

Program Documentation

1) -----

The program description is outlined as: Given two positive integers A and B, write a program to calculate the greatest common divisor (GCD) of the sum of the first A Fibonacci numbers and the sum of the first B Fibonacci numbers. The program can be in either C or Java.

Input: Two positive integers A and B.

Output: The GCD of the sum of the first A Fibonacci numbers and the sum of the first B Fibonacci numbers.

FibonacciGCD.java

The programming language I chose to do this task is Java. I simply created the actual program that contains the algorithm under 'FibonacciGCD.java'. I call the method that does the algorithm "computeGCD()". For now, this is represented simply, before the logic is added:

```
public class FibonacciGCD {  
    public FibonacciGCD() {}  
  
    public int computeGCD(int a, int b) {  
        return -1;  
    }  
}
```

It is easier to first find the sum of the first N Fibonacci numbers, thus I first created a different method which first finds this very thing. I call this "sumOfFirstNFibonacciNumbers()".

```
private static int sumOfFirstNFibonacciNumbers(int n) {  
    // Initialize tracking of Fibonacci numbers  
    int sum = 0;  
    int a = 0;  
    int b = 1;  
    // Loop to find first n Fibonacci numbers and add them to sum  
    for (int i = 0; i < n; i++) {  
        sum += a;  
        int temp = a;  
        a = b;  
        b = temp + b;  
    }  
    // Return sum  
    return sum;  
}
```

Next, it would also be very helpful if there is a method which finds the greatest common divisor between two numbers. Following the Euclidean Algorithm, we can create a recursive method. I called this method "getGCD()".

```
private int getGCD(int a, int b) {  
  
    // Make sure a is greater than b  
    if (a < b) {  
        int temp = a;  
        a = b;  
        b = temp;  
    }  
  
    // Base case  
    if (b == 0) {  
        return a;  
    }  
  
    // Recursive case  
    return getGCD(b, a % b);  
}
```

With these two methods in our grasps, it becomes very easy to find the GCD of the sum of the first A Fibonacci numbers and the sum of the first B Fibonacci numbers.

```
public static int computeGCD(int a, int b) {  
  
    // If a or b is 0, return 0  
    if (a == 0 || b == 0) {  
        return 0;  
    }  
  
    // Find sum of first a and b fibonacci numbers  
    int sumA = sumOfFirstNFibonacciNumbers(a);  
    int sumB = sumOfFirstNFibonacciNumbers(b);  
  
    // Return gcd of sum of first a fibonacci numbers and sum of first b fibonacci numbers  
    int result = getGCD(sumA, sumB);  
    return result;  
}
```

DisplayFibonacci.java

I also made DisplayFibonacci.java class. This class will contain the bulk of what will be shown on the program. That is, apart from the printed texts from FibonacciGCD.java. Thus, the main.java will have to simply call this class's method "run()" to run the entire program.

```
1  package src.main.java;
2  import java.util.Scanner;
3
4  public class DisplayFibonacci {
5      // DisplayFibonacci class is responsible for the user interface of the program
6      private boolean running = true;
7      private Scanner scanner = new Scanner(System.in);
8
9      // Constructor
10     public DisplayFibonacci() {}
11
12     // Main method to run the program
13     public void run() { ...
21
22     // Print the header of the program
23     public void printHeader(){ ...
47
48     // Display the menu commands of the program
49     public boolean menu(){ ...
71
72     // Access the FibonacciGCD class
73     public void calculate(){ ...
83
84     // Get user input for the nth number
85     public int getNumberInput(String nth){ ...
109
110     // Print the footer of the program
111     public void printFooter(){ ...
117 }
118
```

The run() method simply calls on the other methods to run them in the correct order, by repeating the menu() method until the user uses the EXIT command.

```
// Main method to run the program
public void run() {
    printHeader();
    while (running) {
        running = menu();
    }
    printFooter();
    scanner.close();
}
```

The printHeader() and printFooter() methods simply prints out texts that indicates the start and end of the program. They are only made to make the program cleaner to look at; thus the ascii art for the header.

```
public void printHeader(){
    // Ascii art saying SOFTENG 282 and the program description
    System.out.println(x:"\n");
    System.out.println(x:" .00000000. .00000000. 000000000000 000000000000 000000000000 00000 00000 .00000000. ");
    System.out.println(x:" 888' `888' .888' `888. `888' `8 8' `8888' `8 `888' `8 `8888. `888' .888' `888' ");
    System.out.println(x:" `888. `'' 888' `888 888 . 8888 888. . 88888. 888 888' . `'' ");
    System.out.println(x:" `888888. 888 888 88800008 8888 88800008 888 88 888 888 0000. ");
    System.out.println(x:" ... `888. 888. .888 888 ' 8888 888' ' 888 `88888 888. ' 888' ");
    System.out.println(x:" `888. .888 `888. .888' 888 .8888. .888. .8 888 `8888 `888. .888' ");
    System.out.println(x:" `99999999' `99999999' 888 .o8888o. o88800000000 .999. .999. `99999999' \n");
    System.out.println(x:" .000000000. .00000000. .000000000. ");
    System.out.println(x:" 888' `888' `888' `888' 888' `888' ");
    System.out.println(x:" `'' .888' `888. .888' `'' .888' ");
    System.out.println(x:" .888' `888888. .888' ");
    System.out.println(x:" .888' . .888' `888. .888' . ");
    System.out.println(x:" .888' .8' `888. .888' .888' .8' ");
    System.out.println(x:" 00000000009' `99999999' 00000000009' ");
    System.out.println(x:"-----");
    System.out.println(x:"\n");
    System.out.println(x:"Welcome to the Fibonacci GCD Calculator! Insert below which Ath and Bth Fibonacci number do ");
    System.out.println(x:" you want to calculate the GCD of. \n");
}
```

```
public void printFooter(){
    System.out.println(x:"-----");
    System.out.println(x:" Thank you for using the Fibonacci GCD Calculator! Have a great day! \n");
    System.out.println(x:" Author: Toshiro Mendoza ID: 834872958 ");
    System.out.println(x:" SOFTENG 282 - Assignment 2 \n");
}
```

The menu() method asks the user which command they want to use. The repeating nature of this method in the run() method makes it possible for the user to use this program again and again until satisfied of calculating all they need to calculate. The two available commands are:

CALCULATE: Calculate the GCD of the sum of the first A and B Fibonacci numbers.

EXIT: Exit the program.

```
// Display the menu commands of the program
public boolean menu(){
    System.out.println(x:"Please select your next command:\n");
    System.out.println(x:"CALCULATE - Calculate the GCD of the sum of the first A and B Fibonacci numbers");
    System.out.println(x:"EXIT - Exit the program\n");
    System.out.print(s:">> ");
    String command = scanner.nextLine();
    System.out.println(x:"");

    if (command.trim().equalsIgnoreCase(anotherString:"CALCULATE")){
        calculate();
        System.out.print(s:"Press Enter to continue...");
        scanner.nextLine();
        System.out.println(x:"\n-----");
    } else if (command.trim().equalsIgnoreCase(anotherString:"EXIT")){
        return false;
    } else {
        System.out.println(x:"Invalid command. Please try again.\n");
        menu();
    }
    return true;
}
```

The `getNumberInput()` method is the method that obtains the desired A and B needed to calculate the GCD of the sum of the first A and B Fibonacci numbers. This method specifically needs a string input which specifies whether the number being obtained is for number "A" or "B".

```
// Get user input for the nth number
public int getNumberInput(String nth){
    int n = 0;
    boolean haveN = false;

    while (!haveN) {
        System.out.print("Insert "+ nth +": ");
        String input = scanner.nextLine();

        try {
            n = Integer.parseInt(input);
        } catch (Exception e) {
            System.out.println(x:"Invalid input. Please enter a number.");
            continue;
        }

        if (n <= 0) {
            System.out.println("Invalid input. Cardinality "+ nth +" must be greater than 0.");
            continue;
        }
        haveN = true;
    }

    return n;
}
```

Then lastly, the `calculate()` method uses the `getNumberInput()` function to obtain A and B, then uses it to calculate the desired output by calling the `FibonacciGCD` class which contains the desired algorithm.

```
public void calculate(){
    // Get user input for A
    int a = getNumberInput(nth:"A");
    int b = getNumberInput(nth:"B");

    // Initialize FibonacciGCD object
    System.out.println(x:"");
    FibonacciGCD fibonacciGCD = new FibonacciGCD();
    fibonacciGCD.computeGCD(a, b);
}
```

The `Main.java` class thus then simply looks like:

```
package src.main.java;
public class Main {
    Run | Debug
    public static void main(String[] args) {
        DisplayFibonacci display = new DisplayFibonacci();
        display.run();
    }
}
```

2) -----

Next task to do is the following: Add suitable output commands to the GCD algorithm that show the progress of the algorithm.

We can print into the command line the Fibonacci numbers as they are being added into the sums. This tells us the progress of the `sumOfFirstNFibonacciNumbers()`. Thus, I rewrote this method as:

```
private int sumOfFirstNFibonacciNumbers(int n) {  
    // Initialize tracking of Fibonacci numbers  
    int sum = 0;  
    int a = 0;  
    int b = 1;  
  
    // Print first n Fibonacci numbers  
    System.out.print("The first " + n + " fibonacci numbers are: ");  
  
    // Loop to find first n Fibonacci numbers and add them to sum  
    for (int i = 0; i < n; i++) {  
        System.out.print(a + " ");  
        sum += a;  
        int temp = a;  
        a = b;  
        b = temp + b;  
    }  
  
    System.out.print(s:"\n");  
  
    // Return sum  
    return sum;  
}
```

I also modified the `computeGCD()` method to track what the sums are and display what the resulting GCDs are.

```
public int computeGCD(int firstFibonacciSeries, int secondFibonacciSeries) {  
    // If a or b is 0, return 0  
    if (firstFibonacciSeries <= 0 || secondFibonacciSeries <= 0) {  
        System.out.println(x:"Invalid input. Cardinality must be greater than 0.\n");  
        return 0;  
    }  
  
    // Find sum of first a and b fibonacci numbers  
    int sumA = sumOfFirstNFibonacciNumbers(firstFibonacciSeries);  
    System.out.println("Sum of first " + firstFibonacciSeries + " fibonacci numbers: " + sumA);  
  
    int sumB;  
    if (firstFibonacciSeries != secondFibonacciSeries) {  
        sumB = sumOfFirstNFibonacciNumbers(secondFibonacciSeries);  
        System.out.println("Sum of first " + secondFibonacciSeries + " fibonacci numbers: " + sumB);  
    } else {  
        sumB = sumA;  
    }  
  
    // Return gcd of sum of first a fibonacci numbers and sum of first b fibonacci numbers  
    int result = getGCD(sumA, sumB);  
    System.out.println("\nGCD of sum of first " + firstFibonacciSeries + " and " + secondFibonacciSeries + " fibonacci numbers: " + result + "\n");  
    return result;  
}
```

3) -----

Next task to do is the following: Add suitable comments to your code that would help a code reviewer.

4) -----

Next task to do is the following: Provide a readme file that explains your solution to a reader with sample output similar to the output shown in the question above and contains other instructions i.e. how to compile and run the program.

Proper and adequate comments on how the program works are added during the programming process already. The README.md file should be submitted to the same folder as this document.

5) -----

Next task to do is the following: Test your code and document this. Introduce, if useful, bounds for the input.

FibonacciGCDTest.java

For the test, I created FibonacciGCDTest.java class. This test is divided into two subclasses. The first tests the algorithm itself. While the second tests what is displayed when commands are used.

```
1 // Import Junit test classes
2 package src.test.java;
3 import org.junit.Test;
4 import org.junit.Assert;
5 import org.junit.runner.RunWith;
6 import org.junit.runners.Suite;
7 import org.junit.runners.Suite.SuiteClasses;
8
9 // Import input output classes
10 import java.io.ByteArrayInputStream;
11 import java.io.InputStream;
12 import java.io.OutputStream;
13 import java.io.ByteArrayOutputStream;
14 import java.io.PrintStream;
15
16 // Import classes to be tested
17 import src.main.java.FibonacciGCD;
18 import src.main.java.DisplayFibonacci;
19
20
21 @RunWith(Suite.class)
22 @SuiteClasses({
23     FibonacciGCDTest.ComputeGCDTests.class,
24     FibonacciGCDTest.DisplayTests.class
25 })
26 public class FibonacciGCDTest {
27
28     public static class ComputeGCDTests { ...
29
30
31     public static class DisplayTests { ...
32 }
```

ComputeGCDTests Class

The first test subclass is called ComputeGCDTests and is composed of 10 test cases.

```
public static class ComputeGCDTests {  
  
    @Test  
    public void test_example() { ...  
  
    @Test  
    public void test_cardinal_A_is_zero() { ...  
  
    @Test  
    public void test_cardinal_B_is_zero() { ...  
  
    @Test  
    public void test_cardinal_A_and_B_is_zero() { ...  
  
    @Test  
    public void test_cardinal_A_is_negative() { ...  
  
    @Test  
    public void test_cardinal_B_is_negative() { ...  
  
    @Test  
    public void test_cardinal_A_and_B_is_negative() { ...  
  
    @Test  
    public void test_example_2() { ...  
  
    @Test  
    public void test_example_3() { ...  
  
    @Test  
    public void test_B_is_greater_than_A() { ...  
  
}
```

The test methods test_example(), test_example_2(), and test_example_3() are all test cases where the inputs are valid and checks whether the output GCD of sum of first A and B Fibonacci numbers are correct. The first one being the example given in the task specification.

```
@Test  
public void test_example() {  
    // Arrange  
    int a = 4;  
    int b = 3;  
    FibonacciGCD fib = new FibonacciGCD();  
  
    // Act  
    int result = fib.computeGCD(a, b);  
  
    // Assert  
    Assert.assertEquals(expected:2, result);  
}  
  
@Test  
public void test_example_2() {  
    // Arrange  
    int a = 5;  
    int b = 3;  
    FibonacciGCD fib = new FibonacciGCD();  
  
    // Act  
    int result = fib.computeGCD(a, b);  
  
    // Assert  
    Assert.assertEquals(expected:1, result);  
}
```

```
@Test  
public void test_example_3() {  
    // Arrange  
    int a = 9;  
    int b = 6;  
    FibonacciGCD fib = new FibonacciGCD();  
  
    // Act  
    int result = fib.computeGCD(a, b);  
  
    // Assert  
    Assert.assertEquals(expected:6, result);  
}
```


The test methods `test_cardinal_A_is_zero()`, `test_cardinal_B_is_zero()`, and `test_cardinal_A_and_B_is_zero()`, are all tests which look on what happens if at least one of the input numbers to the algorithm is 0. This should cause the algorithm to stop as there is no “0th” Fibonacci.

```
@Test
public void test_cardinal_A_is_zero() {
    // Arrange
    int a = 0, b = 3;
    FibonacciGCD fib = new FibonacciGCD();
    OutputStream outputStream = new ByteArrayOutputStream();
    System.setOut(new PrintStream(outputStream));

    // Act
    fib.computeGCD(a, b);

    // Assert
    Assert.assertEquals("Invalid input. Cardinality must be greater than 0.".trim(), outputStream.toString().trim());
    System.setOut(System.out);
}
```

```
@Test
public void test_cardinal_B_is_zero() {
    // Arrange
    int a = 3, b = 0;
    FibonacciGCD fib = new FibonacciGCD();
    OutputStream outputStream = new ByteArrayOutputStream();
    System.setOut(new PrintStream(outputStream));

    // Act
    fib.computeGCD(a, b);

    // Assert
    Assert.assertEquals("Invalid input. Cardinality must be greater than 0.".trim(), outputStream.toString().trim());
    System.setOut(System.out);
}
```

```
@Test
public void test_cardinal_A_and_B_is_zero() {
    // Arrange
    int a = 0, b = 0;
    FibonacciGCD fib = new FibonacciGCD();
    OutputStream outputStream = new ByteArrayOutputStream();
    System.setOut(new PrintStream(outputStream));

    // Act
    fib.computeGCD(a, b);

    // Assert
    Assert.assertEquals("Invalid input. Cardinality must be greater than 0.".trim(), outputStream.toString().trim());
    System.setOut(System.out);
}
```

This is followed by the `test_cardinal_A_is_negative()`, `test_cardinal_B_is_negative()`, and `test_cardinal_A_and_B_is_negative()`, which are methods that tests what happens to the algorithm if he input is negative. In such cases, the result should be the same when inputs are zero as there is no “Negative Nth” Fibonacci number.

```
@Test
public void test_cardinal_A_is_negative() {
    // Arrange
    int a = -1, b = 3;
    FibonacciGCD fib = new FibonacciGCD();
    OutputStream outputStream = new ByteArrayOutputStream();
    System.setOut(new PrintStream(outputStream));

    // Act
    fib.computeGCD(a, b);

    // Assert
    Assert.assertEquals("Invalid input. Cardinality must be greater than 0.".trim(), outputStream.toString().trim());
    System.setOut(System.out);
}

@Test
public void test_cardinal_B_is_negative() {
    // Arrange
    int a = 3, b = -1;
    FibonacciGCD fib = new FibonacciGCD();
    OutputStream outputStream = new ByteArrayOutputStream();
    System.setOut(new PrintStream(outputStream));

    // Act
    fib.computeGCD(a, b);

    // Assert
    Assert.assertEquals("Invalid input. Cardinality must be greater than 0.".trim(), outputStream.toString().trim());
    System.setOut(System.out);
}

@Test
public void test_cardinal_A_and_B_is_negative() {
    // Arrange
    int a = -1, b = -1;
    FibonacciGCD fib = new FibonacciGCD();
    OutputStream outputStream = new ByteArrayOutputStream();
    System.setOut(new PrintStream(outputStream));

    // Act
    fib.computeGCD(a, b);

    // Assert
    Assert.assertEquals("Invalid input. Cardinality must be greater than 0.".trim(), outputStream.toString().trim());
    System.setOut(System.out);
}
```

Then lastly, there is the `test_B_is_greater_than_A()` method which just checks what happens if numbers A and B are flipped instead of $A > B$ from other tests.

```
@Test
public void test_B_is_greater_than_A() {
    // Arrange
    int a = 3;
    int b = 5;
    FibonacciGCD fib = new FibonacciGCD();

    // Act
    int result = fib.computeGCD(b, a);

    // Assert
    Assert.assertEquals(expected:1, result);
}
```

DisplayTests Class

The second test subclass is called `DisplayTests` and is composed of 3 test cases.

```
public static class DisplayTests {

    @Test
    public void test_menu_exit() { ...

    @Test
    public void test_menu_invalid_command() { ...

    @Test
    public void test_run_calculate_multiple_times() { ...

}
```

The first test method called `test_menu_exit()` tests if the program properly if the EXIT command is selected by the user.

```
@Test
public void test_menu_exit() {
    // Arrange
    String command = "EXIT\n";
    InputStream inputStream = new ByteArrayInputStream(command.getBytes());
    System.setIn(inputStream);
    DisplayFibonacci display = new DisplayFibonacci();

    // Act
    boolean running = display.menu();

    // Assert
    Assert.assertFalse(running);
    System.setIn(System.in);
}
```

The second test method is called `test_menu_invalid_command()` which tests if the program would show "Invalid command. Please try again..."

```
@Test
public void test_menu_invalid_command() {
    // Arrange
    String command = "INVALID\n";
    InputStream inputStream = new ByteArrayInputStream(command.getBytes());
    System.setIn(inputStream);
    DisplayFibonacci display = new DisplayFibonacci();
    OutputStream outputStream = new ByteArrayOutputStream();
    System.setOut(new PrintStream(outputStream));

    // Act
    boolean running = display.menu();
    boolean contains = outputStream.toString().trim().contains(s:"Invalid command. Please try again.");

    // Assert
    Assert.assertTrue(contains);
    Assert.assertTrue(running);
    System.setIn(System.in);
    System.setOut(System.out);
}
```

The last test method is the `test_run_calculate_multiple_times()`. This fully uses the `run()` method instead of just one pass through `menu()`. This checks if the program would let the user calculate different wanted GCD and be able to exit.

```
@Test
public void test_run_calculate_multiple_times() {
    // Arrange
    String command = "CALCULATE\n5\n3\n\nCalculate\n9\n7\n\nEXIT\n";
    InputStream inputStream = new ByteArrayInputStream(command.getBytes());
    System.setIn(inputStream);
    DisplayFibonacci display = new DisplayFibonacci();
    OutputStream outputStream = new ByteArrayOutputStream();
    System.setOut(new PrintStream(outputStream));

    // Act
    display.run();
    boolean containsCalc1 = outputStream.toString().trim().contains(s:"GCD of sum of first 5 and 3 fibonacci numbers: 1");
    boolean containsCalc2 = outputStream.toString().trim().contains(s:"GCD of sum of first 9 and 7 fibonacci numbers: 2");
    boolean containsExit = outputStream.toString().trim().contains(s:"Thank you for using the Fibonacci GCD Calculator! Have a great day!");

    // Assert
    Assert.assertTrue(containsCalc1);
    Assert.assertTrue(containsCalc2);
    Assert.assertTrue(containsExit);
    System.setIn(System.in);
    System.setOut(System.out);
}
```

After making sure that all the test cases are passed we can say that the task is complete!

