软件测试上机报告

第一次上机作业 - Lab 2 Junit and Jacoco

学　　院　智能与计算学部

专　　业　软件工程

姓　　名　郎文翀

学　　号　**3019244247**

年　　级　2019 级

班　　级　软件工程 5 班

1. Experimental requirements

1. Install Junit(4.12), Hamcrest(1.3), Eclemma or Jacoco(newest) in Eclipse (done by last lab) or IDEA (follow the new instruction).
2. Write a java program for the given problem and 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 Eclemma or Jacoco to produce coverage.
    a) Description of the 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)

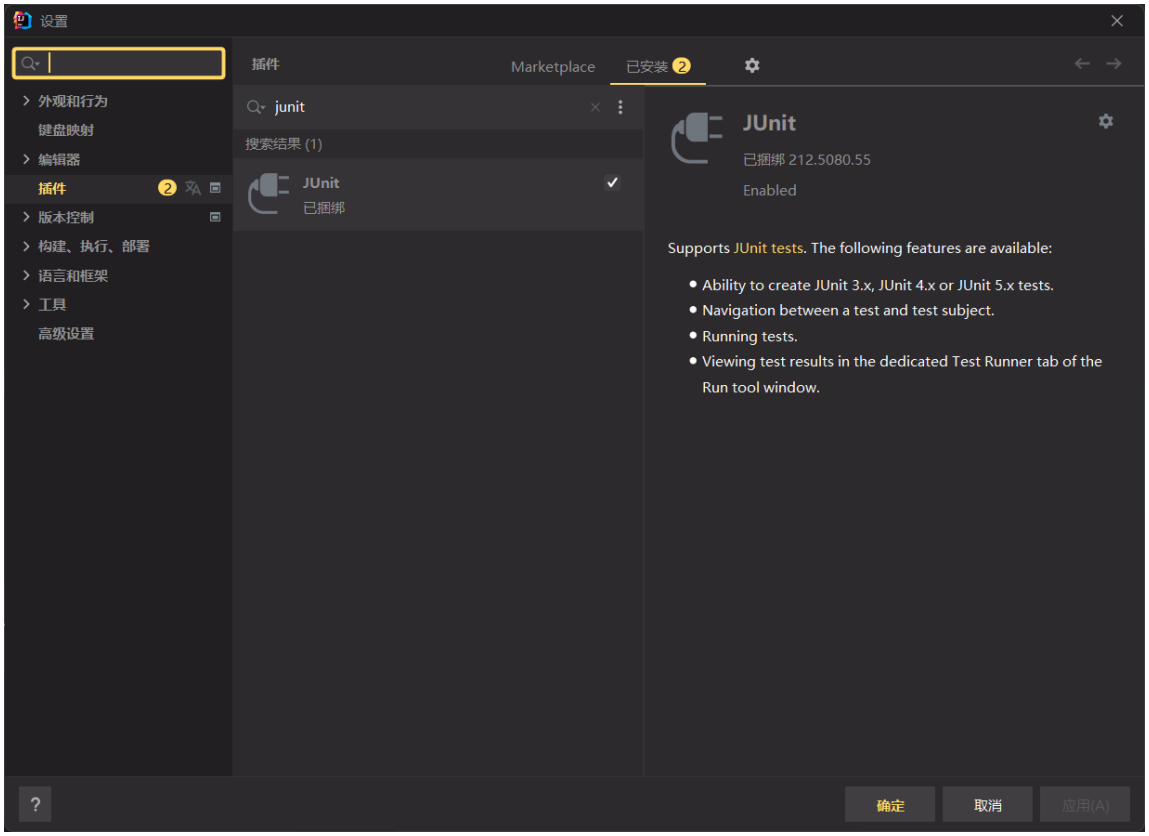
Requirements for the experiment:

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 brief description that you install junit and Eclemma or Jacoco.
   c) The test result and coverage of your tests on the problem.

2. Configuration

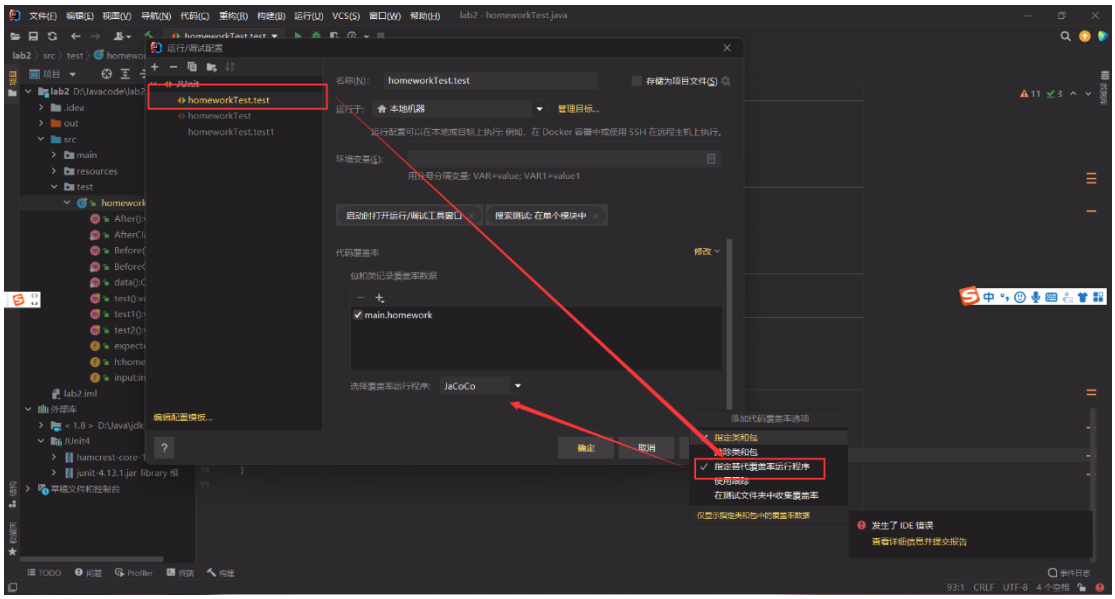
2.1 junit 安装

我使用的是 idea 进行实验，junit 安装较为简单，我们只需要打开 idea 中的设置->插件然后进入插件商店搜索 junit 即可，发现 idea 已经默认提供安装了 junit 如下图所示：



2.2 jacoco 安装

由于我的 idea 版本较新，使用提供的参考文档并未能够成功找到 jacoco 的配置按钮，经过资料查阅以及尝试，最终发现 jacoco 配置安装步骤如下：
我们并不需要使用 maven 进行安装，idea 已经默认安装了 jacoco 插件，我们点击左上方运行环境按钮，点击编辑配置打开如下菜单栏，然后选择我们需要修改的环境，然后重点是点击右下方的覆盖率设置中的修改按钮，将指定覆盖测试运行程序打上对勾，这样才能出现下方的覆盖运行程序选择的按钮，然后我们点击将 idea 切换为 jaococ，然后点击应用保存。



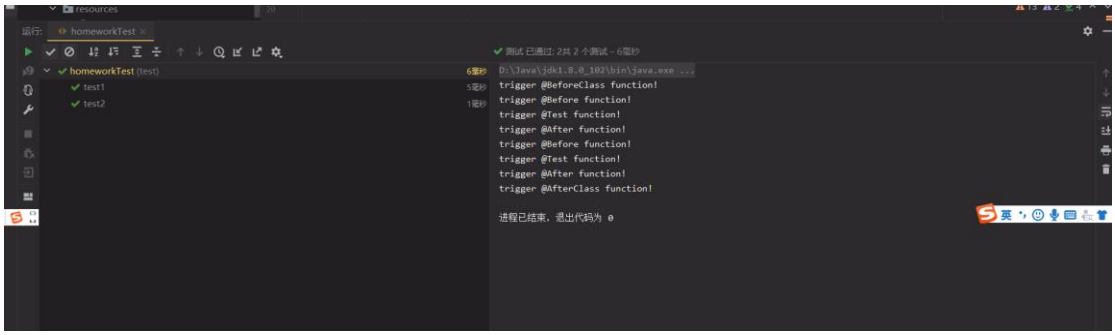
3. Result analysis

3.1 解决问题的代码编写

首先查看题干，大致题意是给出一个数 n，要能够拆分成至少两个数，同时拆分出来的 k 个数（k>=2）的乘积要保证是最大值。很明显是一道 dp 题，假设 dp[i]表示数值 i 经过拆分以后最大的乘积数，那么很明显他和他的拆解数 j 有如下关系
dp[i]=max(dp[i],max(dp[i-j]*j,(i-j)*j)，同时由于 j 是 i 的一个拆解数很明显 j 需要满足 1<j<i，因此我们就可以得到这问题的解决函数 solve()了，同样的我将这个函数写到了 src/main/homework.java 中

3.2 测试代码编写

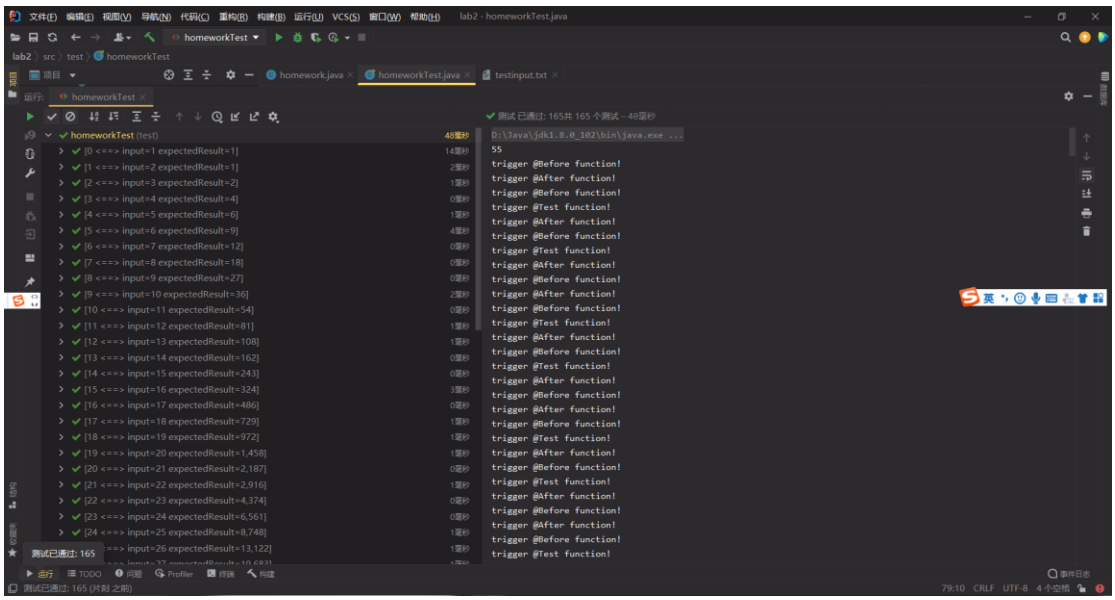
然后我们来编写测试代码，仍然是将测试代码放到 src/test/homeworkTest.java 中，首先我们需要验证一下 junit 中@BeforeClass、@Before、@Test、@After、@AfterClass 几个注解的作用以及时间上的区别。因此我们要个定义几个函数绑定这几个注解，要注意@BeforeClass 和@AfterClass 需要绑定的函数为 static 修饰的静态方法，然后我还是编写了两个 test 函数来进行测试。为了能够看出何时出发这些注解捆绑的函数，我们在每一个函数中都加入一行代码来打印信息：trigger @xxx function。自此我们就完成了实验要求的第一个任务，一下是实验截图：



我们可以看到@BeforeClass 和@AfterClass 都只触发了一次，就是在测试类刚刚创建完成和所有测试函数完成以后触发，因此他们在测试时只触发一次，是针对所有测试的。而@Before 和@After 却在每一个 test 测试前后都有触发，同时都在@Before 和@After 之间。因此我们可以总结出他们之间的顺序关系如下：
一个 JUnit4 的单元测试用例执行顺序为：
@BeforeClass -> @Before -> @Test -> @After -> @AfterClass;
每一个测试方法的调用顺序为：
@Before -> @Test -> @After;

3.4 使用@Runwith 自动导入测试文件进行测试

我们之前的测试都是需要建立一个@test 来进行一次 assertEquals 断言比较测试，但是当需要进行上百上千甚至上万次测试时，很明显如上的方法就很不现实，因此我们需要使用一个能够读取测试文件然后自动装载测试的容器进行测试，也就是使用@Runwith 来进行。
首先我们需要为这个测试类上方注解@RunWith(Parameterized.class)，然后在测试文件中有两个变量，分别是输入的值 n 以及预期结果值，因此我们定义两个变量 input 和 expectedResult 来存储每一次的测试样例。然后我们还需要编写一个读取文件并生成测试样例集的函数，我们需要在这个方法上绑定@parameters 注解声明，同时很明显这个方法是一个返还 list 的方法，每一个元素都是一个数组包含 input 和 expectedResult 整型变量。因此我们在这个函数内部首先需要读取这个文件，通过按行读取，将每一行的字符串读出以后通过 split()按照逗号进行切割存成一个数组，由于切割完成以后会是两个字符串，因此我们还需要将其转换成整型变量，最后再将这个数组作为一个元素存储到 list 中即可。在函数的最上方我们还需要通过(name = "{index} <==> input={0} expectedResult={1}")和对应的变量进行绑定，自此我们就完成了数据集的生成以及自动装载的部分。然后我们最后在新建一个@test 绑定的 test 方法，将 assertEquals()函数中要求的两个变量填写为 expectedResult 和 input 即可。然后我们再次运行即可查看到如下结果：

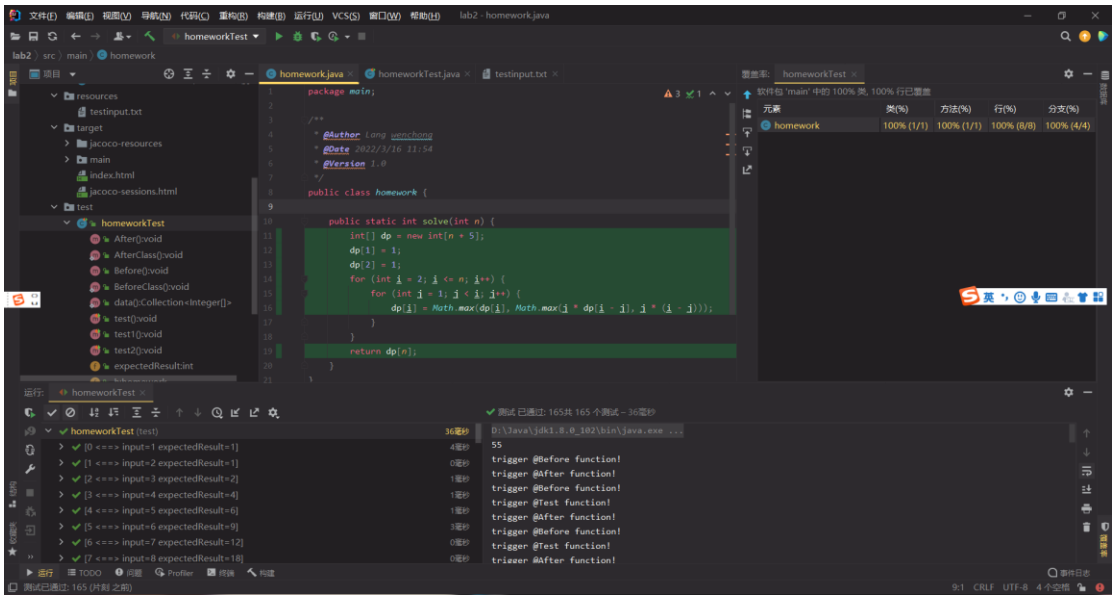


我们可以看到程序自动将测试文件中的所有样例点进行了装载测试，并且最后最终结果全部问成功，说明我们的代码并未发现问题。

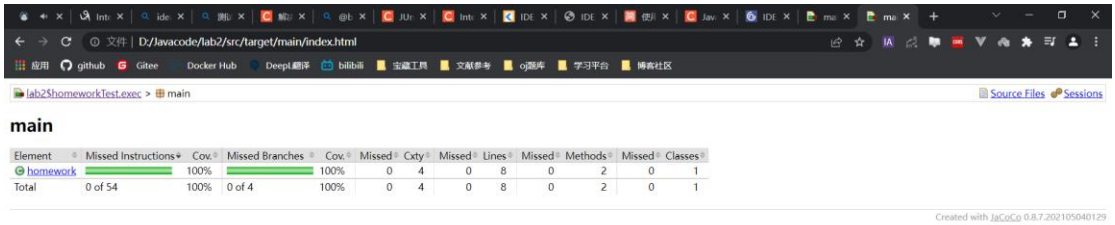
3.5 使用 jacoco 插件查看测试样例的覆盖率

我们知道在测试时并不是样例通过就能说明程序无错误的，样例点需要尽可能的覆盖程序的所有代码和方法，只有这样我们才能保证我们的测试样例考虑的足够全面，通过测试的程序足够健壮，因此接下来我们还要通过 jacoco 插件进行覆盖测试，查看我们的样例覆盖了 homework.java 中我们编写的代码中的那些方法和行。
按照 2 中 jacoco 的配置，我们在在绿色箭头处右键点击选择按照覆盖率运行，然后我们即

可得到分析结果如下图所示：



我们可以看到右侧状态栏啊中显示我们的覆盖率为 100%，同时代码中所有行都有被执行测试，说明我们的样例足够全面。然后我们还可以进一步导出我们的 jacoco 覆盖测试结果，只需要点击导入按钮选择导出路径 ，然后即可再文件夹中看到生成的 html 格式的分析报告，我们打开在浏览器中即可查看到结果：



自此我们就完成了本次实验的所欲任务，代码在如下 4 中给出

4. Source code

这里我再给出实验中使用到的代码，首先是被测试的代码 lab1/src/main/homework.java 然后是 lab2/src/test/homeworkTest.java

```
package main;
```

```
/**
 * @Author Lang wenchong
 * @Date 2022/3/16 11:54
 * @Version 1.0
 */
public class homework {

    public static int solve(int n) {
        int[] dp = new int[n + 5];
        dp[1] = 1;
        dp[2] = 1;
        for (int i = 2; i <= n; i++) {
            for (int j = 1; j < i; j++) {
                dp[i] = Math.max(dp[i], Math.max(j * dp[i - j], j * (i - j)));
            }
        }
        return dp[n];
    }
}
```

然后是 lab2/src/test/homeworkTest.java 中的代码：

```
    package test;

    import main.homework;
    import org.junit.*;
    import org.junit.runner.RunWith;
    import org.junit.runners.Parameterized;

    import java.io.*;
    import java.util.ArrayList;
    import java.util.Collection;
    import java.util.List;

    import static org.junit.Assert.assertEquals;

    /**
     * @Author Lang wenchong
     * @Date 2022/3/16 11:58
     * @Version 1.0
     */

    @RunWith(Parameterized.class)
    public class homeworkTest {

        @Parameterized.Parameter(0)
        public int input;

        @Parameterized.Parameter(1)
        public int expectedResult;
```

```
    @Parameterized.Parameters(name = "{index} <==> input={0} expectedResult={1}")
    public static Collection<Integer[]> data() throws IOException {
        InputStreamReader inputStreamReader = null;
        File f = new File("D:/Javacode/lab2/src/resources/testinput.txt");
        try {
            inputStreamReader = new InputStreamReader(new FileInputStream(f));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        List<Integer[]> list = new ArrayList<>();
        BufferedReader in = new BufferedReader(inputStreamReader);
        String tmp = "";
        tmp = in.readLine();
        while (tmp != null && !tmp.equals("")) {
            String[] arr = tmp.split(",");
            Integer[] item = {Integer.parseInt(arr[0]), Integer.parseInt(arr[1])};
            list.add(item);
            tmp = in.readLine();
        }
        System.out.println(list.size());
        return list;
    }

    public homework h;

    @BeforeClass
    public static void BeforeClass() {
        System.out.println("trigger @BeforeClass function!");
    }

    @Before
    public void Before() throws Exception {
        h = new homework();
        System.out.println("trigger @Before function!");
    }

    @Test
    public void test1() {
        System.out.println("trigger @Test function!");
        assertEquals(12, h.solve(7));
    }

    @Test
    public void test2() {
        System.out.println("trigger @Test function!");
```

```
            assertEquals(108, h.solve(13));
        }

        @Test
        public void test() {
            assertEquals(expectedResult, h.solve(input));
        }

        @After
        public void After() {
            System.out.println("trigger @After function!");
        }

        @AfterClass
        public static void AfterClass() {
            System.out.println("trigger @AfterClass function!");
        }

    }
```

5. 实验心得总结

经过本次实验，我们进一步了解了 junit 中注解的不同的作用，同时使用@runwith 高级语法进行了自动化样例装载测试，大大提高了程序测试的效率。同时我们还尝试引入了 jaococ 插件进行了覆盖率测试分析，可以清晰的看出我们的测试样例是否覆盖了程序的大部分内容从而帮助我们进一步完善我们的测试样例。