
实验二 Junit+eclemma+ant 自动化测试

班级	学号	姓名	指导老师
软件工程 1603 班	2016012963	董佩杰	毛锐

一、实验目的

- 1) 掌握 Junit 的安装及其与 Eclipse 的集成。
- 2) 利用 Junit 进行单元测试。
- 3) 掌握 Junit 中常用 annotation: @Before、@After、@Test、@Ignore、@BeforeClass、@AfterClass 的用法, 了解 @Test 的两个参数 timeout 和 expected 的用法。
- 4) 掌握 Junit 中套件测试和参数化测试的方法。
- 5) 掌握 Eclemma 的安装和使用。
- 6) 基于 Eclemma 的覆盖率测试对 Junit 单元测试覆盖分析, 提升测试质量。
- 7) 利用 Ant 进行自动化测试的配置和执行

二、实验步骤

1. Junit 的安装及其与 Eclipse 的集成。

(两种方法, (1)–(3) 本地安装, (4)Eclipse 集成安装)

(1) 从[Download Junit](#)下载 Junit 压缩包, 把 Junit 压缩包解压到一个物理路径

(2) 记录 Junit4.10.jar 文件所在目录。

(3) 设置环境变量 CLASS_PATH。

(4) 在 Eclipse 菜单 “project” 的子项 “properties” 中选择 “Java Build Path”, 单击 “Libraries” 标签, 添加 JAR, 即选择 junit.jar 或 junit-4.10.jar, 单击打开, 就完成了 Junit 的安装。

2. Junit 单元测试

1. 实习题一 利用 Junit Test Case 生成测试用例的框架, 在框架中设计测试代码, 完成对下面类 Practice_1 中 common_divisor, common_multiple, seek_1 方法的测试。

```
package pkg;

/**
 * @author pprp
 * @category 求解最大公约数和最小公倍数以及查找功能
 */
public class Practice_1 {
    public int common_divisor(int a, int b) { // 求最大公约数
        int c, r;
        if (a < b) {
            c = a;
            a = b;
            b = c;
        }
        r = 1;
        while (r != 0) {
            r = a % b;
            a = b;
            b = r;
        }
        return a;
    }

    public int common_multiple(int a, int b) { // 最小公倍数
        return a * b / common_divisor(a, b);
    }

    public boolean seek_1(int[] a, int x) { // 查找
        boolean flag = false;
        for (int i = 0; i < a.length; i++) {
            if (x == a[i])
                flag = true;
        }
        return flag;
    }

    public static void main(String arg[]) { // 主函数
        int b[] = { 10, 20, 15, 30, 25, 40, 35, 50 };
        int x, y, k;
        x = 12;
        y = 6;
        k = 40;
```

```
        Practice_1 a = new Practice_1();
        System.out.println(" 最大公约数为: " + a.common_divisor(x, y));
        System.out.println(" 最小公倍数为: " + a.common_multiple(x, y));
        System.out.println(" 查找结果为: " + a.seek_1(b, k));
    }
}
```

测试类:

```
package pkg;

import static org.junit.Assert.*;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;

/**
 * @author pprp
 * @category 对 Practice_1 进行测试
 */
public class Practice_1Test {
    private static Practice_1 test1 = null;
    private static int num1;
    private static int num2;
    private static int arr[] =
        ↪ {1, 4, 3, 2, 5, 4, 55, 3, 22, 44, 77, 100, 22, 43, 21, 55, 24, 126, 4, 3, 4, 2, 4, 23, 4, 234, 23, 42342, 3423, 2, 544, 46, 4, 75,
    private static int find;
    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
        test1 = new Practice_1();
        num1 = 12;
        num2 = 6;
        find = 55;
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    @Test
    public void testcommon_multiple() {
        assertEquals(12, test1.common_multiple(num1, num2));
    }
}
```

```

@Test
public void testcommon_divisor() {
    assertEquals(6, test1.common_divisor(num1, num2));
}

@Test
public void testseek_1() {
    assertTrue(test1.seek_1(arr, find));
}
}

```

测试结果:

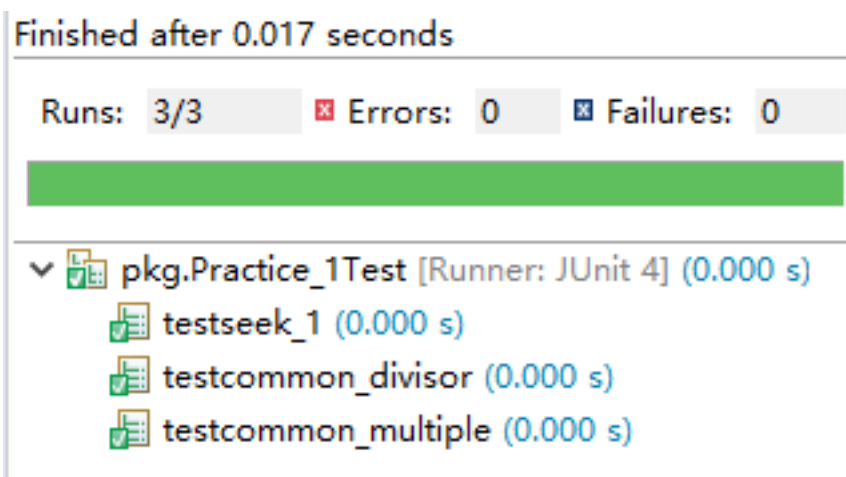


Figure 1: 1557386857969

2. 实习题二 设计判断一个数是不是素数的程序，用基本断言类型实现测试，并用 `setup()` 初始化测试环境。参数化测试数据，并用 `Suite` 方法调用所有单个测试类。

判断素数类:

```

package pkg2;

/**
 * @author pprp
 * @category 判断一个证书是否为素数
 */
public class Prime {

    public boolean isPrime(int num) {

```

```
        for (int i = 2; i < num; i++) {
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }
}

public static void main(String[] args) {
    Prime a = new Prime();
    System.out.println(a.isPrime(9));
}
}
```

测试类:

```
package pkg2;

import static org.junit.Assert.*;

import java.util.Arrays;
import java.util.Collection;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;

/**
 * @author pprp
 * @category 参数化测试，测试素数
 */
@RunWith(Parameterized.class)
public class PrimeTest {

    private Prime test2;
    private int input;
    private boolean output;

    @Before
    public void setUp() throws Exception {
        test2 = new Prime();
    }
}
```

```
}

@After
public void tearDown() throws Exception {

}

@Parameters
public static Collection<Object[]> data() {
    Object[][] object = { { 3, true }, { 7, true }, { 4, false },
        { 5, true }, { 8, false }, { 6, false },
        { 9, false }, { 10, false }, { 11, true },
        { 12, false }, { 13, true }, { 14, false },
        { 42, false }, { 41, true }, { 631, true },
        { 247, false }, { 996, false }, { 96, false } };
    return Arrays.asList(object);
}

public PrimeTest(int input, boolean output) {
    this.input = input;
    this.output = output;
}

@Test
public void testisPrime() {
    assertEquals(output, test2.isPrime(input));
}

}
```

测试结果:



Figure 2: 1557387035056

使用 Suite 对以上两个测试类进行测试:

```
package pkg2;
```

```
import org.junit.runner.RunWith;
```

```
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

/**
 * @author pprp
 * @category 使用 Suite 进行批量测试，此处测试 Practice_1 和 Prime 的测试类
 */
@RunWith(Suite.class)
@SuiteClasses({PrimeTest.class, pkg.Practice_1Test.class})
public class AllTest {

}
```

测试结果为：

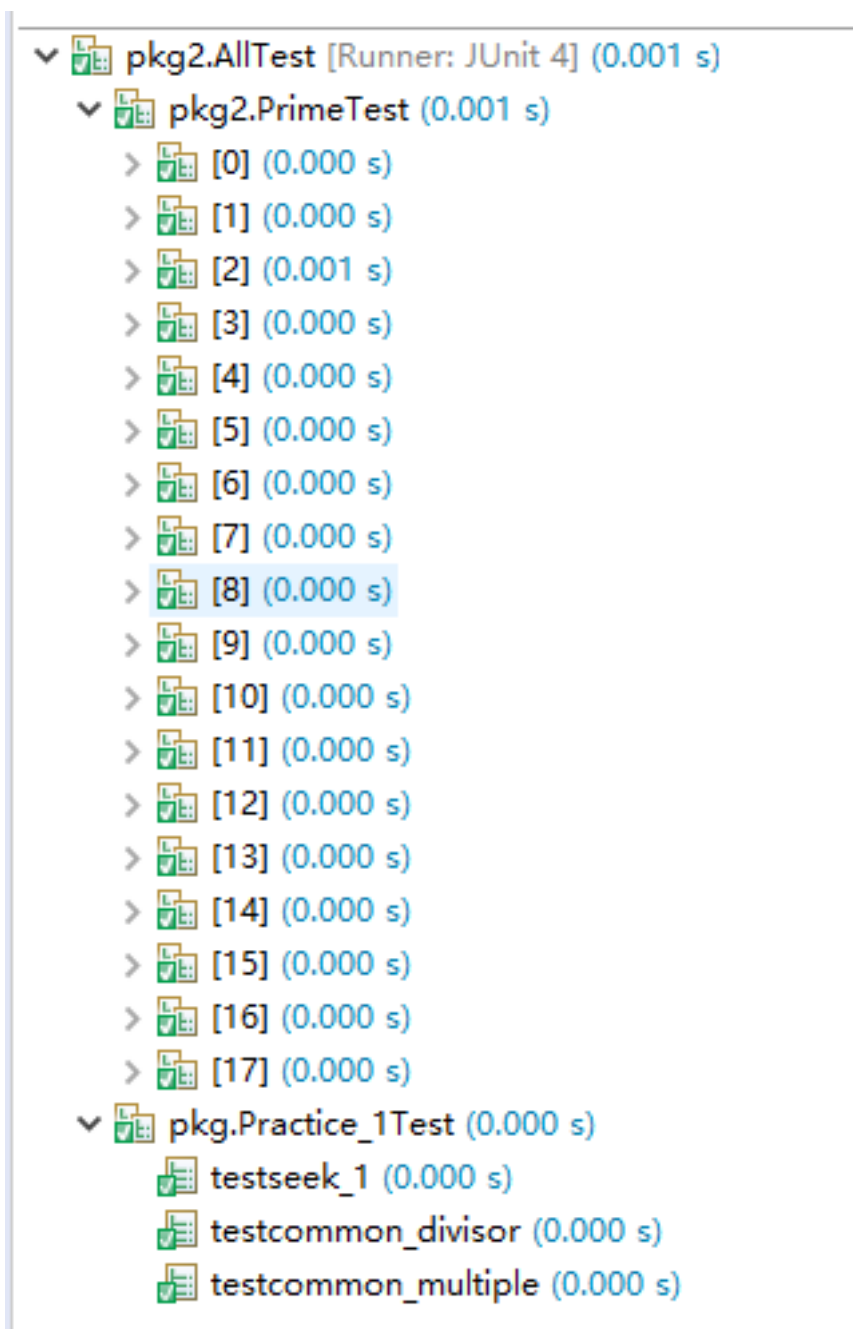


Figure 3: 1557387103442

3. 实习题三 下面是使用 BitSet 来跟踪一年中的那些天是节假日的程序。

```
package pkg3;
```

```
import java.util.BitSet;
```

```
/**
 * @author pprp
 * @category 判断一个数是否存在于这个列表
 */
public class HolidaySked {
    BitSet sked;

    public HolidaySked() {
        sked = new BitSet(365);
        int[] holiday = { 1, 20, 43, 48, 53, 115, 131, 146, 165, 166, 185, 244, 286, 315,
            ↪ 327, 359 };
        // 集合中假日是随机设定的，可根据今年的情况自行调整
        for (int i = 0; i < holiday.length; i++) {
            addHoliday(holiday[i]);
        }
    }

    public void addHoliday(int dayToAdd) {
        sked.set(dayToAdd);
    }

    public boolean isHoliday(int dayToCheck) {
        boolean result = sked.get(dayToCheck);
        return result;
    }

    public static void main(String[] arguments) {
        HolidaySked cal = new HolidaySked();
        if (arguments.length > 0) {
            try {
                int whichDay = Integer.parseInt(arguments[0]);
                if (cal.isHoliday(whichDay)) {
                    System.out.println(whichDay + "is a holiday.");
                } else {
                    System.out.println(whichDay + "is not a holiday.");
                }
            } catch (NumberFormatException nfe) {
                System.out.println("Error: " + nfe.getMessage());
            }
        }
    }
}
```

```
}
```

(1) 请用 TestCase 方法对程序中的 isHoliday() 方法进行 Junit 测试;

```
package pkg3;

import junit.framework.TestCase;

/**
 * @author pprp
 * @category 用 TestCase 方法对程序中的 isHoliday() 方法进行 Junit 测试
 */
public class HolidaySkedCustomTest extends TestCase {
    private HolidaySked cal;
    private int whichDay = 12;

    public void testIsHoliday() {
        cal = new HolidaySked();
        try {
            if (cal.isHoliday(whichDay)) {
                System.out.println(whichDay + "is a holiday.");
            } else {
                System.out.println(whichDay + "is not a holiday.");
            }
        } catch (NumberFormatException nfe) {
            System.out.println("Error: " + nfe.getMessage());
        }
    }
}
```

测试结果:

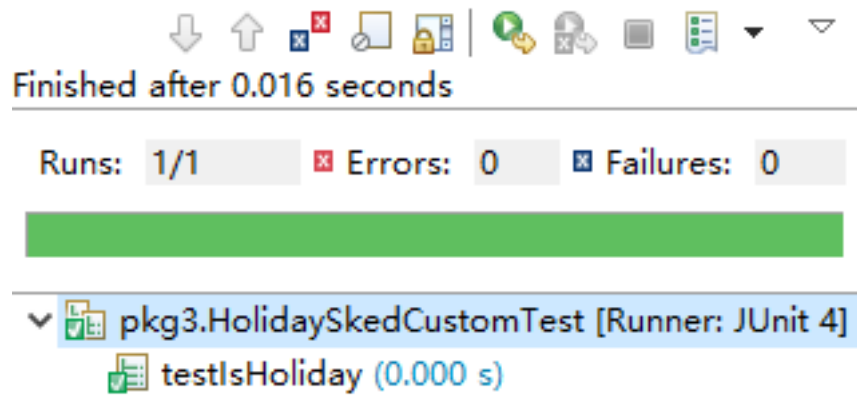


Figure 4: 1557387228219

(2) 用参数化的方法重新设计本题和实验题 1 的测试用例。

本题测试用例：

```
package pkg3;

import static org.junit.Assert.*;

import java.util.Arrays;
import java.util.Collection;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;

/**
 * @author pprp
 * @category 用参数化方法进行测试
 */
@RunWith(Parameterized.class)
public class HolidaySkedTest {
    private int whichday;
    private boolean judge;

    private HolidaySked hs;
    public HolidaySkedTest(int day, boolean judge)
    {
        this.whichday = day;
        this.judge = judge;
    }
}
```

```

        hs = new HolidaySked();
    }

    @Test
    public void test() {
        assertEquals(judge, hs.isHoliday(whichday));
    }

    @Parameters
    public static Collection<Object[]> data() { //1, 20, 43
        Object[][] object = { { 1, true }, { 7, false }, { 4, false },
                               { 20, true }, { 8, false }, { 6, false }, { 43, true } };
        return Arrays.asList(object);
    }
}

```

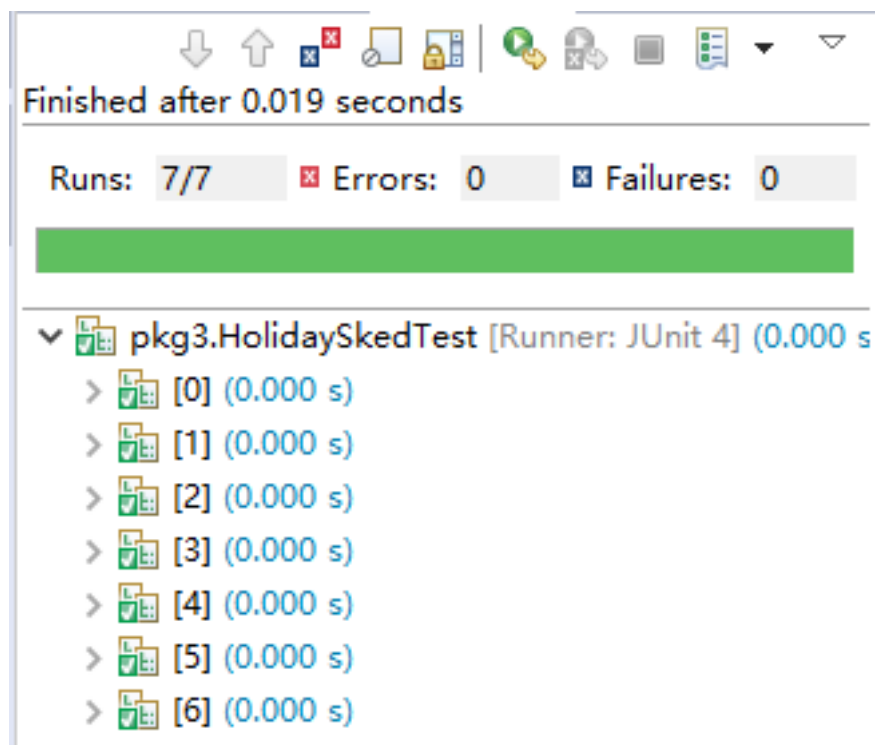


Figure 5: 1557387644744

实验 1 测试用例：（由于参数的异构，所以需要分为两个部分进行测试）

第一部分：最大公约数和最小公倍数

```
package pkg3;
```

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

import java.util.Arrays;
import java.util.Collection;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;

import pkg.Practice_1;

/**
 * @author PC
 * @category 主要用来测试 common_multiple 和 common_divisor 两个方法，用参数化的方法重新设
 * ↪ 计本题
 */

@RunWith(Parameterized.class)
public class Practice_1Test {
    private Practice_1 test2;
    private int num1, num2;
    private int output1, output2; // multiple 公倍数 and divisor 公约数

    public Practice_1Test(int num1, int num2, int output1, int output2)
    {
        this.num1 = num1;
        this.num2 = num2;
        this.output1 = output1;
        this.output2 = output2;
    }

    @Parameters
    public static Collection<Object[]> data() {
        Object[][] object = { { 6, 12, 12, 6 }, { 3, 4, 12, 1 }, { 6, 8, 24, 2 } };
        return Arrays.asList(object);
    }

    @Before
    public void setUp() throws Exception {
```

```
        test2 = new Practice_1();
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void testCommon_divisor() {
        assertEquals(output2, test2.common_divisor(num1, num2));
    }

    @Test
    public void testCommon_multiple() {
        assertEquals(output1, test2.common_multiple(num1, num2));
    }
}
```

测试结果:

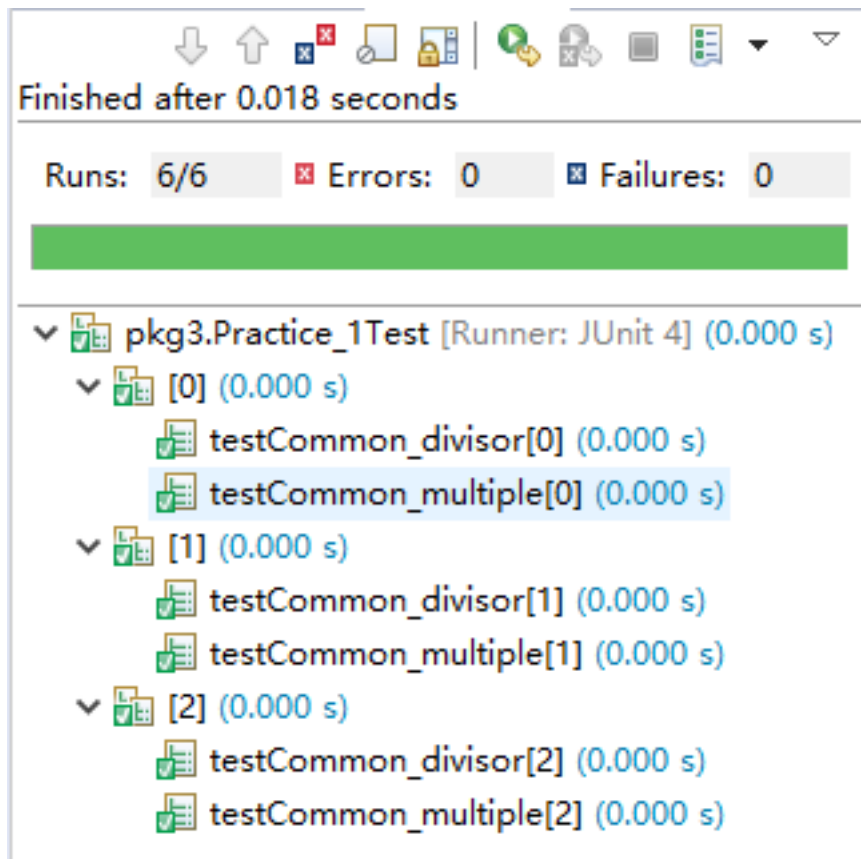


Figure 6: 1557387755721

第二部分：查找功能

```
package pkg3;

import static org.junit.Assert.*;

import java.util.Arrays;
import java.util.Collection;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;

import pkg.Practice_1;
```

```
@RunWith(Parameterized.class)
public class Practice_1TestTest {
    private int arr[] = {1,2,3, 22,44,66, 345,765,432, 1234,6435,7544,8654};
    private int input;
    private Practice_1 test3;

    public Practice_1TestTest(int input) {
        this.input = input;
    }

    @Parameters
    public static Collection<Object[]> data() {
        Object[][] object = {
            { 3 },
            { 22 },
            { 345 } };
        return Arrays.asList(object);
    }

    @Before
    public void setUp() throws Exception {
        test3 = new Practice_1();
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void testseek_1() {
        assertTrue(test3.seek_1(arr, input));
    }
}
```

测试结果:

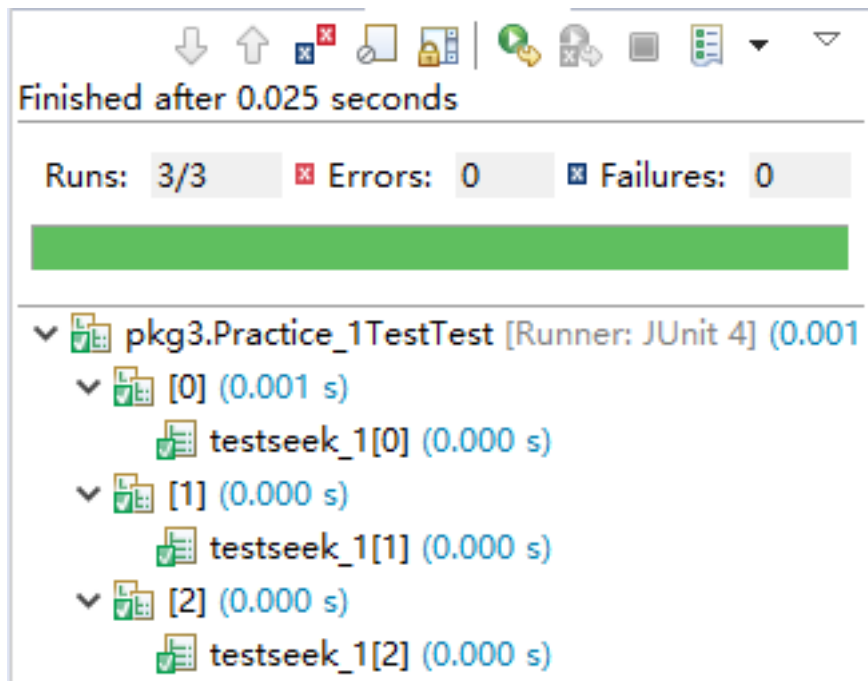


Figure 7: 1557387795760

(3) 再用 Suite 方法对实验 1-3 的所有单个测试类组装进行套件测试。

```
package pkg3;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

import pkg2.PrimeTest;

@RunWith(Suite.class)
@SuiteClasses({Practice_1Test.class,
    PrimeTest.class,
    HolidaySkedCustomTest.class,
    Practice_1TestTest.class,
    pkg.Practice_1Test.class,
    HolidaySkedTest.class})
public class SuiteAllTest {

}
```

测试结果:

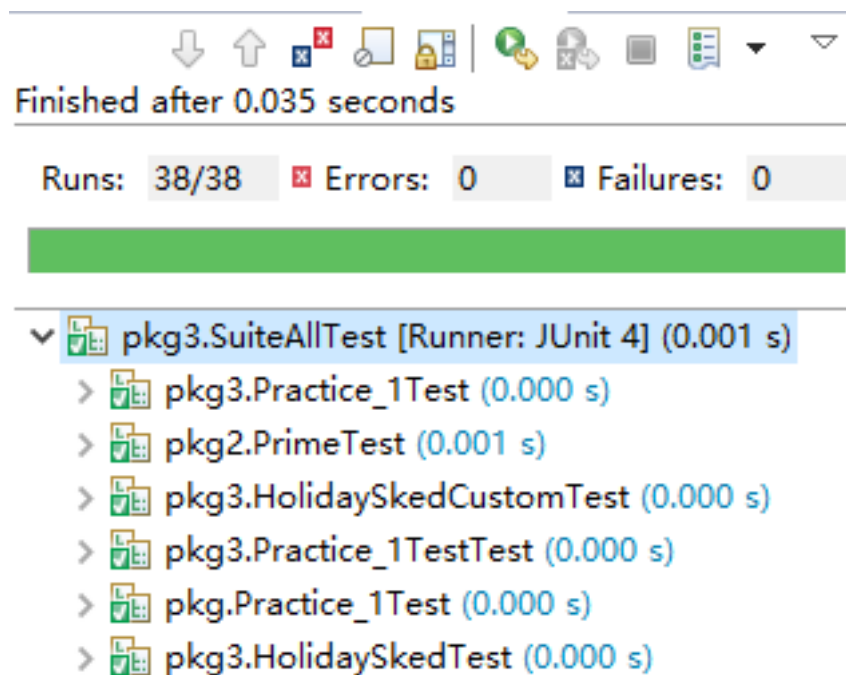


Figure 8: 1557387868226

4. 实习题四 对课本 P56 页的示例程序利用语句覆盖、判定-条件覆盖、条件组合及路径覆盖的角度分别设计测试用例进行自动化测试，同时利用 Eclemma 进行覆盖率分析。(Eclemma 的安装使用见课件)

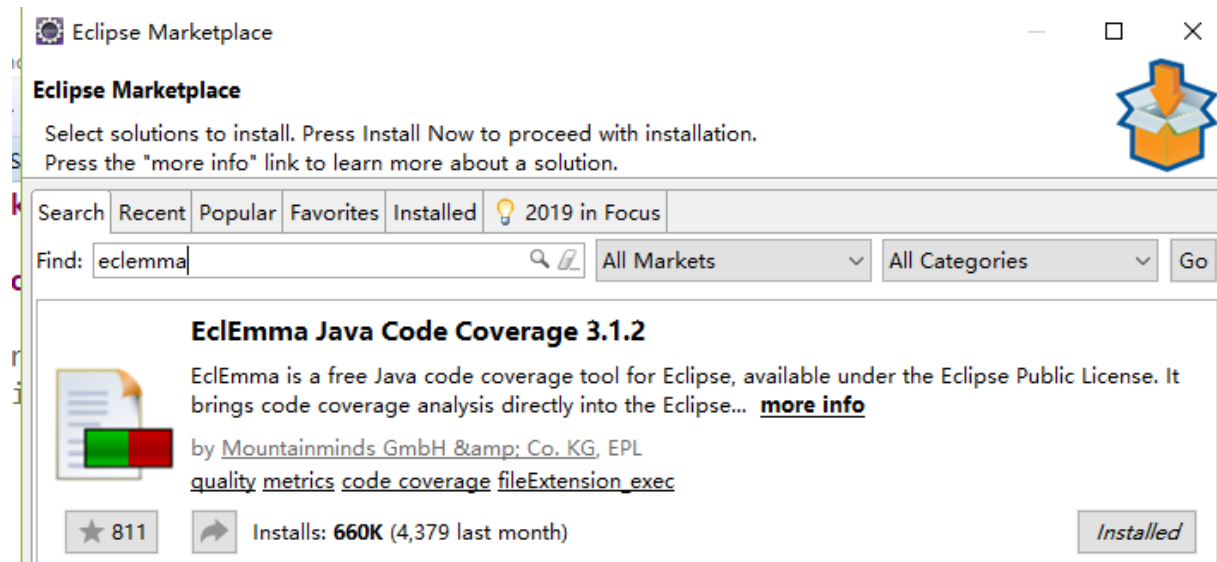


Figure 9: 1557973897890

已安装好 Eclemma

IMG_20190516_110655

5. 实习题五 针对静态测试试验中选择排序，三角形问题，隔一日问题的代码，或者自己开发实现的综合性 Java 项目，在所学测试方法指导下设计测试用例，利用 junit 实现自动化测试，并结合 Eclemma 的覆盖率分析后补充完善所设计的测试用例，提高测试质量。

针对每一个类使用 Junit 进行自动化测试，然后使用 Suite 方法调用所有单个测试类，以下是执行的结果：




























▼  DateProcess		100.0 %	65	0	65
● judge(int[])		100.0 %	26	0	26
● myGetNextDate(int[])		100.0 %	36	0	36
▼  DateProcessTest		100.0 %	153	0	153
● setUp()		100.0 %	6	0	6
● test()		100.0 %	8	0	8
● testJudge()		100.0 %	37	0	37
▼  SelectionSort		100.0 %	54	0	54
● selectionSort(int[])		100.0 %	51	0	51
▼  SelectionSortTest		100.0 %	72	0	72
● setUp()		100.0 %	6	0	6
● test()		100.0 %	15	0	15
▼  Triangle		100.0 %	142	0	142
● judge(int[])		100.0 %	37	0	37
● judgeTriangle(int[])		100.0 %	51	0	51
● selectionSort(int[])		100.0 %	51	0	51
▼  TriangleTest		100.0 %	135	0	135
● setUp()		100.0 %	6	0	6
● test()		100.0 %	33	0	33
● testJudge()		100.0 %	7	0	7
● testSort()		100.0 %	7	0	7

Figure 10: 1558599502366

代码见附件

6. 实习题六 针对实验 5 的 Java 项目，利用 ant 结合 junit 进行自动化测试构建运行。

通过使用 export 将 build.xml 文件生成，然后运行即可。

在这个过程中遇到了一个比较大的问题，一开始将这些文件组织到一个新的工程中的时候，选择将其复制到工程中，然后使用 ant 自动生成 build.xml 文件发现之后在运行 build.xml 的时候无法选中函数，即没有对几个测试类进行调用。一开始以为是环境问题，然后重新检查了 ant 的环境变量等信息，并没有发现大问题。之后在求助了王政宁同学以后得到了解决，先将每个测试类使用 junit 运行一次，然后在进行 export 就可以正常将几个测试类包含在 build.xml 文件中了。具体原因不是很清楚，可能是在运行了以后会生成一些中间文件，ant 是根据这些中间文件判断该类是否是测试类，然后进行 build.xml 的构建。

```
Buildfile: E:\JavaSpace\exp3_pkg6\build.xml
DateProcessTest (1):
[junit] Running pkg6_test.DateProcessTest
[junit] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.109 sec
SelectionSortTest (1):
[junit] Running pkg6_test.SelectionSortTest
[junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.061 sec
TriangleTest (1):
[junit] Running pkg6_test.TriangleTest
[junit] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.061 sec
TestSuites (1):
[junit] Running pkg6_test.TestSuites
[junit] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.069 sec
BUILD SUCCESSFUL
Total time: 3 seconds
```

Figure 11: 1558605140118

三、总结

本次实习经历的时间比较长，前半部分做的比较快，后半部分由于一些环境配置还有其他奇怪怪的原因导致了代码无法运行。总体而言学到了很多有用的东西，感觉这些方法都很实用，可以方便的进行自动化处理。本次实验需要反思的是对时间的把控没有很到位，时间安排不够合理，下次实验尽量往前赶，遇到很难啃的硬骨头及时跟老师和同学进行沟通，推进实验的进度。