实验三　JUnit 单元测试

3.1 实验类型

实验类型为操作型，4 个学时。

3.2 实验目的

（1）　学习使用 JUnit4.X 进行单元测试；

（2）　应用 JUnit4.X 进行单元测试，为工程项目中的类，设计测试类，并且运用白盒测试和黑盒测试方法为类中的方法设计足够充分的测试用例集，从而保证每一个类代码的正确性和健壮性。

3.3 实验环境

Windows 环境，Word 相关的办公软件，JAVA 编程环境(MyEclipse/Eclipse)；JUnit。

3.4 实验报告

题目 **1**、**2**、**4**、**5** 为必做题，题目 **3** 为选做题。实验报告中反映对应的测试用例代码和测试结果。

实验报告命名：实验三 JUnit 单元测试 +学号姓名.pdf。

3.5 实验内容

1. 建一个除法类（Divide），并编写相应的测试用例 Junit test case。

2. 栈（MyStack）具有 4 个方法，应用 JUnit 4.X 构建测试类，对其进行单元测试。

3. 应用 JUnit 4.X 构建测试类：testCoinBox，设计充分的测试用例（可选）。

4. 使用参数化运行器（Parameterized.class）对税收类 tax 进行测试，建立测试类 testTax。

5. 使用套件运行器（Suite.class），建立测试类 testAll，对已经建立的测试类进行套件化的测试。

3.6 实验步骤

JUnit 是一个开源的 Java 编程语言的单元测试框架，最初由 Erich Gamma 和
Kent Beck 编写。Junit 测试是一种白盒测试工具。JUnit 是一套框架，继承 TestCase 类，就可以用 Junit 进行自动测试了。具有 JUnit 经验对于应用“测试驱动开发（TDD）”的程序开发模型是非常重要的。

3.6.1 Junit 包

(1) 从 http://www.junit.org 下载 Junit，打开该链接，会有一个下载链接，下载 Junit4.X.zip，保存在用户机的文件系统中。 (2) 解包 Junit-4.X，得到如图 3-1 的解包文件。

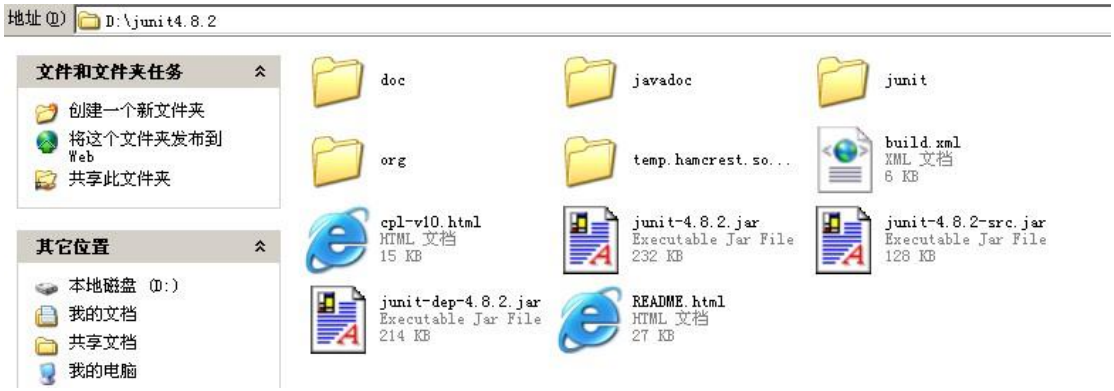


图 1 Junit解包文件

表 1 Junit 文件说明

文件/目录	描　述
junit	是个目录，内有 JUnit 自带的用 JUnit 编写的测试示例程序
javadoc	JUnit 完整的 API 文档
doc	一些文档和文章，包括“Test Infected: Programmers Love Writing Tests”和其它一些资料，可以帮助我们入门。
junit.jar	JUnit 框架结构、扩展和测试运行器的二进制发布
src.jar	JUnit 的源代码，包括一个 Ant 的 buildfile 文件

(3) 配置(以 JUnit4.8.2 为例)。步骤如下：

1、 右击“我的电脑”-“属性”-高级-环境变量；

2、 在系统变量中选择“CLASSPATH”(如果没有则新建一个，变量名

CLASSPATH，变量值 d：\junit4.8.2\junit-4.8.2.jar;d:\junit4.8.2);

3、 如果有 CLASSPATH，将 d：\junit4.8.2\junit-4.8.2.jar;d:\junit4.8.2 加入到变量值即可，多个中间需用；隔开。

(4) 检验：运行中输入 cmd

输入命令：java org.junit.runner.JUnitCore org.junit.tests.AllTests

配置成功，得到下图：

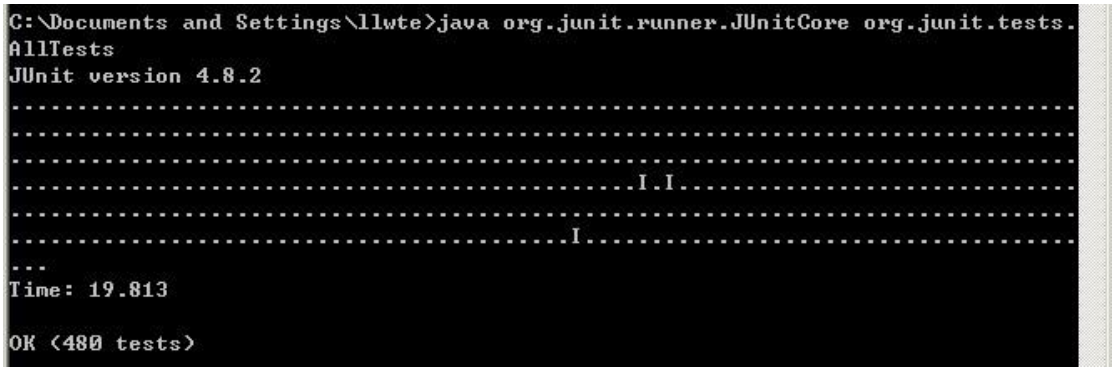


图 2 Junit 配置成功

3.6.2 编写 Junit 测试用例

(1) 新建一 Java Project

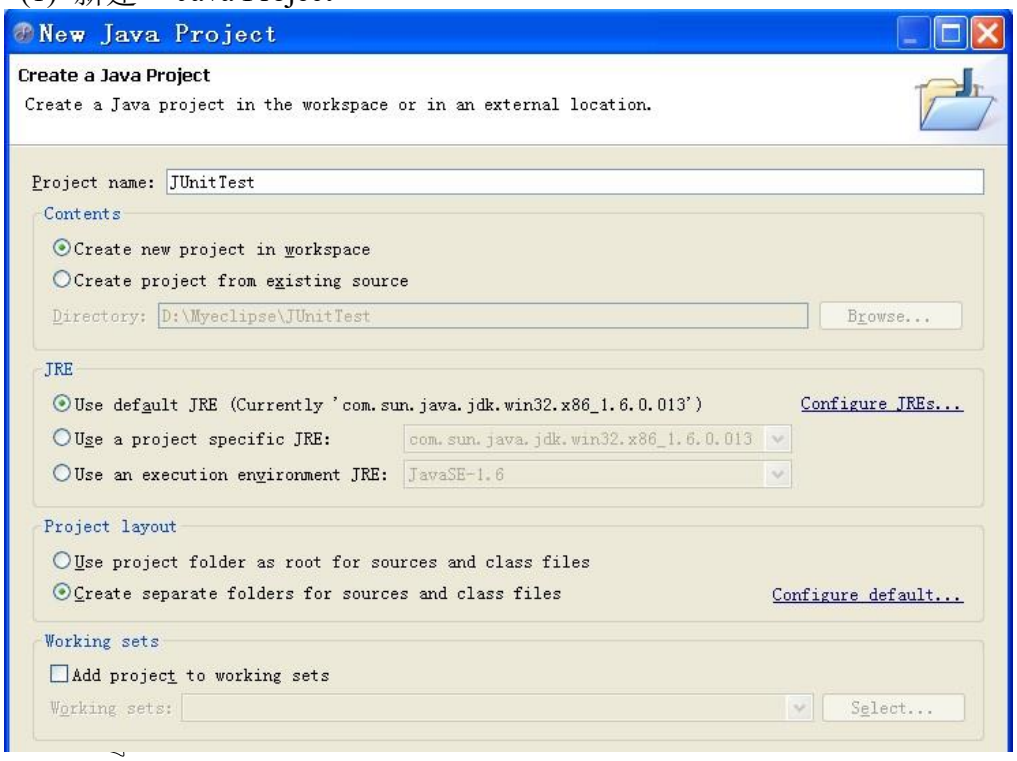


图 3 新建 Java Project

(2) 配置构建路径

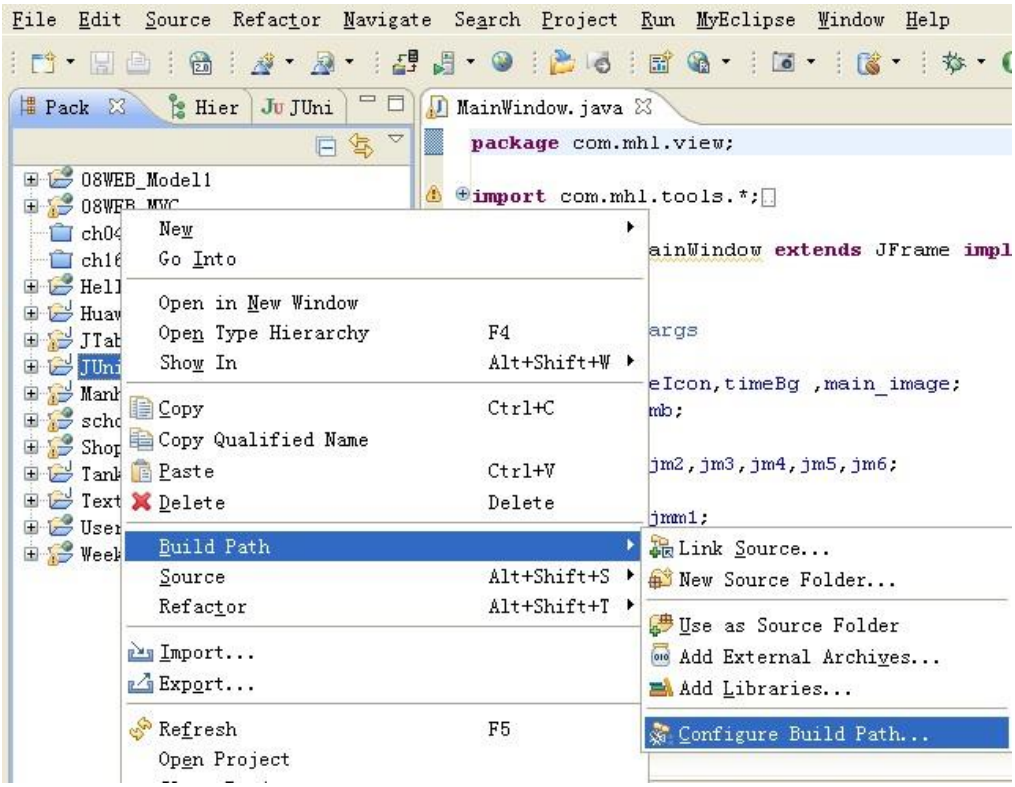


图 4 配置构建路径

(3) Add Library-JUnit 4

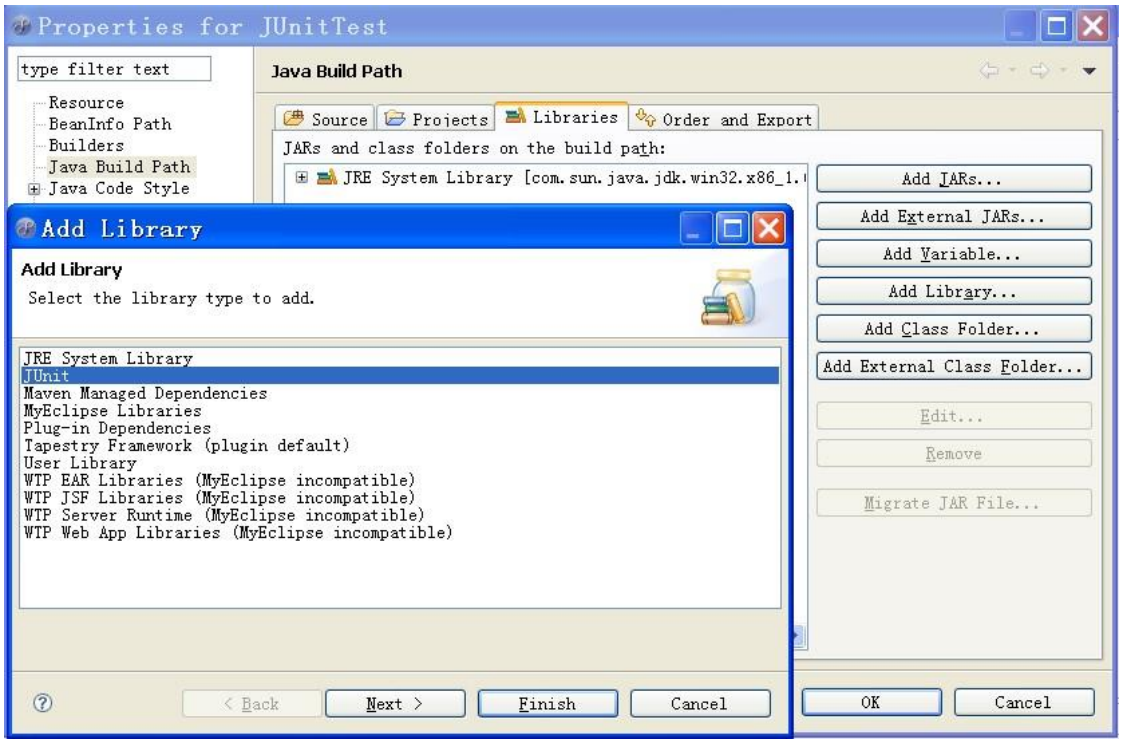


图 5 Add Library

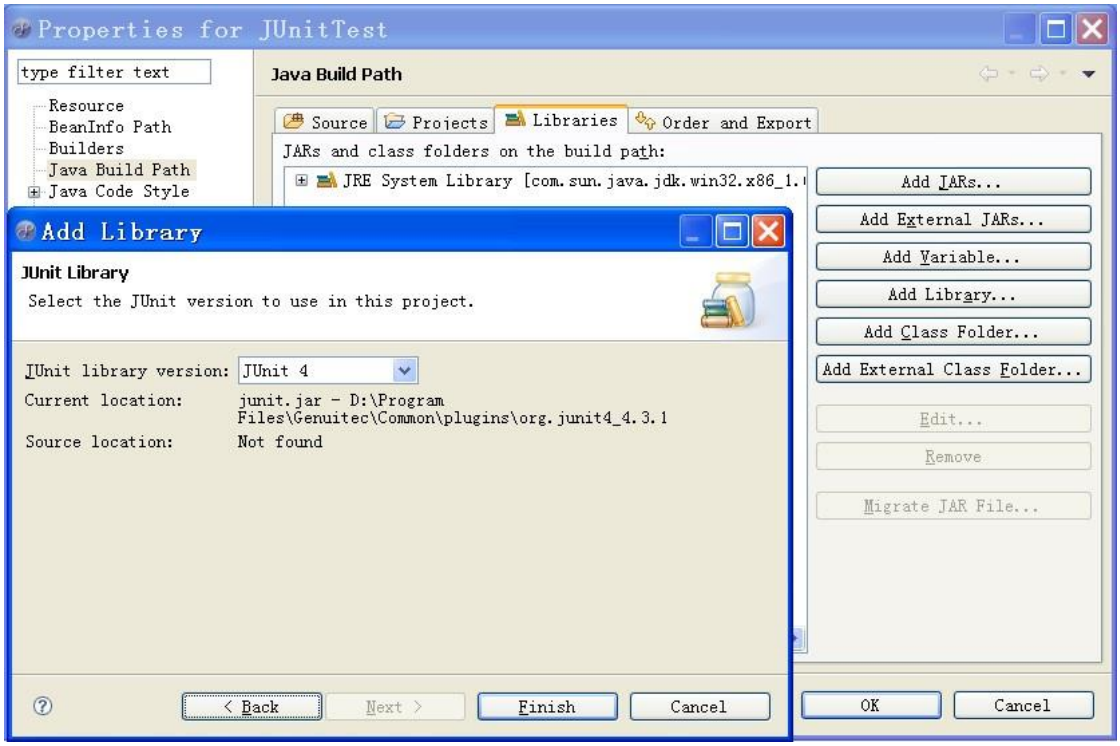


图 6 选择 JUnit 4(1)

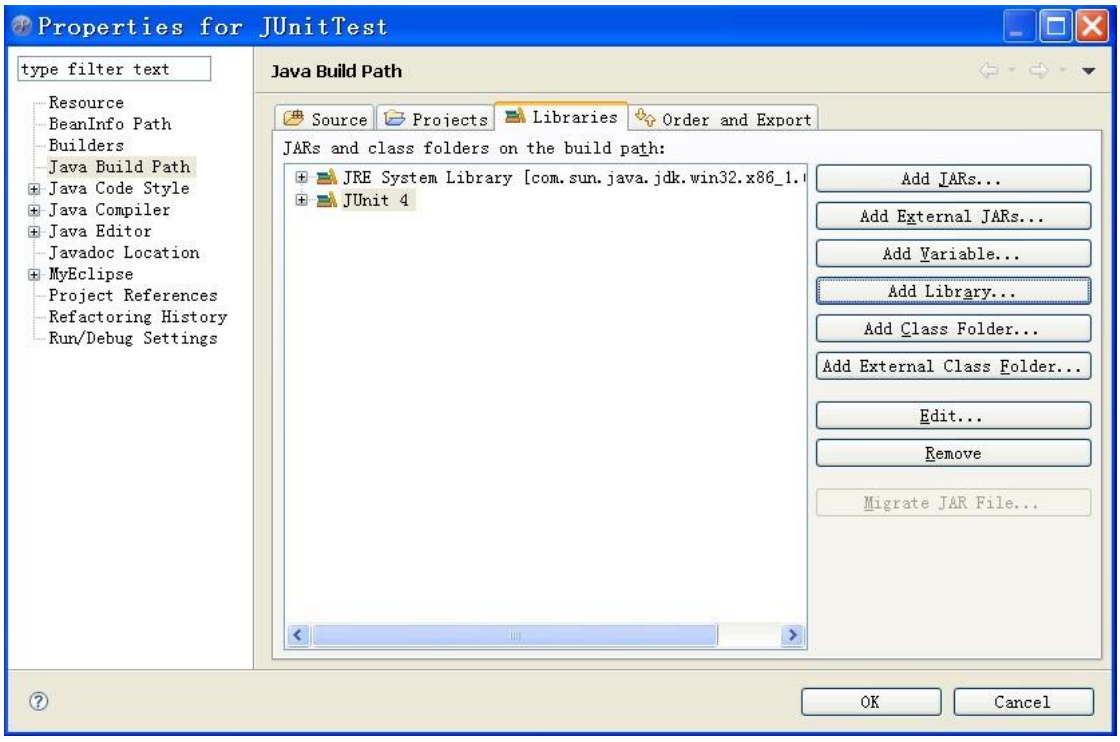


图 7 选择 JUnit 4(2)

(4) 建一个包 com.test 并在此包下建一个除法类：Divide。

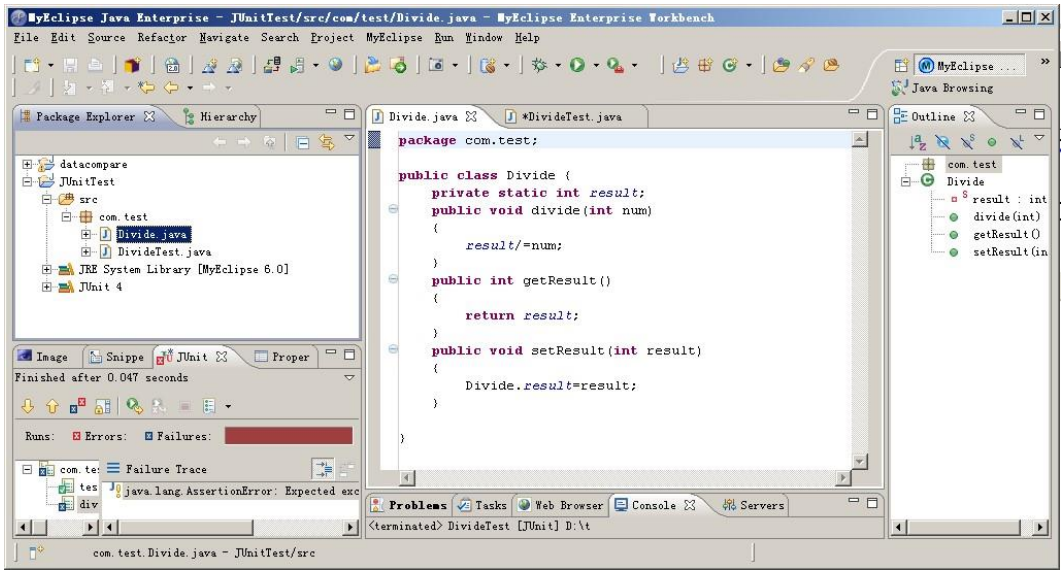


图 8 类 Divide

********** 程序代码：

```
package com.test;


public class Divide {
	private static int result;
	public void divide(int
num)
	{
		result/=num;
	}
	public int getResult()
	{
		return result;
	}
	public void setResult(int result)
	{
		Divide.result=result;
	}
}
```

********** 代码编写完成后，进行调试编译，确保没有语法错误。

(5) 右键 Divide 类

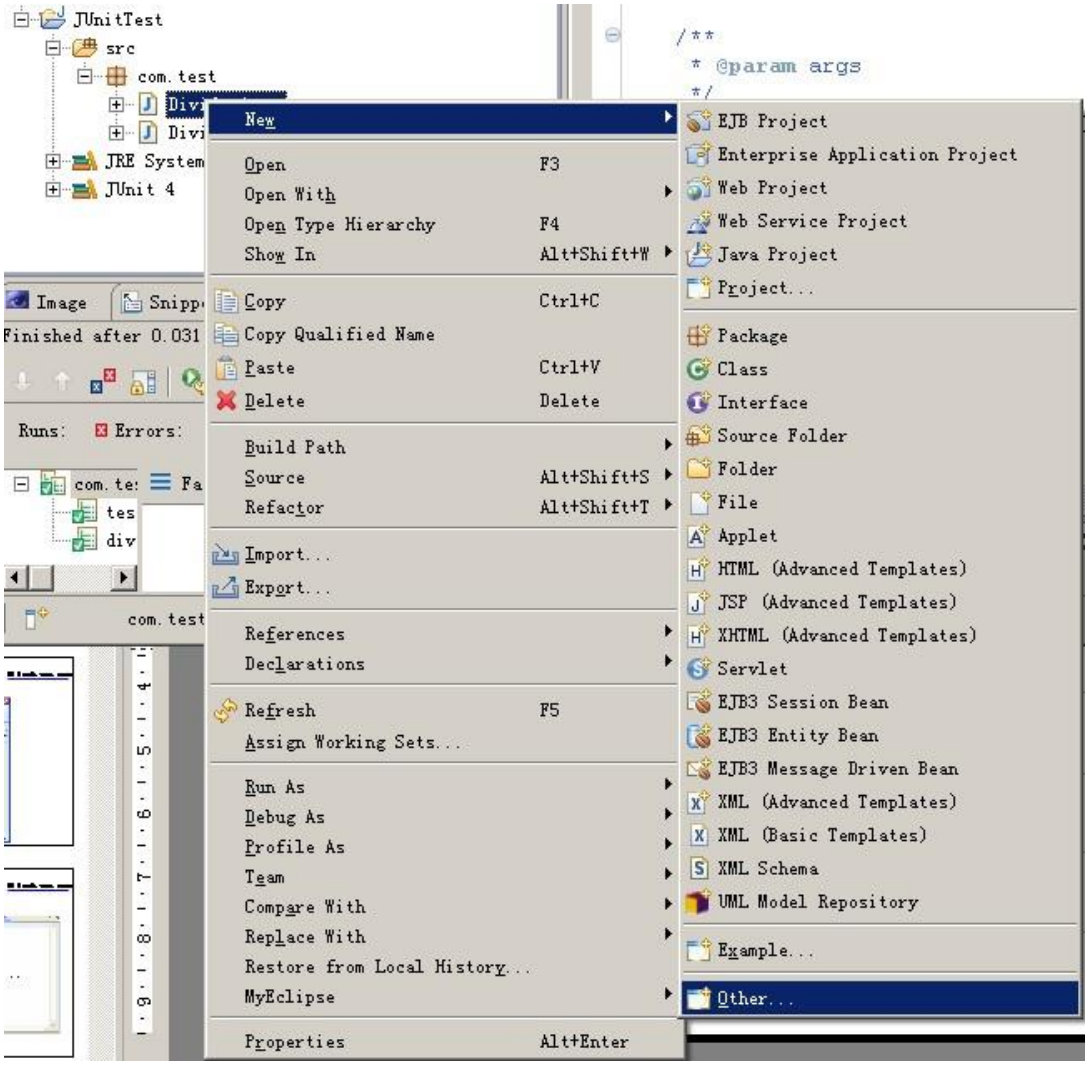


图 9 新建 JUnit Test Case(1)

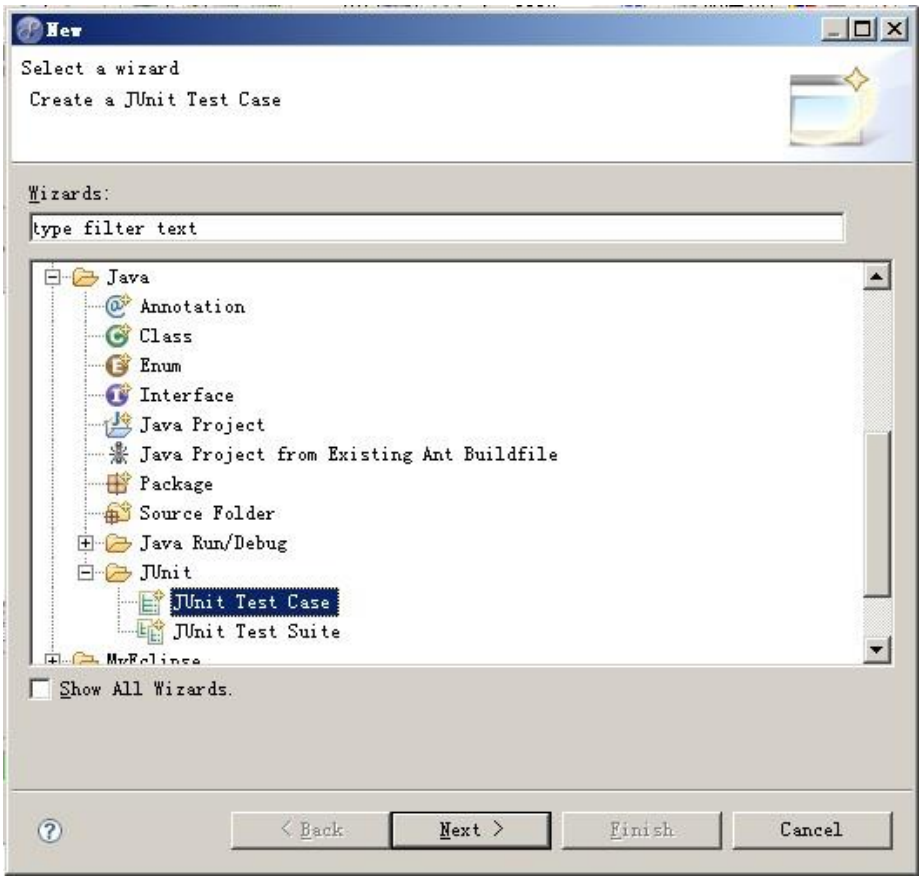


图 10 新建 JUnit Test Case(2)

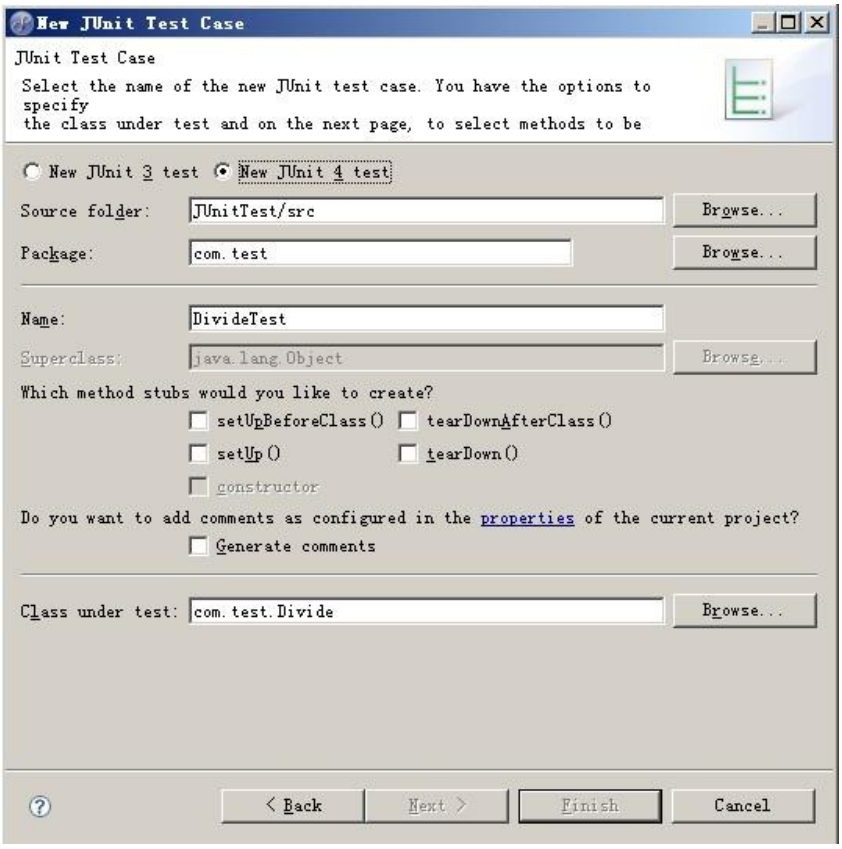


图 11 新建 JUnit Test Case(3)

Myeclipse 会自动为测试类取名：被测试类+Test，单击 Next 就可以了。根据图 12 选择需要进行测试的方法。

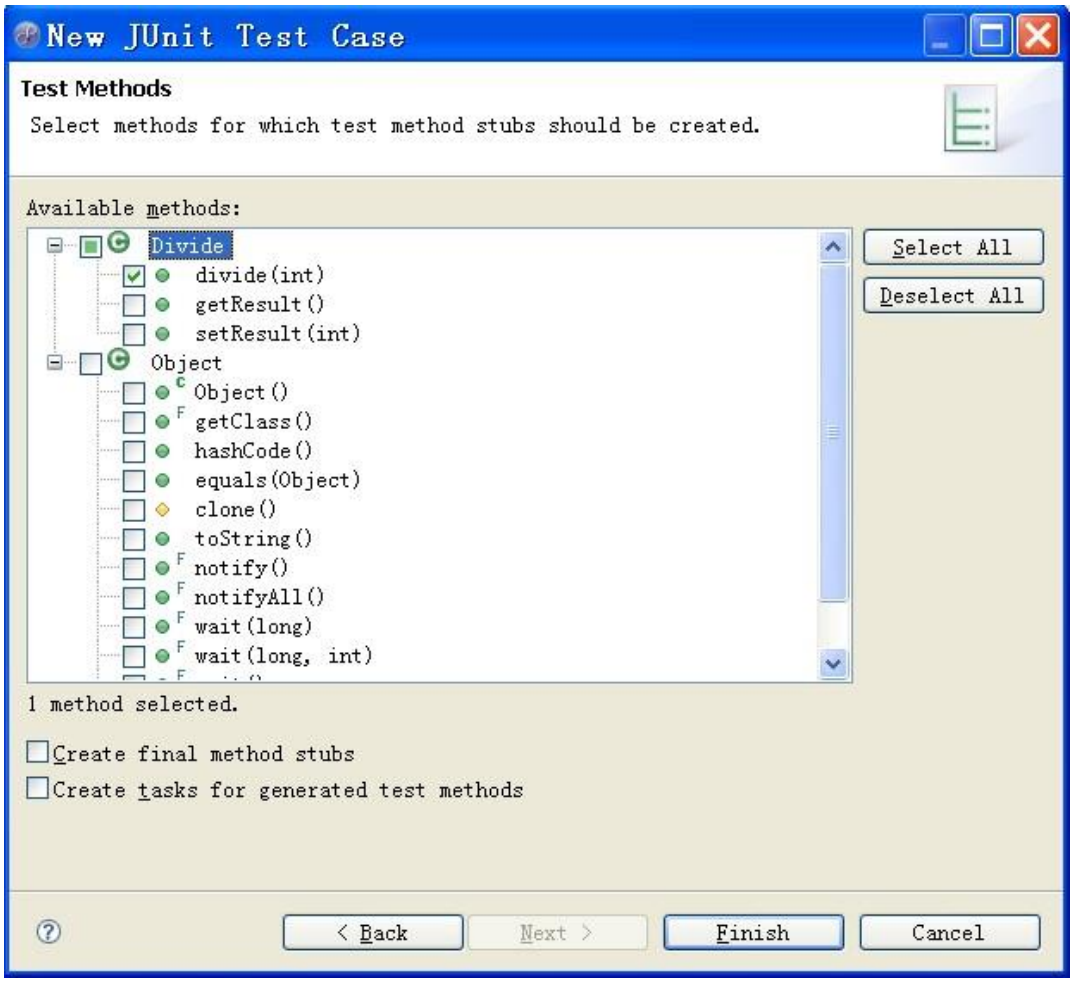


图 12 选择需要测试的方法

(6) 创建测试用例。首先创建一个默认的测试用例。

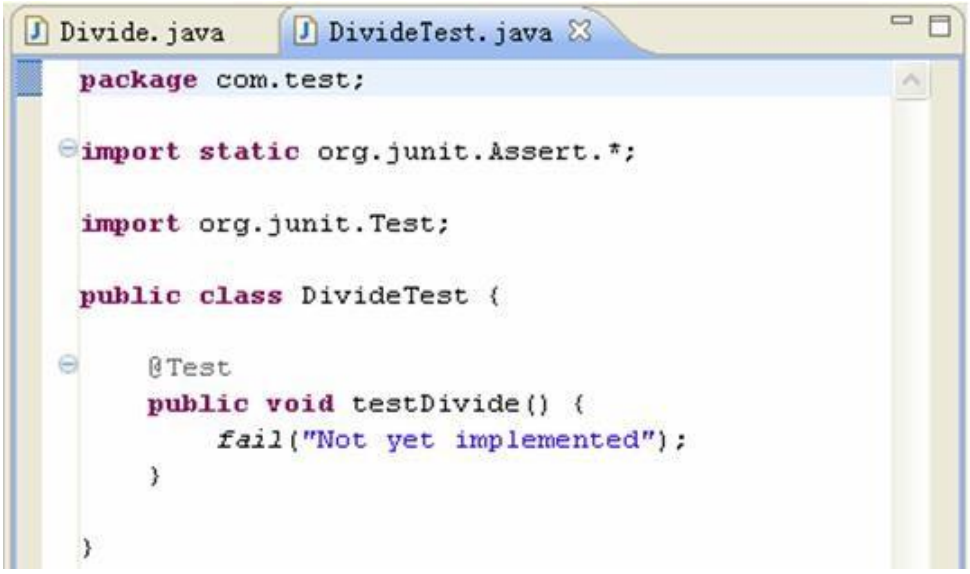


图 13 产生默认的测试用例

(7) 执行测试用例。如图 14 所示。测试结果：红色，测试失败。

注：JUnit 报告测试没有成功，区分为失败（failures）和错误（errors）。失败是你的代码中的 Assert 方法失败引起的；而错误则是代码异常引起

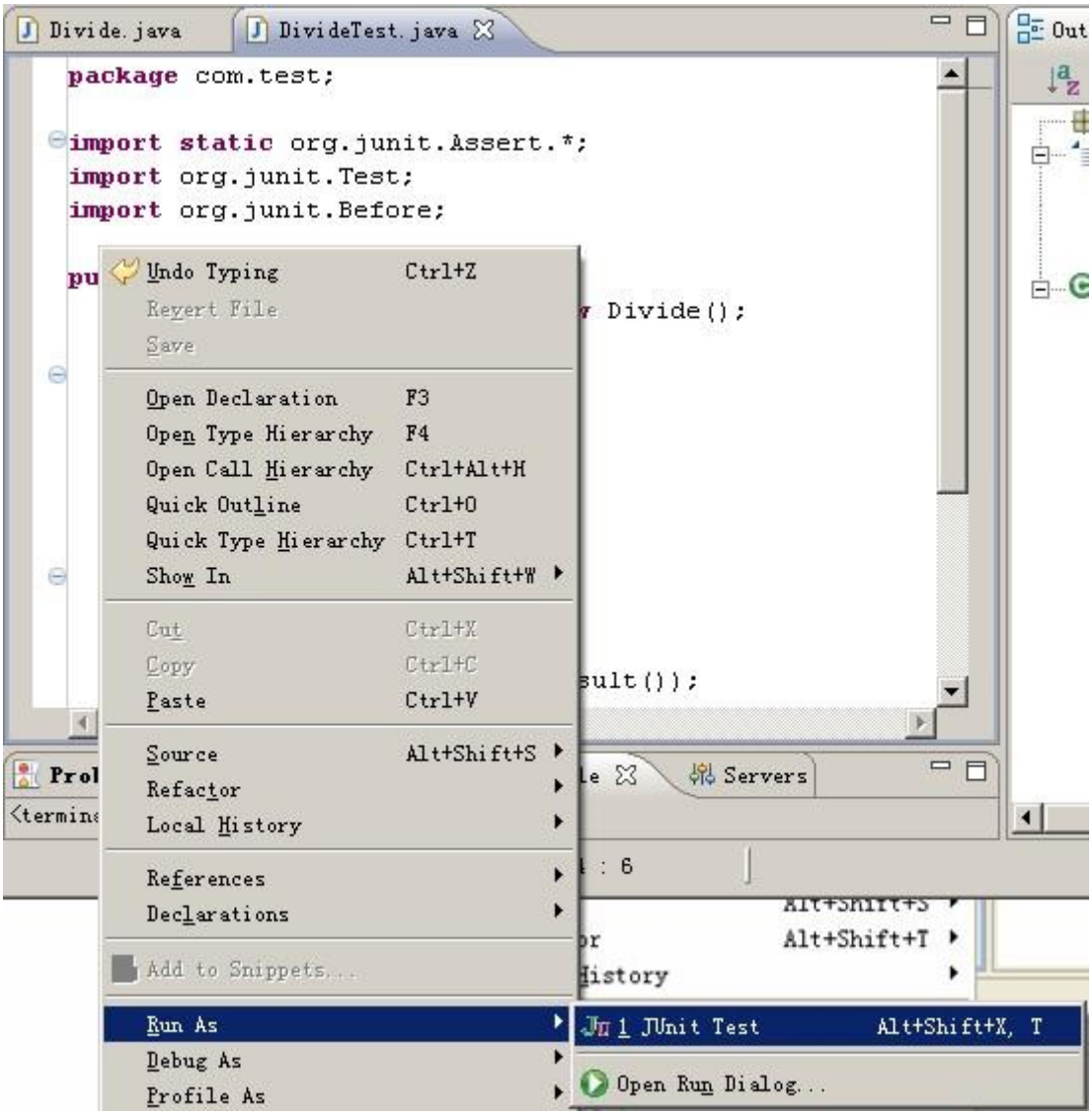


图 14 运行测试用例

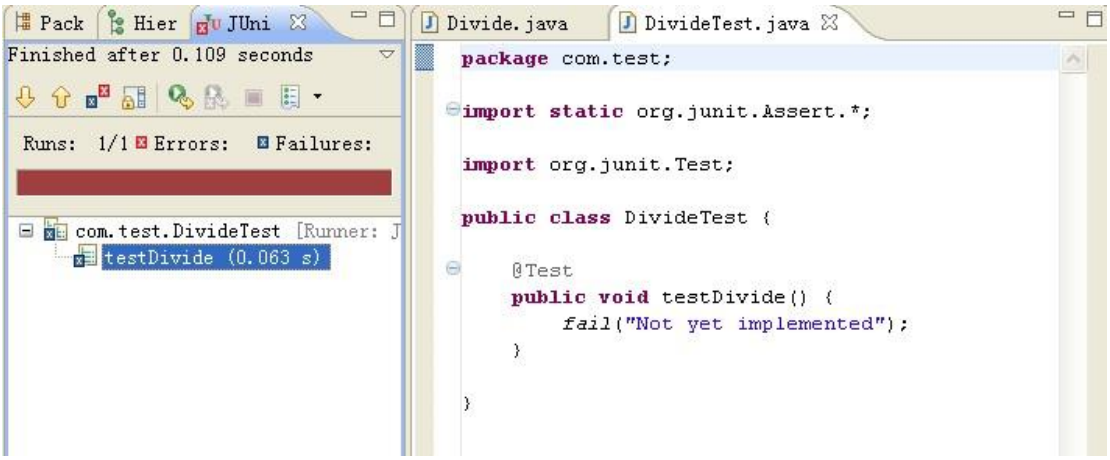


图 15 测试结果

(8) 修改测试用例：DivideTest.java。具体代码如图 16 所示。新测试用例运行后的测试结果如图 17 所示。

```java
package com.test;

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

public class DivideTest {
    private static Divide div=new Divide();
    //初始化
    @Before
    public void init()
    {
        div.setResult(10);
    }
    //除以10
    @Test
    public void testDivide(){
        div.divide(10);
        assertEquals(1,div.getResult());
    }
    //除以0，会有异常
    @Test(expected=Exception.class)
    public final void dividezero()
    {
        div.divide(0);
    }
}
```

图 16 修改后的测试用例

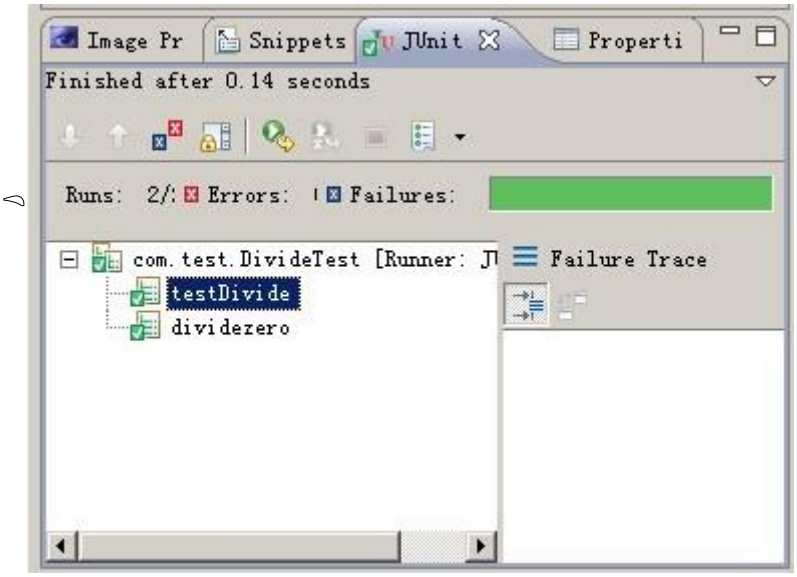


图 17 修改后的测试用例的测试结果

3.6.3 编写栈（**MyStack**）的测试用例

(1) 栈（MyStack）具有 4 个方法：push，pop，top，delete

(2) MyStack 源代码见 MyStack.txt：

****

* package com.junit.test; public class MyStack { private String[] elements; int nextIndex; public MyStack()

```
{
    elements = new String[100];
nextIndex=0;
}

public void push(String element)throws Exception
{
    if(nextIndex==100)
    {
        throw new Exception("数组越界！ ");
    }
    elements[nextIndex++]=element;
}

public String pop()throws Exception
{
    if(nextIndex==0)
    {
        throw new Exception("数组越界！ ");
    }

    return elements[--nextIndex];
}

public String top()throws Exception
{
    if(nextIndex==0)
    {
        throw new Exception("数组越界！ ");
    }

    return elements[nextIndex-1];
```

```
    }

    public void delete(int n)throws Exception
    {
        if(nextIndex-n<0)
        {
            throw new Exception("数组越界！");
        }
        nextIndex -= n;

    }
}
```

****

（4） 测试用例要求：请应用 JUnit 设计 MyStack 类的测试类 testMyStack，构建充分的测试用例，实现对 MyStack 的单元测试，原则为：**Keep the bar green to keep the code clean**。

（5） 采用 JUnit 中运行器方法构建测试套件类 testAll，测试套件中包含 testDivide 类和 testMyStack 类。

3.6.4 编写 CoinBox 测试用例（选做）

(1) 自动售货机：利用自动售货机（Vending Machines）可以销售多种产品，

如冷饮料、热咖啡、糖果、小吃、甚至速冻食品等，可以容易地安装在城市的街道上、学校、工厂里，给人们的生活带来方便。

(2) 自动售货机类图如图 18 所示。

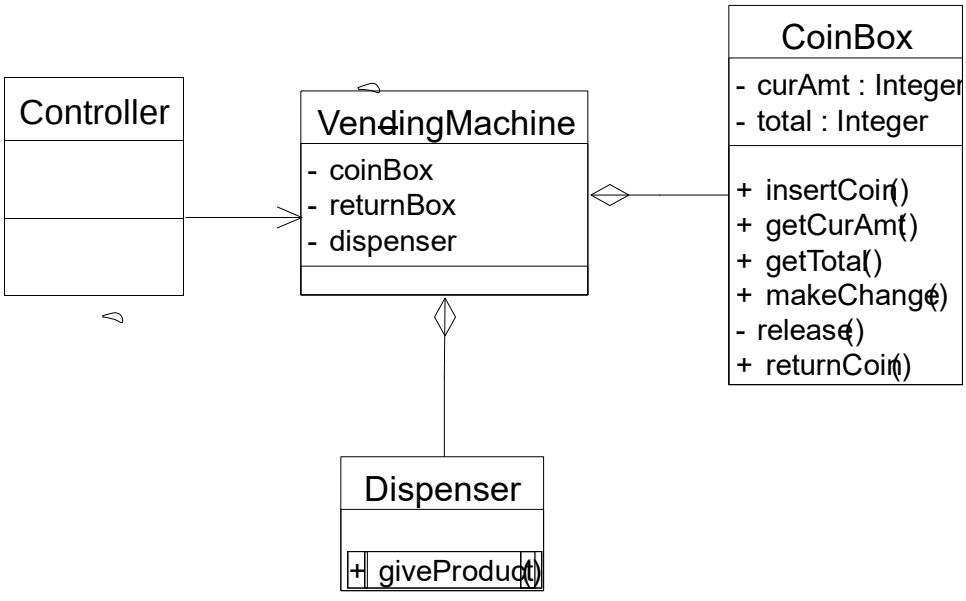


图 18 自动售货机类图

(3) CoinBox 源代码：

```
**********************************************************************
* package coinbox; import java.io.*;
public class CoinBox {
    private int curAmt,
total;
    public void insertCoin(int amt)throws
Exception{        if(amt!=5 && amt!=10 &&
amt!=25){            throw new Exception();
        }
        System.out.println("Current Amount is " +curAmt);
    }
    public int getCurAmt(){
        return curAmt;
    }
    public int getTotal(){
    return total;
    }
    public void makeChange(int price)throws
Exception{        if(price<=0){            throw
new Exception();
        }
        release(curAmt-price);
        curAmt =0;
    total+=price;
    }
    private void release(int amt){
        System.out.println("Release " +amt);
    }
    public void returnCoin(){
        release(curAmt);
    curAmt=0;
    }
}
**********************************************************************
```

（4） 测试用例要求：请应用 JUnit 设计 CoinBox 类的测试类 testCoinBox，运行测试方法，比较测试结果，如果通过测试用例发现了程序中的 bug，请找出原因并更正说明。

（5） 补充类 Dispenser，并实现其中的方法 giveProduct()，并应用 JUnit 构建测试用例(选做)。

3.6.5 应用(Parametrized)运行器编写参数化测试用例

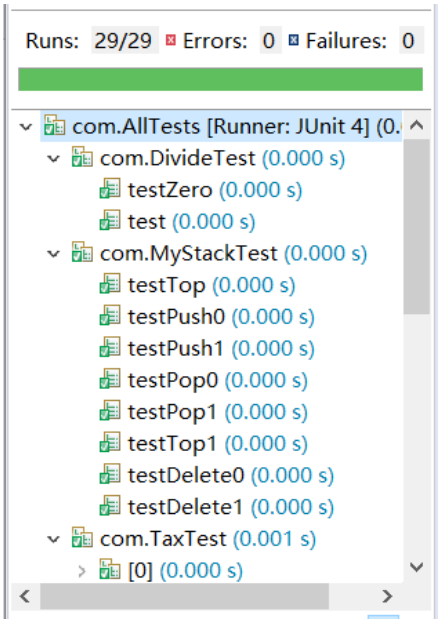
如表 2 所示，针对不同月薪需要缴纳不同的个人所得税计算程序，结合边界值分析法和等价类划分方法，设计充分的测试用例。运用 Parametrized 运行器实现 18 组不同测试数据的测试实例，测试类名称为：testTax。注意测试数据的选择：每一个不同的条件都需要设计两组数据（典型，边界，0）。

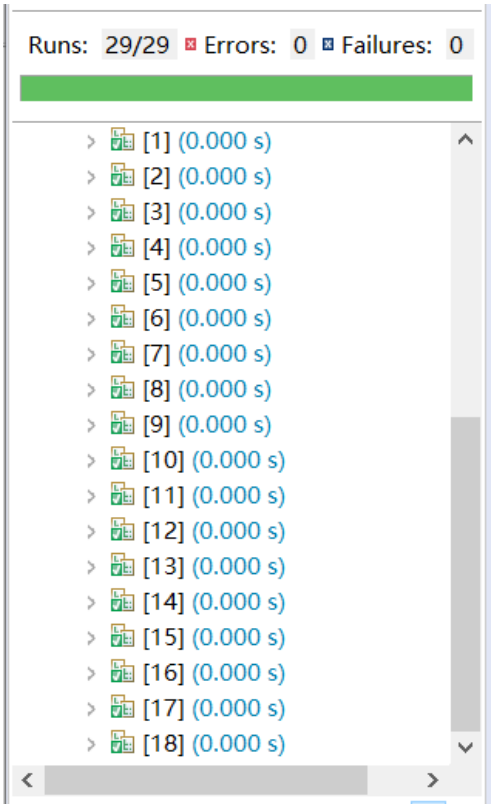
表 2 个税计算表

应纳税所得额(减去起征点 2000 元后的结果)	税率/%
不超过 500 元	5
超过 500 元至 2000 元	10
超过 2000 元至 5000 元	15
超过 5000 元至 20000 元	20
超过 20000 元至 40000 元	25
超过 40000 元至 60000 元	30
超过 60000 元至 80000 元	35
超过 80000 元至 100000 元	40
超过 100000 元	45

3.6.6 应用(Suite)套件运行器编写套件测试用例

运用 Suite 运行器实现的 testDivide.class，testMyStack.class 以及 testTax.class 的测试套件用例，测试类名称为：testAll。





3.7 实验要求

（1）应用 JUnit 4.X 构建测试类：testDivide 和 testMyStack、testCoinBox（可选）、testTax（使用参数化运行器 Parameterized.class）、testAll（使用套件运行器 Suite.class），设计足够多且充分的测试用例。

（3） 实验结果要求给出每个测试用例的测试结果。

（4） 撰写实验报告，实验报告含代码清单：testMyStack 类或 testTax 类，以及测试套件类 testAll。

**********MyStackTest 类**********

```
package com;

import static org.junit.Assert.*;

import org.junit.Test;

public class MyStackTest {

    private MyStack stack = new MyStack();
```

```
private String [] elements;
private int nextIndex;

@Test(expected=Exception.class)
public void testPush0() throws Exception {

    int i;
    for (i=0;i<100;i++) {
        stack.push(i+"");
    }
    stack.push("");
}

@Test
public void testPush1() throws Exception {

    int i;
    for (i=0;i<10;i++) {
        stack.push(i+"");
    }

    for (i=0;i<9;i++) {
        stack.pop();
    }

    assertEquals("0", stack.pop());
}

@Test(expected=Exception.class)
public void testPop0() throws Exception {

    stack.pop();
}

@Test
public void testPop1() throws Exception {

    int i;
    for (i=0;i<10;i++) {
        stack.push(i+"");
    }

    assertEquals("9", stack.pop());
}
```

```
    @Test(expected=Exception.class)
    public void testTop() throws Exception {

        stack.top();
    }

    @Test
    public void testTop1() throws Exception {

        int i;
        for (i=0;i<10;i++) {
            stack.push(i+"");
        }

        assertEquals("9", stack.top());
    }

    @Test(expected=Exception.class)
    public void testDelete0() throws Exception {

        int i;
        for (i=0;i<100;i++) {
            stack.push(i+"");
        }
        stack.delete(101);
    }

    @Test
    public void testDelete1() throws Exception {

        int i;
        for (i=0;i<10;i++) {
            stack.push(i+"");
        }

        stack.delete(9);

        assertEquals("0", stack.top());
    }

}
```

**********TaxTest 类**********

```
package com;
```

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

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

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


@RunWith(Parameterized.class)
public class TaxTest {

    private Tax t = new Tax();
    private double salary;
    private double tax;

    @Parameters
    public static Collection prepareData() {
        Object [][] objects = {{0.0,700.0},{0.0,0.0},{20.0,2400.0},
                {25.0,2500.0},{60.0,2600.0},
                {200.0,4000.0},{330.0,4200.0},{750.0,7000.0},
                {1040.0,7200.0}
                ,{4000.0,22000.0},{5250.0,23000.0},{10000.0,42000.0},
                {12300.0,43000.0},{18000.0,62000.0},{21350.0,63000.0},
                {28000.0,82000.0},
                {32400.0,83000.0},{40000.0,102000.0},{45450.0,103000.0}};
        return Arrays.asList(objects);

    }

    public TaxTest(double tax, double salary) {
        this.salary = salary;
        this.tax = tax;
    }


    @Test
    public void testTax() {
        double result;
        t.setSalary(salary);
```

```
        t.caculate();
        result = t.getTax();
        assertEquals((long)tax,(long) result);
    }

}
**********AllTest 类**********
package com;

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

@RunWith(Suite.class)
@SuiteClasses({ DivideTest.class, MyStackTest.class
    ,TaxTest.class})
public class AllTests {

}
**********DivideTest 类**********
package com;

import static org.junit.Assert.*;

import org.junit.Ignore;
import org.junit.Test;

public class DivideTest {

    private int result=3;
    private Divide d = new Divide();

    @Test
    public void test() {

        d.setResult(result);
        d.divide(1);
        assertEquals(3, d.getResult());
    }

    @Test(expected=Exception.class)
    public void testZero() throws Exception {

        d.setResult(result);
```

```
        d.divide(0);

    }

}
```

3.8 实验思考题

（1）在单元测试中，如何应用 JUnit 动手写一个类的单元测试方法，并让其成功运行？ 导入包、@Test、断言。

（2）JUnit 4 有何特点？对类的测试、参数化测试、测试套件。（3） double/float 数值的断言：将数值类型转化为(long)

assertEquals（expected，fact，0），第三个参数的含义表示两个数值的差值为某一个数值范围内，则认为两个数是相等的