**广州大学学生实验报告**

**开课学院及实验室：**AI实验室 **2023年11月2日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院** | **计算机科学与网络工程学院** | **年级/专业/班** | **软件211/212/213** | **姓名** |  | **学号** |  |
| **实验课程名称** | 软件质量保证与测试实验 | | | | | **成绩** |  |
| **实验项目名称** | 实验一 单元测试实验  实验二 功能测试实验 | | | | | **指导老师** | 李亚 |

教师评语：

**实验一 单元测试实验**

一、实验目的

1、掌握黑盒测试技术和白盒测试技术

2、熟练使用Junit测试框架进行基于Java语言的单元测试

3、掌握基于覆盖和基于独立路径的测试用例设计方法

4、掌握等价类划分法、边界值法、决策表法等测试用例设计方法

二、基本知识

1、单元测试概念

2、Junit测试框架的使用

3、黑盒/白盒测试技术

三、实验环境

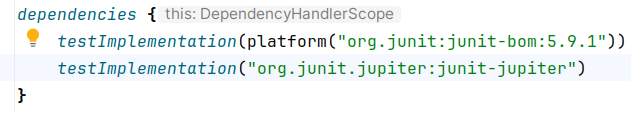
①NeoKylin操作系统+浏览器

②Jetbrains IntelliJ IDEA集成开发环境+Junit 5测试框架

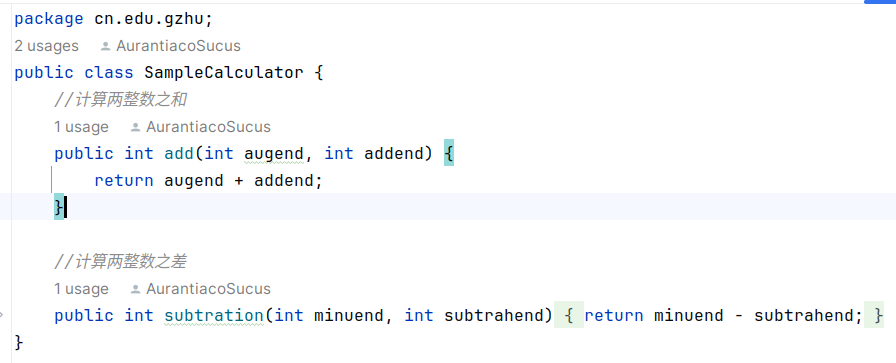
四、实验内容

1、学习Junit框架的使用

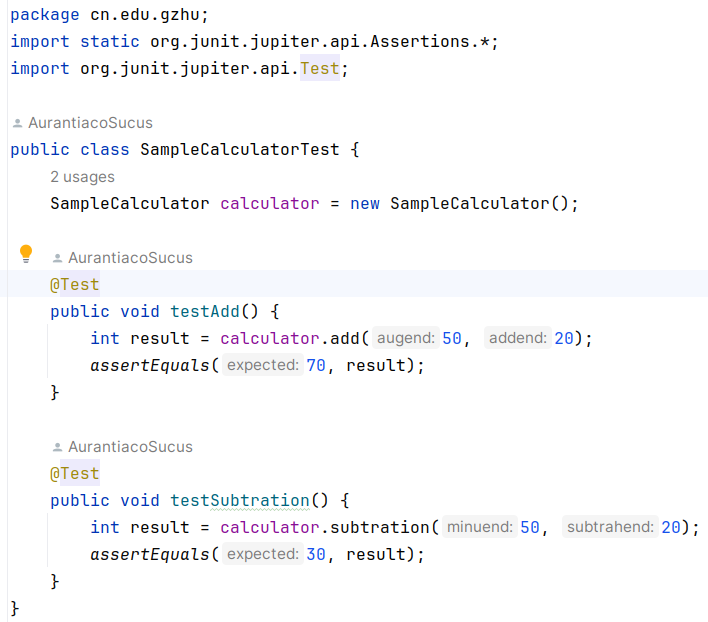
使用IDEA创建Gradle项目，添加JUnit Jupiter支持。



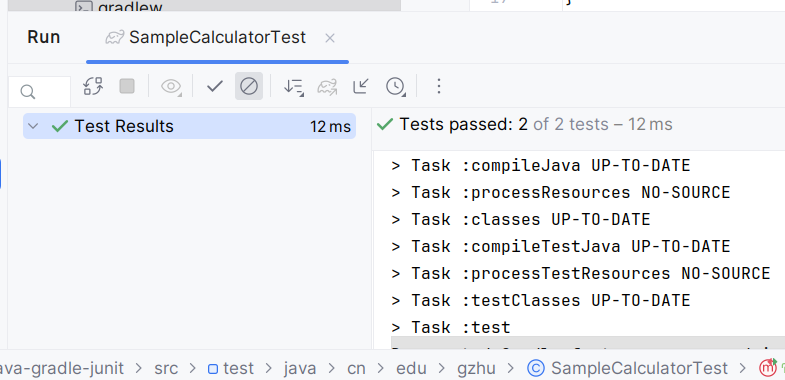
添加业务逻辑。



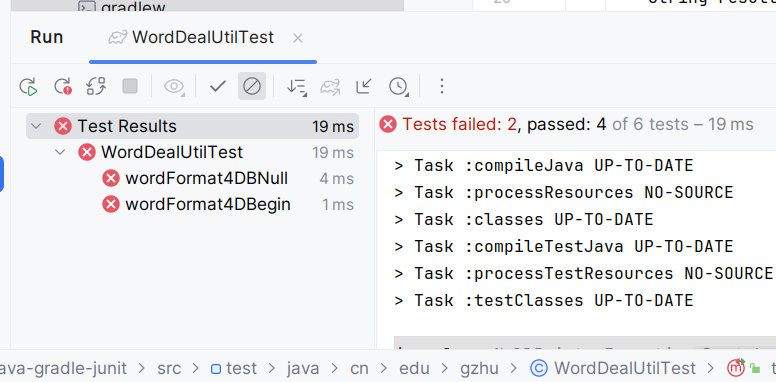
针对其添加测试（到test源文件根）。



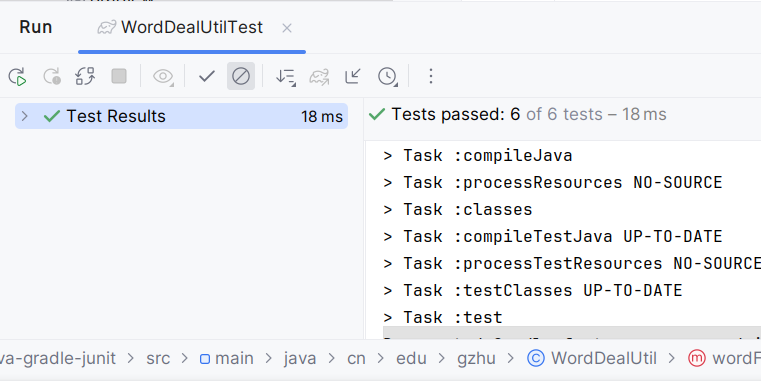
运行测试，通过。



按照要求添加WordDealUtil类，添加单元测试，运行通过。按照要求更改完善测试之后，运行出现错误。

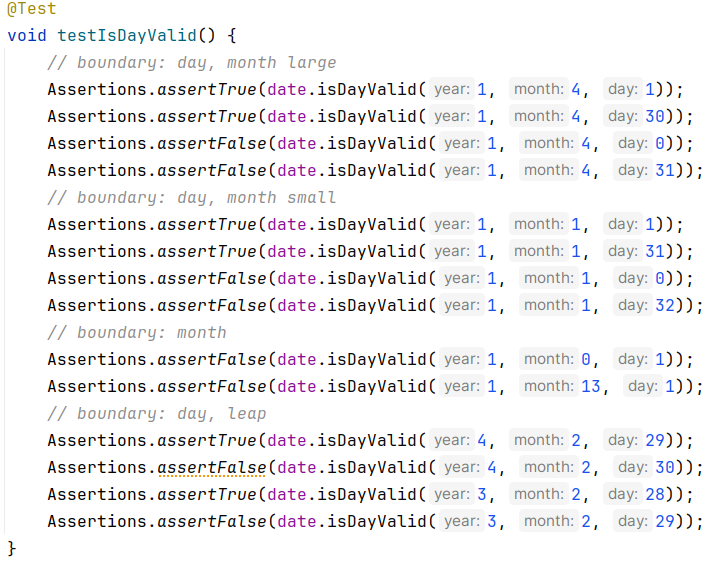


修复漏洞之后，测试通过。



2、使用Junit框架对类Date和类DateUtil进行单元测试。

对闰年、平年分别测试大月小月的边界值，还有2月的边界值，发现倒数第三个assertion出现问题。



源代码中发现逻辑错误，予以修改，通过测试。

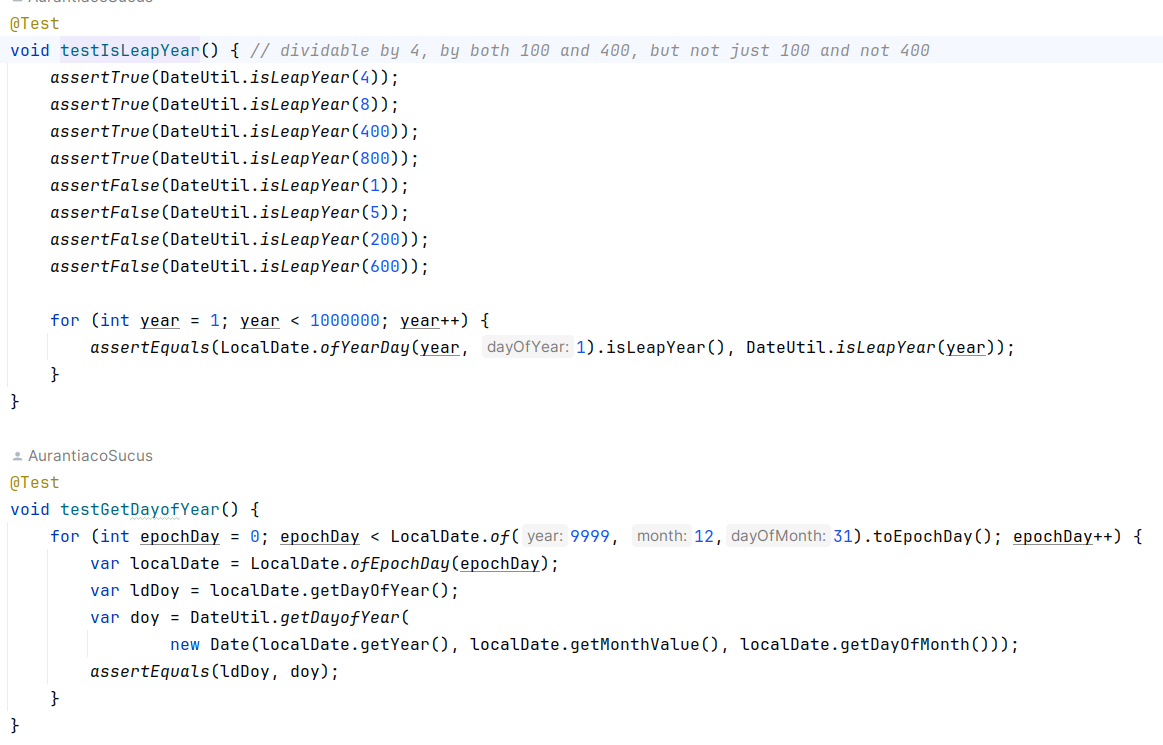


测试1到12月的有效性，和两个无效边界值。年份测试有效年边界值和无效年边界值。

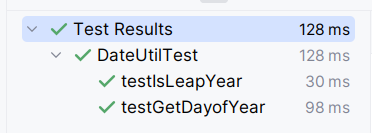


所有测试均通过。

编写针对DateUtil的测试如下：

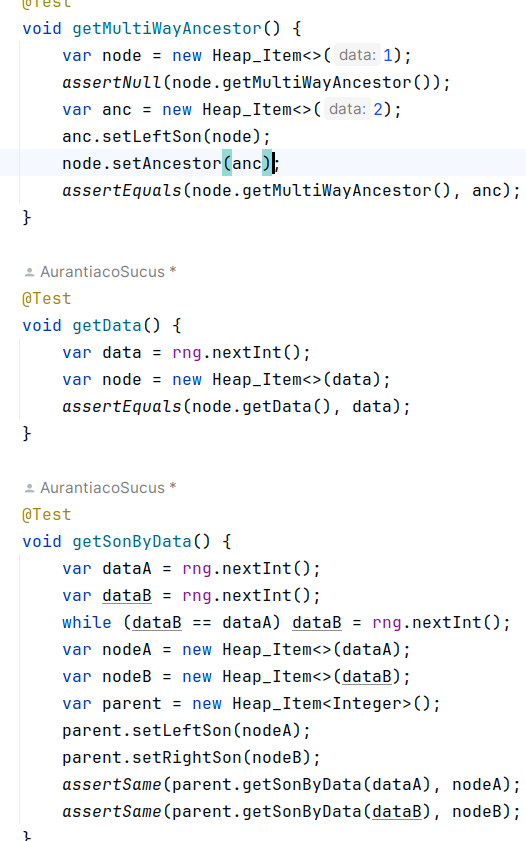


如下可见，测试通过。

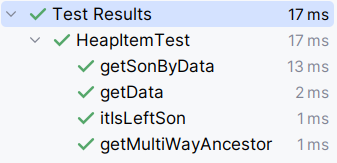


3、使用Junit框架对类Heap\_Item和类BinaryExponentiation进行单元测试。

编写Heap\_Item测试如下：



运行测试，通过。

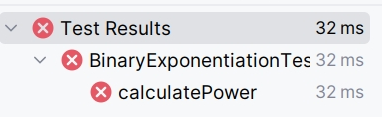


编写BinaryExponentiation测试如下：

（两个函数测试方法同理，故不赘述，仅展示其一）：

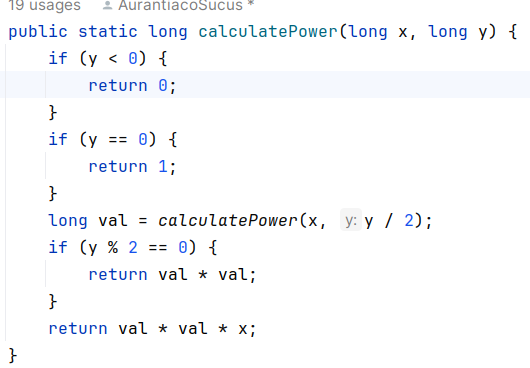


经过测试发现负指数的情况出现了结果不正确的情况。

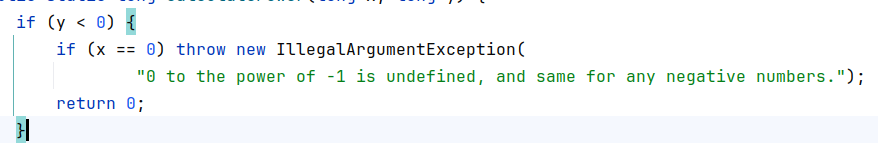


修改代码，添加负指数逻辑。可以推测负指数情况下这两个返回整型值的函数只可能返回0（小数被舍弃）。

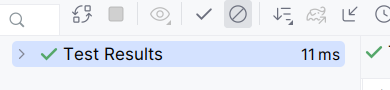
另一个函数的修改同理。修改完成再次运行测试，发现仍然不通过。在输入不合法的情况下没有抛出错误。（无关部分已略过）



再次修改代码，添加相关判断。



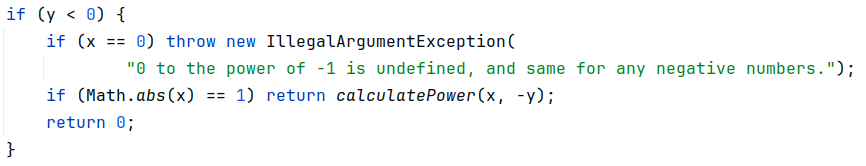
修改后再次测试，所有测试用例通过。



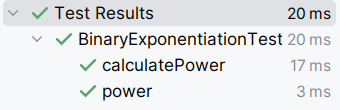
完善测试，添加特殊情况：0^0=1; 1^-1=1;-1^-1=-1；-1的负奇数次幂为-1，-1的负偶数次幂为1。



发现测试不通过，完善判断代码：



再次运行测试，通过。



4、完成以下用例设计

#### A、采用决策表法完成“隔一天日期问题”的测试用例设计

易得等价类表如下：

|  |  |  |
| --- | --- | --- |
| 输入 | 有效等价类 | 无效等价类 |
| month | * 整数 * 1≤month≤12 | * 不是整数 * month＜1 * month＞12 |
| day | * 整数 * 1≤day≤31 | * 不是整数 * day＜1 * day＞31 |
| year | * 整数 | * 不是整数 |

由此，可以推测程序会出现以下不同种类行为：

1. 输出隔一天日期
2. 返回错误信息

简化后的决策表如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 原因 | month合法 | — | — | 0 | 1 |
| day合法 | — | 0 | — | 1 |
| year合法 | 0 | — | — | 1 |
| 结果 | 输出隔一天日期 | 0 | 0 | 0 | 1 |
| 返回错误信息 | 1 | 1 | 1 | 0 |

测试用例：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 输入 | | | 输出 | | |
| month | day | year | month | day | year |
| 1 | 1 | 0 |  |  |  |
| 1 | 0 | 1 |  |  |  |
| 0 | 1 | 1 |  |  |  |
| 1 | 1 | 1 | 1 | 2 | 1 |

根据边界情况考虑的附加用例：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 输入 | | | 输出 | | |
| month | day | year | month | day | Year |
| 1 | 32 | 1 |  |  |  |
| 2 | 30 | 1 |  |  |  |
| 13 | 1 | 1 |  |  |  |

#### B、采用成对测试方法进行配置测试

#### 1）、给定基本配置情况，采用工具设计测试用例

此处给出前30条记录。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PLATFORM | CPUS | RAM | HDD | OS | IE |
| 1a64 | Quad | 128MB | IDE | Win10 | 9 |
| 1a64 | Single | 1GB | SCSI | WinXP | 9 |
| amd64 | Dual | 128MB | SCSI | Win7 | 8 |
| x86 | Quad | 64GB | SCSI | WinXP | 10 |
| amd64 | Single | 64GB | IDE | Win7 | 10 |
| x86 | Quad | 1GB | SCSI | Win10 | 10 |
| amd64 | Dual | 1GB | IDE | Win10 | 10 |
| 1a64 | Single | 1GB | SCSI | WinXP | 8 |
| x86 | Single | 4GB | IDE | Win10 | 9 |
| 1a64 | Dual | 4GB | SCSI | NT4 | 10 |
| 1a64 | Dual | 64GB | IDE | Win7 | 8 |
| x86 | Quad | 64GB | SCSI | NT4 | 9 |
| x86 | Dual | 1GB | SCSI | Win10 | 8 |
| 1a64 | Single | 1GB | SCSI | NT4 | 10 |
| x86 | Quad | 1GB | SCSI | WinXP | 10 |
| 1a64 | Dual | 4GB | SCSI | WinXP | 10 |
| x86 | Single | 4GB | IDE | NT4 | 11 |
| 1a64 | Dual | 64GB | IDE | Win10 | 11 |
| 1a64 | Single | 1GB | SCSI | Win7 | 11 |
| amd64 | Quad | 4GB | IDE | WinXP | 10 |
| amd64 | Dual | 1GB | IDE | NT4 | 8 |
| amd64 | Dual | 128MB | SCSI | Win7 | 11 |
| 1a64 | Dual | 64GB | IDE | Win10 | 9 |
| amd64 | Single | 64GB | IDE | NT4 | 8 |
| 1a64 | Dual | 4GB | SCSI | NT4 | 8 |
| x86 | Dual | 1GB | SCSI | WinXP | 10 |
| 1a64 | Single | 1GB | SCSI | Win7 | 10 |
| 1a64 | Quad | 128MB | IDE | Win10 | 11 |
| x86 | Quad | 64GB | SCSI | Win7 | 11 |

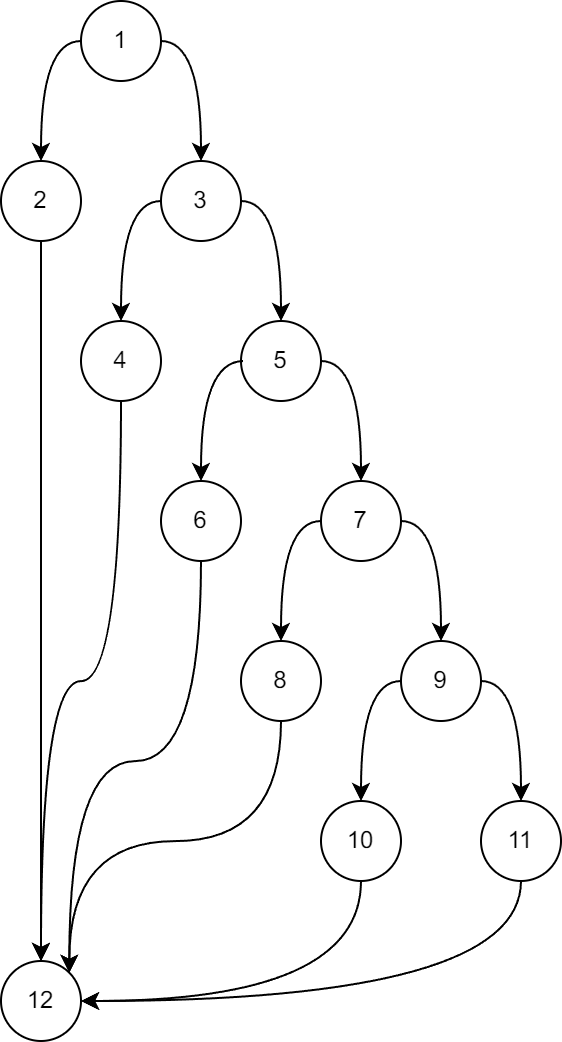
#### 2）、变换或增加配置约束，观察测试用例的变化

这里将RAM移入OS和IE的子模型，其他参数未作修改，可以发现RAM和OS、IE之间的冗余性变小了。此处给出前30条记录：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PLATFORM | CPUS | RAM | HDD | OS | IE |
| x86 | Single | 64GB | IDE | WinXP | 8 |
| x86 | Quad | 1GB | SCSI | Win7 | 8 |
| 1a64 | Dual | 1GB | IDE | Win10 | 10 |
| amd64 | Quad | 128MB | SCSI | Win10 | 8 |
| x86 | Dual | 4GB | IDE | Win7 | 9 |
| 1a64 | Dual | 4GB | IDE | Win7 | 9 |
| x86 | Dual | 4GB | IDE | NT4 | 8 |
| 1a64 | Dual | 1GB | IDE | WinXP | 10 |
| amd64 | Dual | 1GB | SCSI | WinXP | 10 |
| x86 | Quad | 4GB | SCSI | Win7 | 9 |
| x86 | Single | 64GB | IDE | Win10 | 9 |
| 1a64 | Quad | 4GB | IDE | Win7 | 9 |
| 1a64 | Single | 4GB | SCSI | WinXP | 11 |
| x86 | Single | 128MB | IDE | NT4 | 11 |
| x86 | Quad | 64GB | SCSI | Win10 | 9 |
| x86 | Single | 128MB | IDE | WinXP | 9 |
| 1a64 | Single | 4GB | SCSI | NT4 | 8 |
| 1a64 | Single | 1GB | SCSI | Win10 | 10 |
| 1a64 | Dual | 4GB | IDE | NT4 | 8 |
| 1a64 | Quad | 1GB | IDE | Win10 | 11 |
| x86 | Quad | 4GB | SCSI | NT4 | 8 |
| 1a64 | Single | 64GB | SCSI | Win7 | 11 |
| x86 | Dual | 64GB | IDE | NT4 | 10 |
| 1a64 | Quad | 128MB | IDE | Win10 | 8 |
| amd64 | Single | 4GB | IDE | WinXP | 11 |
| x86 | Dual | 128MB | IDE | Win10 | 8 |
| 1a64 | Single | 128MB | SCSI | NT4 | 11 |
| amd64 | Single | 1GB | IDE | Win7 | 8 |
| amd64 | Single | 128MB | IDE | Win7 | 10 |

#### C、基于独立路径的测试用例设计

控制流图如下：



由图可知，环形复杂度为6。

其中独立路径有：

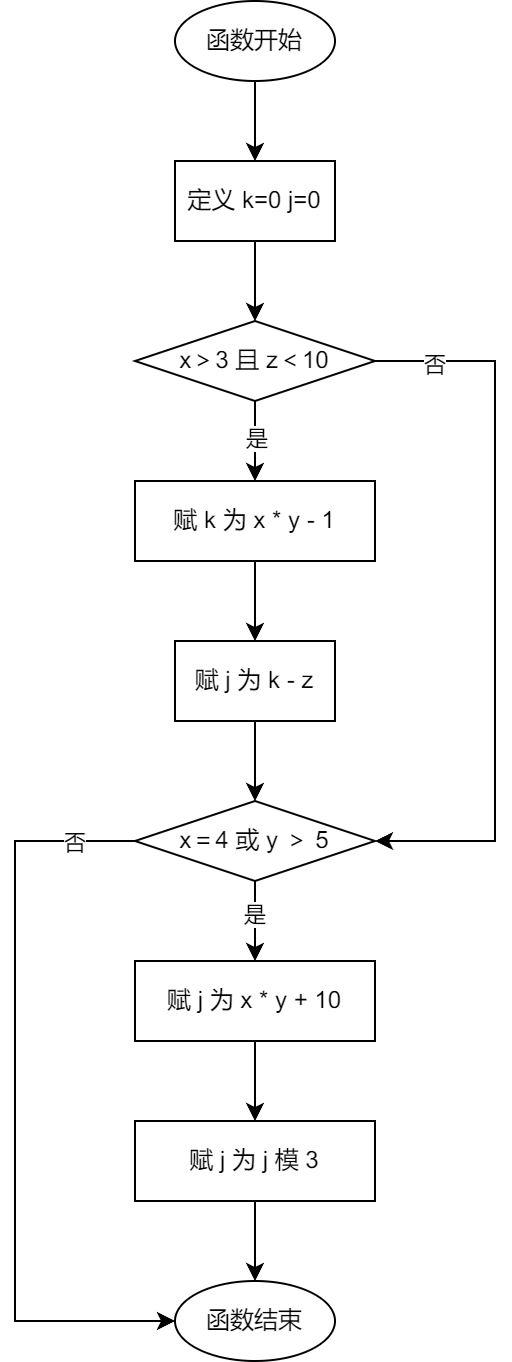
* 1-2-12
* 1-3-4-12
* 1-3-5-6-12
* 1-3-5-7-8-12
* 1-3-5-7-9-10-12
* 1-3-5-7-9-11-12

测试用例设计:

|  |  |
| --- | --- |
| Cstring temp | M\_oper.SetCurSel(x) |
| >= | 0 |
| > | 1 |
| == | 2 |
| <= | 3 |
| < | 4 |
| Hello, World! | 5 |

D、基于逻辑覆盖的测试用例设计

流程图如下：



由题意可知该函数没有返回值。

语句覆盖测试用例：

|  |  |  |
| --- | --- | --- |
| X | y | z |
| 4 | 6 | 9 |

经推演，可知该测试用例达到100%覆盖率。

判定覆盖测试用例：

|  |  |  |
| --- | --- | --- |
| x | y | z |
| 4 | 6 | 9 |
| 0 | 4 | 11 |

经推演，可知两个条件判断的所有分支都得到覆盖。

条件覆盖：

|  |  |  |
| --- | --- | --- |
| x | y | z |
| 4 | 5 | 9 |
| 2 | 6 | 10 |

经推演，可知所有条件的真假情况都得到覆盖。

判定-条件覆盖：

|  |  |  |
| --- | --- | --- |
| x | y | z |
| 4 | 6 | 9 |
| 0 | 4 | 11 |
| 4 | 5 | 9 |
| 2 | 6 | 10 |

条件组合覆盖：

|  |  |  |
| --- | --- | --- |
| x | y | z |
| 4 | 1 | 9 |
| 4 | 1 | 10 |
| 3 | 1 | 10 |
| 5 | 4 | 9 |

五、实验结果（或实验代码）

结果在上文已经给出，下文为代码。

DateTest

package cn.edu.gzhu;  
  
import org.junit.jupiter.api.Assertions;  
import org.junit.jupiter.api.Test;  
  
class DateTest {  
 Date date = new Date(1, 1, 1);  
  
 @Test  
 void testIsDayValid() {  
 *// boundary: day, month large* Assertions.*assertTrue*(date.isDayValid(1, 4, 1));  
 Assertions.*assertTrue*(date.isDayValid(1, 4, 30));  
 Assertions.*assertFalse*(date.isDayValid(1, 4, 0));  
 Assertions.*assertFalse*(date.isDayValid(1, 4, 31));  
 *// boundary: day, month small* Assertions.*assertTrue*(date.isDayValid(1, 1, 1));  
 Assertions.*assertTrue*(date.isDayValid(1, 1, 31));  
 Assertions.*assertFalse*(date.isDayValid(1, 1, 0));  
 Assertions.*assertFalse*(date.isDayValid(1, 1, 32));  
 *// boundary: month* Assertions.*assertFalse*(date.isDayValid(1, 0, 1));  
 Assertions.*assertFalse*(date.isDayValid(1, 13, 1));  
 *// boundary: day, leap* Assertions.*assertTrue*(date.isDayValid(4, 2, 29));  
 Assertions.*assertFalse*(date.isDayValid(4, 2, 30));  
 Assertions.*assertTrue*(date.isDayValid(3, 2, 28));  
 Assertions.*assertFalse*(date.isDayValid(3, 2, 29));  
 }  
  
 @Test  
 void testIsMonthValid() {  
 for (int i = 1; i < 13; i++) {  
 Assertions.*assertTrue*(date.isMonthValid(i));  
 }  
 Assertions.*assertFalse*(date.isMonthValid(0));  
 Assertions.*assertFalse*(date.isMonthValid(13));  
 }  
  
 @Test  
 void testIsYearValid() {  
 Assertions.*assertTrue*(date.isYearValid(1));  
 Assertions.*assertFalse*(date.isYearValid(0));  
 }  
}

DateUtilTest

package cn.edu.gzhu;  
  
import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.jupiter.api.Test;  
  
import java.time.LocalDate;  
import java.time.LocalDateTime;  
  
class DateUtilTest {  
  
 @Test  
 void testIsLeapYear() { *// dividable by 4, by both 100 and 400, but not just 100 and not 400  
 assertTrue*(DateUtil.*isLeapYear*(4));  
 *assertTrue*(DateUtil.*isLeapYear*(8));  
 *assertTrue*(DateUtil.*isLeapYear*(400));  
 *assertTrue*(DateUtil.*isLeapYear*(800));  
 *assertFalse*(DateUtil.*isLeapYear*(1));  
 *assertFalse*(DateUtil.*isLeapYear*(5));  
 *assertFalse*(DateUtil.*isLeapYear*(200));  
 *assertFalse*(DateUtil.*isLeapYear*(600));  
  
 for (int year = 1; year < 1000000; year++) {  
 *assertEquals*(LocalDate.*ofYearDay*(year, 1).isLeapYear(), DateUtil.*isLeapYear*(year));  
 }  
 }  
  
 @Test  
 void testGetDayofYear() {  
 for (int epochDay = 0; epochDay < LocalDate.*of*(9999, 12,31).toEpochDay(); epochDay++) {  
 var localDate = LocalDate.*ofEpochDay*(epochDay);  
 var ldDoy = localDate.getDayOfYear();  
 var doy = DateUtil.*getDayofYear*(  
 new Date(localDate.getYear(), localDate.getMonthValue(), localDate.getDayOfMonth()));  
 *assertEquals*(ldDoy, doy);  
 }  
 }  
}

SampleCalculatorTest

package cn.edu.gzhu;  
import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.jupiter.api.Test;  
  
public class SampleCalculatorTest {  
 SampleCalculator calculator = new SampleCalculator();  
  
 @Test  
 public void testAdd() {  
 int result = calculator.add(50, 20);  
 *assertEquals*(70, result);  
 }  
  
 @Test  
 public void testSubtration() {  
 int result = calculator.subtration(50, 20);  
 *assertEquals*(30, result);  
 }  
}

WordDealUtilTest

package cn.edu.gzhu;  
  
import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.jupiter.api.Test;  
  
class WordDealUtilTest {  
  
 @Test void testWordFormat4DB() {  
 String target = "employeeInfo";  
 String result = WordDealUtil.*wordFormat4DB*(target);  
 *assertEquals*("employee\_info", result);  
 }  
  
 *//测试 null 时的处理情况* @Test public void wordFormat4DBNull(){  
 String target = null;  
 String result = WordDealUtil.*wordFormat4DB*(target);  
 *assertNull*(result);  
 }  
 *//测试空字符串的处理情况* @Test public void wordFormat4DBEmpty(){  
 String target = "";  
 String result = WordDealUtil.*wordFormat4DB*(target);  
 *assertEquals*("", result);  
 }  
 *//测试当首字母大写时的情况* @Test public void wordFormat4DBegin(){  
 String target = "EmployeeInfo";  
 String result = WordDealUtil.*wordFormat4DB*(target);  
 *assertEquals*("employee\_info", result);  
 }  
 *//测试当尾字母为大写时的情况* @Test public void wordFormat4DBEnd(){  
 String target = "employeeInfoA";  
 String result = WordDealUtil.*wordFormat4DB*(target);  
 *assertEquals*("employee\_info\_a", result);  
 }  
 *//测试多个相连字母大写时的情况* @Test public void wordFormat4DBTogether(){  
 String target = "employeeAInfo";  
 String result = WordDealUtil.*wordFormat4DB*(target);  
 *assertEquals*("employee\_a\_info", result);  
 }  
}

BinaryExponentiationTest

package net.mooctest;  
  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class BinaryExponentiationTest {  
  
 @Test  
 void calculatePower() {  
 *// power to zero  
 assertEquals*(1, BinaryExponentiation.*calculatePower*(2, 0));  
 *assertEquals*(1, BinaryExponentiation.*calculatePower*(1, 0));  
 *assertEquals*(1, BinaryExponentiation.*calculatePower*(0, 0));  
 *// power to one  
 assertEquals*(2, BinaryExponentiation.*calculatePower*(2, 1));  
 *// power to zero, negative base  
 assertEquals*(1, BinaryExponentiation.*calculatePower*(-2, 0));  
 *// normal cases  
 assertEquals*(4, BinaryExponentiation.*calculatePower*(2, 2));  
 *assertEquals*(16, BinaryExponentiation.*calculatePower*(4, 2));  
 *// power to one, negative base  
 assertEquals*(-2, BinaryExponentiation.*calculatePower*(-2, 1));  
 *// normal cases, negative base, odd exponent  
 assertEquals*(4, BinaryExponentiation.*calculatePower*(-2, 2));  
 *assertEquals*(16, BinaryExponentiation.*calculatePower*(-4, 2));  
 *// normal cases, negative base, even exponent  
 assertEquals*(-8, BinaryExponentiation.*calculatePower*(-2, 3));  
 *// power to negative numbers  
 assertEquals*(0, BinaryExponentiation.*calculatePower*(2, -1));  
 *assertEquals*(0, BinaryExponentiation.*calculatePower*(3, -3));  
 *assertEquals*(0, BinaryExponentiation.*calculatePower*(-2, -2));  
 *assertEquals*(0, BinaryExponentiation.*calculatePower*(-3, -1));  
 *// division by zero  
 assertThrows*(Exception.class, () -> BinaryExponentiation.*calculatePower*(0, -1));  
 *// special cases  
 assertEquals*(1, BinaryExponentiation.*calculatePower*(0, 0));  
 *assertEquals*(1, BinaryExponentiation.*calculatePower*(1, -1));  
 *assertEquals*(-1, BinaryExponentiation.*calculatePower*(-1, -1));  
 *assertEquals*(1, BinaryExponentiation.*calculatePower*(1, -3));  
 *assertEquals*(-1, BinaryExponentiation.*calculatePower*(-1, -3));  
 *assertEquals*(1, BinaryExponentiation.*calculatePower*(1, -2));  
 }  
  
 @Test  
 void power() {  
 *// power to zero  
 assertEquals*(1, BinaryExponentiation.*power*(2, 0));  
 *assertEquals*(1, BinaryExponentiation.*power*(1, 0));  
 *// power to one  
 assertEquals*(2, BinaryExponentiation.*power*(2, 1));  
 *// power to zero, negative base  
 assertEquals*(1, BinaryExponentiation.*power*(-2, 0));  
 *// normal cases  
 assertEquals*(4, BinaryExponentiation.*power*(2, 2));  
 *assertEquals*(16, BinaryExponentiation.*power*(4, 2));  
 *// power to one, negative base  
 assertEquals*(-2, BinaryExponentiation.*power*(-2, 1));  
 *// normal cases, negative base, odd exponent  
 assertEquals*(4, BinaryExponentiation.*power*(-2, 2));  
 *assertEquals*(16, BinaryExponentiation.*power*(-4, 2));  
 *// normal cases, negative base, even exponent  
 assertEquals*(-8, BinaryExponentiation.*power*(-2, 3));  
 *// power to negative numbers  
 assertEquals*(0, BinaryExponentiation.*power*(2, -1));  
 *assertEquals*(0, BinaryExponentiation.*power*(3, -3));  
 *assertEquals*(0, BinaryExponentiation.*power*(-2, -2));  
 *assertEquals*(0, BinaryExponentiation.*power*(-3, -1));  
 *// division by zero  
 assertThrows*(Exception.class, () -> BinaryExponentiation.*calculatePower*(0, -1));  
 *// special cases  
 assertEquals*(1, BinaryExponentiation.*calculatePower*(0, 0));  
 *assertEquals*(1, BinaryExponentiation.*calculatePower*(1, -1));  
 *assertEquals*(-1, BinaryExponentiation.*calculatePower*(-1, -1));  
 *assertEquals*(1, BinaryExponentiation.*calculatePower*(1, -3));  
 *assertEquals*(-1, BinaryExponentiation.*calculatePower*(-1, -3));  
 *assertEquals*(1, BinaryExponentiation.*calculatePower*(1, -2));  
 }  
}

HeapItemTest

package net.mooctest;  
  
import org.junit.jupiter.api.BeforeAll;  
import org.junit.jupiter.api.Test;  
  
import java.util.Random;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class HeapItemTest {  
 private final Random rng = new Random();  
  
 @Test  
 void getMultiWayAncestor() {  
 var node = new Heap\_Item<>(1);  
 *assertNull*(node.getMultiWayAncestor());  
 var anc = new Heap\_Item<>(2);  
 anc.setLeftSon(node);  
 node.setAncestor(anc);  
 *assertEquals*(node.getMultiWayAncestor(), anc);  
 }  
  
 @Test  
 void getData() {  
 var data = rng.nextInt();  
 var node = new Heap\_Item<>(data);  
 *assertEquals*(node.getData(), data);  
 }  
  
 @Test  
 void getSonByData() {  
 var dataA = rng.nextInt();  
 var dataB = rng.nextInt();  
 while (dataB == dataA) dataB = rng.nextInt();  
 var nodeA = new Heap\_Item<>(dataA);  
 var nodeB = new Heap\_Item<>(dataB);  
 var parent = new Heap\_Item<Integer>();  
 parent.setLeftSon(nodeA);  
 parent.setRightSon(nodeB);  
 *assertSame*(parent.getSonByData(dataA), nodeA);  
 *assertSame*(parent.getSonByData(dataB), nodeB);  
 }  
  
 @Test  
 void itIsLeftSon() {  
 var nodeA = new Heap\_Item<>(1);  
 var nodeB = new Heap\_Item<>(2);  
 var parent = new Heap\_Item<>(3);  
 parent.setLeftSon(nodeA);  
 parent.setRightSon(nodeB);  
 *assertTrue*(parent.itIsLeftSon(nodeA));  
 *assertFalse*(parent.itIsLeftSon(nodeB));  
 *assertFalse*(nodeA.itIsLeftSon(parent));  
 }  
}

六、问题与体会

1. 遇到了JDK太新导致Gradle无法运行的问题，通过安装JDK（LTS）17解决。
2. 遇到了跨Linux（Arch、neoKylin）和Windows（LTSC 2021）多平台开发的编码问题，通过强制JVM使用Unicode编码解决。
3. 这次试验中我提高了开发测试的技巧，深刻了解了各种测试方法，同时学习了JUnit Jupiter的简单使用。