Principles of Java Language with Applications, PIC20A
D. Hyde
Spring 2020

Homework 1
Due 11am, Friday, April 10, 2020

For problem 1, write a `public class` named `MyMath` with 1 member function `sqrt`. (So the filename must be `MyMath.java`.) For problem 2, add 1 member function named `main` to your `MyMath` class, so you can run `MyMath.java` as a program.

We may take up to 20% of the total marks for poor style; make sure to name your variables reasonably, indent properly, and comment sufficiently. Submit `MyMath.java`.

Problem 1: (Square root)

In this homework assignment, write a function that computes the square root of a given `double` up to a certain precision. This function should be a member function of of `class MyMath` and should have signature

```
public static double sqrt(double d)
```

You will use a technique called *bisection* or *binary search* for its implementation.

First assume the input, which we call d, is between 0 and 1. Then we know that $l \leq \sqrt{d} \leq u$, where $l = 0$ and $u = 1$. Here, l represents a lower bound for the (yet unknown) value of $\sqrt{d}$, and u represents an upper bound.

To perform bisection, take the midpoint $m = (l+u)/2$ and check whether $\sqrt{d}$ is within the interval $[l, m]$ or $[m, u]$. To do so, check whether $d \leq m^2$, since if $d \leq m^2$ then $\sqrt{d} \leq m$. If $d \leq m^2$ then $\sqrt{d}$ is in the interval $[l, m]$. So perform the update $u = m$. If $m^2 < d$ then $\sqrt{d}$ is in the interval $[m, u]$. So perform the update $l = m$.

By iterating this process, we get a successively tighter interval $[l, u]$ that contains $\sqrt{d}$. If the length of this interval is small enough, i.e., if $u - l$ is small enough, we terminate this process; the midpoint $m = (l + u)/2$ will be very close to the true value of $\sqrt{d}$. For the purpose of this assignment, you can terminate when the interval is smaller than 10^{-10}.

Finally, we have to deal with the case when $d > 1$. One way to handle this case is by *normalizing* the input. The approach is based on the simple insight

$$\begin{aligned}\sqrt{d} &= 2\sqrt{d/4}\\ \sqrt{d} &= 2^2\sqrt{d/4^2}\\ \sqrt{d} &= 2^3\sqrt{d/4^3}\\ &\vdots\end{aligned}$$

So loop and divide d by 4 until you reach a value less than or equal to 1. Compute the square root of the dividend, and make sure to multiply back the appropriate power of 2.

You may not use `Math.sqrt` in the implementation of `MyMath.sqrt`.

Problem 2: (Testing)

Write code that compares the speed of your `MyMath.sqrt` with `Math.sqrt` in the way specified below. This code should go into the `public static` member function named `main` with return type `void`.

Using a loop, evaluate `Math.sqrt` on $10,000,000$ random numbers between 0 and 100. These random numbers should be generated by calling `100*Math.random()`. Measure the time it takes to complete this task using `System.currentTimeMillis`. Do the same with `MyMath.sqrt`. You may want to consult:
`https://docs.oracle.com/javase/7/docs/api/java/lang/System.html#currentTimeMillis()`

This time measurement includes the time it takes to generate the random numbers. With a separate loop, measure the time it takes to generate the random numbers without evaluating the square root.

Putting these measurements together, output, to the command line, the average execution times per evaluation of the two square root functions with the execution time of the random number generation subtracted out.

You will see that `Math.sqrt` is far superior over `MyMath.sqrt`. This is to be expected since standard math functions like `Math.sqrt` are implemented and optimized by a group of experts over many hours.