南 京 理 工 大 学

软件测试技术实验

姓　　名: 蒋旭钊　　**学 号:** 918106840727

学院(系): 计算机科学与工程学院

专　　业: 计算机科学与技术

课　　程: 软件测试技术

2021 年　12 月

1. 问题描述

输入年月日，判断该日期是星期几。

2. 被测核心代码及控制流图

我的代码编写主要分为了两个版本，两个版本主要是基于Junit的测试发现缺陷的，进而对此进行了改进。第一个版本没有考虑到输入类超界的问题，产生了许多意想不到的结果，在第二个版本中我针对超界进行了处理，针对第二个版本展示了黑盒测试的代码。针对白盒测试，考虑到将输入的超界判断单独写开来会产生较大的圈复杂度，因此将它们封装成一个函数，展示了封装后的白盒代码，依据白盒测试代码绘制控制流图，以下为两个代码：

2.1核心代码

```
//黑盒测试

//输入年月日，判断该日期是星期几。

public int findDay(double year,double month,double day)
{
    double leapYear=0;
    double noLeapYear=0;
    double days_1=0;
    double days_2=0;
    double days=0;
    int findday=0;

    //年月日是否越界
    if(year<=0)
    {
        return -1;
    }
    if(month<=0||month>=13)
    {
        return -1;
    }
    if(day<=0||day>=32)
    {
```

```
        return -1;
    }

    //年月日是否为整数
    if((int)year!=year)
    {
        return -1;
    }
    if((int)month!=month)
    {
        return -1;
    }
    if((int)day!=day)
    {
        return -1;
    }

    //判断第 1 年到第 year-1 年中闰年的个数
    int i;
    for (i = 1;i < year;i++) {
        if ((i % 4 == 0 && i % 100 != 0) || (i % 400 == 0)) {
            leapYear ++;

        } else {
            noLeapYear ++;

        }
    }
    //通过年个数算出总天数
    days_1= leapYear*366+noLeapYear*365;

    //判断第 year 年中第 1 月到第 month-1 月的天数
    int j;
    for (j = 1;j < month;j++) {
        switch (j) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12: days_2 += 31;break;
            case 4:
            case 6:
```

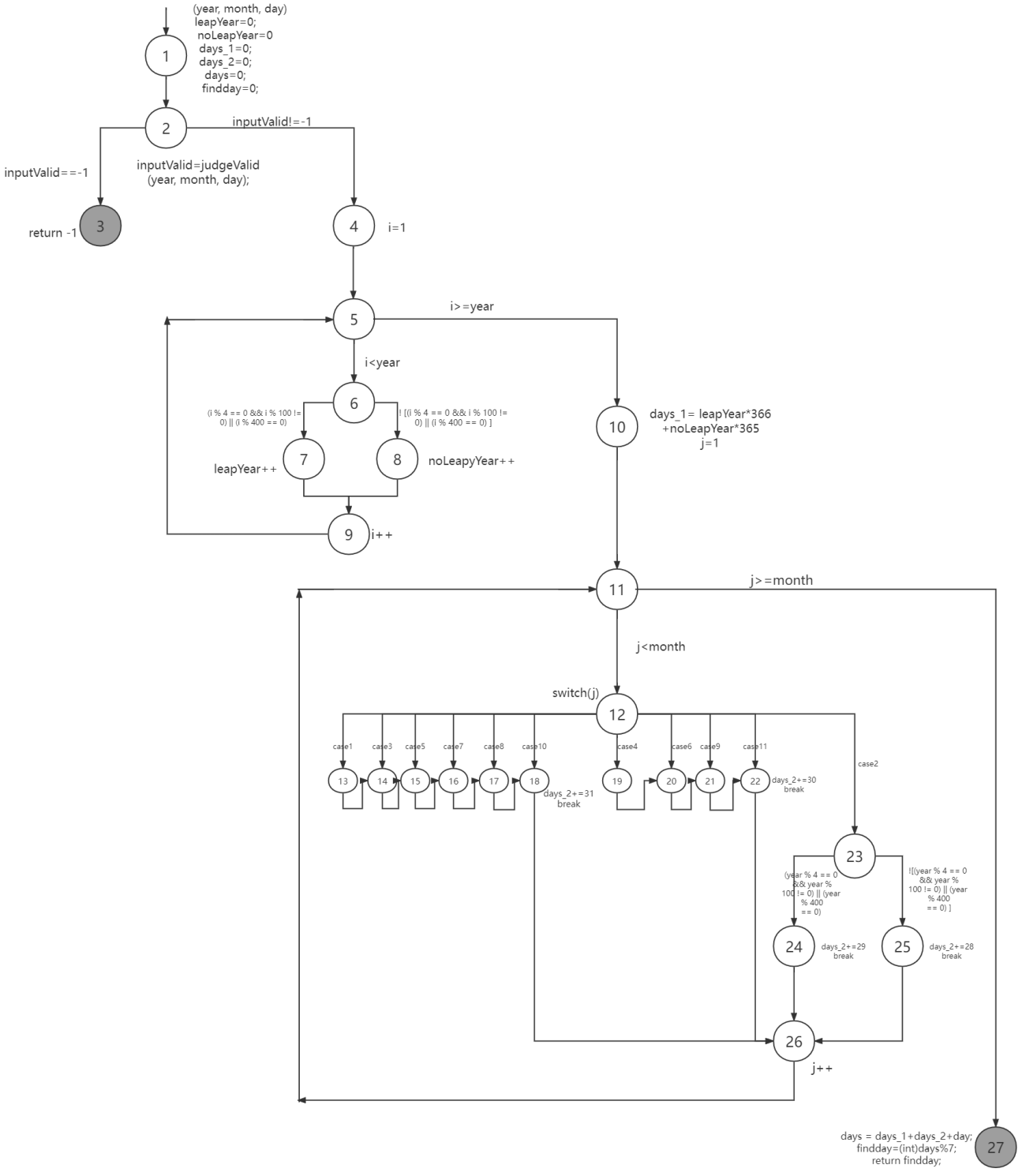
```
            case 9:
            case 11:days_2 += 30;break;
            case 2: if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))
                days_2 += 29;
            else
                days_2 += 28;
                break;
        }
    }

    //总天数
    days = days_1+days_2+day;

    //计算星期数
    findday=(int)days%7;

    //0:Sunday
    //1:Monday
    //2:Tuesday
    //3:Wednesday
    //4:Thursday
    //5:Friday
    //6:Saturday
    return findday;
}
```

```
//白盒测试

//输入年月日，判断该日期是星期几。

public int findDay(double year,double month,double day)
{
    double leapYear=0;
    double noLeapYear=0;
    double days_1=0;
    double days_2=0;
    double days=0;
    int findday=0;
    int inputValid;

    inputValid=judgeValid(year, month, day);
    if (inputValid==-1) return -1;
```

```
//判断第 1 年到第 year-1 年中闰年的个数
int i;
for (i = 1;i < year;i++) {
    if ((i % 4 == 0 && i % 100 != 0) || (i % 400 == 0)) {
        leapYear ++;

    } else {
        noLeapYear ++;

    }
}
//通过年个数算出总天数
days_1= leapYear*366+noLeapYear*365;

//判断第 year 年中第 1 月到第 month-1 月的天数
int j;
for (j = 1;j < month;j++) {
    switch (j) {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:days_2 += 31;break;
        case 4:
        case 6:
        case 9:
        case 11:days_2 += 30;break;
        case 2: if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))
            days_2 += 29;
        else
            days_2 += 28;
            break;
    }
}

//总天数
days = days_1+days_2+day;

//计算星期数
findday=(int)days%7;

//0:Sunday
```

```
    //1:Monday
    //2:Tuesday
    //3:Wednesday
    //4:Thursday
    //5:Friday
    //6:Saturday
    return findday;
}

public int judgeValid(double year,double month,double day)
{
    //年月日是否越界
    if(year<=0)
    {
        return -1;
    }
    if(month<=0||month>=13)
    {
        return -1;
    }
    if(day<=0||day>=32)
    {
        return -1;
    }

    //年月日是否为整数
    if((int)year!=year)
    {
        return -1;
    }
    if((int)month!=month)
    {
        return -1;
    }
    if((int)day!=day)
    {
        return -1;
    }
    return 0;
}
```

2.2控制流图


(year, month, day)
leapYear=0;
noLeapYear=0
days_1=0;
days_2=0;
days=0;
findday=0;
1
2
inputValid!=-1
inputValid==-1
inputValid=judgeValid
(year, month, day);
return -1
3
4
i=1
5
i>=year
i<year
6
(i % 4 == 0 && i % 100 != 0) || (i % 400 == 0)
! [(i % 4 == 0 && i % 100 != 0) || (i % 400 == 0)]
7
8
leapYear++
noLeapyYear++
9
i++
10
days_1= leapYear*366
+noLeapYear*365
j=1
11
j>=month
j<month
switch(j)
12
case1
case3
case5
case7
case8
case10
case4
case6
case9
case11
case2
13
14
15
16
17
18
19
20
21
22
days_2+=31
break
days_2+=30
break
23
(year % 4 == 0
&& year %
100 != 0) || (year
% 400
== 0)
![(year % 4 == 0
&& year %
100 != 0) || (year
% 400
== 0)]
24
25
days_2+=29
break
days_2+=28
break
26
j++
days = days_1+days_2+day;
findday=(int)days%7;
return findday;
27

3. 黑盒测试用例设计

3.1等价类

考虑到该问题年月日都有较为明确的划分区间，故采用等价类方法能够清晰描述测试问题。

考虑到划分出来的无效等价类过多，若采取强一般和强健壮等价类测试将占用非常多的测试用例和篇幅，但是为了体现测试的普遍性以及检验我的学习成果，在下面我还是划分了弱一般、弱健壮两种等价类测试方法。

等价类划分表：

输入条件	有效等价类	编号	无效等价类	编号
Year	Year 为闰年	1		
	Year 不为闰年	2		
	Year 为正整数	3	Year 为非正数	13
			Year 为非整数	14
Month	Month=1, 3, 5, 7, 8, 10	4	Month<1	15
	Month=4, 6, 9, 11	5	Month>12	16
	Month = 2	6	Month 为非整数	17
	Month = 12	7		
Day	1<=day<=27	8	Day<1	18
	Day=28	9	Day 为非整数	19
	Day=29	10		
	Day=30	11		
	Day=31	12	Day>31	20

测试用例（弱一般等价类测试）：

测试用例编号	输入数据			预期输出	覆盖等价类
	Year	Month	Day		
1.	1999	1	13	3	2,4,8
2.	1998	4	28	2	2,5,9
3.	2000	2	29	2	1,6,10
4.	2001	12	30	0	2,7,11
5.	2017	3	31	5	2,4,12

测试用例（弱健壮等价类测试）：

测试用例编号	输入数据			预期输出	覆盖等价类
	Year	Month	Day		
1.	1999	1	13	3	2,4,8
2.	1998	4	28	2	2,5,9
3.	2000	2	29	2	1,6,10
4.	2001	12	30	0	2,7,11
5.	2017	3	31	5	2,4,12
6.	-1	1	13	-1	13
7.	2016.1	1	13	-1	14
8.	1999	0	13	-1	15
9.	1999	13	13	-1	16
10.	1999	1.1	13	-1	17
11.	1999	1	0	-1	18
12.	1999	1	1.1	-1	19
13.	1999	11	32	-1	20

4. 白盒测试用例设计

4.1 基路径测试

经统计边数为40，节点数为27，只有1个连通图，计算可得基路径条数为15条

序号	路径
1.	1,2,3
2.	1,2,4,5,10,11,27
3.	1,2,4,5,10,11,12,13,14,15,16,17,18,26,11,27
4.	1,2,4,5,10,11,12,23,25,26,11,12,13,14,15,16,17,18,26,11,27
5.	1,2,4,5,10,11,12,14,15,16,17,18,26,11,12,23,25,26,11,12,13,14,15,16,17,18,26,11,27

6.	1,2,4,5,10,11,12,19,20,21,22,26,11,12,14,15,16,17,18,26,11,12,23,25,26,11,12,13,14,15,16,17,18,26,11,27
7.	1,2,4,5,10,11,12,15,16,17,18,26,11,12,19,20,21,22,26,11,12,14,15,16,17,18,26,11,12,23,25,26,11,12,13,14,15,16,17,18,26,11,27
8.	1,2,4,5,6,8,9,5,10,11,27
9.	1,2,4,5,6,8,9,5,6,8,9,5,10,11,27
10	1,2,4,5,6,8,9,5,6,8,9,5,6,8,9,5,10,11,27
11	1,2,4,5,6,7,9,5,6,8,9,5,6,8,9,5,6,8,9,5,10,11,27
12	1,2,4,5,6,8,9,5,6,7,9,5,6,8,9,5,6,8,9,5,6,8,9,5,10,11,27
13	1,2,4,5,6,8,9,5,6,8,9,5,6,8,9,5,10,11,12,13,14,15,16,17,18,26,11,27
14	1,2,4,5,6,8,9,5,6,8,9,5,6,8,9,5,10,11,12,23,24,26,11,12,13,14,15,16,17,18,26,11,27
15	1,2,4,5,6,8,9,5,6,8,9,5,6,8,9,5,10,11,12,14,15,16,17,18,26,11,12,23,24,26,11,12,13,14,15,16,17,18,26,11,27

基路径覆盖测试用例如下：

测试用例编号	输入数据			预期输出	覆盖路径
	Year	Month	Day		
1.	1.1	1	1	-1	1
2.	1	1	1	1	2
3.	1	2	1	4	3
4.	1	3	1	4	4
5.	1	4	1	0	5
6.	1	5	1	2	6
7.	1	6	1	5	7
8.	2	1	1	2	8
9.	3	1	1	3	9

10.	4	1	1	4	10
11.	5	1	1	6	11
12.	6	1	10	2	12
13.	4	2	1	0	13
14.	4	3	1	1	14
15.	4	4	1	4	15

4.2 Prime路径测试

根据Simple Path迭代得到的Prime路径如下所示：

Simple Path路径长度	路径（忽略的容易从图中直接观察到）
0	易从图中得
1	易从图中得
2	[1,2,3]
4	[5,6,7,9,5],[5,6,8,9,5],[11,12,18,26,11],[11,12,22,26,11]
5	[11,12,17,18,26,11],[11,12,21,22,26,11],[11,12,23,24,26,11], [11,12,23,25,26,11]
6	[1,2,4,5,10,11,27],[11,12,16,17,18,26,11] [11,12,20,21,22,26,11]
7	[11,12,15,16,17,18,26,11],[11,12,19,20,21,22,26,11]
8	[11,12,14,15,16,17,18,26,11]

9	[11,12,13,14,15,16,17,18,26,11]

序号	Prime路径
1	[1,2,3]
2	[5,6,7,9,5]
3	[5,6,8,9,5]
4	[11,12,18,26,11]
5	[11,12,22,26,11]
6	[11,12,17,18,26,11]
7	[11,12,21,22,26,11]
8	[11,12,23,24,26,11]
9	[11,12,23,25,26,11]
10	[1,2,4,5,10,11,27]
11	[11,12,16,17,18,26,11]
12	[11,12,20,21,22,26,11]
13	[11,12,15,16,17,18,26,11]
14	[11,12,19,20,21,22,26,11]
15	[11,12,14,15,16,17,18,26,11]
16	[11,12,13,14,15,16,17,18,26,11]

Prime路径覆盖测试用例如下：

测试用例编号	输入数据			预期输出	覆盖路径
	Year	Month	Day		
1	1.1	1	1	-1	1
2	1	12	1	6	4,5,6,7,9,10,11,12,13,14,15,16
3	5	1	1	6	2,3,10
4	4	12	1	3	3,4,5,6,7,8,10,11,12,13,14,15,16

4.3 数据流测试

考虑到整张控制流图有26个结点，对所有结点进行数据流测试手算工作量过大，故仅对1，2，3，4，5，6，7，8，9，10，11，27几个结点开展数据流测试。

结点的Def&Use：

Node	Def	Use
1	year,month,day,leapYear,noLeapYear, days_1,days_2,days,findday	year,month,day
2	inputValid	year,month,day
3		
4	i	
5		
6		
7	leapYear	leapYear

8	noLeapYear	noLeapYear
9	i	i
10	days_1, j	leapYear,noLeapYear
11		
27	days,findday	days_1,days_2,day,days, findday

边的Def&Use:

Edge	Use
(1,2)	
(2,3)	inputValid
(2,4)	inputValid
(4,5)	
(5,6)	i, year
(6,7)	i
(6,8)	i
(7,9)	
(8,9)	
(9,5)	
(5,10)	i, year

(10,11)	
(11,27)	j,month

DU对：

变量	DU Pairs
year	（1，2）
month	（1，2）
day	（1，2）
leapYear	(1, 7), (7, 10)
noLeapYear	(1, 8), (8, 10)
days_1	(10, 27)
days_2	(1, 27)
days	(1, 27)
findday	(1, 27)
i	(4, (5, 6))
j	(10, (11, 27))
inputValid	(2, (2, 3))

DU Paths：

序号	DU Paths

1	[1,2]
2	[1,2,4,5,6,7]
3	[7,9,5,10]
4	[1,2,4,5,6,8]
5	[8,9,5,10]
6	[10,11,27]
7	[1,2,4,5,10,11,27]
8	[4,5,6]
9	[10,11,27]
10	[2,3]

数据流测试用例如下：

测试用例编号	输入数据			预期输出	覆盖路径
	Year	Month	Day		
1	1.1	1	1	-1	1,10
2	5	1	1	6	1,2,3,4,5,6,7,8,9

5. 基于Junit的单元测试实践

5.1 弱一般等价类测试

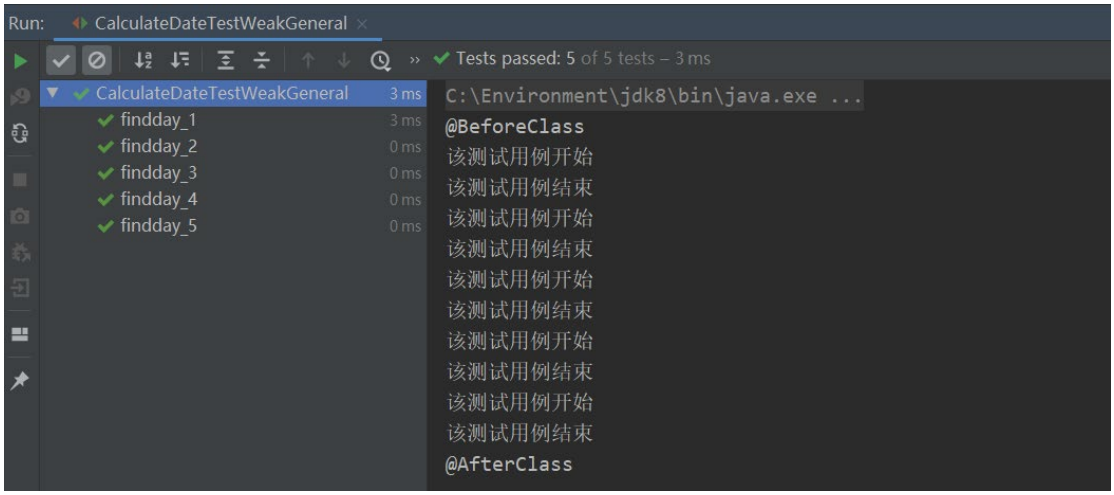
弱一般等价类测试采用了Junit提供的@Test注释，针对上面的弱一般等价类测试表设置了测试用例。

核心代码：

```
//弱一般等价类测试

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

public class CalculateDateTestWeakGeneral {

    private static CalculateDate calDate = new CalculateDate();
    @BeforeClass
    public static void setupBeforClass() throws Exception{
        System.out.println("@BeforeClass");
    }
    @AfterClass
    public static void teardownAfterClass() throws Exception{
        System.out.println("@AfterClass");
    }
    @Before
    public void setUp() throws Exception {
        System.out.println("该测试用例开始");
    }
    @After
    public void tearDown() throws Exception {
        System.out.println("该测试用例结束");
    }
    @Test
    public void findday_1() {
        calDate.findDay(1999,1,13);
        assertEquals(3,calDate.getFindday());
    }
    @Test
    public void findday_2() {
        calDate.findDay(1998,4,28);
        assertEquals(2,calDate.getFindday());
    }
    @Test
    public void findday_3() {
        calDate.findDay(2000,2,29);
        assertEquals(2,calDate.getFindday());
    }
    @Test
    public void findday_4() {
        calDate.findDay(2001,12,30);
        assertEquals(0,calDate.getFindday());
    }
    @Test
```

```
    public void findday_5() {
        calDate.findDay(2017,3,31);
        assertEquals(5,calDate.getFindday());
    }
}
```

执行截图：



5.2 弱健壮等价类测试

弱健壮等价类测试采用了TestSuite套件测试，以addTestSuite方法添加了扩展了TestCase类的JunitTestCaseDemo类。测试方法需要用“test***”的形式进行书写。

核心代码：

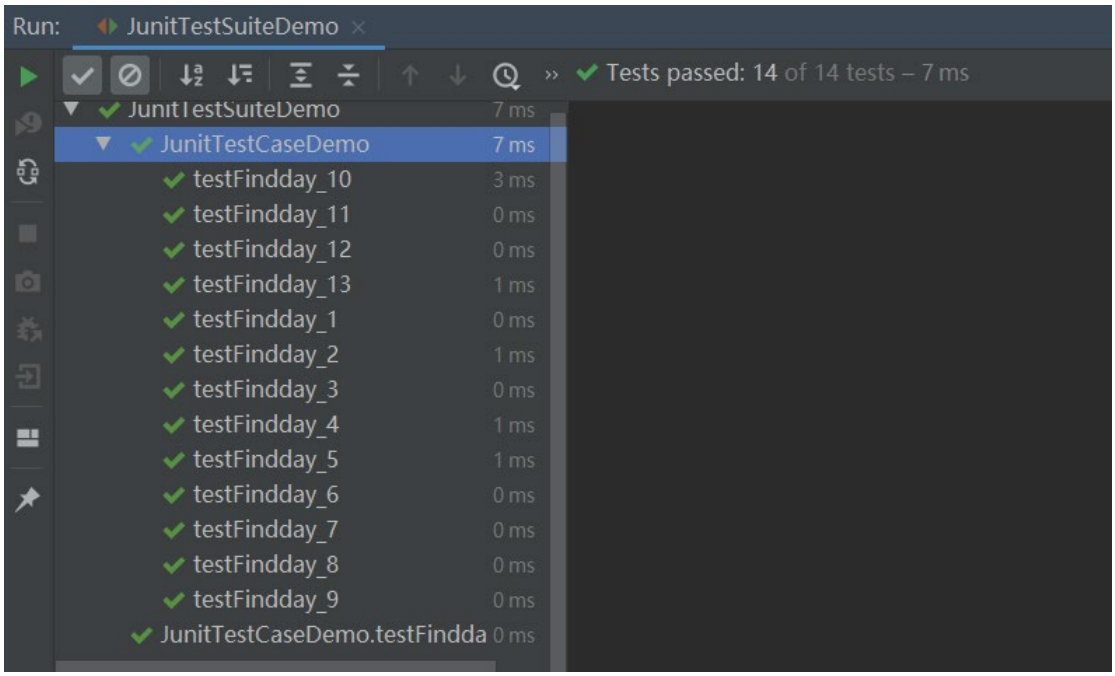
```
//弱健壮等价类测试

import junit.framework.TestCase;

public class JunitTestCaseDemo extends TestCase {

    public JunitTestCaseDemo(String name)
    {
        super(name);
    }

    public void testFindday_1() {
        CalculateDate2 calDate= new CalculateDate2();
```

```
        int day= calDate.findDay(1999,1,13);
        assertEquals(3,day);
    }

    public void  testFindday_2() {
        CalculateDate2 calDate= new CalculateDate2();
        int day= calDate.findDay(1998,4,28);
        assertEquals(2,day);
    }

    public void  testFindday_3() {
        CalculateDate2 calDate= new CalculateDate2();
        int day= calDate.findDay(2000,2,29);
        assertEquals(2,day);
    }

    public void  testFindday_4() {
        CalculateDate2 calDate= new CalculateDate2();
        int day= calDate.findDay(2001,12,30);
        assertEquals(0,day);
    }

    public void  testFindday_5() {
        CalculateDate2 calDate= new CalculateDate2();
        int day= calDate.findDay(2017,3,31);
        assertEquals(5,day);
    }

    public void  testFindday_6() {
        CalculateDate2 calDate= new CalculateDate2();
        int day= calDate.findDay(-1,1,13);
        assertEquals(-1,day);
    }

    public void  testFindday_7() {
        CalculateDate2 calDate= new CalculateDate2();
        int day= calDate.findDay(2016.1,1,13);
        assertEquals(-1,day);
    }

    public void  testFindday_8() {
        CalculateDate2 calDate= new CalculateDate2();
        int day= calDate.findDay(1999,0,13);
        assertEquals(-1,day);
```

```
    }

    public void  testFindday_9() {
        CalculateDate2 calDate= new CalculateDate2();
        int day= calDate.findDay(1999,13,13);
        assertEquals(-1,day);
    }

    public void  testFindday_10() {
        CalculateDate2 calDate= new CalculateDate2();
        int day= calDate.findDay(1999,1.1,13);
        assertEquals(-1,day);
    }

    public void  testFindday_11() {
        CalculateDate2 calDate= new CalculateDate2();
        int day= calDate.findDay(1999,1,0);
        assertEquals(-1,day);
    }

    public void  testFindday_12() {
        CalculateDate2 calDate= new CalculateDate2();
        int day= calDate.findDay(1999,1,1.1);
        assertEquals(-1,day);
    }

    public void  testFindday_13() {
        CalculateDate2 calDate= new CalculateDate2();
        int day= calDate.findDay(1999,11,32);
        assertEquals(-1,day);
    }
}

import junit.framework.*;

public class JunitTestSuiteDemo {
    public static TestSuite suite(){
        // Create TestSuite Class testSuite
        TestSuite testSuite = new TestSuite("WeakStrong TestSuite");

        // Add test set to testSuite
        testSuite.addTestSuite(JunitTestCaseDemo.class);

        // Add single test to testSuite
        Test singleTest =
```

```
TestSuite.createTest(JunitTestCaseDemo.class,"testFindday_13");
        testSuite.addTest(singleTest);

        return testSuite;
    }
}
```

执行截图：



6. 基于TestNG的单元测试实践

主要应用了数据驱动的TestNG测试方法，主要是训练使用csv文件中的测试数据驱动进行驱动测试。

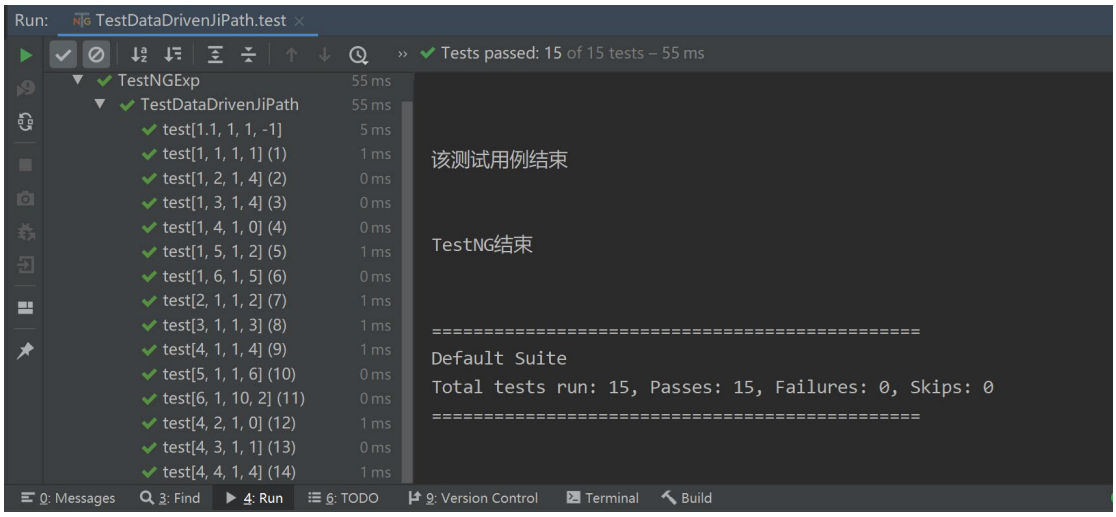
6.1 基路径测试

核心代码：

```
//基路径测试

import org.testng.annotations.*;
import util.CsvUtil;
import java.io.IOException;
```

```
import static org.junit.Assert.assertEquals;

public class TestDataDrivenJiPath {
    public static String filePath = "src/main/resources/JiTest.csv";//文件相对路径

    @DataProvider(name = "testData")
    public static Object[][] words() throws IOException{
        return CsvUtil.getTestData(filePath);
    }
    @Test(dataProvider = "testData")
    public void test(String year, String month, String day, String findDay)
    {
        CalculateDate2 calDate = new CalculateDate2();
        int result=
calDate.findDay(Double.parseDouble(year.trim()),Double.parseDouble(month.trim()),Do
uble.parseDouble(day.trim()));
        int a =Integer.parseInt(findDay.trim());
        assertEquals(a,result);
    }

    @BeforeTest
    public void start() throws Exception {
        System.out.println("TestNG 开始");
    }
    @AfterTest
    public void end() throws Exception {
        System.out.println("TestNG 结束");
    }
    @BeforeMethod
    public void setUp() throws Exception {
        System.out.println("该测试用例开始");
    }
    @AfterMethod
    public void tearDown() throws Exception {
        System.out.println("该测试用例结束");
    }
}
```

执行截图：



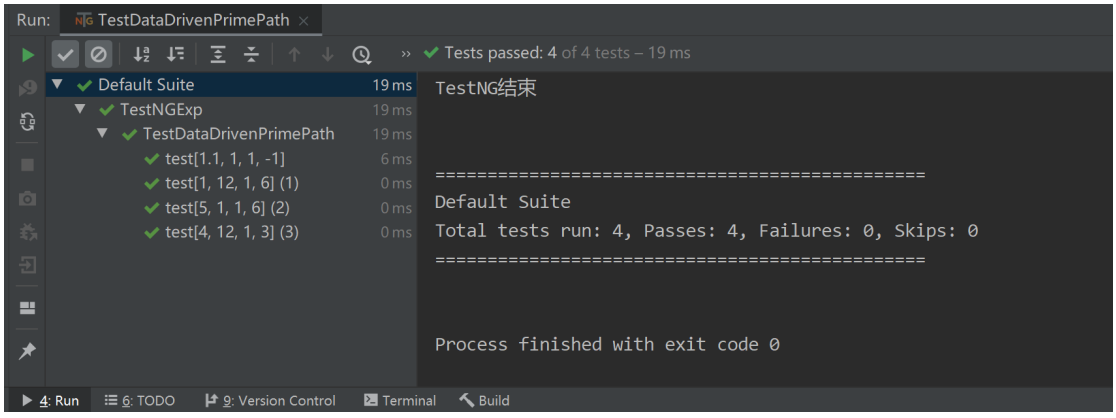
6.2 Prime路径测试

核心代码：

```
//Prime 路径测试

import org.testng.annotations.*;
import util.CsvUtil;
import static org.junit.Assert.*;
import java.io.IOException;

public class TestDataDrivenPrimePath {
    public static String filePath = "src/main/resources/PrimeTest.csv";//文件相对路
径

    @DataProvider(name = "testData")
    public static Object[][] words() throws IOException {
        return CsvUtil.getTestData(filePath);
    }

    @Test(dataProvider = "testData")
    public void test(String year, String month, String day, String findDay)
    {
        CalculateDate2 calDate = new CalculateDate2();
        int result=
calDate.findDay(Double.parseDouble(year.trim()),Double.parseDouble(month.trim()),Do
uble.parseDouble(day.trim()));
        int a =Integer.parseInt(findDay.trim());
        assertEquals(a,result);
```

```
    }


    @BeforeTest
    public void start() throws Exception {
        System.out.println("TestNG 开始");
    }
    @AfterTest
    public void end() throws Exception {
        System.out.println("TestNG 结束");
    }
    @BeforeMethod
    public void setUp() throws Exception {
        System.out.println("该测试用例开始");
    }
    @AfterMethod
    public void tearDown() throws Exception {
        System.out.println("该测试用例结束");
    }
}
```

执行截图：



6.3 数据流测试

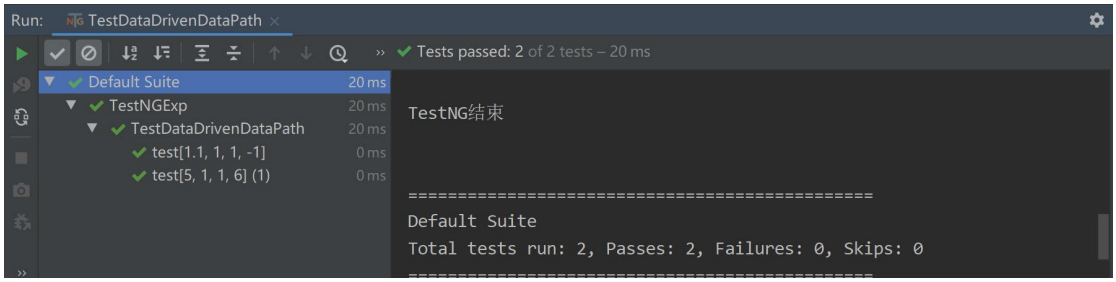
核心代码：

```
//数据流测试
```

```
import org.testng.annotations.*;
import util.CsvUtil;
import java.io.IOException;
import static org.junit.Assert.*;
public class TestDataDrivenDataPath {
    public static String filePath = "src/main/resources/DataTest.csv";//文件相对路径

    @DataProvider(name = "testData")
    public static Object[][] words() throws IOException{
        return CsvUtil.getTestData(filePath);
    }

//    @Test(dataProvider = "testData")
//    public void test(String year, String month, String day, String findDay)
//    {
//        CalculateDate2 calDate = new CalculateDate2();
//        int result=
calDate.findDay(Double.parseDouble(year.trim()),Double.parseDouble(month.trim()),Do
uble.parseDouble(day.trim()));
//        int a =Integer.parseInt(findDay.trim());
//        assertEquals(a,result);
//    }

    @Test(dataProvider = "testData")
    public void test(String year, String month, String day)
    {
        CalculateDate2 calDate = new CalculateDate2();
        int result=
calDate.findDay(Double.parseDouble(year.trim()),Double.parseDouble(month.trim()),Do
uble.parseDouble(day.trim()));
        System.out.println(result);
    }

    @BeforeTest
    public void start() throws Exception {
        System.out.println("TestNG 开始");
    }
    @AfterTest
    public void end() throws Exception {
        System.out.println("TestNG 结束");
    }
    @BeforeMethod
    public void setUp() throws Exception {
        System.out.println("该测试用例开始");
```

```
    }
    @AfterMethod
    public void tearDown() throws Exception {
        System.out.println("该测试用例结束");
    }
}
```

执行截图：



7. 测试体会

本次实验是软件测试的一次重要实践，让我又一次增强了自己的动手能力。本次实验综合运用了黑盒测试、白盒测试的各种测试方法，对我写的代码进行了验证。

在黑盒测试中，我选用了等价类测试的方法，因为考虑到我“日期输入判断星期几”的题目，对于年月日的输入范围已经有了明确的界限，所以用等价类测试方法能在保证各种有无缺陷、单缺陷、多缺陷的情况下有良好的检测效果。在我的第一个版本中，弱一般等价类测试用例能够全部通过，但是当输入的日期超界或者为小数的时候，测试用例就无法通过。因此我对此进行了迭代改进，在第二个版本中首先对输入进行有效性判断，尝试对无效类进行一部分的处理。从中我可以知道，在编写代码的时候，不仅应该对有效类进行处理，而且还需要保证无效类的特殊处理情况。

在白盒测试中，我综合运用了基路径测试、Prime 路径测试、数据流测试等多种方法。这一切的前提都是做出一张好的控制流图，因此我运用画图软件，成功美观地将代码转化为了 DFG 图，其中 if 条件判断、循环、switch 语句都进行了很好地转化，最终转化出了一张 27 个节点的复杂度非常高的图。因此，

我在接下来的测试中需要十分耐心仔细地找好路径，以防止不出错。此外，我还在此基础上进行了 Def,Use 的标注，以此来产生 DU 对，最后也很好地测试了数据流。

通过这次实验，我对于书本知识的理解又更上了一层楼，可以说是收获颇丰的。