**package** junitTest;

**import** java.util.Arrays;

**import** org.junit.Test;

**public** **class** Demo1 {

**public** **static** **void** main(String[] args) {

System.***out***.println("HelloWorld");

}

@Test

**public** **void** sort() {

**int**[] array = **new** **int**[] { 1, 10, 88, 4, 8, 5, 100, 34, 55 };

**for** (**int** i = 0; i < array.length - 1; i++) {

**for** (**int** j = i + 1; j < array.length; j++) {

**if** (array[i] > array[j]) {

**int** tmp = array[j];

array[j] = array[i];

array[i] = tmp;

}

}

}

System.***out***.println(Arrays.*toString*(array));

}

**public** **static** String sort1() {

**int**[] array = **new** **int**[] { 1, 2, 3, 4, 8, 5, 100, 34, 55 };

**for** (**int** i = 0; i < array.length - 1; i++) {

**for** (**int** j = i + 1; j < array.length; j++) {

**if** (array[i] > array[j]) {

**int** tmp = array[j];

array[j] = array[i];

array[i] = tmp;

}

}

}

**return** Arrays.*toString*(array);

}

}

**package** junitTest;

**import** org.junit.Test;

**import** junit.framework.Assert;

/\*\*

\* 在以前判断方法正误：

\* 1.如果需要测试，都需要在main方法上调用，实际上每个类的main方法也可看成一种测试手段

\* 2.结果都需要人工对比来判断程序正误

\*

\* 有了JUnit后：

\* 1.对方法的测试可模块化进行

\* 2.在测试类中规定程序的正误判断

\* 3.点击方法名测试一个方法，点击类名测试测试类中所有Test方法

\*

\* 如果只是判断结果：

\* 1.可以使用断言 Assert.assertSame(expected, actual);

\* 2.断言是JUnit框架的部分

\*

\* jUnit规范

\* 1.一个类如果需要测试，那么就应该对应着一个测试类，测试类的命名规范：被测试类名+Test

\* 2.一个被测试的方法一般对应着一个测试的方法，测试方法命名规范：test+被测试的方法名

\*

\* 偷懒技巧：

\* 1.直接在源代码的方法标记@Test，测试完记得删去

\* 2.@Test方法不能标记static方法，也不能带有形参，可以另外写个小方法在旁边，

\* 然后调用带有形参的方法

\*

\* **@author** Administrator

\*

\*/

// 测试类

**public** **class** Demo2 {

@Test

**public** **void** testSort() {

String ar = Demo1.*sort1*();

**if** (ar == "haha") {

**throw** **new** RuntimeException();

} **else** {

System.***out***.println("fuck");

}

Assert.*assertTrue*(ar != "haha");

}

}

**package** junitTest;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** org.junit.After;

**import** org.junit.Before;

**import** org.junit.Test;

/\*\*

\* 当所测试的方法需要准备环境时，为了让测试方法更纯粹，将准备测试环境和删除测试环境分离在外面

\* 在jUnit框架中，有@Before和@After两个注解用来分别处理

\* 不过，当批量测试几个需要准备环境的方法时，程序会不断调用准备和删除，此时，可以通过-

\* -@BeforeClass和@AfterClass注解来标记，但此时将准备和删除改为静态方法static

\*

\*/

**public** **class** Demo3 {

//准备测试的环境

@Before

//@BeforeClass

**public** **void** beforeRead(){

System.***out***.println("Before");

}

//读取文件数据，吧文件

@Test

**public** **void** readFile() **throws** IOException{

FileInputStream fileInputStream = **new** FileInputStream("F:\\a.txt");

**int** content = fileInputStream.read();

System.***out***.println("内容" + content);

fileInputStream.close();

}

//第二个测试的方法

@Test

**public** **void** haha(){

}

//清理测试的环境

@After

//@AfterClass

**public** **void** afterRead(){

System.***out***.println("After");

}

}