天津大学

软件测试技术第一次实验报告

学　　院　软件学院

专　　业　软件工程

年　　级　2016 级

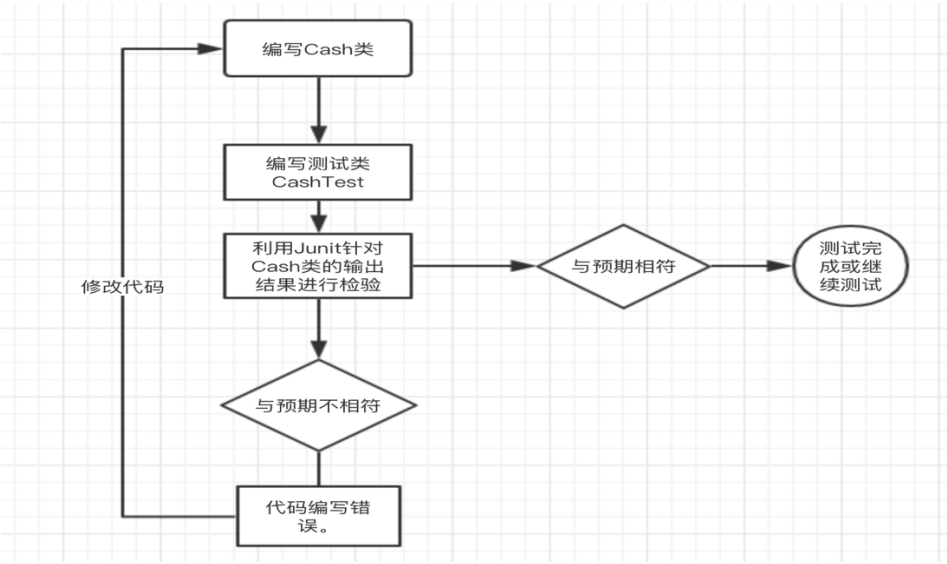
姓　　名　梁哲粼

2019 年 3 月 12 日

软件测试技术第一次实验报告

一、需求分析（描述具体需求）
给定一张 50 元，一张 20 元，两张 5 元纸币和 3 枚一元硬币，给定一个数字，编写程序检验是否能拿出和该数字相等的钱，并利用 Junit 进行测试。

二、概要设计（简单描述设计思路，配合 UML 图）
总共需要两个类，即一个代码实现类和一个测试类。实现类负责检验给定数字能否由所拥有的钱数组成，而测试类则检验代码编写者的预期结果是否和系统给出的结果相一致



三、详细设计（详细描述具体如何实现，附代码及说明）

1：在 eclipse 中对所选工程引入下载好的 Junit 和 Hamcrest 包

2：在 help-EclipseMarketplace 中下载 Eclemma 并安装

3：新建 Change 类，并编写 changeTest 方法,其中 changeTest 方法需给定 5 个参数，即：

50 元纸币的张数、20 元纸币的张数、5 元纸币的张数、1 元硬币的枚数、给定要匹配的数字

代码如下

```
public class Change {
    /*5个参数分别是
     * 50元纸币的张数、20元纸币的张数、5元纸币的张数、1元硬币的枚数、给定想要匹配数字
     */
  public boolean changeTest(int numberOfFif, int numberOfTw, int numberOfFi,int numberOfOn, int cash) {
      //所需50元的张数
      int orderFif = cash/50;
      //若所需50元张数大于给定张数，则减去给定张数的钱，否则减去所需张数的钱
      cash = orderFif > numberOfFif ? cash-numberOfFif*50 : cash-orderFif*50;
      //以此类推
      int orderTw = cash/20;
      cash = orderTw > numberOfTw ? cash-numberOfTw*20 : cash-orderTw*20;
      int orderFi = cash/5;
      cash = orderFi > numberOfFi ? cash-numberOfFi*5 : cash-orderFi*5;
      cash = cash > numberOfOn ? cash-numberOfOn*1 : 0;
      if(cash==0) return true;
      return false;
  }
}
```

4：新建 TestChange 类，对 Change 类代码正确性进行测试。

```
@Test
    //测试函数
    public void test() {
        assertEquals(true, new Change().changeTest(1,1,2,3,1));
        assertEquals(true, new Change().changeTest(1,1,2,3,73));
        assertEquals(true, new Change().changeTest(5,10,12,3,498));
        assertEquals(true, new Change().changeTest(1,1,2,3,8));
        assertEquals(true, new Change().changeTest(4,51,32,7,333));
        assertEquals(false, new Change().changeTest(1,1,2,3,4));
        assertEquals(false, new Change().changeTest(1,1,2,3,9));
        assertEquals(true, new Change().changeTest(10,2,5,4,281));
        assertEquals(false, new Change().changeTest(1,1,2,3,59));
```

```
        assertEquals(false, new Change().changeTest(7,1,2,3,109));
    }
```

源代码和测试代码的覆盖率如下图

（测试代码的红色部分是错误事例）

```
public class Change {
    /*5个参数分别是
     * 50元纸币的张数、20元纸币的张数、5元纸币的张数、1元硬币的枚数、给定想要匹配数字
     */
  public boolean changeTest(int numberOfFif, int numberOfTw, int numberOfFi,int numberOfOn, int cash) {
      //所需50元的张数
      int orderFif = cash/50;
      //若所需50元张数大于给定张数，则减去给定张数的钱，否则减去所需张数的钱
      cash = orderFif > numberOfFif ? cash-numberOfFif*50 : cash-orderFif*50;
      //以此类推
      int orderTw = cash/20;
      cash = orderTw > numberOfTw ? cash-numberOfTw*20 : cash-orderTw*20;
      int orderFi = cash/5;
      cash = orderFi > numberOfFi ? cash-numberOfFi*5 : cash-orderFi*5;
      cash = cash > numberOfOn ? cash-numberOfOn*1 : 0;
      if(cash==0) return true;
      return false;
  }
}
```

（测试代码⬇）

```
public class TestChange {

    @Test
    //测试函数
    public void test() {
        assertEquals(true, new Change().changeTest(1,1,2,3,1));
        assertEquals(true, new Change().changeTest(1,1,2,3,73));
        assertEquals(true, new Change().changeTest(5,10,12,3,498));
        assertEquals(true, new Change().changeTest(1,1,2,3,8));
        assertEquals(true, new Change().changeTest(4,51,32,7,333));
        assertEquals(false, new Change().changeTest(1,1,2,3,4));
        assertEquals(false, new Change().changeTest(1,1,2,3,9));
        assertEquals(true, new Change().changeTest(10,3,5,4,281));
        assertEquals(false, new Change().changeTest(1,1,2,3,59));
        assertEquals(false, new Change().changeTest(7,1,2,3,109));
    }
    @Test
    //测试函数
    public void test1() {
        assertEquals(true,  new Change().changeTest(4,5,2,3,89));

    }
    @Test
    //测试函数
    public void test2() {
        assertEquals(true,  new Change().changeTest(8,8,5,1,109));

    }
    @Test
    //测试函数
    public void test3() {
        assertEquals(false, new Change().changeTest(7,1,2,3,100));

    }
    @Test
    //测试函数
    public void test4() {
        assertEquals(false, new Change().changeTest(13,5,8,10,235));

    }
    @Test
    //测试函数
    public void test5() {
        assertEquals(true,  new Change().changeTest(4,4,4,2,119));
    }

}
```

四、调试分析（在实验过程中遇到的问题以及如何解决）

在 Change 类代码的编写过程中进行了代码优化：

第一版的代码只是按照要求对给定 1 张 50 元，1 张 20 元，两张五元和 3 枚一元硬币进行代码编写；

第二版的代码可以通过给定钱的数目来进行检验，但由于是用 while 进行钱数的判断，所以存在代码冗余，且性能不好。

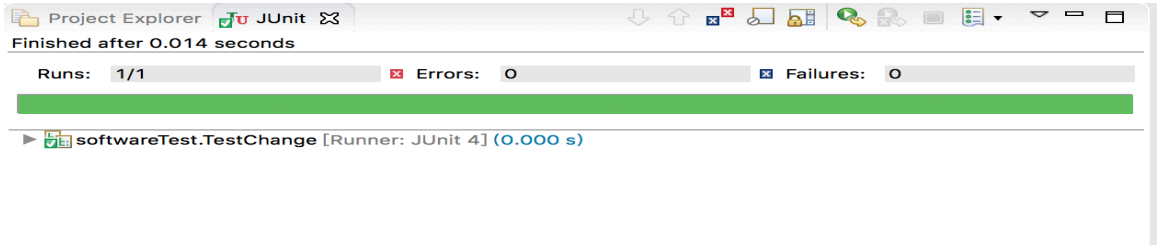
第三版代码通过比较所需钱数和给定钱数，来选择减去所需的张数还是给定的张数，使用三目运算符使代码更简洁易懂，且性能较好。可以从覆盖率看出。

五、测试结果（描述输入和输出）

给定如下 10 个测试样例

```
assertEquals(true, new Change().changeTest(1,1,2,3,1));
assertEquals(true, new Change().changeTest(1,1,2,3,73));
assertEquals(true, new Change().changeTest(5,10,12,3,498));
assertEquals(true, new Change().changeTest(1,1,2,3,8));
assertEquals(true, new Change().changeTest(4,51,32,7,333));
assertEquals(false, new Change().changeTest(1,1,2,3,4));
assertEquals(false, new Change().changeTest(1,1,2,3,9));
assertEquals(true, new Change().changeTest(10,2,5,4,281));
assertEquals(false, new Change().changeTest(1,1,2,3,59));
assertEquals(false, new Change().changeTest(7,1,2,3,109));
```

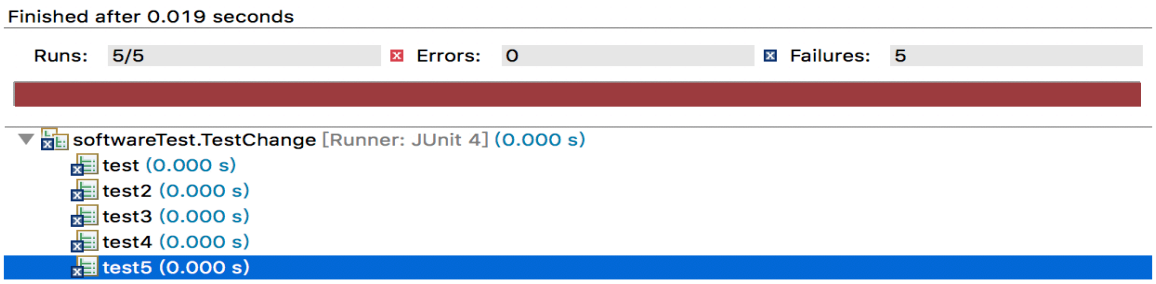
输出结果如下：



给定几个错误的测试样例如下：

```
assertEquals(true,  new Change().changeTest(4,5,2,3,89));
assertEquals(true,  new Change().changeTest(8,8,5,1,109));
assertEquals(false, new Change().changeTest(7,1,2,3,100));
assertEquals(false, new Change().changeTest(13,5,8,10,235));
assertEquals(true,  new Change().changeTest(4,4,4,2,119));
```

输出结果如下：



六、总结

1. 了解了 Junit、Hamcrest 以及 Eclemma 的安装
2. 通过实践了解了 Junit 的注解用法

即：

- **@BeforeClass** - 表示在类中的任意 public static void 方法执行之前执行
- **@AfterClass** - 表示在类中的任意 public static void 方法执行之后执行
- **@Before** - 表示在任意使用@Test 注解标注的 public void 方法执行之前执行
- **@After** - 表示在任意使用@Test 注解标注的 public void 方法执行之后执行
- **@Test** - 使用该注解标注的 public void 方法会表示为一个测试方法