实验报告三

|  |  |
| --- | --- |
| 姓名 | Fyl |
| 学号 | SA18225079 |
| 工具 | Win10;Eclipse;Junit; |

目录

[1. 引言](#_Toc23396_WPSOffice_Level1) [5](#_Toc23396_WPSOffice_Level1)

[1.1. 测试标识](#_Toc28302_WPSOffice_Level2) [5](#_Toc28302_WPSOffice_Level2)

[1.2. 系统概述](#_Toc24667_WPSOffice_Level2) [5](#_Toc24667_WPSOffice_Level2)

[1.3. 文档概述](#_Toc12486_WPSOffice_Level2) [5](#_Toc12486_WPSOffice_Level2)

[2. 源程序需求描述](#_Toc28302_WPSOffice_Level1) [5](#_Toc28302_WPSOffice_Level1)

[2.1. 需求概述](#_Toc23738_WPSOffice_Level2) [5](#_Toc23738_WPSOffice_Level2)

[2.2. 需求样例](#_Toc5427_WPSOffice_Level2) [6](#_Toc5427_WPSOffice_Level2)

[3. 详细的测试结果](#_Toc24667_WPSOffice_Level1) [7](#_Toc24667_WPSOffice_Level1)

[3.1. 编写Junit单元测试代码](#_Toc13170_WPSOffice_Level2) [8](#_Toc13170_WPSOffice_Level2)

[3.1.1. 基本原理](#_Toc28302_WPSOffice_Level3) [8](#_Toc28302_WPSOffice_Level3)

[3.1.2. 测试分析（附流程图、控制流图）](#_Toc13170_WPSOffice_Level3) [9](#_Toc13170_WPSOffice_Level3)

[3.1.3. 编写Junit测试代码](#_Toc32702_WPSOffice_Level3) [14](#_Toc32702_WPSOffice_Level3)

1. 测试结果概述 15

[4.1. 测试总体评估 1](#_Toc10575_WPSOffice_Level2)5

[4.2. 改进建议](#_Toc30603_WPSOffice_Level2) [15](#_Toc30603_WPSOffice_Level2)

1. 评估 15

[5.1. 软件能力](#_Toc27149_WPSOffice_Level2) [16](#_Toc27149_WPSOffice_Level2)

[5.2. 缺陷和限制](#_Toc29910_WPSOffice_Level2) [17](#_Toc29910_WPSOffice_Level2)

[5.3. 建议](#_Toc32702_WPSOffice_Level2) [17](#_Toc32702_WPSOffice_Level2)

[5.4. 结论](#_Toc9901_WPSOffice_Level2) [17](#_Toc9901_WPSOffice_Level2)

1. **引言**

## 测试标识

本文档适用系统：Windows 10；

本文档使用软件：NumericConvertToRMB.exe;Eclipse;Junit;

## 系统概述

本系统是人民币数字大写转换系统。

输入：输入数字保留小数点后两位，整数部分位数不大于13位，且不能为负数（人民币没有负数），也不能包含非数字字符。

输出：如3003，应输出人民币叁仟零叁元整。

## 文档概述

本测试文档为人民币数字大写转换的单元测试报告，文档内容包括单元测试方法展开的测试设计、测试用例和测试结果分析。

1. **源程序需求描述**

## 需求概述

系统的具体需求如下：

1）中文大写金额数字应用壹、贰、叁、肆、伍、陆、柒、捌、玖、拾、佰、仟、万、亿、元、角、分、零、整(正)等字样。

2）中文大写金额数字到"元"为止的，在"元"之后，应写"整"(或"正")字，在"角"之后，可以不写"整"(或"正")字。

3）中文大写金额数字前应标明"人民币"字样，大写金额数字有"分"的，"分"后面不写"整"(或"正")字。

4）大写金额数字应紧接"人民币"字样填写，不得留有空白。

5）阿拉伯数字小写金额数字中有"0"时，中文大写应按照汉语语言规律、金额数字构成和防止涂改的要求进行书写。

## 需求样例

为更好地理解系统需求，举例如下：

1、阿拉伯数字中间有"0"时，中文大写要写"零"字，如￥1409.50，应写成：人民币壹仟肆佰零玖元伍角。

2、阿拉伯数字中间连续有几个"0"时，中文大写金额中间只写一个"零"字，如￥6007.14，应写成：人民币陆仟零柒元壹角肆分。

3、阿拉伯金额数字万位和元位是"0"，或者数字中间连续有几个"0"，万位、元位也是"0"，但千位、角位不是"0"时，中文大写金额中只写一个零字，也可以不写"零"字。如￥1680.32，应写成：人民币壹仟陆佰捌拾元叁角贰分，又如￥107000.53，应写成：人民币壹拾万零柒仟元伍角叁分。

4、阿拉伯金额数字角位是"0"，而分位不是"0"时，中文大写金额"元"后面应写"零"字。如￥16409.02，应写成人民币：壹万陆仟肆佰零玖元零贰分；又如￥325.04，应写成人民币叁佰贰拾伍元零肆分。

1. **详细的测试结果**

单元测试是编写测试代码，用来检测特定的、明确的、细颗粒的功能。单元测试并不一定保证程序功能是正确的，更不保证整体业务是准备的。

单元测试不仅仅用来保证当前代码的正确性，更重要的是用来保证代码修复、改进或重构之后的正确性。

一般来说，单元测试任务包括

1．接口功能测试：用来保证接口功能的正确性。

2．局部数据结构测试（不常用）：用来保证接口中的数据结构是正确的

1．比如变量有无初始值

2．变量是否溢出

3．边界条件测试

1．变量没有赋值（即为NULL）

**2．变量是数值（或字符)**

**1．主要边界：最小值，最大值，无穷大（对于DOUBLE等）**

**2．溢出边界（期望异常或拒绝服务）：最小值-1，最大值+1**

**3．临近边界：最小值+1，最大值-1**

3．变量是字符串

1．引用“字符变量”的边界

2．空字符串

3．对字符串长度应用“数值变量”的边界

4．变量是集合

1．空集合

2．对集合的大小应用“数值变量”的边界

3．调整次序：升序、降序

5．变量有规律

1．比如对于Math.sqrt，给出n^2-1，和n^2+1的边界

4．所有独立执行通路测试：保证每一条代码，每个分支都经过测试

1．代码覆盖率

1．语句覆盖：保证每一个语句都执行到了

2．判定覆盖（分支覆盖）：保证每一个分支都执行到

3．条件覆盖：保证每一个条件都覆盖到true和false（即if、while中的条件语句）

4．路径覆盖：保证每一个路径都覆盖到

5．各条错误处理通路测试：保证每一个异常都经过测试

## 编写Junit单元测试代码

### 基本原理

1．使用黑盒测试中的弱健壮等价类分析方法以及边界值分析方法设计测试用例，根据测试用例，编写Junit测试代码。

2．使用白盒测试中的分支/条件分析方法以及基本路径分析方法设计测试用例，根据测试用例，编写Junit测试代码。

3．在上面的基础上完成以下测试要求

1．普通合法数据是否正确处理（等价类）

2．普通非法数据是否正确处理（等价类）

3．边界内最接近边界的合法数据是否正确处理（边界值）

4．边界外最接近边界的非法数据是否正确处理（边界值）

5．N次循环的第0、1、n次是否有错（基本路径测试or分支/条件）

6．运算或判断中取最大最小值时是否有错（基本路径测试or分支/条件）

7．数据流、控制流中刚好等于、大于或小于确定的比较值时是否出错（基本路径测试）

### 测试分析（附流程图、控制流图）

1．弱健壮等价类测试用例如下表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例 | 操作 | 数据 | 期望结果 | 实际结果 | 测试状态 |
| 1-1 | 不输入数据，并进行转换 | - | 提示不执行 | 程序进入死循环 | Fail |
| 1-2 | 输入非法数据，并执行转换 | xxx | 提示不执行 | 程序无法执行，直接退出 | Fail |
| 1-3 | 输入多个小数点，并执行转换 | 1...1 | 提示不执行 | 人民币壹元 | Fail |
| 1-4 | 输入超大数据 | 100000000000000000000000000 | 提示不执行 | 程序无法执行，直接退出 | Fail |
| 1-5 | 输入负值 | -1 | 提示不执行 | 程序无法执行，直接退出 | Fail |
| 1-6 | 输入高精度数据 | 1004.135 | 人民币壹仟零肆元壹角四分 | 人民币壹仟零肆元壹角叁分 | Pass |
| 1-7 | 输入包含连续零的数据 |
| 1-8 | 输入不含零的数据 | 1234.56 | 人民币壹仟贰佰叁拾肆元伍角陆分 | 人民币壹仟贰佰叁拾肆元伍角陆分 | Pass |
| 1-9 | 输入角位是“0”，分位不是“0”的数据 | 1.01 | 人民币壹元零壹分 | 人民币壹元零壹分 | Pass |
| 1-10 | 输入前几位是“0”的数据 | 001223 | 人民币壹仟贰佰贰拾叁元整 | 人民币万零壹仟贰佰贰拾叁元整，程序继续运行 | Fail |
| 1-11 | 输入到元为止的数据 |
| 1-12 | 输入角位、分位都为“0”的数据 | 1.00 | 人民币壹元整 | 人民币壹元整 | Pass |

2．单缺陷边界值健壮性测试用例如下表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例 | 操作 | 数据 | 期望结果 | 实际结果 | 测试状态 |
| 2-1 | 输入Max value+1，并进行转换 | 10000000000000.00 | 提示不执行 | 人民币壹拾元整,程序继续执行 | Fail |
| 2-2 | 输入Max value，并执行转换 | 999999999 999.99 | 人民币玖仟玖佰玖拾玖亿玖仟玖佰玖拾玖万玖仟玖佰玖拾玖元玖角玖分 | 人民币玖仟玖佰玖拾玖亿玖仟玖佰玖拾玖万玖仟玖佰玖拾玖元玖角玖分 | Pass |
| 2-3 | 输入Min value，并执行转换 | 0.00 | 人民币零元整 | 人民币零元整 | Pass |
| 2-4 | 输入Min value-1，并进行转换 | -1 | 提示不执行 | 程序无法执行，直接退出 | Fail |

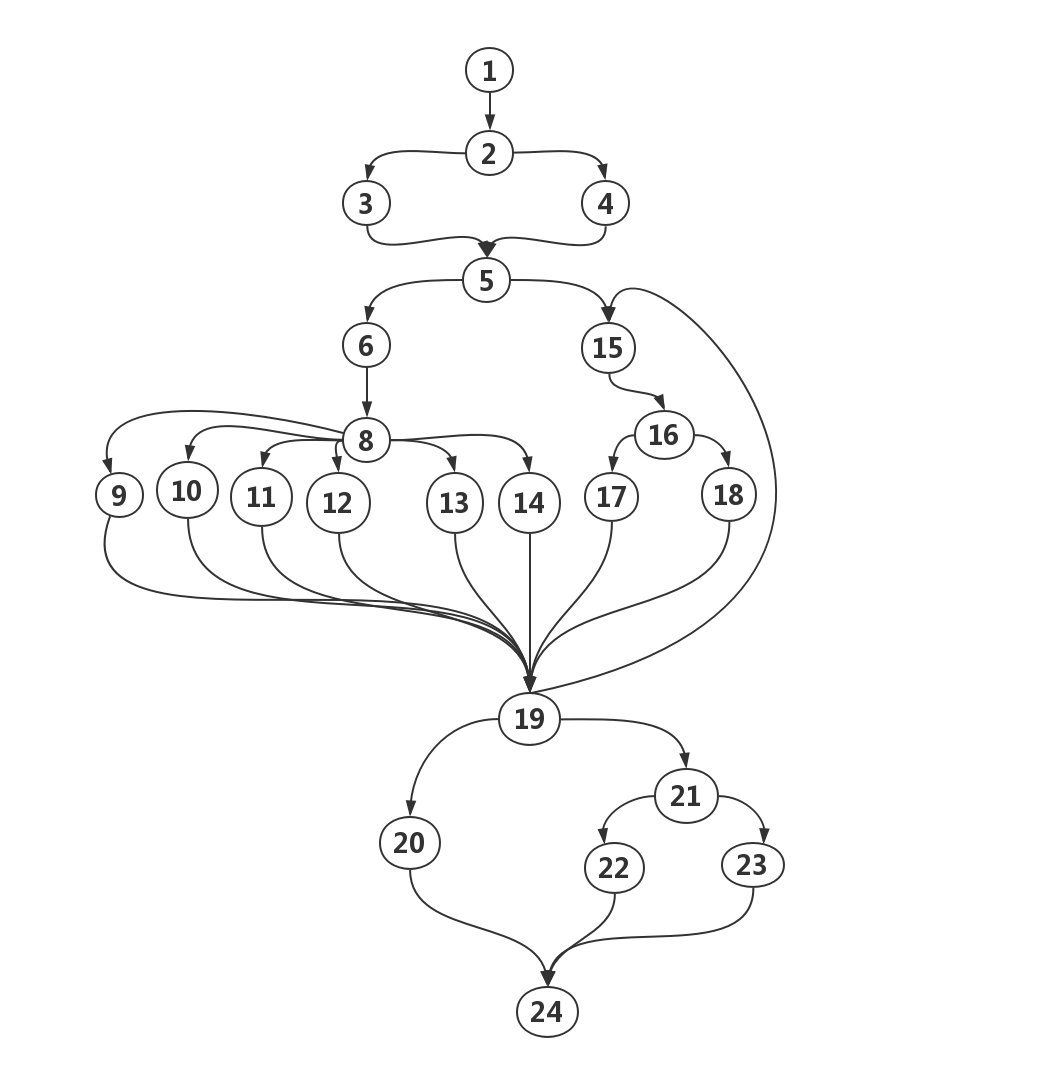
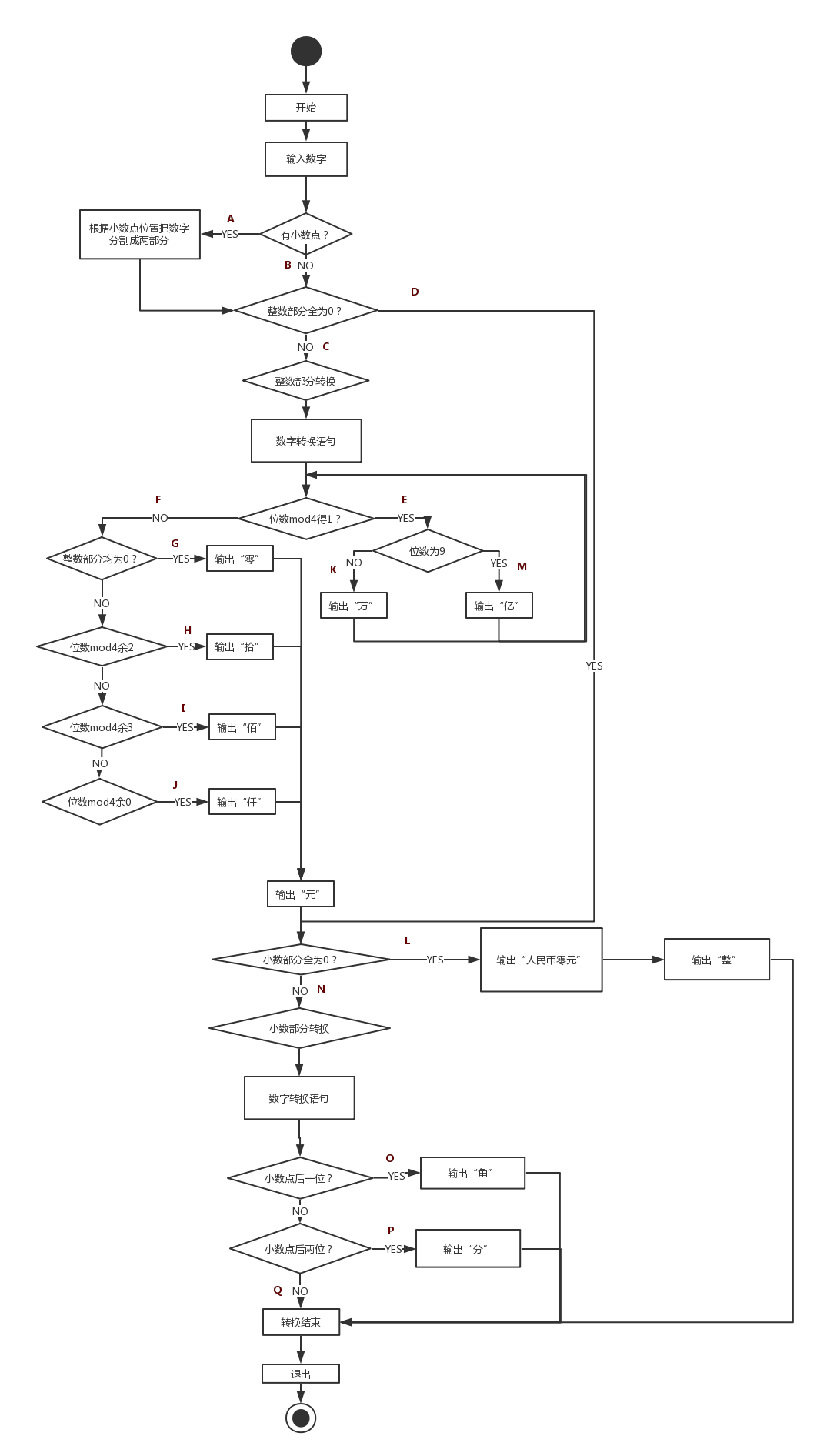
3．采用“判定／条件覆盖”标准设计测试用例

|  |  |  |  |
| --- | --- | --- | --- |
| 测试数据 | 执行路径 | 预期结果 | 实际结果 |
| 4567.89 | RSACTWFGHIJZNVWOPQY | 人民币肆仟伍佰陆拾柒元捌角玖分 | 人民币肆仟伍佰陆拾柒元捌角玖分 |
| 123456789.00 | RSACTWFGHIJZLQY | 人民币壹亿贰仟叁佰肆拾伍万陆仟柒佰捌拾玖元整 | 人民币壹亿贰仟叁佰肆拾伍万陆仟柒佰捌拾玖元整 |
| 4.11 | RSACTWFNVWOPQY | 人民币肆元壹角壹分 | 人民币肆元壹角壹分 |
| 8.00 | RSACTWFZLY | 人民币捌元整 | 人民币捌元整 |
| 0.00 | RSACDLY | 人民币零元整 | 人民币零元整 |
| 6789 | RSBCTWFGHIJZLY | 人民币陆仟柒佰捌拾玖元整 | 人民币陆仟柒佰捌拾玖元整 |

4．采用“基本路径方法”设计测试用例

|  |  |  |  |
| --- | --- | --- | --- |
| 测试数据 | 执行路径 | 预期结果 | 实际结果 |
| 7834.23 | RSACTWFGHIJZNVWOPQY | 人民币柒仟捌佰叁拾肆元贰角叁分 | 人民币柒仟捌佰叁拾肆元贰角叁分 |
| 222222200.00 | RSACTWFGHIJZLQY | 人民币贰亿贰仟贰佰贰拾贰万贰仟贰佰元整 | 人民币贰亿贰仟贰佰贰拾贰万贰仟贰佰元整 |
| 6.06 | RSACTWFNVWOPQY | 人民币陆元零陆分 | 人民币陆元零陆分 |
| 9.00 | RSACTWFZLY | 人民币玖元整 | 人民币玖元整 |
| 0.00 | RSACDLY | 人民币零元整 | 人民币零元整 |
| 6896 | RSBCTWFGHIJZLY | 人民币陆仟捌佰玖拾陆元整 | 人民币陆仟捌佰玖拾陆元整 |

并根据部分测试用例，编写Junit测试代码



### 编写Junit测试代码

为提高代码的重用度，使用Junit 提供的参数化测试方式编写测试代码。编写步骤如下：

1．为准备使用参数化测试的测试类指定特殊的运行器 org.junit.runners.Parameterized。

2．为测试类声明几个变量，分别用于存放期望值和测试所用数据。

3．为测试类声明一个使用注解 org.junit.runners.Parameterized.Parameters 修饰的，返回值为 java.util.Collection 的公共静态方法，并在此方法中初始化所有需要测试的参数对。

4．为测试类声明一个带有参数的公共构造函数，并在其中为第二个环节中声明的几个变量赋值。

5．编写测试方法，使用定义的变量作为参数进行测试。

代码如下：

**package** com.su.test;

**import** **static** org.junit.Assert.\*;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** org.junit.Assert;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Parameterized;

**import** org.junit.runners.Parameterized.Parameters;

**import** com.su.util.NumericConvertToRMB;

/\*\*

\* **@title** NumericConvertToRMBTest.java

\* **@author** Shuai

\* **@date** 2016-6-5下午7:58:41

\*/

@RunWith(Parameterized.**class**)

**public** **class** NumericConvertToRMBTest {

**private** String expected;

**private** **double** target;

@Parameters

**public** **static** Collection words(){

**return** Arrays.*asList*(**new** Object[][]{

{"人民币壹亿零壹拾元零零角零分",100000010.00}, {"输入不合法", -1.00}, {"人民币零元零角零分", 0.00}, {"人民币零元壹分", 0.01}, {"人民币零元壹角壹分", 0.11},

{"人民币玖仟玖佰玖拾玖亿玖仟玖佰玖拾玖万玖仟玖佰玖拾玖元玖角玖分", 999999999999.99},

{"人民币玖仟玖佰玖拾玖亿玖仟玖佰玖拾玖万玖仟玖佰玖拾玖元玖角捌分", 999999999999.98},

{"输入不合法", 10000000000000.00},

{"输入不合法", -0.99},

{"人民币壹亿零壹拾元零壹分", 100000010.01},

{"人民币壹亿元零零角零分", 100000000.00},

{"人民币壹元零零角零分", 1},

{"人民币壹元贰角叁分", 1.234}, {"人民币壹元壹角", 1.10},

{"人民币零元壹角零分", 0.10},

{"人民币零元壹分", 0.01}

});

}

/\*\*

\* 参数化测试必须的构造函数

\*/

**public** NumericConvertToRMBTest(String expected , **double** target){

**this**.expected = expected;

**this**.target = target;

}

/\*\*

\* 测试 numericConvertToRMB 人民币大小写转换

\*/

@Test

**public** **void** testNumericConvertToRMB() {

String result="";

**if**(target>=10000000000000.00||target < 0.00){

result="输入不合法";

}**else**{

result=NumericConvertToRMB.*numericConvertToRMB*(target);

}

Assert.*assertEquals*(expected,result);

}

}

1. **测试结果概述**

## 测试总体评估

测试结果和预期结果一致，预期结果是根据系统需求所设计的，故总体符合需求说明。

## 改进建议

健壮性有待提升，对测试用例的覆盖程度不够。

1. **评估**

## 软件能力

软件完成需求描述的功能，对边界情况和各种条件组合情况也满足输出要求。

## 缺陷和限制

健壮性有待提升，测试用例无法穷举，可能有位置错误，没测出来。

## 建议

健壮性有待提升，测试多采用的弱健壮性测试，对测试用例的覆盖程度不够。

## 结论

功能相对完善，基本符合设计的测试用例。本次实验内容主要为学习Junit4的单元测试工具方法，并实际运用在实例上。主要收获应该是对Junit单元测试方法有了加深理解，也对软件测试在项目开发中的作用有了清晰的认识。