UNIT
TEST

JUnit Test Fibonacci calculation .
Test Plan A - Positive Fibonacci

Kashaf Arshad , 217459

Molk Agbary , 217464

Product Quality and Management
Hoschule Heilbronn - HNN

Contents

Class under test: **FibonacciRecursive**
Test Object for the class : **FibonacciRecursiveTest**
This class calculates the Fibonacci number with a recursive algorithm.

1 Method under test: FibonacciRec ()

This method calls the recursive logic from within and prints the Fibonacci value. It accepts only integer values. It also has input validation to check if the input integer is within allowed ranges and raises InvalidInputException when validation fails.

Input (n)	**an integer**
Result (n)	**The Fibonacci number is printed using `System.out.println()` in the format "F(n) = Fibonacci value"**
Example (n)	**input $n = 0$ gives the results as "F(0) = 0".**

1.1 Devising Test Cases for FibonacciRec ()

1.1.1 Logical Test Cases

The test method FibonacciRec allows computation of Fibonacci number only within the range 0 – 46. Based on this, the following relation between input and results can be derived

Negative integers	**Invalid Input Exception**
Integers from 0 to 46	Fibonacci value is printed in format **"F(0) = 0"**
Positive Integers above 46	**Invalid Input Exception**

According to Spillner chapter 2, texts can be misinterpreted and it is better represented using mathematical formulae. Hence, the following logical or abstract test cases are derived:

Logical Test Case	**1**	**2**	**3**
Input value n (an integer)	$n < 0$	$0 \leq n \leq 46$	$n > 46$
Expected Results	Invalid Input Exception	Fibonnacci value of n in the format "F(0) = 0" should be printed to the output stream	Invalid Input Exception

1.1.2 Concrete Test Cases

According to Sommerville chapter 8, A good rule of thumb for test case selection is to choose test cases on the boundaries of the partitions, plus cases close to the midpoint of the partition. Thus, the concrete test cases are created by employing the partition testing strategy on the above logical test cases. We arrive at the following test cases based on the partitioning strategy:

Concrete Test Case No	**Input Value n (int)**	**Expected Results**
1	-100	InvalidInputException
2	-1	InvalidInputException
3	0	F(0) = 0
4	23	F(23) = 28657
5	46	F(46) = 1836311903
6	47	InvalidInputException
7	100	InvalidInputException

2 Method under test: Main ()

This is the main method of class. It triggers the method FibonacciRec () mentioned above, and processes the exceptions raised from it. It additionally does input check to ensure that the user has given some input and also validates that it is an integer that the user has supplied. Main () prints error messages when any error is detected. As these input validations are also essential for the fail proof execution of the program, it is imperative that these are also tested.

2.1 Devising Test Cases for Main()

2.1.1 Logical Test Cases

Main () method does a pre-check before calling the Fibonacci computation. Hence, the objectives of testing Main () are to test the correctness of user input. There is no need to test all the Fibonnaci boundary conditions here, as it gets tested as part of FibonacciRec (). Based on this, the relation between input and results are listed below:

For empty user input	**Raise error message.**
For non-integer user input	**Raise error message.**
For correct user input	**Execute Fibonacci logic successfully.**
For any exceptions raised from within FibonacciRec ():	**Raise error message.**

Hence, the following logical or abstract test cases are derived:

Logical Test Case	**1**	**2**	**3**	**4**
Input value n (an integer)	Blank	Non-Integer (Value which cannot be parsed into integer. Eg: Char string, float)	Integer between 0 and 46	$n < 0$ or $n > 46$
Expected Results	Error message: "Enter the arguments"	Error message: "Invalid Input. Please enter a Positive integer"	Fibonacci value of n in the format "F(0) = 0"	Error message from Invalid Input Exception

2.1.2 Concrete Test Cases

The below concrete test cases are created by employing the partition testing strategy on the above logical test cases:

Concrete Test Case No	**Input Value n (int)**	**Expected Results**
1	Empty String	Enter the arguments
2	Kashaf	"Invalid Input. Please enter a Positive integer"
3	-1	F(0) = "Invalid Input. Please enter a Positive integer"
4	23	F(23) = 28657
5	47	"Fibonacci number is too large for this input. Please enter a number less than 47!"

These test cases represent one entry from each of the input and output equivalence partitions for main (). Thus, it meets the testing requirement for all the input validations, printing of all the error messages, and a success scenario.

3 Fibonacci Recursive Algorithm

3.1 Main Class

```
import java.util.InputMismatchException;

public class FibonacciRecursive {
    private static final int MAX_N_VALUE = 46;

    public static void main(String[] args) {

        if (args.length > 0) {
            try {
                int n = Integer.parseInt(args[0]);
                new FibonacciRecursive(n);

            } catch (InvalidInputException
            | NumberFormatException e) {
                System.out.println("Invalid Input.
                Please enter a positive integer");
            }
            catch ( InputMismatchException e) {
                System.out.println("Fibonacci number
                is too large for this input.
                Please enter a number less than 47!");
            }
        } else System.out.println("Enter the arguments");
    }
    public FibonacciRecursive(int n)
    throws InputMismatchException, InvalidInputException {
        validateInput(n);
        System.out.println("F(" + n + ") = "
        + calculateFibonacci(n));
    }

    private void validateInput(int n)
    throws InputMismatchException, InvalidInputException {
        if (n < 0) {
            throw new InvalidInputException();
        }
        if (n > MAX_N_VALUE) {
```

```
            throw new InputMismatchException();
        }

    }

    private int calculateFibonacci(int n) {
        // Recursive algorithm
        if (n == 0) return 0;
        else if (n == 1) return 1;
        else return calculateFibonacci(n - 2)
        + calculateFibonacci(n - 1);
    }

}
```

3.2 Test Class

```
import org.junit.jupiter.api.*;

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import java.util.InputMismatchException;

import static org.junit.jupiter.api.Assertions.*;

class FibonacciRecursiveTest {
    private FibonacciRecursive fib;
    private final ByteArrayOutputStream outContent
    = new ByteArrayOutputStream();
    private final PrintStream originalOut = System.out;

    @BeforeEach
    public void setUp() {
        fib = null;
        System.setOut(new PrintStream(outContent));
    }

    @AfterEach
    public void tearDown() {
        System.setOut(originalOut);
    }

    @Test
    void main_EmptyString() {
        String[] args = new String[]{};
        FibonacciRecursive.main(args);
        assertEquals("Enter the arguments\n",
        outContent.toString());
    }

    @Test
    void main_NonInteger() {
        String[] args = new String[]{"Kashaf"};
        FibonacciRecursive.main(args);
        assertEquals("Invalid Input.
        Please enter a positive integer\n", outContent.toString());
    }

    @Test
    void main_Valid() {
```

```
        String[] args = new String[]{"23"};
        FibonacciRecursive.main(args);
        assertEquals("F(23) = 28657\n",
        outContent.toString());
    }

    @Test
    void main_NegativeInteger() {
        String[] args = new String[]{"-1"};
        FibonacciRecursive.main(args);
        assertEquals("Invalid Input.
        Please enter a positive integer\n", outContent.toString());
    }

    @Test
    void main_LargeInteger() {
        String[] args = new String[]{"47"};
        FibonacciRecursive.main(args);
        assertEquals("Fibonacci number is too
        large for this input. Please enter a number less than 47!\n", outCont
    }

    @Test
    void FibonacciRec_Negative() {
        assertThrows(InvalidInputException.class,
                () -> fib = new FibonacciRecursive(-100));
    }

    @Test
    void FibonacciRec_NegativeBoundary() {
        assertThrows(InvalidInputException.class,
                () -> fib = new FibonacciRecursive(-1));
    }

    @Test
    void FibonacciRec_LowerBoundary()
    throws InputMismatchException, InvalidInputException {
        fib = new FibonacciRecursive(0);
        assertEquals("F(0) = 0\n",
        outContent.toString());
    }

    @Test
    void FibonacciRec_MidRange()
    throws InputMismatchException, InvalidInputException {
```

```
        fib = new FibonacciRecursive(23);
        assertEquals("F(23) = 28657\n",
        outContent.toString());
    }

    @Test
    void FibonacciRec_UpperBoundary()
    throws InputMismatchException, InvalidInputException {
        fib = new FibonacciRecursive(46);
        assertEquals("F(46) = 1836311903\n",
        outContent.toString());
    }

    @Test
    void FibonacciRec_ExceedLimitBoundary() {
        assertThrows(InputMismatchException.class,
                () -> fib = new FibonacciRecursive(47));
    }

    @Test
    void FibonacciRec_ExceedLimit() {
        assertThrows(InputMismatchException.class,
                () -> fib = new FibonacciRecursive(100));
    }
}
```

3.3 Invalid Input Exception Class

```
public class InvalidInputException extends Exception{
    public InvalidInputException(){
    }

    public InvalidInputException(String str){
        super(str);
    }
}
```

3.3.1 Test Log

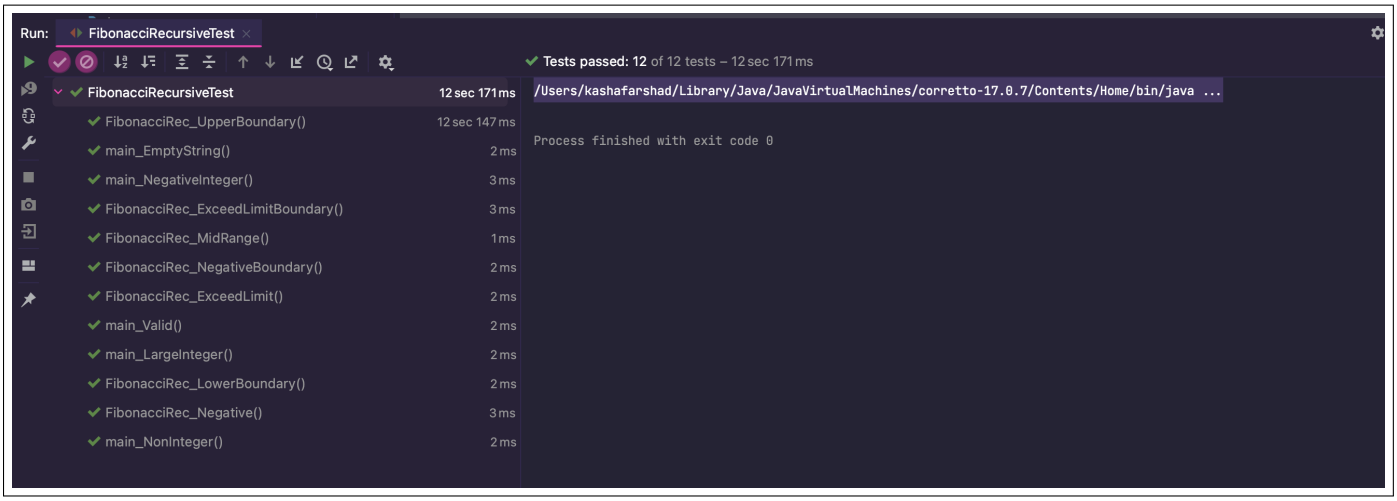

Figure 1: Test Log of Plan A - Positive Fibonacci