**Junit:**

一个单元测试框架

官网：

<http://junit.org/junit4/>

Jar包下载：

<https://github.com/junit-team/junit4/wiki/Download-and-Install>

创建java项目--》JunitProject--》创建类JunitDemo--》在类中写方法，如下：

**package** com.jnmd.junit;

**public** **class** JunitDemo1 {

**private** **static** **int** *result*; // 静态变量，用于存储运行结果

**public** **void** add(**int** n) {

*result* = *result* + n;

}

**public** **void** substract(**int** n) {

*result* = *result* - n;

}

**public** **void** divide(**int** n) {

*result* = *result* / n;

}

**public** **void** square(**int** n) {

*result* = n \* n;

}

**public** **void** clear(){

*result* = 0;

}

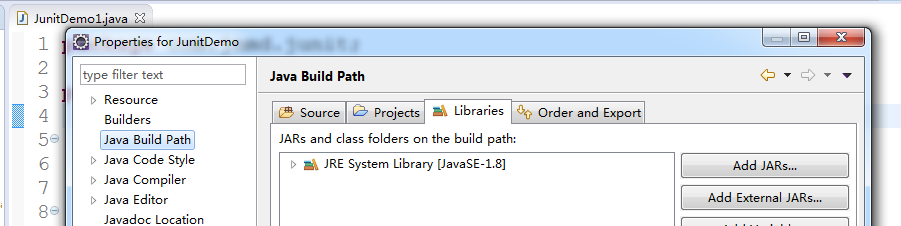
**public** **int** getResult(){

**return** *result*;

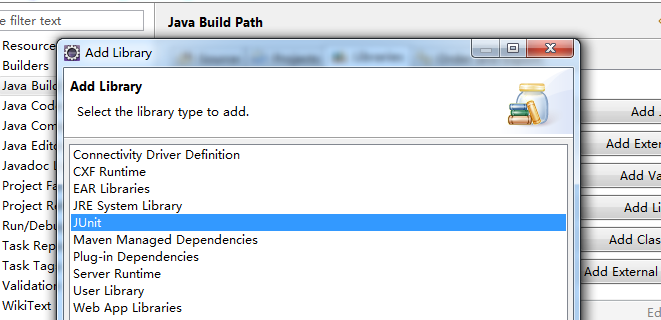
}

}

导入Junit的jar文件，项目名右键，属性---》Java Build Path



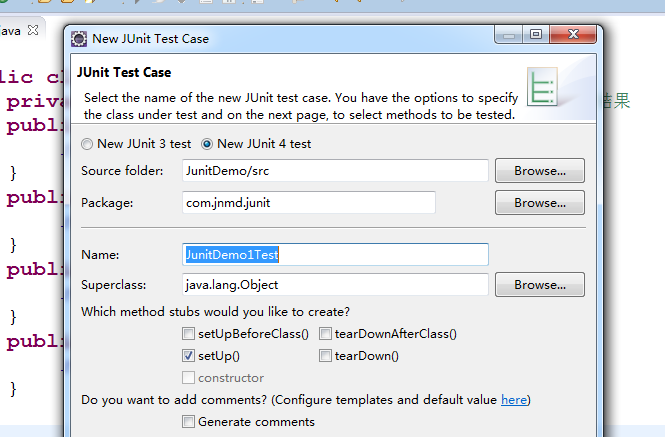
选择添加类库：



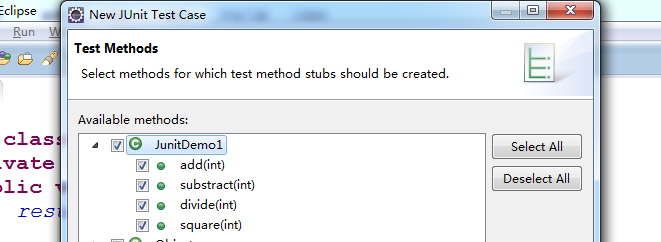
即可导入Junit的jar文件。

对要进行测试的Java文件，创建对应的测试Java文件；

文件名---》右键--》new---》Junit Test case



点击next：



选择要进行测试的方法。

之后系统会自动生成一个新类JunitDemo1Test,即测试框架类，里面包含一些空的测试用例。只需要将这些测试用例稍作修改即可使用。

断言：

assert：

是JDK1.4后加入的新功能。主要是用于代码开发和测试时期，用于对某些关键数据的判断，如果这个关键数据不是程序所预期的数据，程序就提出警告或退出。

断言关键字语法格式：

assert 表达式; //表达式为boolean类型，如果为false程序会退出。

assert 表达式1：表达式2；//表达式1是一个boolean类型，表达式2是一个基本类型或者是Object类型，如果表达式1为true，则忽略表达式2；如果表达式1为false，则运行表达式2，然后程序退出.

Assert：

Junit jar包中的一个对象，对方法进行了封装。

1.assertTrue/False([String message]，boolean condition);  
 判断一个条件是true还是false。

2.fail([String message,]);  
 失败，可以有消息，也可以没有消息。  
 3.assertEquals([String message,]Object expected,Object actual);  
 判断是否相等，可以指定输出错误信息。  
 第一个参数是期望值，第二个参数是实际的值。  
 4.assertNotNull/Null([String message,]Object obj);  
 判读一个对象是否非空(非空)。  
 5.assertSame/NotSame([String msg,]Object expected,Object actual);  
 判断两个对象是否指向同一个对象。

**package** com.jnmd.junit;

**import** **static** org.junit.Assert.\*;

**import** org.junit.Before;

**import** org.junit.Test;

**public** **class** JunitDemo1Test {

**private** **static** JunitDemo1 *jd* = **new** JunitDemo1();

@Before

**public** **void** setUp() **throws** Exception {

*jd*.clear();

}

@Test

**public** **void** testAdd() {

*jd*.add(5);

*assertEquals*(5, *jd*.getResult());

}

@Test

**public** **void** testSubstract() {

*jd*.substract(3);

*assertEquals*(-3, *jd*.getResult());

}

}

注意：

1测试方法上必须使用@Test进行修饰  
    2测试方法必须使用public void 进行修饰，不能带任何的参数  
    3新建一个源代码目录来存放我们的测试代码，即将测试代码和项目业务代码分开  
    4测试类所在的包名应该和被测试类所在的包名保持一致  
    5测试单元中的每个方法必须可以独立测试，测试方法间不能有任何的依赖  
    6测试类使用Test作为类名的后缀（不是必须）  
    7测试方法使用test作为方法名的前缀（不是必须）

8测试用例是用来达到测试想要的预期结果，而不能测试出程序的逻辑错误。

@Test:表示该方法是一个测试方法，返回值类型是void类型，无参数列表。

@Before:任何测试之前要做的操作

@After:任何测试之后要做的操作

@BeforeClass：所有方法运行前执行，静态方法，通常用于配置文件加载，初始化等。

@AfterClass：所有方法运行结束后执行，静态方法，通常用于释放资源等。

限时测试：

@Test(timeout = 1000)

Eg:测试无限循环

timeout属性表明设定的时间，单位为毫秒。

异常测试：

@Test(expected = ArithmeticException.**class**)

**public** **void** testDivide(){

JunitDemo1 *jd* = **new** JunitDemo1();

*jd*.divide(0);

}

标注的expected属性，将要检验的异常传递给它，这样JUnit框架就能自动帮我们检测是否抛出了指定的异常。

Runner运行器：

问题：Junit框架是如何进行测试的呢？

在 JUnit 中有很多个 Runner ，他们负责调用测试代码，每一个 Runner 都有各自的特殊功能，根据需要选择不同的Runner 来运行测试代码。如果没有指定Runner，Junit有一个默认Runner来运行测试代码。

想要指定Runner，需使用标注@RunWith。

注意：该标注是用来修饰类的，只要指定了Runner，那么该测试类中的方法都是被这个Runner调用。

参数化测试：

Eg:计算某一个数的平方。

public void square（int num）{

result = num \* num;

}

对上边的方法进行测试，如果编写测试用例，需考虑正数，负数，0三种情况：

@Before

public void setUp(){

jd.clear();

}

@Test

Public void test1(){

jd.square(4);

}

@Test

Public void test2(){

jd.square(0);

}

@Test

Public void test3(){

jd.square(-4);

}

为了简化类似这样的测试，Junit提出了参数化测试，编写一个方法，即可完成测试。

**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;

@RunWith(Parameterized.**class**)

**public** **class** JunitDemo1Test2 {

**private** **static** JunitDemo1 *jd* = **new** JunitDemo1();

**private** **int** param; //存储测试数据

**private** **int** result; //存储测试结果

**public** JunitDemo1Test2(**int** param,**int** result) {//属性初始化

**this**.param = param;

**this**.result = result;

}

@Parameters

**public** **static** Collection data(){ //方法为静态，返回值为Collection类型

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

{4,16},

{0,0},

{-4,16}

});

}

@Test

**public** **void** test() {

*jd*.square(param);

*assertEquals*(result, *jd*.getResult());

}

}

该案例中注意：数组中的数据是没有顺序的，但是进行实例化时候一定要按照数组的顺序。