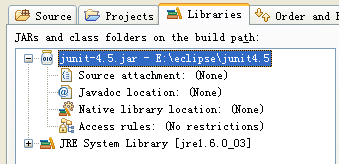
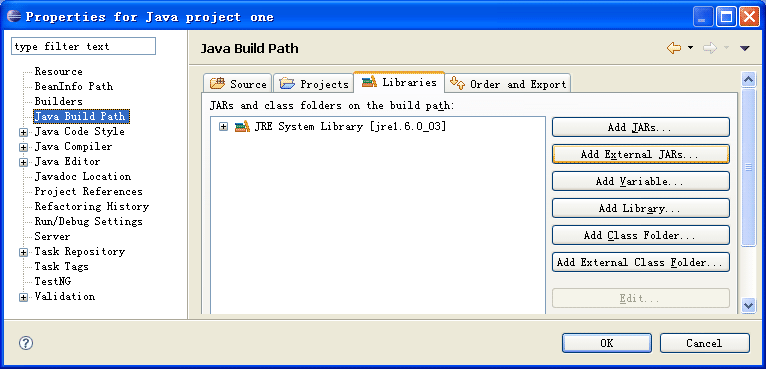
**实验一：JUnit**

* **JUnit三分钟教程 ---- 快速起步**

JUnit是个好东西，做大点的项目离不开这东西，实际中用的时候也因为时间和人力的关系，仅仅写个TestCase（测试单元），运行一下看看就扔过去了，过后也不维护测试代码。

JUnit这东西，说复杂也复杂，说简单也简单，关于JUnit的叙述解说网上一堆一堆，可是就找不到个简单点又能说明问题的附带有完整代码可运行的例子。下面给出一个三分钟的入门的例子大家看看。

JUnit安装



**说在前面：**

当测试类中有两个测试用例testMethod1和testMethod2时，在测试类运行时会分别创建一个新的TestCase子类对象，并引起TestCase中的setUp（用@Before注解）和tearDown函数（用@After注解）分别执行一遍。因此，在进行单元测试的过程中，如有必要我们可以在setUp当中进行一些初始化操作（如类的某些属性的赋值操作），在tearDown中进行一些扫尾工作（如类中某些对象所持有资源的释放）。

* **测试类示例**

一、写测试对象（业务相关的类）

public class Calculator {

    public double add(double n1, double n2) {

        return n1 + n1;

}

   public double subtract(double n1, double n2) {

        return n1 – n2;

}

}

二、写测试用例

测试类1

import junit.framework.TestCase;

public class TestCalculator extends TestCase {

@Test

    public void testAdd(){

        Calculator calculator=new Calculator();

        double result=calculator.add(1,2);

        assertEquals(3,result,0);

}

@Test

public void testSubtract(){

Calculator calculator=new Calculator();

        double result=calculator.subtract(2,1);

        assertEquals(1,result,0);

    }

}

测试类2

import junit.framework.TestCase;

public class TestCalculator2 extends TestCase  {

    public void testAdd(){

        Calculator calculator=new Calculator();

        double result=calculator.add(1.1,2.2);

        assertEquals(3.3,result,0);

}

public void testSubtract(){

        Calculator calculator=new Calculator();

        double result=calculator.subtract(2.2,1.1);

        assertEquals(1.1,result,0);

    }

}

* **TestSuite功能练习**

三、写测试单元

import junit.framework.TestSuite;

import junit.framework.Test;

import junit.textui.TestRunner;

public class TestAll extends TestSuite {

public static Test suite() {

**//****TestSuite功能练习**

        TestSuite suite = new TestSuite("TestSuite Test");

        suite.addTestSuite(TestCalculator.class);

suite.addTestSuite(TestCalculator2.class);

suite.addTest(new TestCalculator());

//添加一个具体的测试用例

suite.addTest(TestSuite.createTest(TestCalculator.class,“testAdd”));

        return suite;

    }

    public static void main(String args[]){

        TestRunner.run(suite());

    }

}

/\* suite方法命名规则如下：

1）必须以“suite”方法命名；

2）suite方法的访问修饰权限必须为public；

3）.suite方法必须为静态方法；

4）.suite方法必须没有参数。

\*/

**另一种写法：**

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

import org.junit.runners.Suite.SuiteClasses;

//RunWith表示这个类是一个suite的类

@RunWith(Suite.class)

//说明这个类中包含哪些测试组件

@SuiteClasses({TestCalculator.class,

               TestCalculator2.class})

public class TestSuite {

    /\* 测试原则：

     \* 1、建议创建一个专门的source folder-->test来编写测试类代码

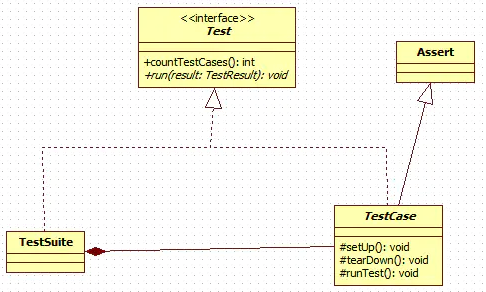
     \* 2、测试类的包应该保持和需要测试的类一致

     \* 3、测试单元中的每一个测试方法都必须可以独立执行，没有顺序

     \*    测试方法之间不能有任何的依赖性

     \*/

}



在执行JUnit测试引擎的run方法时会调用TestSuite的的run方法，TestSuite在执行自身run方法时会遍历所有TestCase对象的run方法，同一个TestCase子类的run方法会根据自身所包含的测试用例个数被执行相应的次数。

四、运行单元测试的main()方法

* **参数化功能练习**

Calculator对应的JUnit测试类：CalculatorTest.java。

import static org.junit.Assert.assertEquals;

import java.util.Arrays;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.junit.runners.Parameterized;

import org.junit.runners.Parameterized.Parameters;

@RunWith(value = Parameterized.class)

public class CalculatorTest {

private int numberA;

private int numberB;

private int expected;

//parameters pass via this constructor

public CalculatorTest(int numberA, int numberB, int expected) {

this.numberA = numberA;

this.numberB = numberB;

this.expected = expected;

}

//Declares parameters here

@Parameters(name = "{index}: add({0}+{1})={2}")

public static Iterable<Object[]> data1() {

return Arrays.asList(new Object[][] {

{ 1, 1, 2 },

{ 2, 2, 4 },

{ 8, 2, 10 },

{ 4, 5, 9 }

});

}

@Test

public void test\_add() {

Calculator calculator=new Calculator();

assertEquals(expected,calculator.add(numberA, numberB));

}

}