Software Testing, Lab 2, March 9. 2023.

Tasks:

1. Write a java program for the given problem:

Given a positive integer N, split it into the sum of at least two positive integers, and maximize the product of these integers. Returns the largest product value M.

Input:

N a positive integer

Output:

M the largest product value

Sample Input:

10

Sample Output

36 (10 = 3 + 4 + 4， 3 *3*4 = 36)

2. Test the program with **Advanced Junit Usage**.

a) Add `@Before`, `@BeforeClass`, `@After`, `@AfterClass` to different functions, and write logs in each function, observe the execution order.

b) Given the input list `testinput.txt`, each line with a test input and expected output, separated by comma, eg. `10,36` means `10 = 3 + 3 + 4` and the result should be `3 * 3 * 4 = 36`. Each line only contains one test case. Use `@RunWith(Parameterized.class)` to load the test file, and test all test cases.

3. Use Cobertura to produce coverage.

Requirements for the experiment:

1. Finish the tasks above individually.
2. Please send your experiment report to 智慧树, the following information should be included in your report:

a) Your java code and Junit test program.

b) The test result and coverage of your tests on the problem.

Submission deadline:

23:59 March 15, 2023.

Task1: java program for the problem.

使用动态规划的方法来解决这个问题，通过建立一个数组来存储每个子问题的最大乘积值。首先设置基本情况，即N=1时最大乘积值为1。然后通过自下而上的方式解决子问题，对于每个i从2到N，枚举每个可能的拆分方案，计算每个拆分方案的乘积值，然后选择乘积值最大的拆分方案作为dp[i]的值。最后，程序返回dp[N]的值作为最大乘积值。

```
public class Solution {
    public static int splitInteger(int n) {
        // Create an array to store the maximum product values for each subproblem
        int[] dp = new int[n + 1];

        // Base case
        dp[1] = 1;

        // Solve subproblems in bottom-up manner
        for (int i = 2; i <= n; i++) {
            for (int j = 1; j <= i / 2; j++) {
                dp[i] = Math.max(dp[i], Math.max(j, dp[j]) * Math.max(i - j, dp[i -
j]));
            }
        }

        // Return the maximum product value
        return dp[n];
    }

}
```

Task2:Test the program with Advanced Junit Usage.

```
import org.junit.*;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;

import java.io.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Date;
import java.util.logging.FileHandler;
import java.util.logging.LogRecord;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;

@RunWith(Parameterized.class)
public class TestFile {
    private final Integer inputNumber;
    private final Integer expectedResult;
```

```
    private static Logger logger = Logger.getLogger(TestFile.class.getName());


    private Solution maxSplit;


    @BeforeClass
    public static void testBeforeClass() {
        FileHandler fileHandler = null;
        try {
            fileHandler = new
FileHandler("src\\test\\java\\com\\ljz\\lab2\\test.log", false);
            fileHandler.setFormatter(new SimpleFormatter() {
                @Override
                public String format(LogRecord record) {
                    return record.getLevel() + ":" + new Date(record.getMillis()) +
" " + record.getMessage() + "\n";
                }
            });
            logger.addHandler(fileHandler);
        } catch (IOException e) {
            e.printStackTrace();
        }
        logger.info("testBeforeClass ok");
    }

    @Before
    public void testBefore() {
        maxSplit = new Solution();
        logger.info("testBefore ok");
    }

    public TestFile(Integer inputNumber, Integer expectedResult) {
        this.inputNumber = inputNumber;
        this.expectedResult = expectedResult;
    }

    //读取文件
    public static Object[][] toArrayByInputStreamReader2(String name) {
        ArrayList<String> arrayList = new ArrayList<>();
        try {
            File file = new File(name);
            InputStreamReader input = new InputStreamReader(new
                    FileInputStream(file));
            BufferedReader bf = new BufferedReader(input);
            String str;
            while ((str = bf.readLine()) != null) {
                arrayList.add(str);
            }
            bf.close();
            input.close();
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }

        int length = arrayList.size();
        int width = arrayList.get(0).split(",").length;
        Object[][] array = new Object[length][width];
        for (int i = 0; i < length; i++) {
            for (int j = 0; j < width; j++) {
                String s = arrayList.get(i).split(",")[j];
                array[i][j] = Integer.parseInt(s);
            }
        }

        return array;
    }

    @Parameterized.Parameters
    public static Collection Numbers() {
        Object[][] array;
        String path = "src\\test\\java\\com\\ljz\\lab2\\testinput.txt";
        array = TestFile.toArrayByInputStreamReader2(path);
        return Arrays.asList(array);
    }

    @Test
    public void testNumber() {
        logger.info("Testing--" + "Input Number is : " + inputNumber + " Expected
Result is : " + expectedResult);
        Assert.assertEquals(expectedResult, (Integer)
                maxSplit.splitInteger(inputNumber));
        logger.info("Test passed");
    }

    @After
    public void testAfter() {
        logger.info("testAfter ok");
    }

    @AfterClass
    public static void testAfterClass() {
        logger.info("testAfterClass ok");
    }
}
```

上述代码中使用了`@RunWith(Parameterized.class)`注释来表示使用参数化测试，即输入输出数据是从外部文件中读取的，而不是写死在代码中。

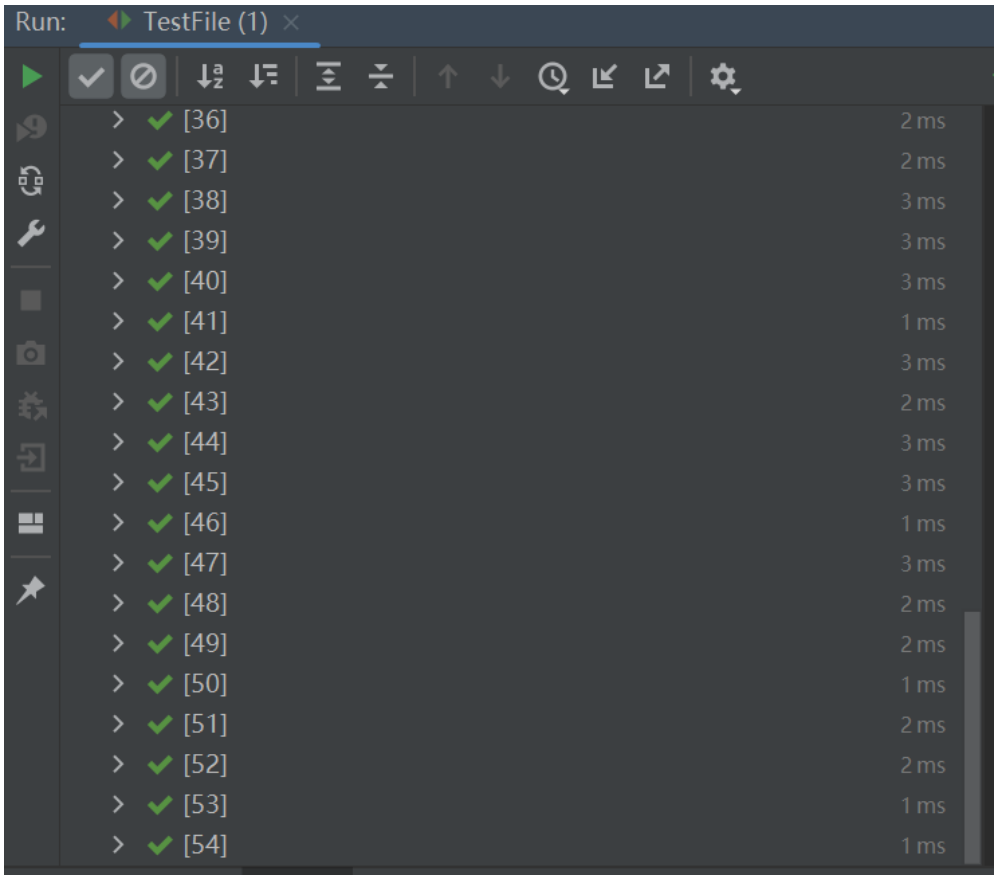
该测试类使用了`@Parameterized.Parameters`注释和`toArrayByInputStreamReader2`方法来读取外部文件中的输入输出数据，然后将其传递给构造函数。对于每个测试用例，JUnit将使用构造函数中的输入数据来初始化测试类的成员变量，然后运行`testNumber`方法来测试给定输入时输出是否与预期输出匹配。

此外，测试类中还包含了一些JUnit提供的方法，例如`@BeforeClass`、`@Before`、`@After`和`@AfterClass`。`@BeforeClass`和`@AfterClass`分别在测试类中的所有测试方法执行前和执行后运行一次，`@Before`和`@After`分别在测试类中的每个测试方法执行前和执行后运行一次。在这里，这些方法被用于日志记录，以便跟踪测试过程中的执行情况。

以单个测试结果为例，其日志结果输出如下。不难发现一个JUnit4单元测试用例执行顺序为：@BeforeClass -> @Before -> @Test -> @After -> @AfterClass;

```
INFO:Thu Mar 09 18:29:54 CST 2023 testBeforeClass ok
INFO:Thu Mar 09 18:29:54 CST 2023 testBefore ok
INFO:Thu Mar 09 18:29:54 CST 2023 Testing--Input Number is : 1 Expected Result is :
1
INFO:Thu Mar 09 18:29:54 CST 2023 Test passed
INFO:Thu Mar 09 18:29:54 CST 2023 testAfter ok
INFO:Thu Mar 09 18:29:54 CST 2023 testAfterClass ok
```

测试结果如下，55个测试全部通过。



Task3:Use Cobertura to produce coverage

Coverage Report - com.ljz.lab2

Package	# Classes	Line Coverage	Branch Coverage	Complexity
com.ljz.lab2	1	100% 7/7	100% 4/4	3

Classes in this Package	Line Coverage	Branch Coverage	Complexity
Solution	100% 7/7	100% 4/4	3

Report generated by Cobertura 2.1.1 on 23-3-9 下午6:48.