实验一　单元测试实验（8 学时）

一、实验目的

1、掌握单元测试用例的设计
2、熟悉使用 Junit 测试框架进行基于 Java 语言的单元测试
3、掌握基于覆盖的测试用例设计
4、掌握基于独立路径的测试用例设计

二、基本知识

1、单元测试概念
2、Junit 测试框架的使用
3、白盒测试技术，包括基于覆盖的测试用例设计方法和基于独立路径的测试用例设计方法

三、实验环境

①windows 操作系统+IE 浏览器
②Eclipse 集成开发环境+Junit 测试框架

四、实验内容

1、学习 Junit 框架的使用

可通过以下两个示例进行学习。

A、Junit 使用方法示例 1

1）把 Junit 引入当前项目库中
新建一个 Java 工程—coolJUnit，打开项目 coolJUnit 的属性页 -> 选择“Java Build Path”子选项 -> 点选“Add Library…”按钮 -> 在弹出的“Add Library”对话框中选择 JUnit（图 1），并在下一页中选择版本 Junit 4 后点击“Finish”按钮。这样便把 JUnit 引入到当前项目库中了。

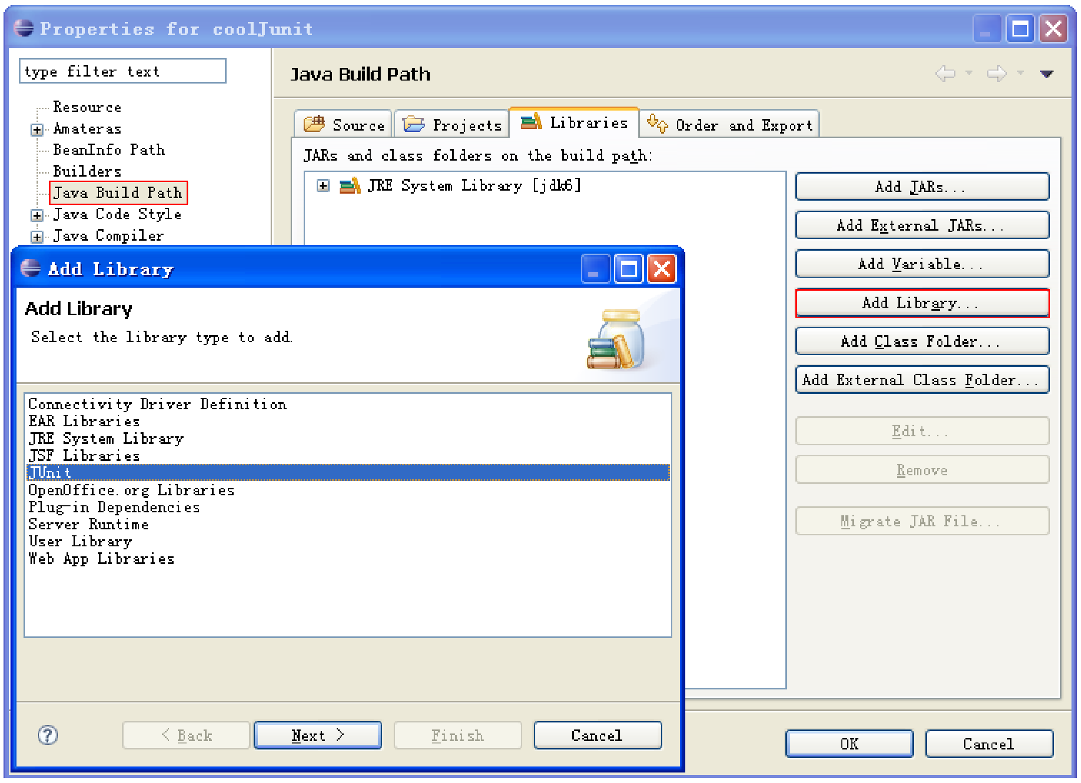


图 1 为项目添加 JUnit 库

2）新建单元测试代码目录

单元测试代码是不会出现在最终软件产品中的，所以最好为单元测试代码与被测试代码创建单独的目录，并保证测试代码和被测试代码使用相同的包名。这样既保证了代码的分离，同时还保证了查找的方便。遵照这条原则，在项目 coolJUnit 根目录下添加一个新目录 testsrc，并把它加入到项目源代码目录中。（见 图 2、3）。

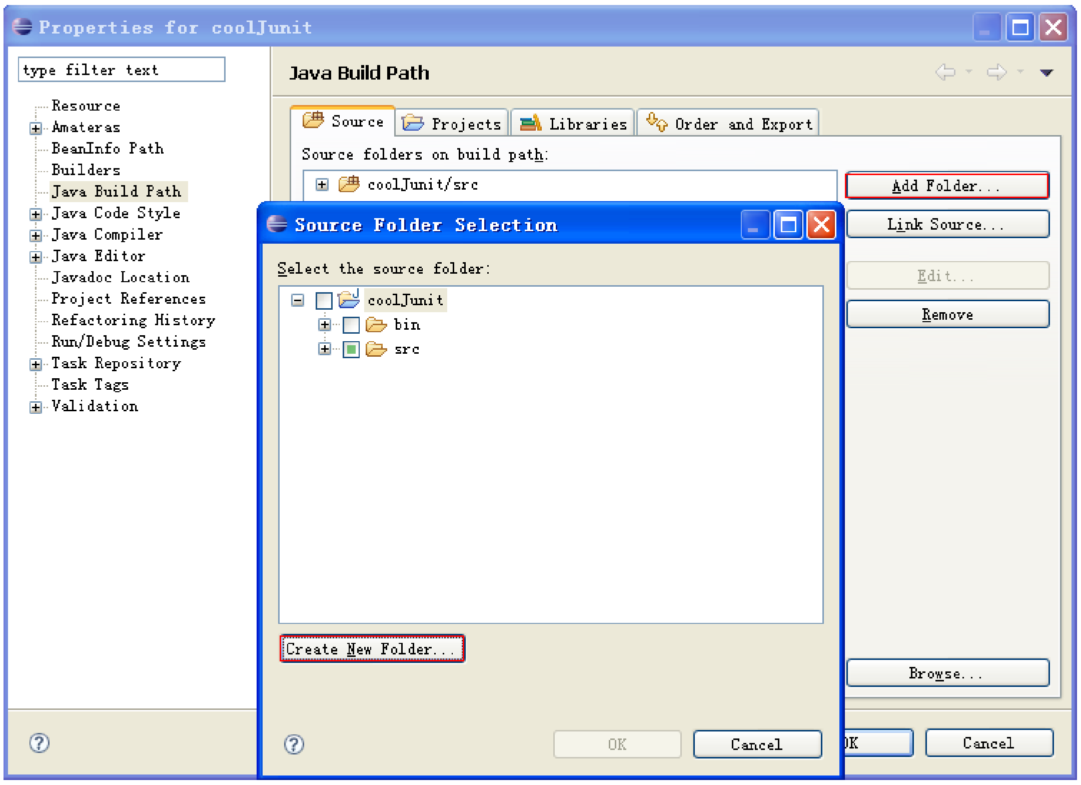


图 2 新建测试代码目录

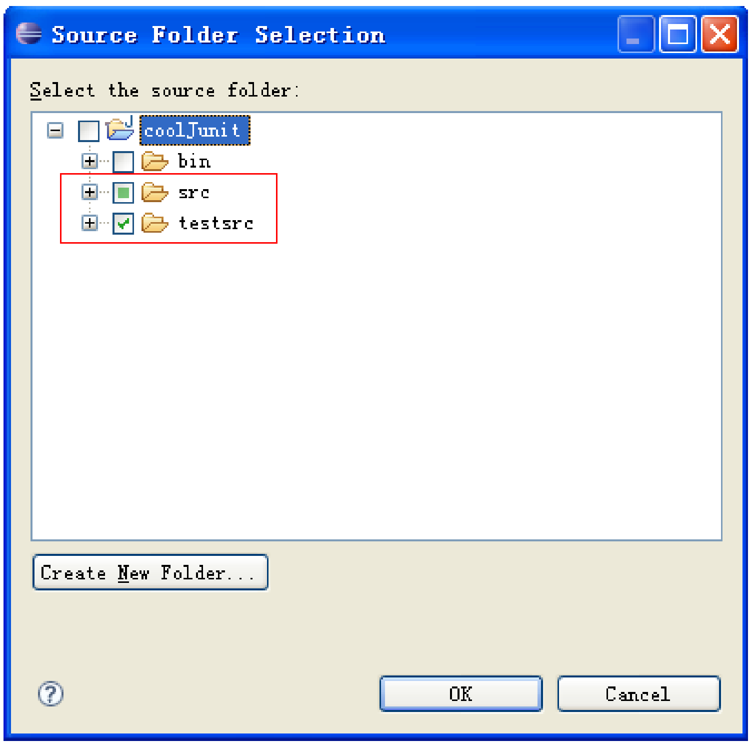


图 3 添加测试代码目录

3）在工程中添加类

添加类 SampleCaculator，类中有两个方法，分别计算加减法。编译代码。

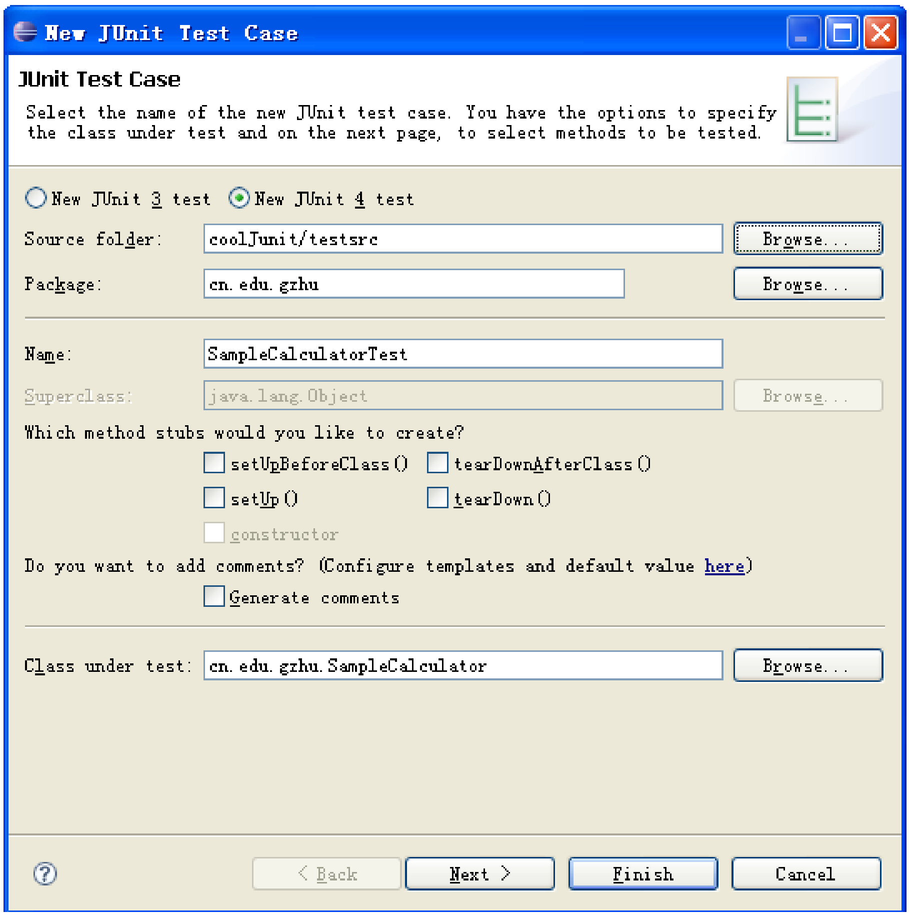
```
package cn.edu.gzhu;

public class SampleCalculator {
    //计算两整数之和
    public int add(int augend, int addend){
        return augend + addend;
    }

    //计算两整数之差
    public int subtration(int minuend, int subtrahend){
        return minuend - subtrahend;
    }
}
```

4）写单元测试代码

为类**SampleCalculator**添加测试用例。在资源管理器**SampleCalculator.java**文件处右击选**new>**选**Junit Test Case**（见图**4**），**Source foler**选择**testsrc**目录，点击**next**，选择要测试的方法，这里把**add**和**subtration**方法都选上，最后点**finish**完成。



JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

( ) New JUnit 3 test [✓ New JUnit 4 test]

Source folder: coolJunit/testsrc

Package: cn.edu.gzhu

Name: SampleCalculatorTest

Superclass: java.lang.Object

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()

☐ setUp() ☐ tearDown()

☐ constructor

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Class under test: cn.edu.gzhu.SampleCalculator

图**4** 新建测试用例

Junit自动生成测试类**SampleCalculatorTest**，修改其中的代码（如下）。

其中**assertEquals**断言，用来测试预期目标和实际结果是否相等。

assertEquals([Sting message], expected, actual)

expected是期望值（通常都是硬编码的），**actual**是被测试代码实际产生的值，**message**是一个可选的消息，如果提供的话，将会在发生错误时报告这个消息。

如想用断言来比较浮点数（在**Java**中是类型为**float**或者**double**的数），则需指定一个额外的误差参数。

assertEquals([Sting message], expected, actual, tolerance)

其它断言参见课本和参考书介绍。测试方法需要按照一定的规范书写：

1. 测试方法必须使用注解 **org.junit.Test** 修饰。

2. 测试方法必须使用 **public void** 修饰，而且不能带有任何参数。

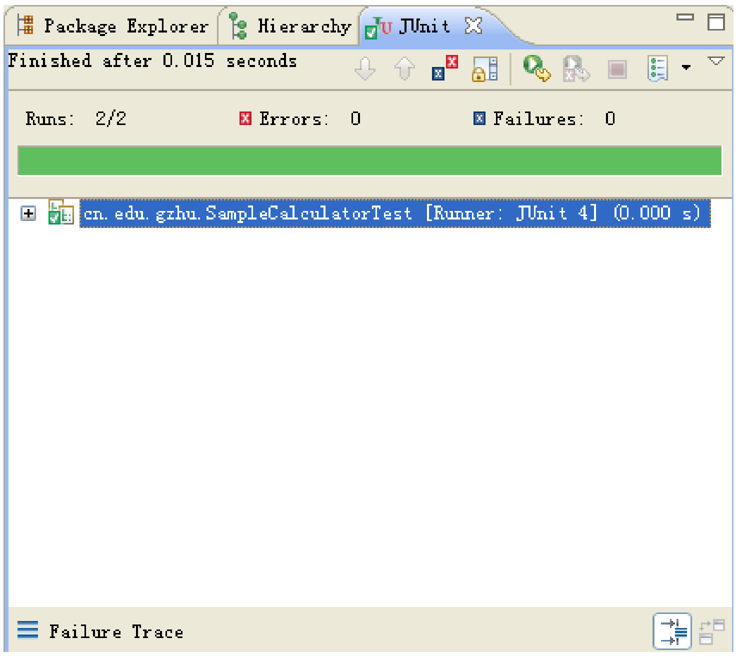
```
package cn.edu.gzhu;
import static org.junit.Assert.*;
import org.junit.Test;

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);
    }

}
```

5）查看运行结果

在测试类上点击右键，在弹出菜单中选择 **Run As JUnit Test**。运行结果如下图，绿色的进度条提示我们，测试运行通过了。



```
Finished after 0.015 seconds
Runs: 2/2        Errors: 0        Failures: 0
cn.edu.gzhu.SampleCalculatorTest [Runner: JUnit 4] (0.000 s)
```

图5 示例1运行结果

B、Junit 使用方法示例 2

1）在工程中添加类

类 WordDealUtil 中的方法 wordFormat4DB( )实现的功能见文件注释。

```
package cn.edu.gzhu;
package cn.edu.gzhu;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class WordDealUtil {
    /**
     * 将Java对象名称（每个单词的头字母大写）按照
     * 数据库命名的习惯进行格式化
     * 格式化后的数据为小写字母，并且使用下划线分割命名单词
     *
     * 例如：employeeInfo 经过格式化之后变为 employee_info
     *
     * @param name Java对象名称
     */
    public static String wordFormat4DB(String name){
        Pattern p = Pattern.compile("[A-Z]");
        Matcher m = p.matcher(name);
        StringBuffer strBuffer = new StringBuffer();

        while(m.find()){
            //将当前匹配子串替换为指定字符串，
            //并且将替换后的子串以及其之前到上次匹配子串之后的字符串段添加到
一个StringBuffer对象里
            m.appendReplacement(strBuffer, "_"+m.group());
        }
        //将最后一次匹配工作后剩余的字符串添加到一个StringBuffer对象里
        return m.appendTail(strBuffer).toString().toLowerCase();
    }

}
```

2）写单元测试代码

```
package cn.edu.gzhu;
import static org.junit.Assert.*;
import org.junit.Test;
public class WordDealUtilTest {

    @Test
    public void testWordFormat4DB() {
        String target = "employeeInfo";
        String result = WordDealUtil.wordFormat4DB(target);

        assertEquals("employee_info", result);
    }


}
```

3）进一步完善测试用例

单元测试的范围要全面，如对边界值、正常值、错误值的测试。运用所学的测试用例的设计方法，如：等价类划分法、边界值分析法，对测试用例进行进一步完善。继续补充一些对特殊情况的测试：

```
    //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 wordFormat4DBBegin(){
        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);
    }
```

4）查看分析运行结果，修改错误代码

再次运行测试。JUnit 运行界面提示我们有两个测试情况未通过测试（见图 6），当首字母大写时得到的处理结果与预期的有偏差，造成测试失败（failure）；而当测试对 null 的处理结果时，则直接抛出了异常——测试错误（error）。显然，被测试代码中并没有对首字母

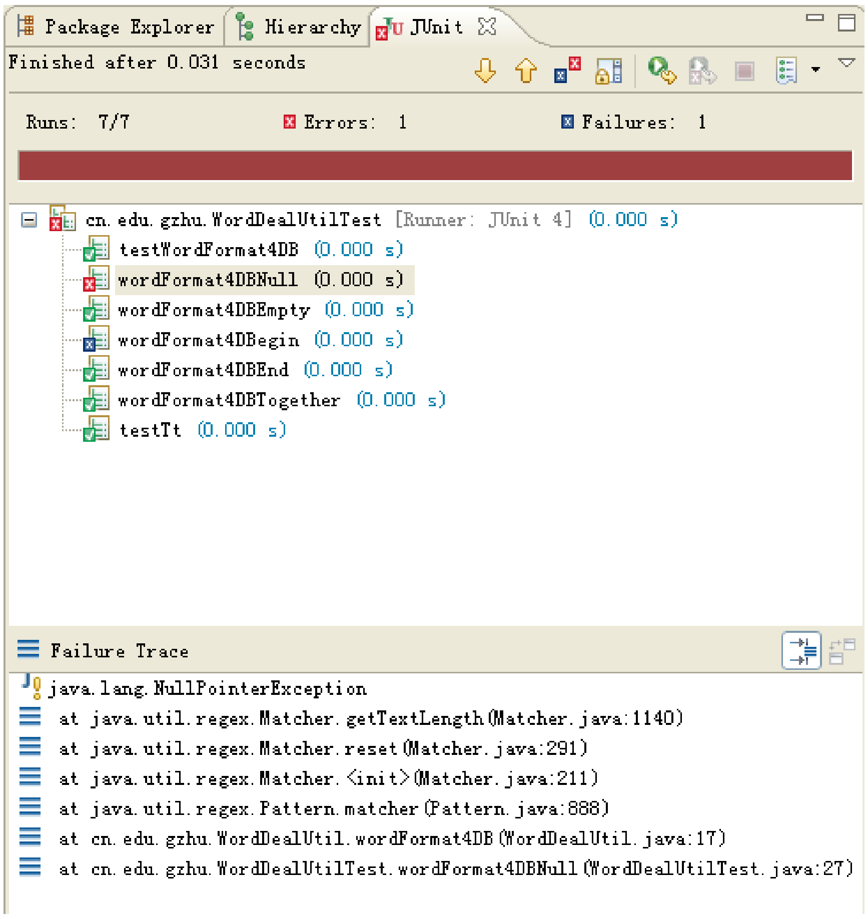
大写和 null 这两种特殊情况进行处理，修改如下：

```
    //修改后的方法 wordFormat4DB
        public static String wordFormat4DB(String name){

            if(name == null){
                return null;
            }

            Pattern p = Pattern.compile("[A-Z]");
            Matcher m = p.matcher(name);
            StringBuffer sb = new StringBuffer();

            while(m.find()){
                if(m.start() != 0)
                    m.appendReplacement(sb, ("_"+m.group()).toLowerCase());
            }
            return m.appendTail(sb).toString().toLowerCase();
        }
```



```
Finished after 0.031 seconds

Runs:  7/7          Errors:  1          Failures:  1

cn.edu.gzhu.WordDealUtilTest [Runner: JUnit 4] (0.000 s)
    testWordFormat4DB (0.000 s)
    wordFormat4DBNull (0.000 s)
    wordFormat4DBEmpty (0.000 s)
    wordFormat4DBBegin (0.000 s)
    wordFormat4DBEnd (0.000 s)
    wordFormat4DBTogether (0.000 s)
    testTt (0.000 s)

java.lang.NullPointerException
  at java.util.regex.Matcher.getTextLength(Matcher.java:1140)
  at java.util.regex.Matcher.reset(Matcher.java:291)
  at java.util.regex.Matcher.<init>(Matcher.java:211)
  at java.util.regex.Pattern.matcher(Pattern.java:888)
  at cn.edu.gzhu.WordDealUtil.wordFormat4DB(WordDealUtil.java:17)
  at cn.edu.gzhu.WordDealUtilTest.wordFormat4DBNull(WordDealUtilTest.java:27)
```

图 6 示例 2 运行结果

2、使用 Junit 框架对类 Date 和类 DateUtil（参见附录）进行单元测试。

只对包含业务逻辑的方法进行测试，包括：
类 Date 中的
isDayValid(int year, int month, int day)
isMonthValid(int month)
isYearValid(int year)

类 DateUtil 中的
isLeapYear(int year)
getDayofYear(Date date)

要求设计合理测试用例，用 Junit 进行测试，分析测试结果，并对错误代码进行修改。

3、使用 Junit 框架对类 Heap_Item 和类 BinaryExponentiation（参见附录）进行单元测试。

只对包含业务逻辑的方法进行测试，包括：
类 Heap_Item 中的
getMultiWayAncestor()
getSonByData(T paCriterion)
itIsLeftSon(Heap_Item<T> paNode)
getData()

类 BinaryExponentiation 中的
calculatePower(long x, long y)
power(long N, long M)

要求设计合理测试用例，用 Junit 进行测试，并使用断言杀死异常。

4、完成以下用例设计

A、基于逻辑覆盖的测试用例设计，程序段如下：

```
void work ( int x, int y, int z){
    int k = 0, j = 0;
    if ((x>3) && (z<10)) {
        k = x*y - 1;
        j = k – z;
    }
    if ((x == 4) || (y>5)) {
        j = x*y + 10;
    }
    j=j % 3;
}
```

步骤 1：将源代码转化为程序流程图。

步骤 2：按照不同的覆盖准则（包括语句覆盖、判定覆盖、条件覆盖、判定-条件覆盖、条件组合覆盖）设计测试用例。

B、基于独立路径的测试用例设计，程序段如下：

```
1 void ReadPara ( Cstring temp)
2 {
3    if (temp == “>=”)
4       m_oper.SetCurSel (0);
5    else
6    {
7       if (temp == “>”)
8          m_oper.SetCurSel (1);
9       else
10      {
11         if (temp == “==”)
12            m_oper.SetCurSel (2);
13         else
14         {
15            if (temp == “<=”)
16               m_oper.SetCurSel (3);
17            else
18               {
19                  if (temp == “<”)
20                     m_oper.SetCurSel (4);
21                  else
22                     m_oper.SetCurSel (5);
23               }
24            }
25         }
26   }
27    return;
28 }
```

步骤 1：画出这段代码的控制流图；
步骤 2：计算环形复杂度；
步骤 3：列出独立路径；
步骤 4：设计测试用例。

五、实验要求

1、做好实验预习，掌握并熟悉本实验中所使用的测试环境及相应的测试软件。
2、写出实验报告，内容包括：
① 实验目的
② 实验环境
完成实验内容 2：
③ 给出测试用例
④ 分析测试结果

⑤ 给出改进后的代码
完成实验内容 3：
⑥ 完成测试用例设计，使用断言杀死异常
完成实验内容 4：
⑦ 按步骤完成测试用例设计

附录

类 Date:

```
package cn.edu.gzhu;

/**
 * @author Liya
 *
 */
public class Date {
	public Date() {
	}

	public Date(int year, int month, int day) {
		super();
		if (this.isDayValid(year, month, day) && this.isMonthValid(month)
				&& this.isYearValid(year)) {
			this.year = year;
			this.month = month;
			this.day = day;
		}else{
			throw new IllegalArgumentException("Please check your input!");
		}
	}

	/**
	 * 0 < year
	 */
	private int year = -1;

	/**
	 * 1 <= month <= 12
	 */
	private int month = -1;

	/**
	 * 1 <= day <=31 the day should in the scope [1, 30] when the month is: 4,
	 * 6, ,9, 11 the day should in the scope [1, 31] when the month is: 1, 3, 5,
	 * 7, 8, 10, 12 the day should in the scope [1, 28] when the month is 2 and
	 * the year is not leap year @see the day should in the scope [1, 29] when
	 * the month is 2 and the year is leap year @see
	 */
	private int day = -1;
```

```
    public boolean isDayValid(int year, int month, int day) {
        if ((month == 4 || month == 6 || month == 9 || month == 11) && (day <= 30 && day >= 1))
            return true;
        if ((month == 4 || month == 6 || month == 9 || month == 11) && (day > 30 || day < 1))
            return false;
        if ((month == 1 || month == 3 || month == 5 || month == 7|| month == 8 || month == 10 || month == 12)
&& (day <= 31 && day >= 1))
            return true;
        if ((month == 1 || month == 3 || month == 5 || month == 7|| month == 8 || month == 10 || month == 12)
&& (day > 31 || day < 1))
            return false;

        if(month == 2 && DateUtil.isLeapYear(year) && (day >= 1 || day <= 29)) return true;
        if(month == 2 && DateUtil.isLeapYear(year) && (day < 1 || day > 29)) return false;

        if(month == 2 && !DateUtil.isLeapYear(year) && (day >= 1 && day <= 28)) return true;
        if(month == 2 && !DateUtil.isLeapYear(year) && (day < 1 || day > 28)) return false;
        return false;
    }

    public boolean isMonthValid(int month) {
        return month >= 1 && month <= 12;
    }

    public boolean isYearValid(int year) {
        return year > 0;
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        if (this.isYearValid(year)) {
            this.year = year;
        }else{
            throw new IllegalArgumentException("Please check your input!");
        }
    }

    public int getMonth() {
        return month;
    }
```

```
	public void setMonth(int month) {
		if (this.isMonthValid(month)) {
			this.month = month;
		}else{
			throw new IllegalArgumentException("Please check your input!");
		}
	}

	public int getDay() {
		return day;
	}

	public void setDay(int day) {
		if (this.year == -1 || this.month == -1)
			throw new IllegalStateException("You should set the year and month before day!");
		if (this.isDayValid(year, month, day)) {
			this.day = day;
		}else{
			throw new IllegalArgumentException("Please check your input!");
		}
	}

}
```

类 DateUtil

```
package cn.edu.gzhu;

/**
 * This is a utility for Date
 *
 * @author Liya
 *
 */
public class DateUtil {
	public DateUtil() {
	}

	/**
	 * @param year
	 * @return true if year % 4 == 0 and year % 100 != 0 true if year % 100 == 0
 *             and year % 400 == 0 false otherwise
 */
public static boolean isLeapYear(int year) {
        if (year % 4 == 0 && year % 100 != 0)
                return true;
        if (year % 100 == 0 && year % 400 != 0)
                return false;
        if (year % 100 == 0 && year % 400 == 0)
                return true;
        return false;
}

/**
 *
 * @param date
 * @return day of year，改日期是一年中的第几天
 */
public static int getDayofYear(Date date) {
        int sum = 0;
        if (isLeapYear(date.getYear())) {
                switch (date.getMonth()) {
                case 1:
                        sum = 0;
                        break;
                case 2:
                        sum = 31;
                        break;
                case 3:
                        sum = 60;
                        break;
                case 4:
                        sum = 91;
                        break;
                case 5:
                        sum = 121;
                        break;
                case 6:
                        sum = 152;
                        break;
                case 7:
                        sum = 182;
                        break;
                case 8:
                        sum = 213;
```

```
				break;
			case 9:
				sum = 244;
				break;
			case 10:
				sum = 274;
				break;
			case 11:
				sum = 305;
				break;
			case 12:
				sum = 335;
				break;
			default:
				System.out.print("data error");
				break;

			}
		} else {
			switch (date.getMonth()) {
			case 1:
				sum = 0;
				break;
			case 2:
				sum = 31;
				break;
			case 3:
				sum = 59;
				break;
			case 4:
				sum = 90;
				break;
			case 5:
				sum = 120;
				break;
			case 6:
				sum = 151;
				break;
			case 7:
				sum = 181;
				break;
			case 8:
				sum = 212;
				break;
```

```
                    case 9:
                        sum = 243;
                        break;
                    case 10:
                        sum = 273;
                        break;
                    case 11:
                        sum = 304;
                        break;
                    case 12:
                        sum = 334;
                        break;
                    default:
                        System.out.print("data error");
                        break;
                    }
                }
                sum = sum + date.getDay();
                return sum;
        }
}
```

类 Heap_Item

```
package net.mooctest;

/**
 *
 * @author juraj67
 * @param <T>
 */
public class Heap_Item <T extends Comparable<T>> {

    private Heap_Item ancestor;
    private Heap_Item leftSon;
    private Heap_Item rightSon;
    private T data;

    /**
     * Constructor creates an empty Heap_Item
     */
    public Heap_Item() {
    }

    /**
```

```
  * Constructor creates a new Heap_Item
  * @param data
  */
public Heap_Item(T data) {
	this.data = data;
}

/**
  * Copy constructor makes a copy of Heap_Item
  * @param paCopy_HeapItem
  */
public Heap_Item(Heap_Item<T> paCopy_HeapItem) {
	this.leftSon = paCopy_HeapItem.getLeftSon();
	this.rightSon = paCopy_HeapItem.getRightSon();
	this.data = paCopy_HeapItem.getData();
	this.ancestor = paCopy_HeapItem.getAncestor();
}

/**
  * Method returns the ancestor in binary form
  */
public Heap_Item<T> getAncestor() {
	return ancestor;
}

/**
  * Method sets the ancestor of the heap (in binary form)
  * @param ancestor
  */
public void setAncestor(Heap_Item ancestor) {
	this.ancestor = ancestor;
}

/**
  * Method returns the ancestor of the multi-way pairing heap
  * @return
  */
public Heap_Item<T> getMultiWayAncestor() {
	if(this.ancestor != null) {
		if(this.ancestor.itIsLeftSon(this)) {
			return this.ancestor;
		} else {
			Heap_Item<T> node = this.ancestor;
			while(!node.getAncestor().itIsLeftSon(node)) {
```

```
                    node = node.getAncestor();
                }
                return node.getAncestor();
            }
        }
        return null;
    }

    /**
     * Method returns the left son of heap
     * @return
     */
    public Heap_Item<T> getLeftSon() {
        return this.leftSon;
    }

    /**
     * Method returns the right son of heap
     * @return
     */
    public Heap_Item<T> getRightSon() {
        return this.rightSon;
    }

    /**
     * Method returns heap data
     * @return
     */
    public T getData() {
        return (this.data == null ? null : this.data);
    }

    /**
     * Method sets the heap data
     * @param data
     */
    public void setData(T data) {
        this.data = data;
    }

    /**
     * Set left son of the heap
     * @param leftSon
     */
```

```
public void setLeftSon(Heap_Item leftSon) {
	this.leftSon = leftSon;
}

/**
  * Set right son of the heap
  * @param rightSon
  */
public void setRightSon(Heap_Item rightSon) {
	this.rightSon = rightSon;
}

/**
  * Method returns true if the heap has a left son
  * @return
  */
public boolean hasLeftSon() {
	return (this.leftSon != null);
}

/**
  * Method returns true if the heap has a right son
  * @return
  */
public boolean hasRightSon() {
	return (this.rightSon != null);
}

/**
  * Method removes the left son from the heap
  */
public void removeLeftSon() {
	this.leftSon = null;
}

/**
  * Method removes the right son from the heap
  */
public void removeRightSon() {
	this.rightSon = null;
}

/**
  * Method returns an item that matches the criteria
  * @param paCriterion
  * @return
  */
public Heap_Item<T> getSonByData(T paCriterion) {
    if(this.hasLeftSon() && this.leftSon.getData().compareTo(paCriterion) == 0) {
        return this.leftSon;
    } else if(this.hasRightSon() && this.rightSon.getData().compareTo(paCriterion) == 0) {
        return this.rightSon;
    } else {
        return null;
    }
}

/**
  * Method removes an item that matches the criteria
  * @param paCriterion
  */
public void removeChild(T paCriterion) {
    if(this.hasLeftSon() && this.leftSon.getData().compareTo(paCriterion) == 0) {
        this.leftSon = null;
        return;
    }
    if(this.hasRightSon() && this.rightSon.getData().compareTo(paCriterion) == 0) {
        this.rightSon = null;
        return;
    }
}

/**
  * Method returns true, if the parameter matches the left son of the heap
  * @param paNode
  * @return
  */
public boolean itIsLeftSon(Heap_Item<T> paNode) {
    return (this.leftSon == paNode ? true : false);
}

/**
  * Method replaces a son who meets the criteria
  * @param paCriterion
  * @param paNode
  * @return
  */
public boolean replaceChild(T paCriterion, Heap_Item<T> paNode) {
```

```
            if(this.hasLeftSon() && this.leftSon.getData().compareTo(paCriterion) == 0) {
                this.leftSon = paNode;
                return true;
            } else if(this.hasRightSon()&& this.rightSon.getData().compareTo(paCriterion) == 0){
                this.rightSon = paNode;
                return true;
            } else {
                return false;
            }
        }
}
```

类 BinaryExponentiation

```
package net.mooctest;

//Java Program to Implement Binary Exponentiation (power in log n)

/*
* Binary Exponentiation is a method to calculate a to the power of b.
* It is used to calculate a^n in O(log n) time.
*
* Reference:
* https://iq.opengenus.org/binary-exponentiation/
*/

public class BinaryExponentiation {

  // recursive function to calculate a to the power of b
  public static long calculatePower(long x, long y) {
        if (y == 0) {
            return 1;
        }
        long val = calculatePower(x, y / 2);
        if (y % 2 == 0) {
            return val * val;
        }
        return val * val * x;
  }

  // iterative function to calculate a to the power of b
  long power(long N, long M) {
        long power = N, sum = 1;
        while (M > 0) {
            if ((M & 1) == 1) {
```

```
            sum *= power;
        }
        power = power * power;
        M = M >> 1;
    }
    return sum;
 }
}
```