Technical documentation

Module 5: Documentation

Jack Oporto

October 10, 2020

Professor Al Kafaf

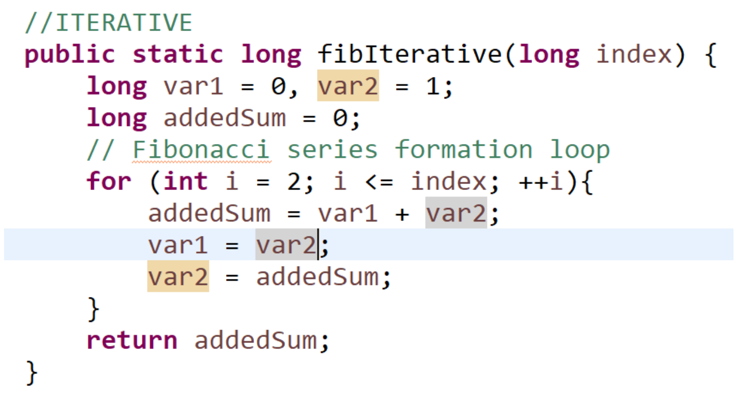
Summary:

This program demonstrates the differences in runtime performance between an iterative and recursive function. While recursive functions are good for most tasks, iterative functions run faster when dealing with enormous numbers.

The enormous numbers in this example use the Fibonacci sequence.

On run, the program will run calculations for the given number of times to calculate a Fibonacci number (40). Once it has completed both recursive and iterative calculations, a graph will display the milliseconds it took to calculate the number at each index.

Iterative function



```
//ITERATIVE
public static long fibIterative(long index) {
    long var1 = 0, var2 = 1;
    long addedSum = 0;
    // Fibonacci series formation loop
    for (int i = 2; i <= index; ++i){
        addedSum = var1 + var2;
        var1 = var2;
        var2 = addedSum;
    }
    return addedSum;
}
```

In this function, the index is passed in and used to determine what Fibonacci number we're calculating.

Since the Fibonacci number always deals with two separate numbers, our initial values are the first two Fibonacci numbers, var1 and var2, (0, 1)

First they are added to each other and placed in addedSum.

Then var2's value is placed into var1 and addedSum's value is placed into var2.

The loop repeats to completion.

Recursive function

```
//RECURSIVE (Calls itself)
public static long fibRecursive(long index) {
    if (index == 0) {
        return 0;
    } else if (index == 1) {
        return 1;
    } else {
        return fibRecursive(index - 1) + fibRecursive(index - 2);
    }
}
```

In this function, the index is passed in and used to determine which Fibonacci number we're calculating.

The function performs two checks on every use: If the index is 0, the value is 0. If the index is 1, the value is 1. These are the first two Fibonacci numbers so no further calculation is required.

Ex: fibRecursive(3)

    return fibRecursive(3-1) + fibRecursive(3-2) -> return fibRecursive(3)

If the index is greater than 1, it calls its function within itself, returning the sum of the previous number and the number before that in the index.

JavaFX

The code utilizes the built in NumberAxis and LineChart objects to draw the graph.

```
//This gets the data and puts it in the corresponding chart
for(int i = 0; i<=40 ; i++) {
    series1.getData().add(new XYChart.Data(i, getRecTime(i)));
    series2.getData().add(new XYChart.Data(i, getIterTime(i)));
}
```

This code generates the data for our points and goes up to index 40.

Time calculation

```
//This returns back the milliseconds it took to perform the calculation
public static long getRecTime(long index) {
    long startTime = System.nanoTime();
    fibRecursive(index);
    long endTime = System.nanoTime();

    return (endTime-startTime);
}
```

This code passes in the index from the for loop described above and uses it to determine which Fibonacci number is calculated. First it records the time in nanoseconds using the System, then it runs the calculation (Recursive or iterative depending on the loop, both are nearly identical). Once the calculation is complete the time is recorded again.

It subtracts the end time by the start time and the difference is the total time it took to calculate the Fibonacci number at that index.