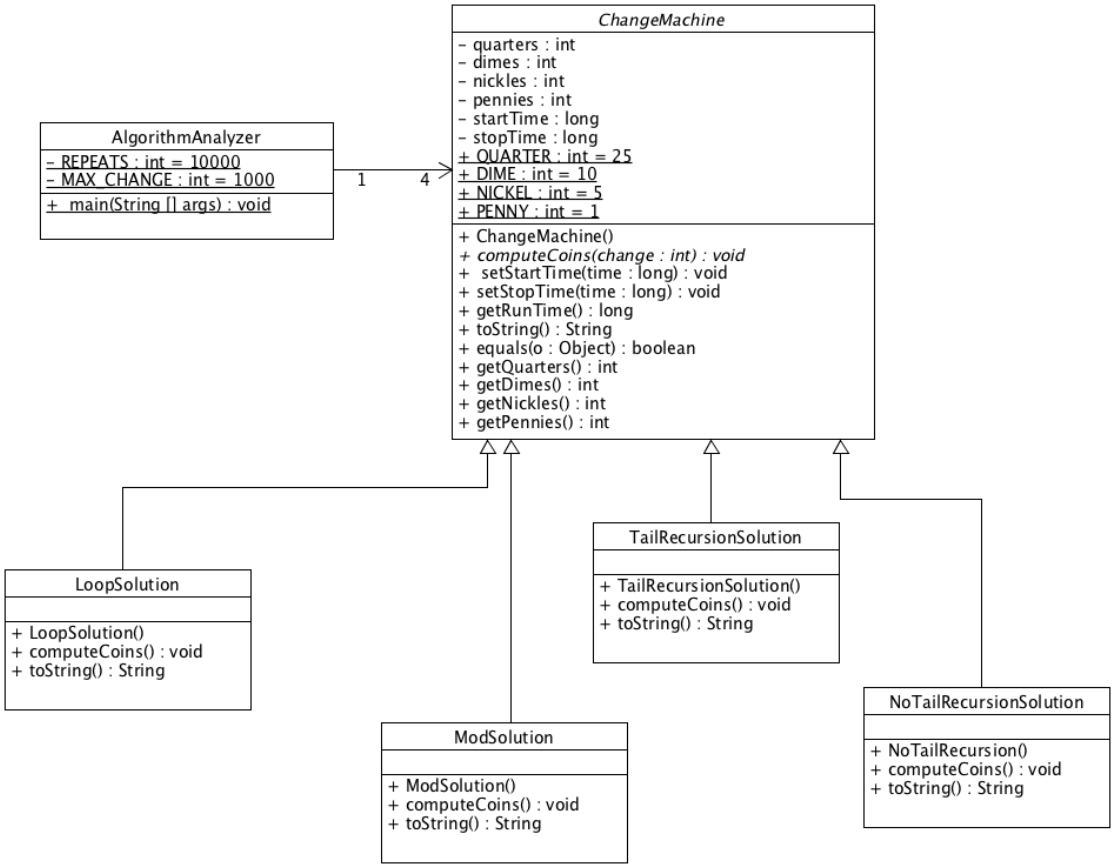

Figure 1. Assignment 3 UML Class Diagram

Submission:

Your submission should be 1 zipped file uploaded to blackboard containing:

- AlgorithmAnalyzer.java which runs the simulations for the ChangeMachine. You may either take the median or mean of the 10000 (REPEATS) runs to analyze your algorithms for computing change from 1 cent to 1000 cents.
- ChangeMachine.java is an abstract class containing all of the methods and fields common to the different algorithms. The method computeCoins must be abstract.
- LoopSolution.java, ModSolution.java, TailRecursiveSolution.java, NoTailRecursiveSolution.java
- ChangeMachineTest.java which verifies the solutions are correct for each algorithm.

In addition, you must write a short paragraph on your blackboard submission that identifies the 'fastest' algorithm of the four, along with its winning time and whether Java supports tail recursion optimization or not based on the timing of your two different recursive algorithms.

Extra Credit:

Consider adding the ability to return bills too ($1, 5, 10, 50, 100 only). Submit your redesigned UML Diagram and implementation of the 'fastest' change computing class. You should have a new driver AlgorithmAnalyzer2.java.