实验一：开发者测试

一、实验目的

理解和掌握使用Eclemma测试工具对Java语言编写的程序进行语句覆盖测试

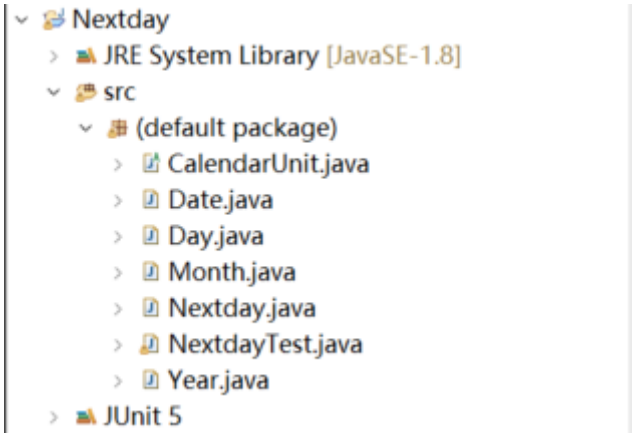
二、实验环境

Eclipse、Windows11、JDK1.8

三、实验内容

①安装配置eclipse：在之前的学习中已经配置完成。

②创建项目Nextday



③编写测试代码（详见NextdayTest.java）

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

import org.junit.Test;

import org.hamcrest.CoreMatchers;
import org.junit.Assert;
import org.junit.Ignore;
import org.junit.Test;

import static org.junit.Assert.assertThat;
import static org.junit.Assert.fail;

/**
 *
 * @author 楚逸飞
 * @version 1.0
 */
public class NextdayTest {
```

```
/*
 * MONTH_EXCEPTION will be thrown when the month is not valid
 */
private final String MONTH_EXCEPTION = "Not a valid month";

/*
 * DAY_EXCEPTION will be thrown when the day is not valid
 */
private final String DAY_EXCEPTION = "Not a valid day";

/*
 * YEAR_EXCEPTION will be thrown when the year is not valid
 */
private final String YEAR_EXCEPTION = "Not a valid year";

/*
 * 等价类测试（有效类）
 */
@Test(timeout = 1000)
public void firstTest() {

    Date date; //初始日期
    Date fact; //实际执行结果
    Date expected; //所期望的正确结果

    /*
     * common
     */
    date = new Date(12, 5, 2021);
    fact = Nextday.nextDay(date);
    fact.printDate();
    expected = new Date(12, 6, 2021);
    Assert.assertEquals(fact, expected);

    /*
     * 测试month 31天的月份
     */
    date = new Date(1, 31, 2021);
    fact = Nextday.nextDay(date);
    fact.printDate();
    expected = new Date(2, 1, 2021);
    Assert.assertEquals(fact, expected);

    /*
     * 测试month 30天的月份
     */
    date = new Date(4, 30, 2021);
    fact = Nextday.nextDay(date);
    fact.printDate();
    expected = new Date(5, 1, 2021);
    Assert.assertEquals(fact, expected);

    /*
     * year测试
     */
    date = new Date(12, 31, 2020);
    fact = Nextday.nextDay(date);
    fact.printDate();
```

```
        expected = new Date(1, 1, 2021);
        Assert.assertEquals(expected, fact);

        /*
         * 特殊年份 公元前后
         */
        date = new Date(12, 31, -1);
        fact = Nextday.nextDay(date);
        fact.printDate();
        expected = new Date(1, 1, 1);
        Assert.assertEquals(expected, fact);

        /*
         * 闰年 测试三种情况 主要是的是4的倍数但不是100的倍数或者是400的倍数 测试2000 2020
         */
        date = new Date(2, 29, 2020);
        fact = Nextday.nextDay(date);
        fact.printDate();
        expected = new Date(3, 1, 2020);
        Assert.assertEquals(expected, fact);

        date = new Date(2, 29, 2000);
        fact = Nextday.nextDay(date);
        fact.printDate();
        expected = new Date(3, 1, 2000);
        Assert.assertEquals(expected, fact);


        /*
         * 非闰年
         */
        date = new Date(2, 28, 2021);
        fact = Nextday.nextDay(date);
        fact.printDate();
        expected = new Date(3, 1, 2021);
        Assert.assertEquals(expected, fact);

        String s = date.toString();

        Year year = new Year(2021);
        year.equals(null);
        Month month = new Month(12, year);
        month.equals(null);
        Day day = new Day(5, month);
        day.equals(null);
        Nextday nextDay = new Nextday();

        date.equals(null);
    }

    /*
     * 等价类测试（无效类）
     */
    @Test
    public void secondTest() {
        Object[][] testArr = {
                //month day year
                {3, 1, 0, YEAR_EXCEPTION},//没有公元0年
```

```
                {-1, 1, 2021, MONTH_EXCEPTION},//month 不能为非正整数
                {13, 1, 2021, MONTH_EXCEPTION},//month 不能大于12
                {29, 2, 2100, MONTH_EXCEPTION},//测试特殊年份2100 属于既是4的倍数又是100的倍数 不是
闰年 所以2月没有29日
                {1, -1, 2021, DAY_EXCEPTION},//day 不能为非正整数
                {2, 30, 2020, DAY_EXCEPTION},//2月没有30日
                {2, 29, 2021, DAY_EXCEPTION},//非闰年2月没有29
                {3, 32, 2021, DAY_EXCEPTION},//day不能大于31
                {4, 31, 2021, DAY_EXCEPTION}//小月没有31日
        };
        for (Object[] test : testArr) {
            try {
                Date date = new Date((int) test[0], (int) test[1], (int) test[2]);
                Date d = Nextday.nextDay(date);
                fail();
            } catch (IllegalArgumentException e) {
                assertThat(e.getMessage(), CoreMatchers.containsString((String) test[3]));
            }
        }
    }
}
```

④进行覆盖率测试

```java
//package net.mooctest;
public class Date {
    private Day d;
    private Month m;
    private Year y;

    public Date(int pMonth, int pDay, int pYear) {
        y = new Year(pYear);
        m = new Month(pMonth, y);
        d = new Day(pDay, m);
    }

    public void increment() {
        if (!d.increment()) {
            if (!m.increment()) {
                y.increment();
                m.setMonth(1, y);
            }
            d.setDay(1, m);
        }
    }

    public void printDate() {
        System.out.println(m.getMonth() + "/" + d.getDay() + "/" + y.getYear());
    }

    public Day getDay() {
        return d;
    }

    public Month getMonth() {
        return m;
    }

    public Year getYear() {
        return y;
    }

    public boolean equals(Object o) {
        if (o instanceof Date) {
            if (this.y.equals(((Date) o).y) && this.m.equals(((Date) o).m)
                    && this.d.equals(((Date) o).d))
                return true;
        }
        return false;
    }

    public String toString() {
        return (m.getMonth() + "/" + d.getDay() + "/" + y.getYear());
    }
}

```

```java
//package net.mooctest;
public class Day extends CalendarUnit {
    private Month m;

    public Day(int pDay, Month m) {
        setDay(pDay, m);
    }

    public boolean increment() {
        currentPos += 1;
        if (currentPos <= m.getMonthSize())
            return true;
        else
            return false;
    }

    public void setDay(int pDay, Month m) {
        setCurrentPos(pDay);
        this.m = m;
        if (!this.isValid()) {
            throw new IllegalArgumentException("Not a valid day");
        }
    }

    public int getDay() {
        return currentPos;
    }

    public boolean isValid() {
        if (m != null && m.isValid())
            if (this.currentPos >= 1 && this.currentPos <= m.getMonthSize())
                return true;
        return false;

    }

    public boolean equals(Object o) {
        if (o instanceof Day) {
            if (this.currentPos == ((Day) o).currentPos
                    && this.m.equals(((Day) o).m))
                return true;
        }
        return false;
    }
}

```

```java
//package net.mooctest;
public class Month extends CalendarUnit {
    private Year y;
    private int[] sizeIndex = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

    public Month(int pMonth, Year y) {
        setMonth(pMonth, y);
    }

    public void setMonth(int pMonth, Year y) {
        setCurrentPos(pMonth);
        this.y = y;
        if (!this.isValid()) {
            throw new IllegalArgumentException("Not a valid month");
        }
    }

    public int getMonth() {
        return currentPos;
    }

    public int getMonthSize() {
        if (y.isLeap())
            sizeIndex[1] = 29;
        else
            sizeIndex[1] = 28;
        return sizeIndex[currentPos - 1];
    }

    public boolean increment() {
        currentPos += 1;
        if (currentPos > 12)
            return false;
        else
            return true;
    }

    public boolean isValid() {
        if (y != null && y.isValid())
            if (this.currentPos >= 1 && this.currentPos <= 12)
                return true;
        return false;

    }

    public boolean equals(Object o) {
        if (o instanceof Month) {
            if (this.currentPos == ((Month) o).currentPos
                    && this.y.equals(((Month) o).y))
                return true;
        }
        return false;
    }
}

```

```java
//package net.mooctest;
public class Year extends CalendarUnit {
    public Year(int pYear) {
        setYear(pYear);
    }

    public void setYear(int pYear) {
        setCurrentPos(pYear);
        if (!this.isValid()) {
            throw new IllegalArgumentException("Not a valid year");
        }
    }

    public int getYear() {
        return currentPos;
    }

    public boolean increment() {//年份增加，但是并不允许0的年份出现
        currentPos = currentPos + 1;
        if (currentPos == 0)
            currentPos = 1;
        return true;
    }

    public boolean isLeap() {//判断闰年
        if (currentPos >= 0
                && (((currentPos % 4 == 0) && (currentPos % 100 != 0)) || (currentPos % 400 == 0)))
            return true;
        else if (currentPos < 0
                && ((((currentPos * -1) % 4 == 1) && ((currentPos * -1) % 100 != 1)) || ((currentPos * -1) % 400 == 1)))
            return true;
        return false;
    }

    protected boolean isValid() {//判断年份是否有效
        if (this.currentPos != 0)
            return true;
        return false;
    }

    public boolean equals(Object o) {//判断是否为同一年
        if (o instanceof Year) {
            if (this.currentPos == ((Year) o).currentPos)
                return true;
        }
        return false;
    }
}
```

Coverage ×

NextdayTest (2022年11月9日 上午11:34:11)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Nextday	97.6 %	976	24	1,000
src	97.6 %	976	24	1,000
(default package)	97.6 %	976	24	1,000
Year.java	100.0 %	106	0	106
NextdayTest.java	95.2 %	474	24	498
Nextday.java	100.0 %	20	0	20
Month.java	100.0 %	153	0	153
Day.java	100.0 %	82	0	82
Date.java	100.0 %	131	0	131
CalendarUnit.java	100.0 %	10	0	10

四、实验分析与总结

（eclipse的测试中会将测试代码本身也算入覆盖率的计算，这导致正常测试时，代码中的catch语句常常无法到达而没法实现完全覆盖）

本次实验中熟悉了Eclemma测试工具的使用。实验中使用了黑盒测试与白盒测试相结合的方法，在黑盒测试中，运用了上课的时候讲到的等价类划分方法以及边界值分析方法。在设计测试用例的时候先根据需求划分等价类，从每个等价类中选取一个或两个转换成对应的测试用例进行测试，随后再使用边界值测试法进行补充。对应于代码中的firstTest函数中是有效等价类；secondTest中是无效等价类。又由于我们可以在看到源码，所以再结合白盒测试进行进一步补充。

这让我认识到测试并不是一项简单的工作，而是一项“复杂且具有创造力，需要智慧的挑战性工作”。测试的时候要充分理解需求，才能更好地进行基于需求的测试（黑盒测试）；也需要有良好的代码基础，才能更好地进行基于代码的测试（白盒测试）。对于程序外部结构、内部逻辑结构的分析要全面。测试用例的设计要尽可能少，但要尽可能覆盖更多的情况。不过，我也充分认识到，即使是一个简单的程序，其输入情况、路径情况都是非常非常多的，更不必说实际的工程项目。