实验2使用Junit进行单元测试

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 组 别 |  | 实验人员 | KAFLE SAMRAT | | | | |
| 实验项目名称 | 使用Junit进行单元测试 | | | | | | |
| 实 验 类 别 | 验证型□ 设计型□ 综合型□ | | | | | | |
| 实 验 工 具 | Eclipse | | | 实验地点 | J13-132 | 实验日期 | 2020.12.7 |
| 学生实验小结 | 编写被测程序有些困难，好在坚持下来把源程序写完了，通过Junit对编写的源程序进行测试，开始时发现了错误，实验要求的程序功能是命令行检测出语法错误后就停止运行，但是自己的程序并没有这个判断功能，后来修改代码后测试成功。 | | | | | | |
| 教师评语 | 指导教师签名：  年 月 日 | | | | | | |
| 实验成绩 |  | | | | | | |

1实验内容和目标

1.1实验内容

单元测试是最小粒度的测试，以测试某个功能或代码块。单元测试多采用白盒测试技术，本实验主要学习通过白盒测试方法设计测试用例，并使用Junit对编写的程序进行单元测试

1.2实验目标

(1)掌握使用白盒测试方法设计测试用例

(2)掌握Junit进行参数化单元测试的方法

2实验方案设计

2.1实验环境

Eclipse、junit

2.2 实验原始数据

The core code of the Command class

for (int i = 1; i < s.length - 1; i++) {

if (s[i].charAt(0) == '-' && s[i + 1].charAt(0) == '-' && s[i].length() == 2)// 不带参数的命令

{

// Gets the letter of the command

char a = s[i].charAt(1);

// Matches whether the command exists without arguments

for (j = 0; j < format.length(); j++) {

// If the command is matched, it is not known if the parameters are required

if (format.charAt(j) == a) {

String s1;

// Determine whether there are parameters

if (j != format.length() - 1 || format.charAt(j) + 1 != ':')// No parameters are required

s1 = "-" + a;// Make sure it is the correct command

else

break;// Arguments are required, but not available

if (!ts.contains(s1))// If the command is not in the list

ts.add(s1);

}

}

if (j == format.length() - 1)// The command does not match, and pops out directly

{

isTrue = false;

break;

}

}

// Command with arguments

else if (s[i].charAt(0) == '-' && s[i + 1].charAt(0) != '-' && s[i].length() == 2) {

char a = s[i].charAt(1);// Determine the check command letters

for (j = 0; j < format.length() - 1; j++) {

// Format has a belt: match

if (format.charAt(j) == a && format.charAt(j + 1) == ':') {

ff = 1;

String s1 = "-" + a;

String s2 = s[i + 1];// Parameters of the S1 instruction

StringBuffer sbb = new StringBuffer();

sbb.append(s1 + " " + s2);// Complete instruction with arguments

String s3 = new String(sbb);// Convert to String type

int flag = 0;

// Update the command with arguments

if (!ts.isEmpty()) {

for (int k = 0; k < ts.size() - 1; k++) {

// If the command in the linked list library has the command with arguments

if (ts.get(k).charAt(1) == a && ts.get(k + 1).charAt(0) != '-') {

flag = 1;

ts.remove(k);

ts.add(s3);// The update operation

}

}

}

// If the list is empty

else {

ts.add(s3);

}

// This directive does not exist in the linked list, add it directly

if (flag == 0) {

StringBuffer sbb1 = new StringBuffer();

sbb.append(s1 + " " + s2);

String s33 = new String(sbb1);

ts.add(s33);

}

} else

continue;

}

// Format does not have this parameter String number

if (ff != 1) {

isTrue = false;

break;

}

}

}

2.2.2 White Box Test Test Case Design for CommandOut Method in Command Class:

Judge1：

if (s[i].charAt(0) == '-' && s[i + 1].charAt(0) == '-' && s[i].length() == 2)

judge2：

if (j != format.length() - 1 || format.charAt(j) + 1 != ':')

judge3：

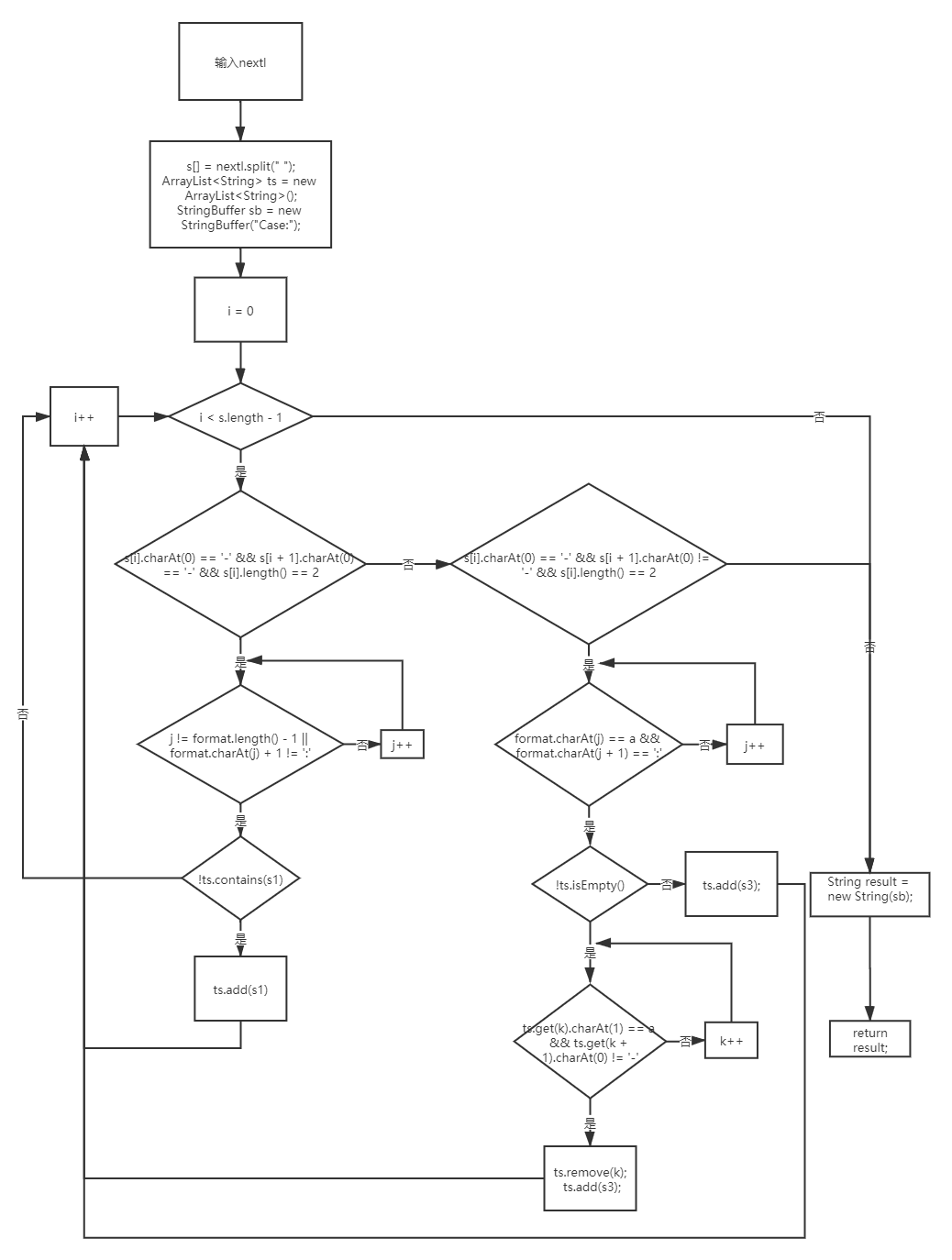
if (format.charAt(j) == a && format.charAt(j + 1) == ':')

judge4：

if (!ts.isEmpty())

|  |  |  |
| --- | --- | --- |
| The test case | Cover conditions | Covering the judgment |
| ls -a -l -b | Can be inserted without arguments | Judge the Y branch of 1;  Judge the Y branch of 2;  Judge the N branch of 3 |
| ls -w document -x | Insert the first time with a parameter | Judge the N branch of 1;  Judge the Y branch of 2;  Judge the Y branch of 3;  judge the N branch of 4 |
| ls -w write1 -a -w write2 | Take a second parameter update | Judge the N branch of 1  Judge the Y branch of 2;  Judge the Y branch of 3;  Judge the Y branch of 4 |
| ls -a -n | Terminate if no parameter condition is not met | Judge the Y branch of 1  Judge the N branch of 2  Judge the N branch of 3 |
| Ls -a -b doucument | Terminate if the parameter condition is not met | Judge the Y branch of 1  Judge the N branch of 1  Judge the Y branch of 2  Judge the N branch of 3 |
|  |  |  |

2.2.3 Command class flow chart:



2.2.4 Path test:

|  |
| --- |
| ls -a -l -b |
| ls -w document -x |
| ls -w write1 -a -w write2 |
| ls -a -n |
| Ls -a -b doucument |

2.2.5 Core code of CommandOut class

@Parameters

public static Collection<Object[]> data(){

Object[][] objects= {

{"ls -a -l -a documents -b","Case:-a -l"},

{"ls","Case:"}

};

return Arrays.asList(objects);

}

@Test

public void testCommandOut() {

// Test the first case

assertThat(c.commandOut(this.input),is(this.expected));

}

3实验步骤和结果

3.1 Steps of compiling the program under test

3.1.1 start eclispe

3.1.2

Write a command line analyzer that analyzes which options are included in a given command line. The program is described as follows: The command line consists of a number of strings separated by exactly one space. The first of these strings is the name of the command-line tool, which is made up of lowercase letters and is not processed by your program. It may include several options after the tool name, and then it may include some parameters that are not options.

There are two types of options: options with arguments and options without arguments. A valid form of a no-argument option is a minus sign followed by a single lowercase letter, such as "-a" or "-b". The argument option consists of two strings separated by Spaces, the former formatted in the same way as the no-argument option, and the latter, an argument to the option, which is a non-empty string of lowercase letters, numbers, and a minus sign.

The author of the command line tool provides you with a format string that specifies which options his command line tool should accept. The string consists of a number of lowercase letters and a colon, where each lowercase letter represents an option that the program accepts. If the lowercase letter is followed by a colon, it represents an option with arguments; otherwise, it is a no-argument option. "Ab: m:" for example, says the program accepts three options, namely "-a" (with no parameters), "-" b (parameters), and "-m" (parameters).

The author of the command line tool has prepared a command line to test your program. For the command line, your tool should always analyze backwards. When your tool encounters a string that is neither a valid option nor an argument to a valid option, the analysis stops. The remaining unparsed portions of the command line do not constitute options for the command, so your program should ignore them. The Command program is written as shown in 2.2.1.

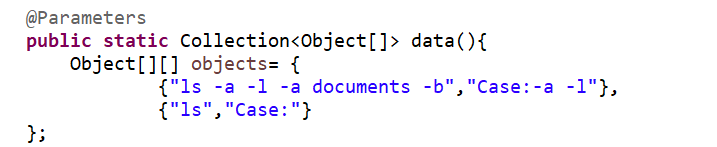
3.2 Steps to design test cases using logical coverage

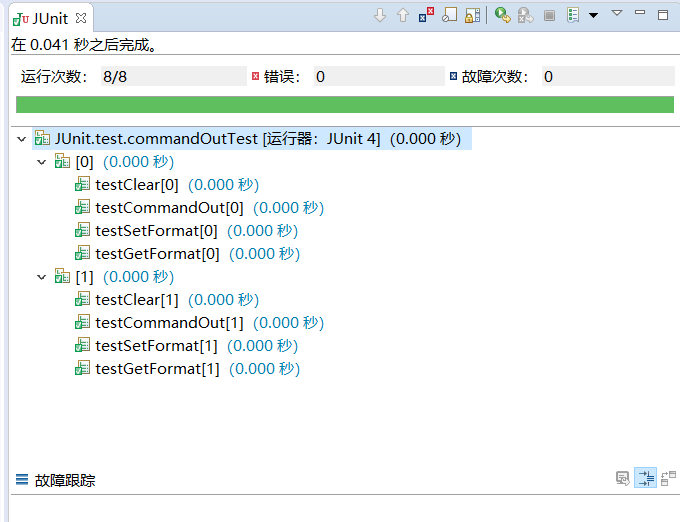
According to the logic covered in white box testing method designed for task one Command commandOut method in the class of test case design, requirements to determine - conditions covering (/ conditions covering the basic idea is: design enough test cases, make run these test cases, the application under test all of the possible results are carried out at least once, and each judge each of the conditions of all possible outcomes appear at least once.)

3.3 Steps to build a testing framework using JUnit

Write the test class CommandOutTest

3.4 Test the program written and analyze the test results





4实验结果分析

结果正确。