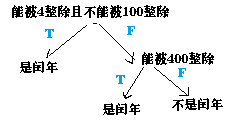
**1.试题编号：H2-1《人力资源管理系统》应用程序单元测试**

任务一：判断闰年

**解答：**

①根据本模块附件录1设计测试用例表的要求分别设计测试用例。(20分)

表 1 设计测试用例

|  |  |
| --- | --- |
| 用例要求 | 测试用例 |
| 语句覆盖 | year=2004  year=2000  year=2001 |
| 判定覆盖 | year=2004  year=2000  year=2100 |
| 条件覆盖 | year=2000  year=2100  year=2001 |
| 判定/条件覆盖 | year=2004  year=2000  year=2100  year=2001 |
| 组合覆盖 | year=2004  year=2100  year=2000  year=2001 |
| 路径覆盖 | year=2004  year=2000  year=2100 |

②使用 xunit 工具和路径覆盖测试用例编写单元测试程序，展示测试结果，分析测试结果。(10分)

类代码：

**public** **class** Year {

**public** **boolean** isLeepYear(**int** year){

//能被4整除且不能被100整除

**if**((year%4==0)||(year%100!=0)){

**return** **true**;

//能被400整除

}**else** **if**(year%400==0){

**return** **true**;

}**else**{

**return** **false**;

}

}

测试类代码：

import static org.junit.Assert.\*;

import org.junit.Test;

public class YearTest {

@Test

public void test() {

Year y=new Year();

assertEquals(y.isLeepYear(2000),true);

assertEquals(y.isLeepYear(2100),false);

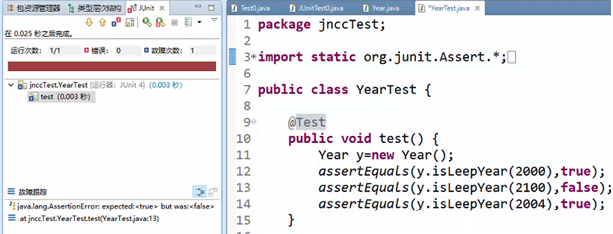
assertEquals(y.isLeepYear(2004),true);

assertEquals(y.isLeepYear(2001),false);

}

}

测试结果：有BUG



用例测试结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例要求 | 测试用例 | 预测结果 | 运行结果 | 结论 |
| 路径覆盖 | year=2004 | True | True |  |
| year=2000 | True | True |  |
| year=2100 | false | True | BUG |

分析BUG原因：2100年能被4整除，也能被100整除，正确判断为不是闰年

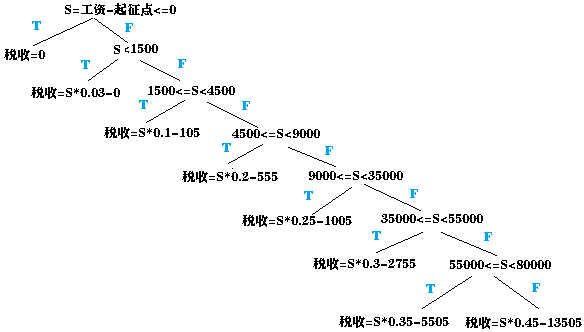
测试2100年属于闰年，从程序year%100!=0前后找原因，原来是||或运算符的原因，正确符号应该是&&与运算符才对。

**if**((year%4==0)||(year%100!=0))错误

正确的应该是**if**((year%4==0)&&(year%100!=0))

任务二：计算工资纳税额

1. 根据本模块附件录1设计测试用例表的要求分别设计测试用例。(20分)

表 1 设计测试用例

|  |  |
| --- | --- |
| 用例要求 | 测试用例 |
| 语句覆盖 | Salary=3500,deduct=3500  Salary=4999,deduct=3500  Salary=5000,deduct=3500  Salary=8000,deduct=3500  Salary=12500,deduct=3500  Salary=38500,deduct=3500  Salary=58500,deduct=3500  Salary=83500,deduct=3500 |
| 判定覆盖 | Salary=3500,deduct=3500  Salary=4999,deduct=3500  Salary=5000,deduct=3500  Salary=8000,deduct=3500  Salary=12500,deduct=3500  Salary=38500,deduct=3500  Salary=58500,deduct=3500  Salary=83500,deduct=3500 |
| 条件覆盖 | Salary=3500,deduct=3500  Salary=4999,deduct=3500  Salary=5000,deduct=3500  Salary=8000,deduct=3500  Salary=12500,deduct=3500  Salary=38500,deduct=3500  Salary=58500,deduct=3500  Salary=83500,deduct=3500 |
| 判定/条件覆盖 | Salary=3500,deduct=3500  Salary=4999,deduct=3500  Salary=5000,deduct=3500  Salary=8000,deduct=3500  Salary=12500,deduct=3500  Salary=38500,deduct=3500  Salary=58500,deduct=3500  Salary=83500,deduct=3500 |
| 组合覆盖 | Salary=3500,deduct=3500  Salary=4999,deduct=3500  Salary=5000,deduct=3500  Salary=8000,deduct=3500  Salary=12500,deduct=3500  Salary=38500,deduct=3500  Salary=58500,deduct=3500  Salary=83500,deduct=3500 |
| 路径覆盖 | Salary=3500,deduct=3500  Salary=4999,deduct=3500  Salary=5000,deduct=3500  Salary=8000,deduct=3500  Salary=12500,deduct=3500  Salary=38500,deduct=3500  Salary=58500,deduct=3500  Salary=83500,deduct=3500 |

public class Salary {

/\*\*

\*个人所得税具体计算方法

\*注意：本程序中假设工资已经减去五险一金

\*@param salary工资

\*@param deduct起征点

\*@return 个人所得税

\*/

public double selfValue(int salary,int deduct){

double sefValue=0;

if(salary<1500){

sefValue=(double)(salary-deduct)\*0.03-0;

}else if(salary>=1500&&salary<4500){

sefValue=(double)(salary-deduct)\*0.1-105;

}else if(salary>=4500&&salary<9000){

sefValue=(double)(salary-deduct)\*0.2-555;

}else if(salary>=9000&&salary<35000){

sefValue=(double)(salary-deduct)\*0.25-1005;

}else if(salary>=35000&&salary<55000){

sefValue=(double)(salary-deduct)\*0.3-2755;

}else if(salary>=55000&&salary<80000){

sefValue=(double)(salary-deduct)\*0.35-5505;

}else {

sefValue=(double)(salary-deduct)\*0.45-13505;

}

return sefValue;

}

}

②使用 xunit 工具和路径覆盖测试用例编写单元测试程序，展示测试结果，分析测试结果。(10分)

import static org.junit.Assert.\*;

import org.junit.Test;

public class TestSalary {

@Test

public void test() {

Salary S=new Salary();

assertEquals((Double)S.selfValue(3500, 3500),(Double)0.0);

assertEquals((Double)S.selfValue(4999, 3500),(Double)44.97);

assertEquals((Double)S.selfValue(5000, 3500),(Double)45.0);

assertEquals((Double)S.selfValue(8000, 3500),(Double)345.0);

assertEquals((Double)S.selfValue(12500, 3500),(Double)1245.0);

assertEquals((Double)S.selfValue(38500, 3500),(Double)7745.0);

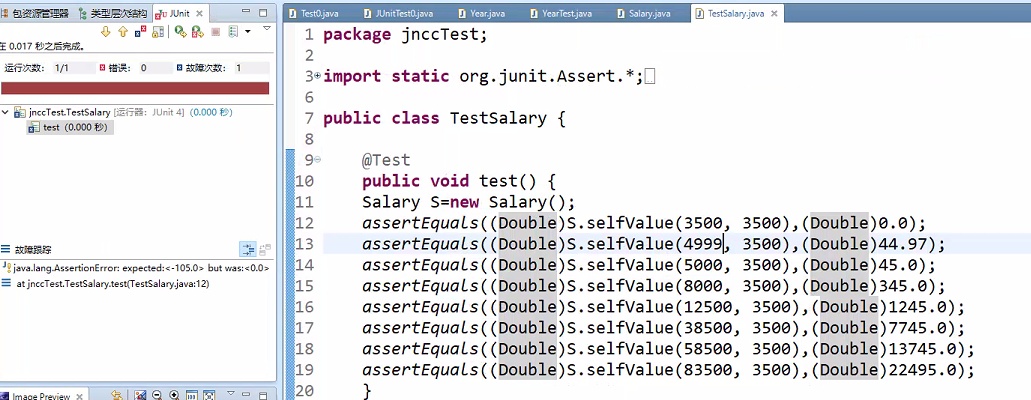
assertEquals((Double)S.selfValue(58500, 3500),(Double)13745.0);

assertEquals((Double)S.selfValue(83500, 3500),(Double)22495.0);

}

}

测试结果：有BUG



用例测试结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用例要求 | 测试用例 | 预测结果 | 运行结果 | 结论 |
| 路径覆盖 | salary=3500 | 0 | 105 | Bug |
| salary=4999 | 44.97 | 255.2 | Bug |
| salary=5000 | 45 | 255 | Bug |
| salary=8000 | 345 | 345 |  |
| salary=12500 | 1245 | 1245 |  |
| salary=38500 | 7745 | 7745 |  |
| salary=58500 | 13745 | 13745 |  |
| salary=83500 | 22495 | 22495 |  |

分析BUG原因：salary小于3500的起征点，应该不用收税，程序不应该以salary判断，应该以salary-deduct判断，且应该增加没有salary-deduct<=0的情况。

有BUG的代码：

if(salary<1500){

sefValue=(double)(salary-deduct)\*0.03-0;

正确的代码：

if(salary-deduct<=0){

sefValue=0;

}Else if(salary-deduct<1500){

sefValue=(double)(salary-deduct)\*0.03-0;