Hogeschool van Amsterdam

Datastructures

Assignment 1A

Tijmen Stor, 500752989 IS205
HBO-ICT Software Engineering
Marcio Fuckner

Index

Introduction

In this report, I will answer the questions on assignment 1A. These questions involve the Fibonacci sequence written in Java code. The main focus of this report is looking at the separate assignments within the assignment. We'll be looking at a small pre-written code that calculates and then displays the Fibonacci sequence. During this assignment, I will find out that the code is inefficient after running it for a while, which will lead to me creating a new method to calculate the Fibonacci numbers. This batch of code will proof to be more efficient and shorten the process of calculating the numbers.

Assignments

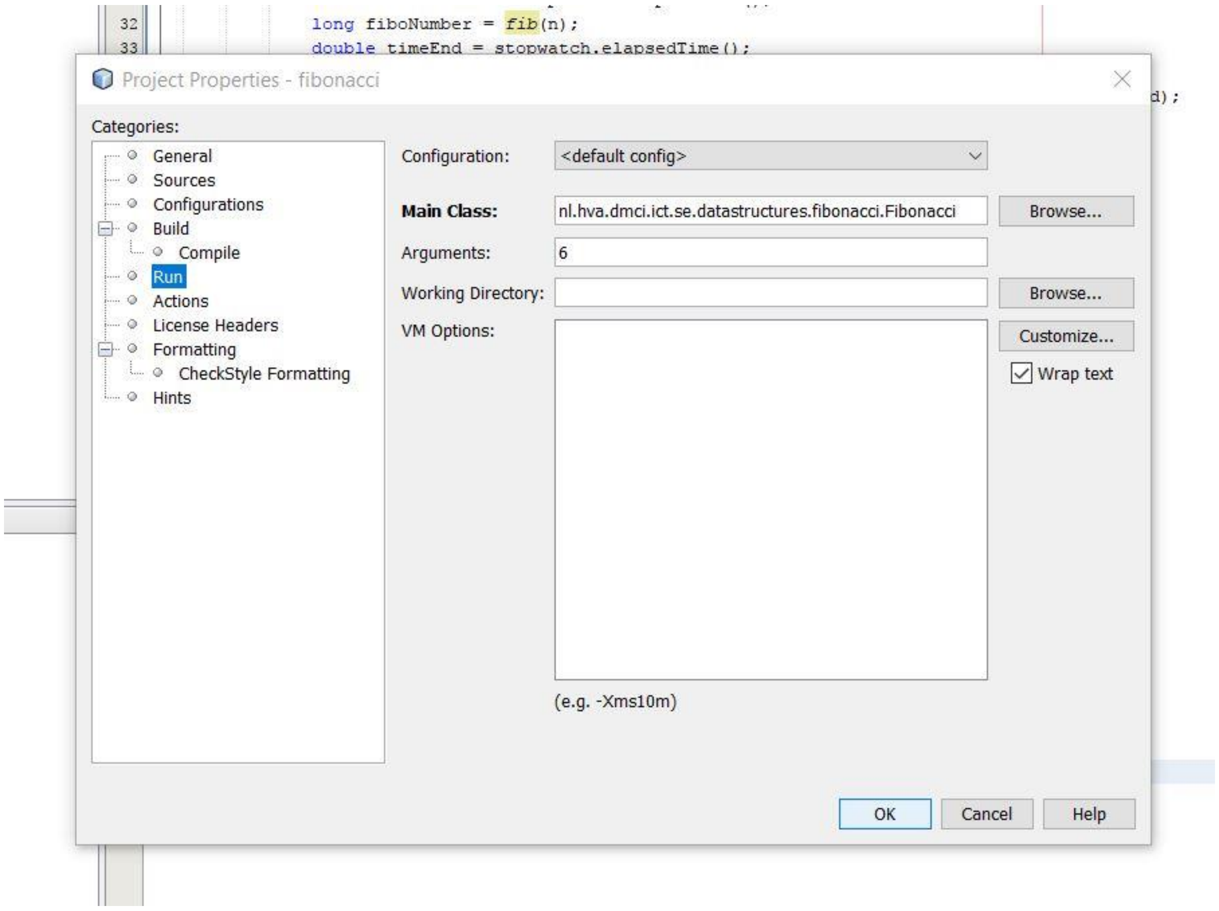
In this chapter, I will answer the questions from assignment 1A.

Sub-assignment 1A-1

In this sub-assignment, I will be answering the question:

Fibonacci.main can be given command line arguments. You are able to give those through command prompt, but also through the IDE you are using. How do you give a command line argument to the main within your IDE?

For every Java project I make, I use Netbeans IDE. Within the IDE, right click on the project. Click on properties. This opens up a new window. In this window, click on the run tab. In here, you can fill in the arguments to give to the main. This looks something like this:



```
long fiboNumber = fib(n);
double timeEnd = stopwatch.elapsedTime();
```

Project Properties - fibonacci

Configuration: <default config>

Main Class: nl.hva.dmci.ict.se.datastructures.fibonacci.Fibonacci

Arguments: 6

Working Directory:

VM Options:

(e.g. -Xms10m)

Here you can see the main class Fibonacci being chosen and the arguments given to it. In this instance, Fibonacci won’t go further than the sixth number.

Sub-assignment 1A-2

In this sub-assignment, we will be looking at the code from the main and why it is so slow. The question here is:

The implementation from Fibonacci is really slow. Analyse the algorithm and describe why it is so slow.

Disclaimer: My Netbeans would not install a RTF Copy/Paste plugin, so I am manually copying and pasting code.

The Fibonacci code to calculate the numbers currently look like this:

```
    public static long fib(int n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
}
```

N means which Fibonacci number it currently is and the method returns the value that is connected to that number. Here is why it is so slow. After the else, the method is calling itself twice again to calculate the numbers. This means for every calculation in the method, two new ones have to be calculated yet again. In theory this means, that every number value is being calculated again. When the Fibonacci number is low, it’s not a problem. But when the number rises, it has to calculate every old number again in the row, all the way down to the first number. This takes a lot of time and computer power.

Sub-assignment 1A-3

This sub-assignment is where the coding starts. The question is as follows:

Rewrite the method fib(int n) so it becomes faster. You will have to write your own new method with the same arguments. Your method will look like this: mijnFib(int n). What is the drawback from your method in comparison to the pre-written method?

Rewriting this code was fairly hard, because you need to find another logic answer to calculating each number, with the added benefit that it has to be faster than the old code. Instead of making the method call itself to calculate new values, the new method will be focused on calculating the number instantly within the method. It looks like this:

```
    private static int a = 0;
    private static int b = 1;

   public static long mijnFib(long n) {
        if (n < 2) {
            return n;
        } else {
            n = a + b;
            a = b;
            b = n;
            return n;
        }

    }
}
```

In this method, the value n which was used as the number at which Fibonacci was is used to display the value too. The value is being changed into a + b. In this case, a is the value of two Fibonacci numbers back and b is the value of one Fibonacci number back. After the calculation, the values of a and b are calculated again to be new values, or more specific, to move up a Fibonacci number value. This is a faster method, but not the answer to the question yet.

The drawback between the new method and the old method is that it uses a bit more code and makes use of two new pre-defined variables. The downside to this is that the code is bigger, but the method is way faster.

Sub-assignment 1A-4

This question involves numbers changing from positive into negative. The question is as follows:

What happens with the values at F(93)? How does this happen and how can you solve it?

What happens at F(93), is that the value is too big to fit into the long type. Because of that, the value will change into a negative value. To fix this issue, changing a couple of variables into a new data type called BigInteger. BigInteger has a greater length than long and can thus store bigger values. The code will look like this.

```
    private static int a = 0;
    private static int b = 1;
    private static BigInteger c = new BigInteger(String.valueOf(a));
    private static BigInteger d = new BigInteger(String.valueOf(b));

    public static void main(String[] args) {

        final int SIZE;

        if (args.length == 0) {
            SIZE = 1000;
        } else {
            SIZE = Integer.parseInt(args[0]);
        }

        Stopwatch stopwatch = new Stopwatch();

        System.out.printf(" n %15s  (  lap \t total)\n", "fib(n)");
        for (int n = 1; n <= SIZE; n++) {
            double timeStart = stopwatch.elapsedTime();
            BigInteger fiboNumber = mijnFib(n);
            double timeEnd = stopwatch.elapsedTime();
            double lapTime = timeEnd - timeStart;
            System.out.printf("%2d %15d  (%.3f \t %.3f)\n", n, fiboNumber, lapTime, timeEnd);
        }
    }
    public static BigInteger mijnFib(long n) {
        if (n < 2) {
            BigInteger k = new BigInteger(String.valueOf(n));
            return k;
        } else {
            BigInteger k = new BigInteger(String.valueOf(n));
            k = c.add(d);
            c = d;
            d = k;

            return k;
        }

    }
}
```

Changing the data types within the method will preserve the values in its original state and takes care of the problem where numbers will become negative.

Conclusion

The big issue with the old method, was that the method was using itself multiple times to calculate the older values. After creating a new method which instantly calculates the new value and changes the old values to move up a Fibonacci number. This is more efficient and takes less processing power. The issue with the numbers changing into negative numbers after a certain number, was solved by changing data types. Long is only able to hold numbers under 9223372036854775807 and above -9223372036854775808. Because the numbers we produced became larger than the reach of Long, we had to change the data type to BigInteger, which can store way bigger numbers. This did mean we had to change a lot of data types within the code and creating new variables.