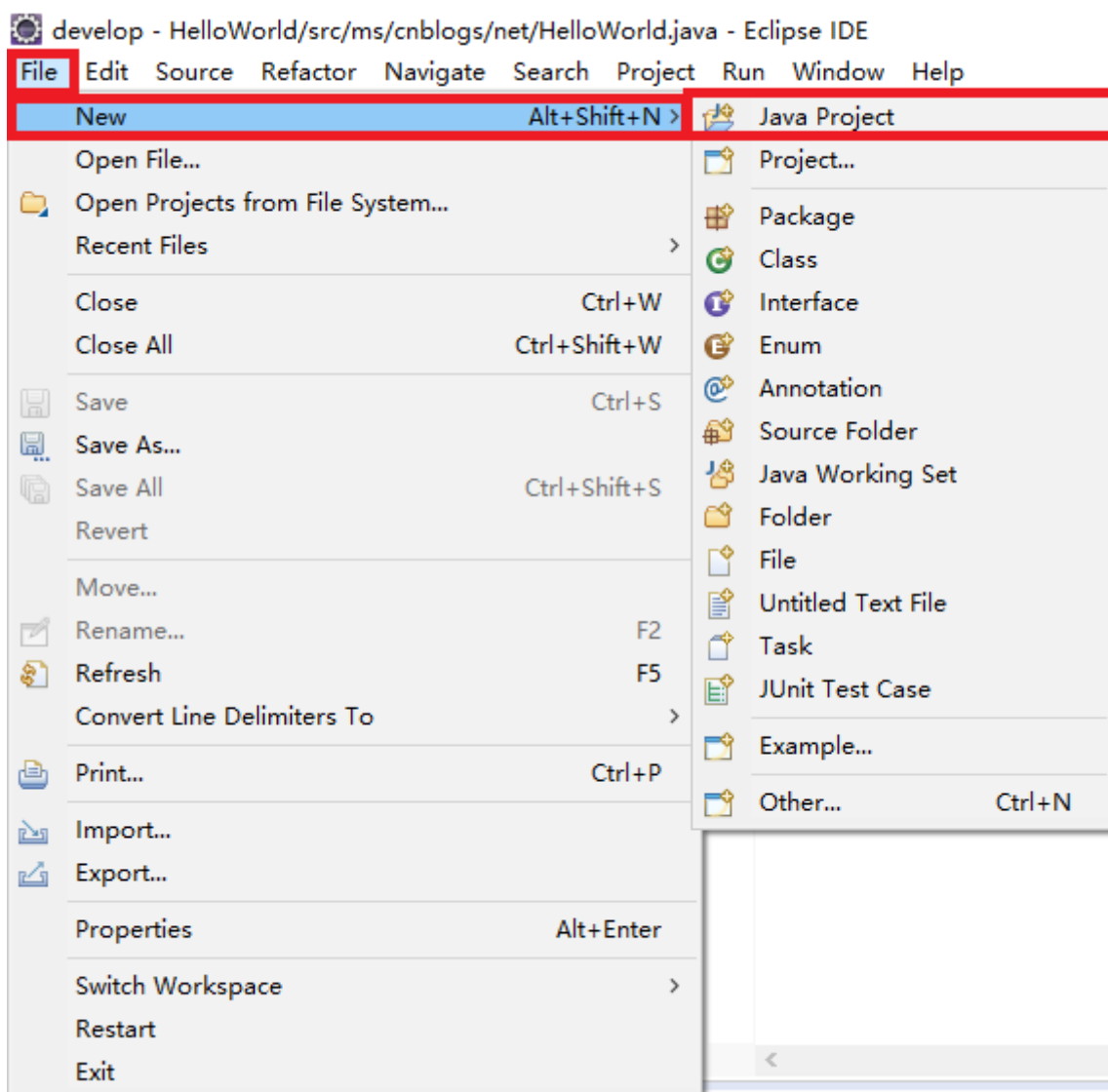


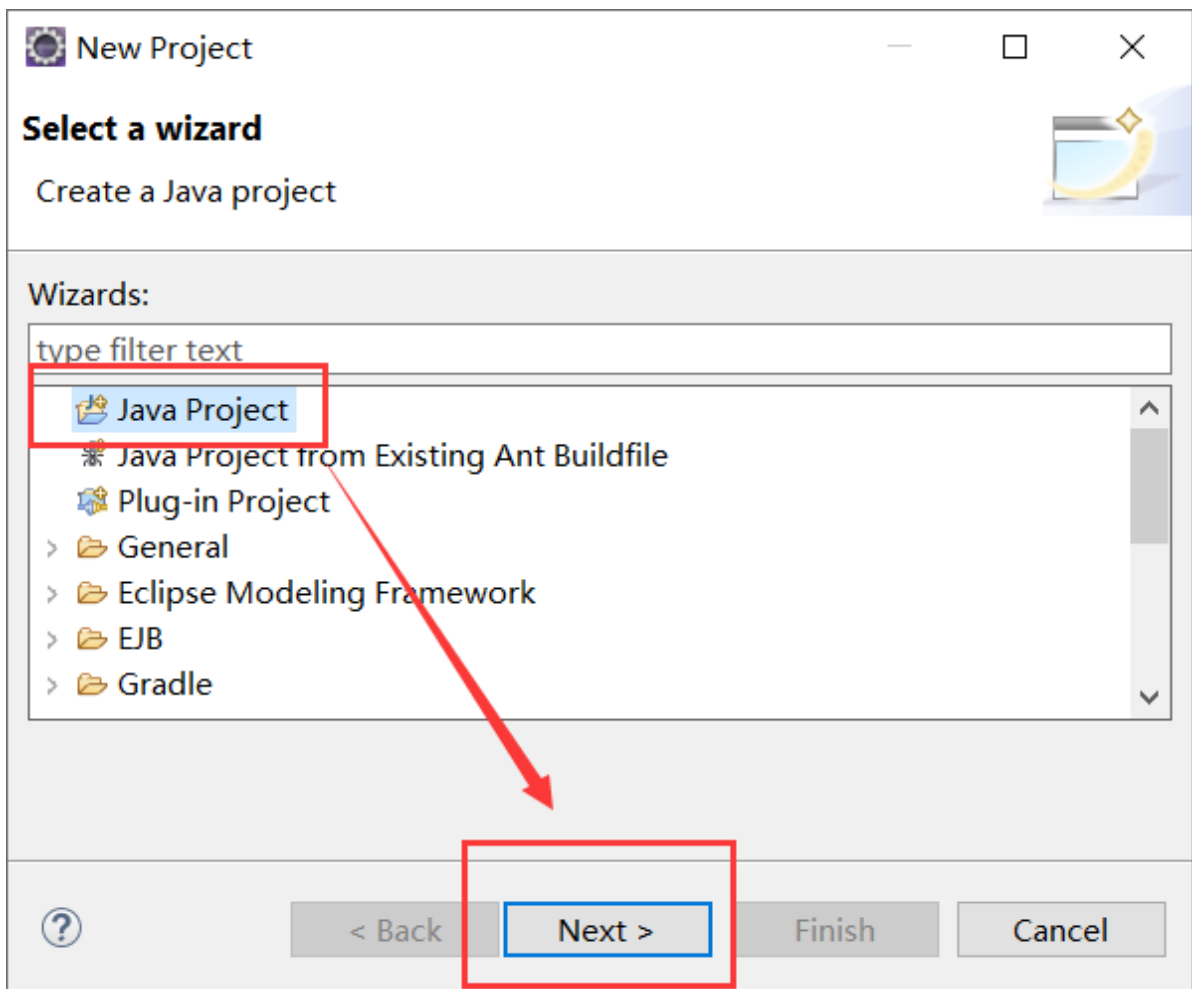
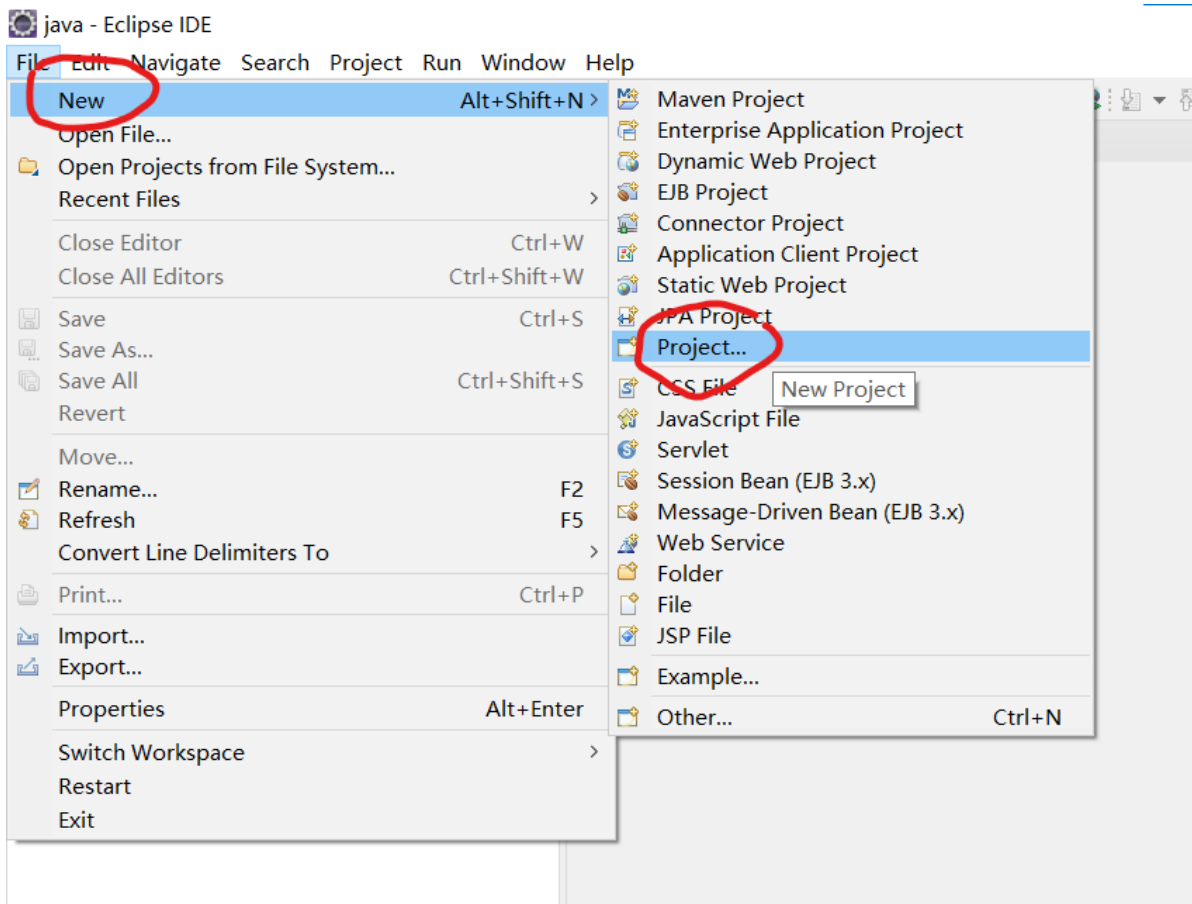
# Eclipse中使用JUnit进行单元测试

## 1、新建java项目

这是一般情况下新建Java项目的样式，有的Eclipse版本或者其他方面不同，界面可能不同



如果不是这个界面也没有关系，也可以在Project中找到



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: 单元测试 这里写项目名称，建议使用英文

☒ Use default location

Location: C:\Users\Administrator\Desktop\Code\java\单元测试 Browse...

JRE

☒ Use an execution environment JRE: JavaSE-11

☐ Use a project specific JRE: JavaJDK

☐ Use default JRE 'JavaJDK' and workspace compiler preferences [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

☐ Add project to working sets New...

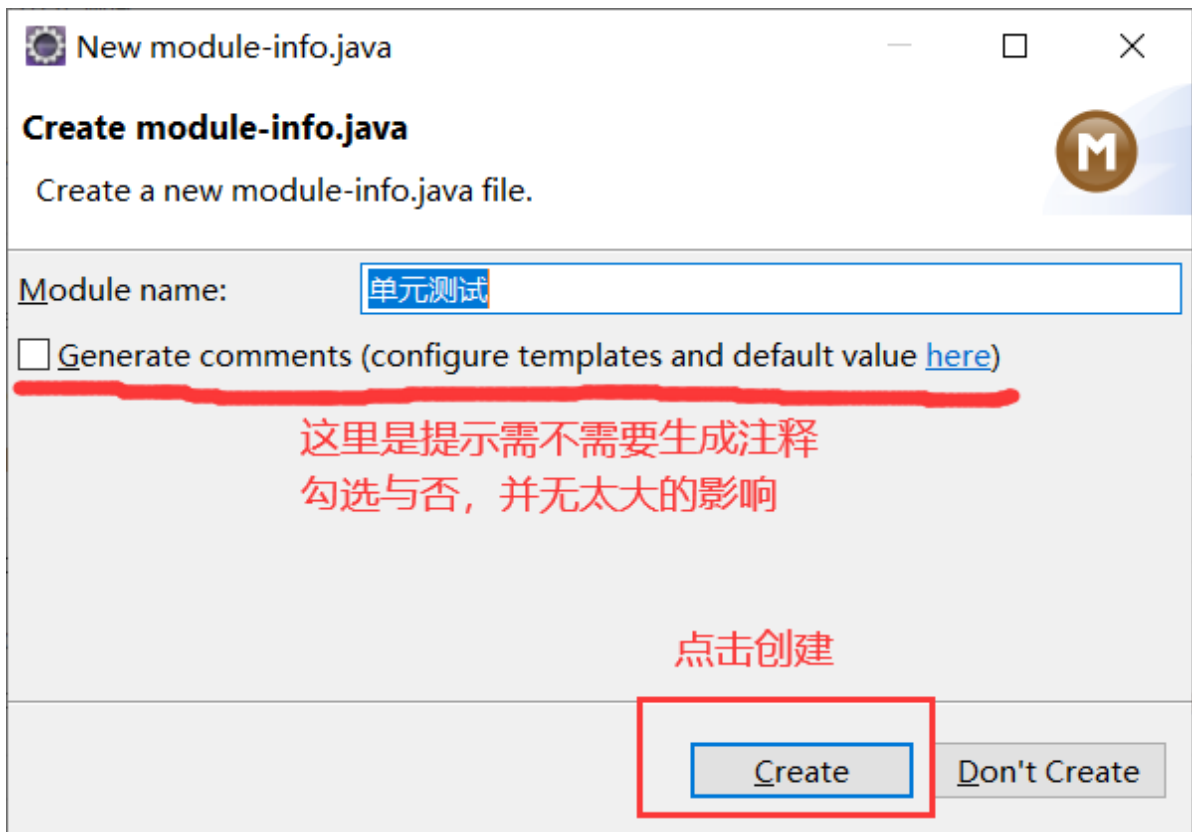
Working sets: Select...

点击Next可以继续设置其他内容，在这里就不演示了，可以自行查阅相关资料

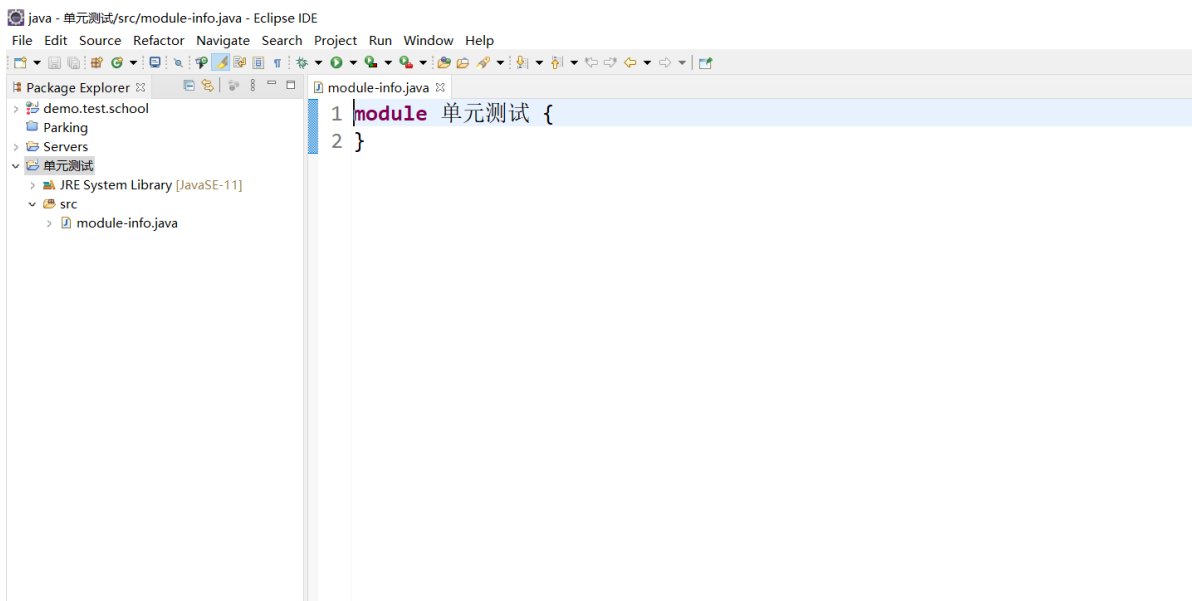
点击下一步

< Back Next > Finish Cancel

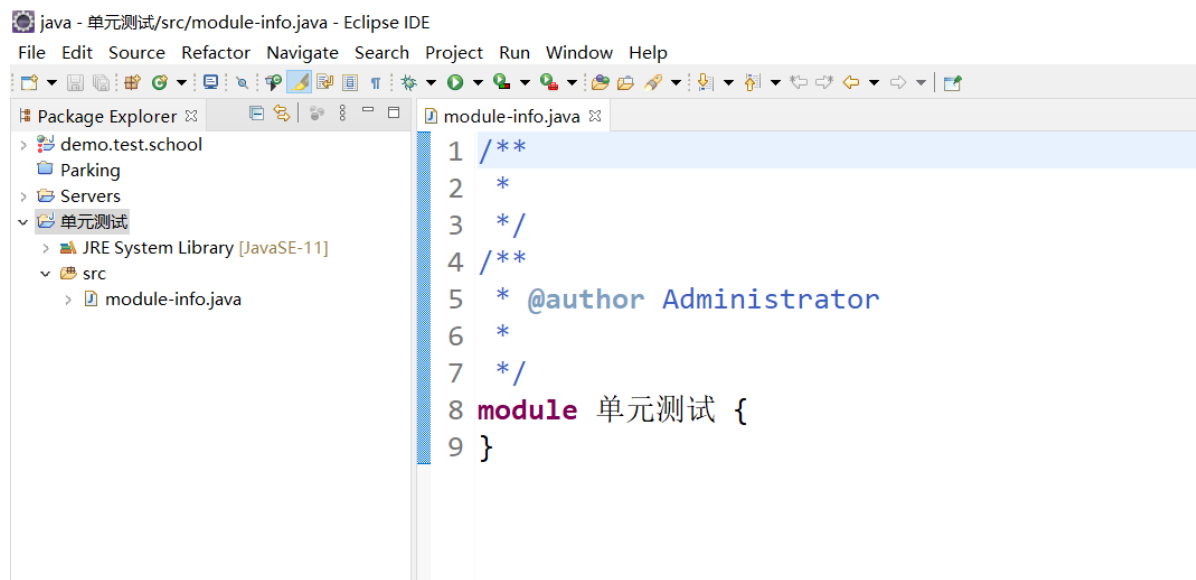
新版（2020）可能会出现以下提示



这是没有选择生成注释

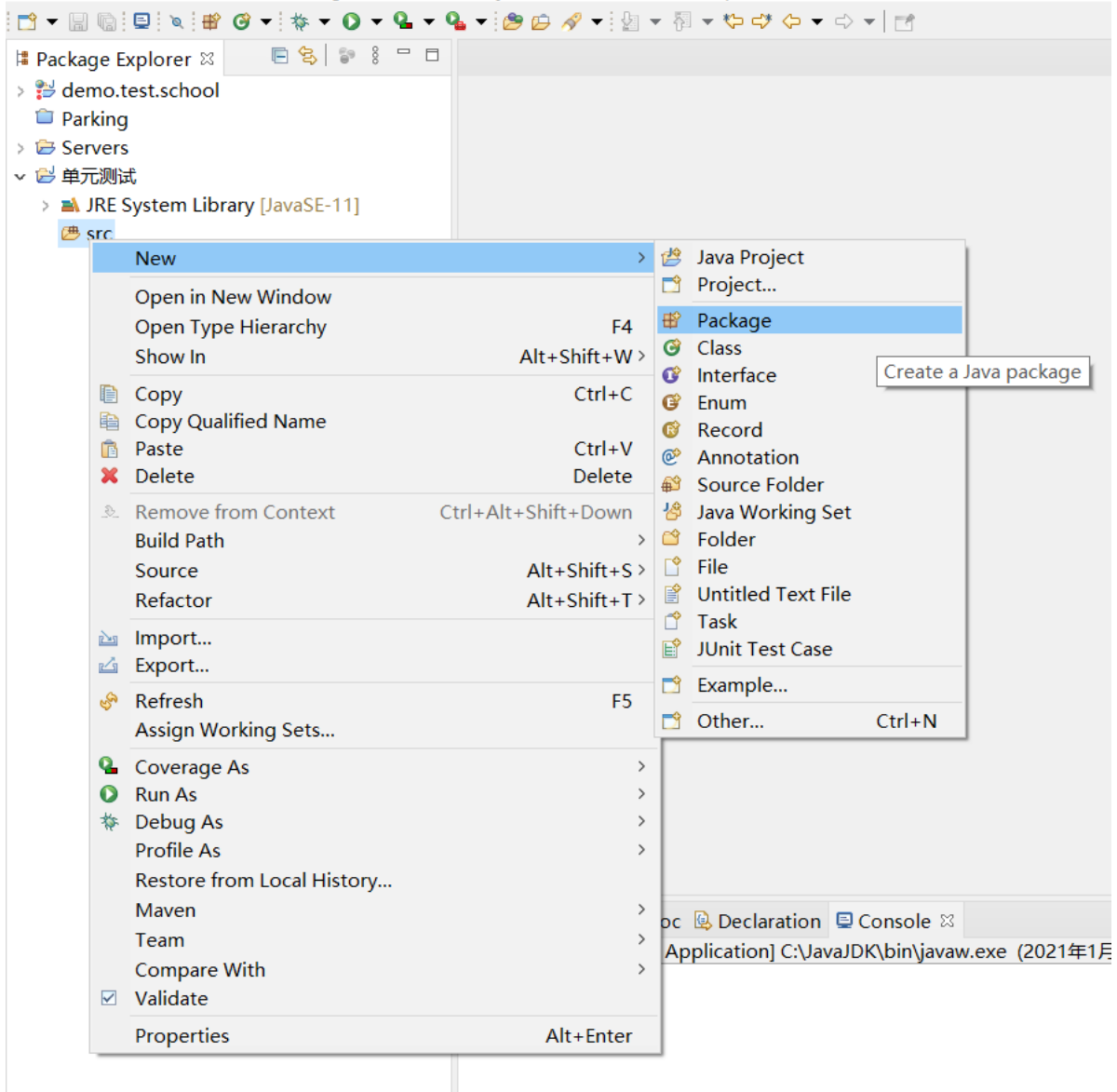


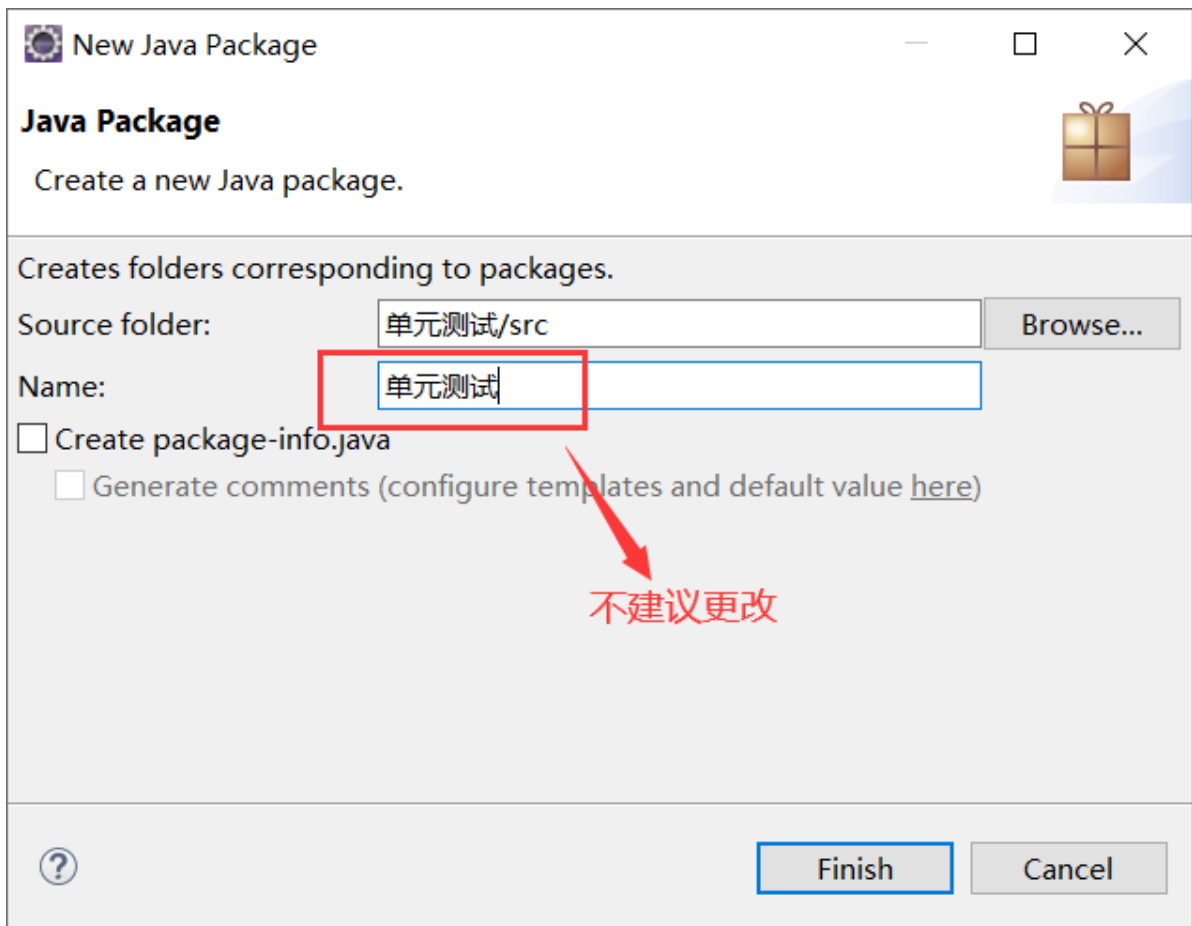
这是选择生成注释



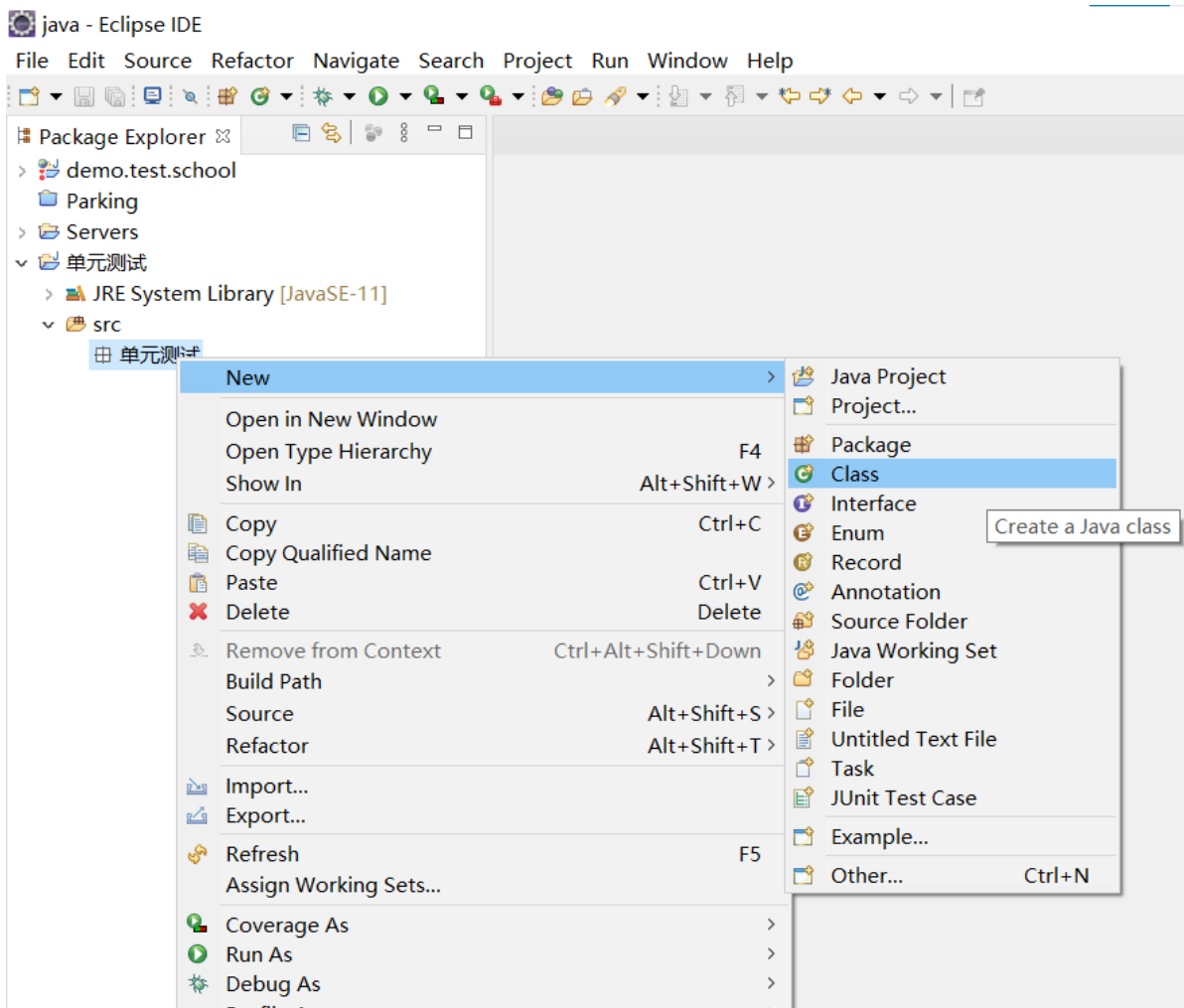
两者并无太多差别，但是如果选择生成注释的，中途还会提示是否继续生成代码视图选项，选择生成即可


右键src，创建一个包（Package）





## 继续创建class选项



 New Java Class

**Java Class**

Create a new Java class.

Source folder: 单元测试/src Browse...

Package: 单元测试 Browse...

☐ Enclosing type: 输入英文 建议第一个字母大写 Browse...

Name: Test

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add...  
Remove

Which method stubs would you like to create?

☒ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

? Finish Cancel

## 创建成功后的页面

java - 单元测试/src/单元测试/Test.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

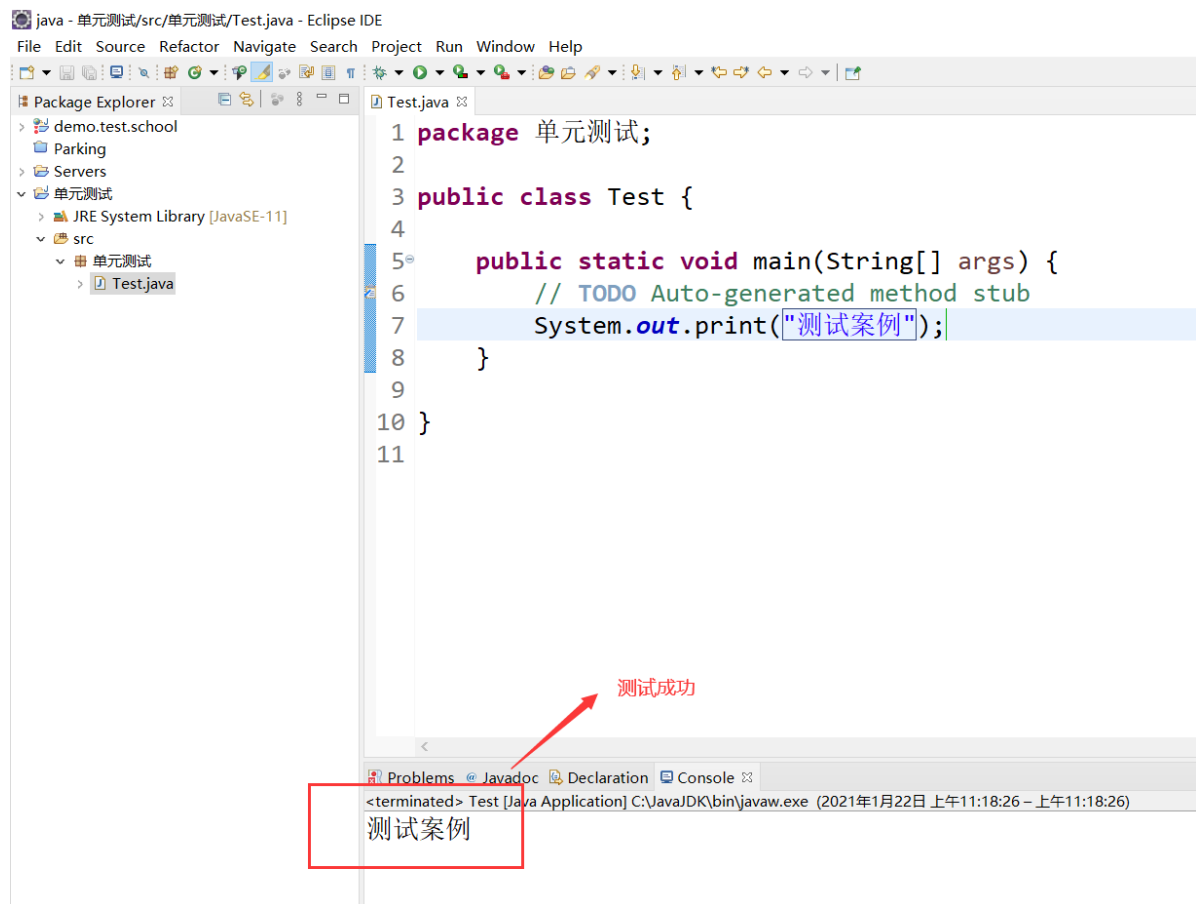
Package Explorer

- demo.test.school
  - Parking
  - Servers
  - 单元测试
    - JRE System Library [JavaSE-11]
    - src
      - 单元测试
        - Test.java

```
1 package 单元测试;  
2  
3 public class Test {  
4  
5     public static void main(String[] args) {  
6         // TODO Auto-generated method stub  
7  
8     }  
9  
10 }  
11
```

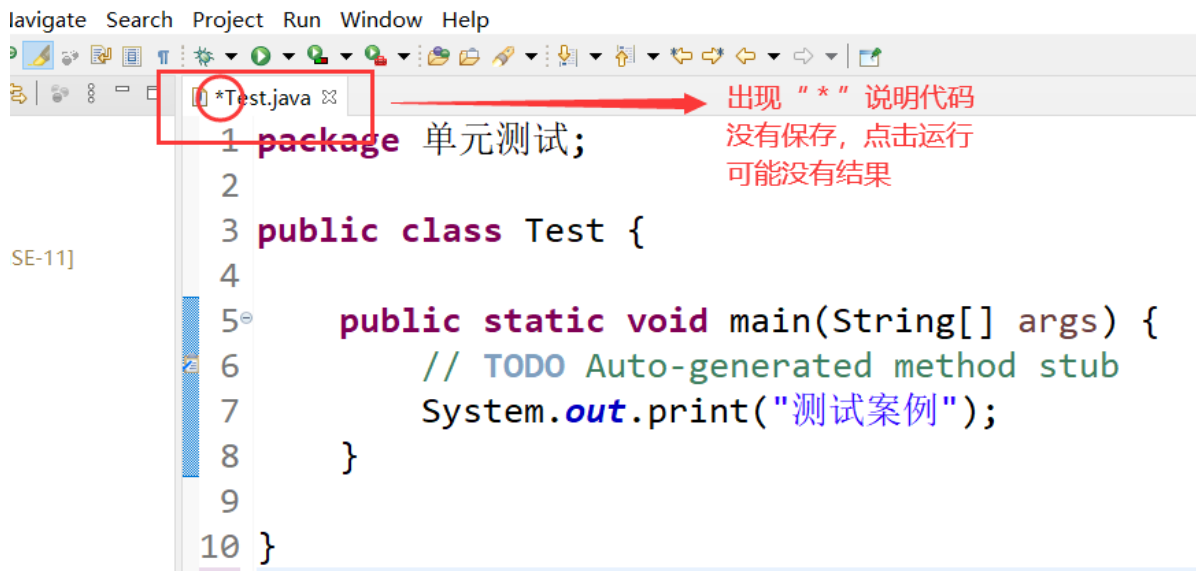
先写一个简单的输出语句来测试一下





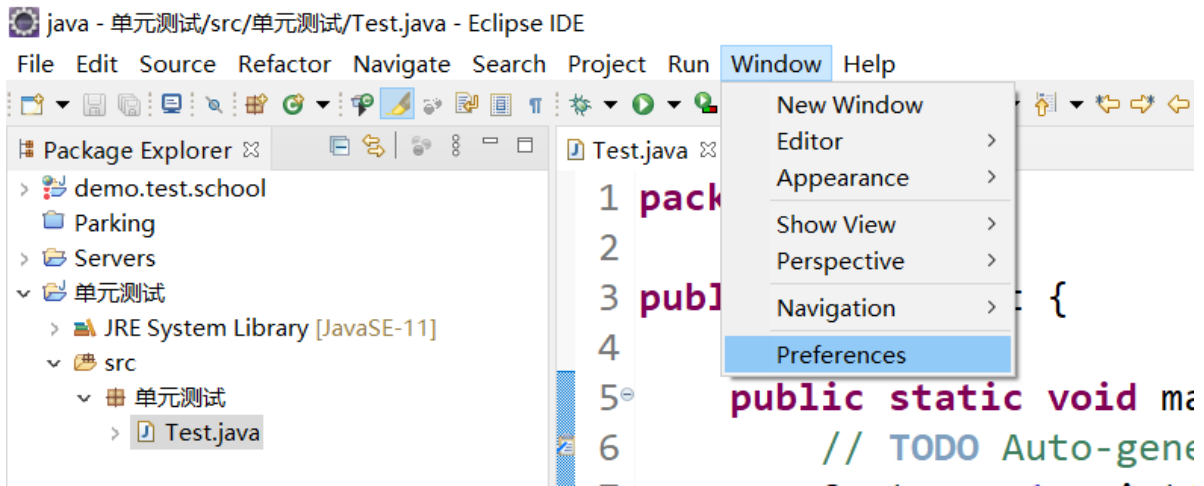
**注意:**

在写完代码运行后，可能会出现点击运行而无法自动保存，导致程序运行失败的结果

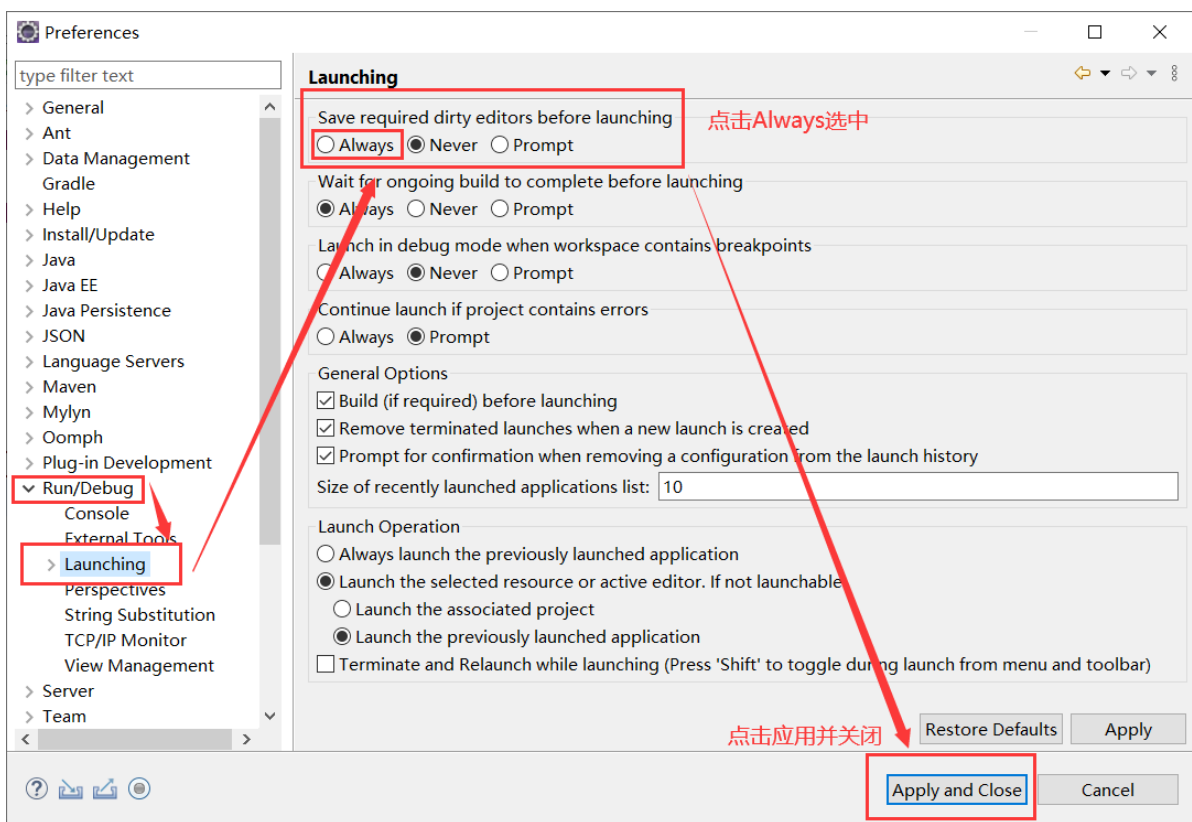


这种情况需要更改Eclipse的运行设置

点击Window中的Preferences



找到Run/Debug, 点击Launching, 将第一个选项改为Always应用即可防止以上问题



## 2、导入JUnit单元测试导入项目

### 1、编写好用于测试的案例

编写Calculator类, 能够简单实现加减乘除、平方、开方的计算器类, 然后对这些功能进行单元测试。这个类并不是很完美, 故意保留了一些Bug用于演示, 这些Bug在注释中都有说明。

实例代码:

```

package 单元测试;

public class Calculator {
    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 subtract(int n) {
        result = result - 1; //Bug: 正确的应该是
    结果=result-n
    }
    public void multiply(int n) {
    } // 此方法尚未写好
    public void divide(int n) {
        result = result / n;
    }
    public void square(int n) {
        result = n * n;
    }
    public void squareRoot(int n) {
        for (; ; ) ; //Bug : 死循环
    }
    public void clear() { // 将结果清零
        result = 0;
    }
    public int getResult() {
        return result;
    }
}

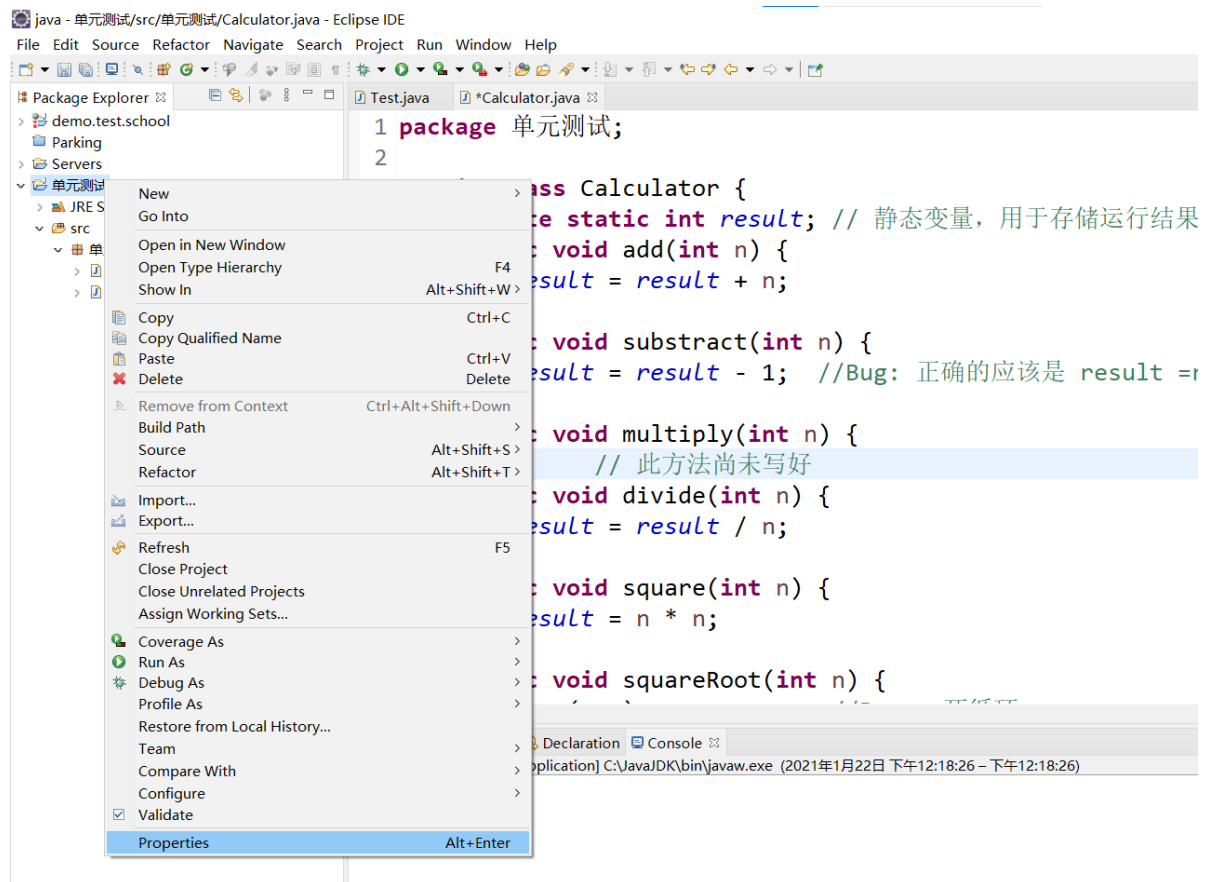
```

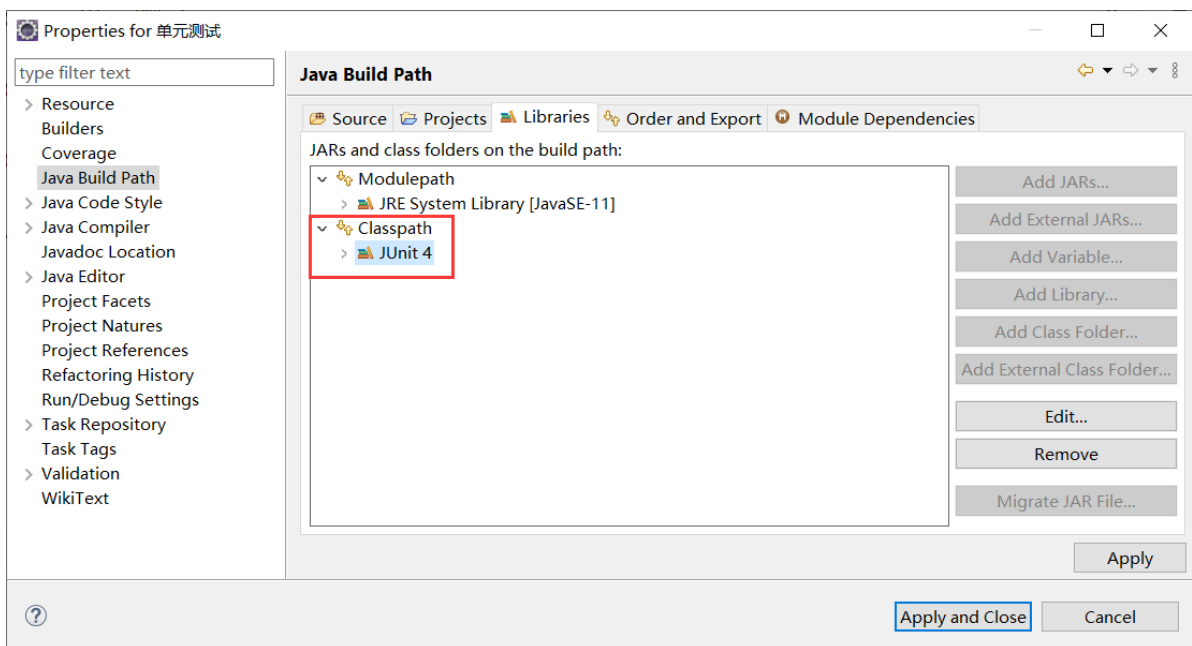
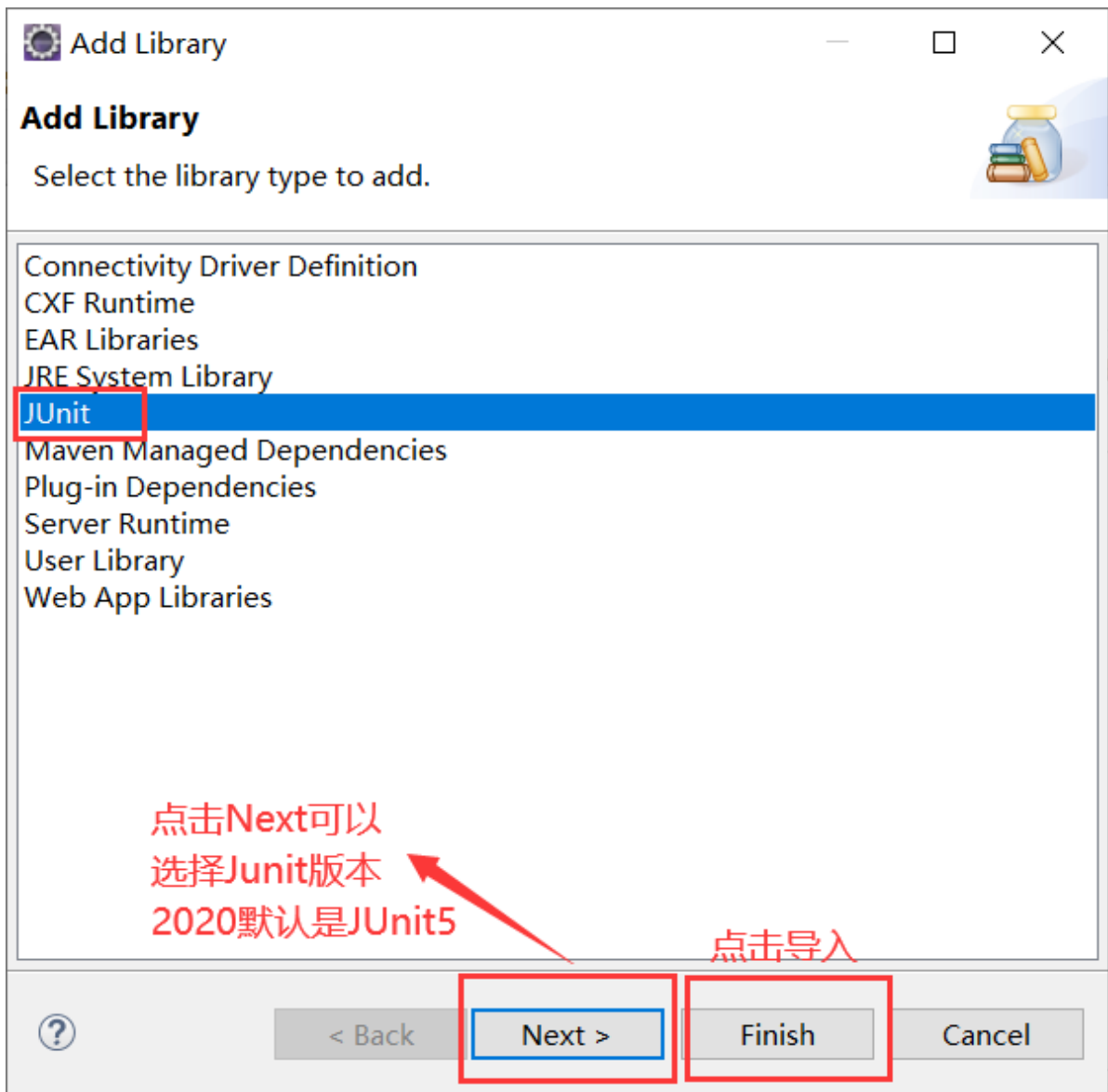
## 2、将JUnit单元测试包导入项目

右键项目，点击属性

注意:

是点击最外层的项目，而不是src，也不是里面的包（Package）和类（Class）

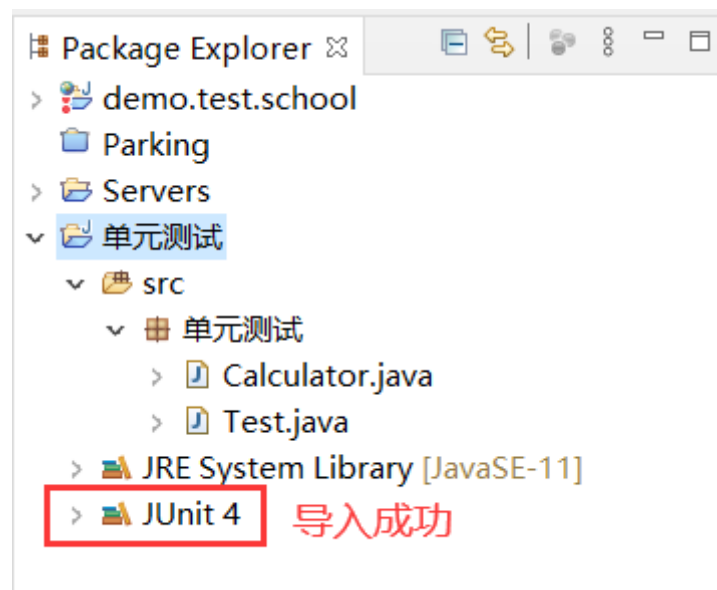




**特别注意:**

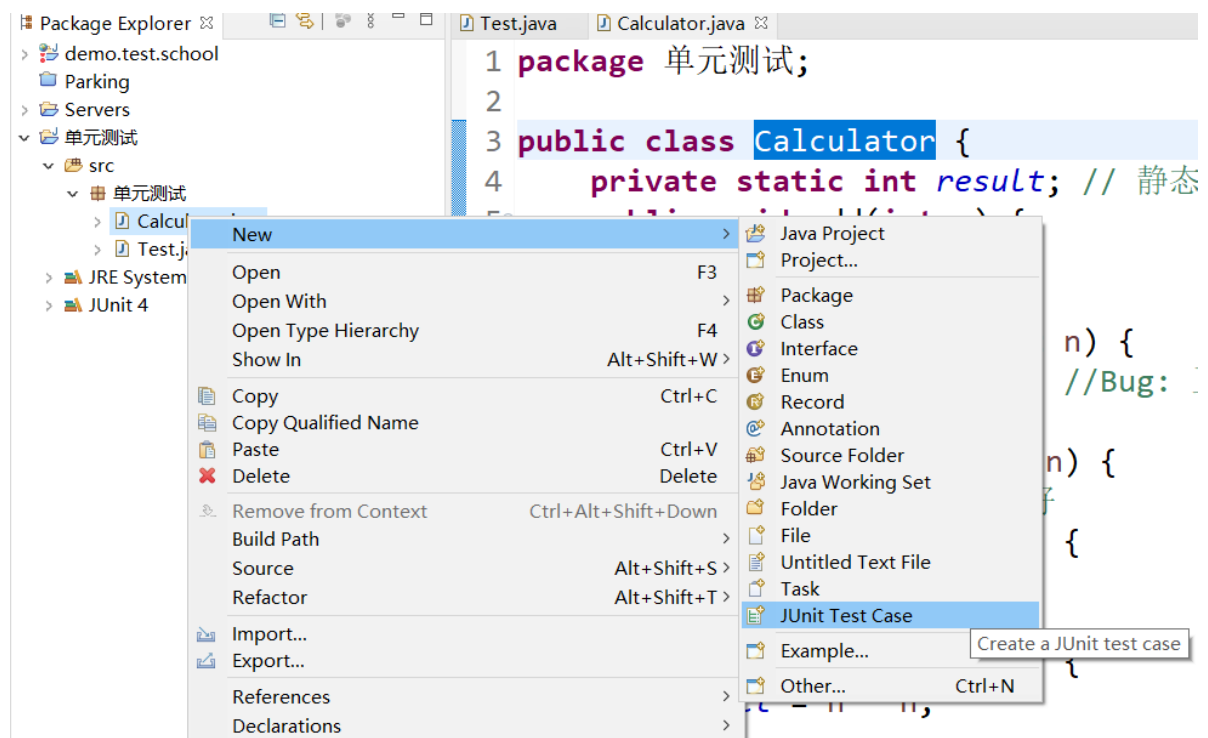
在导入JUnit时，如果导入的路径为Modulepath，可能会出现无法运行的情况

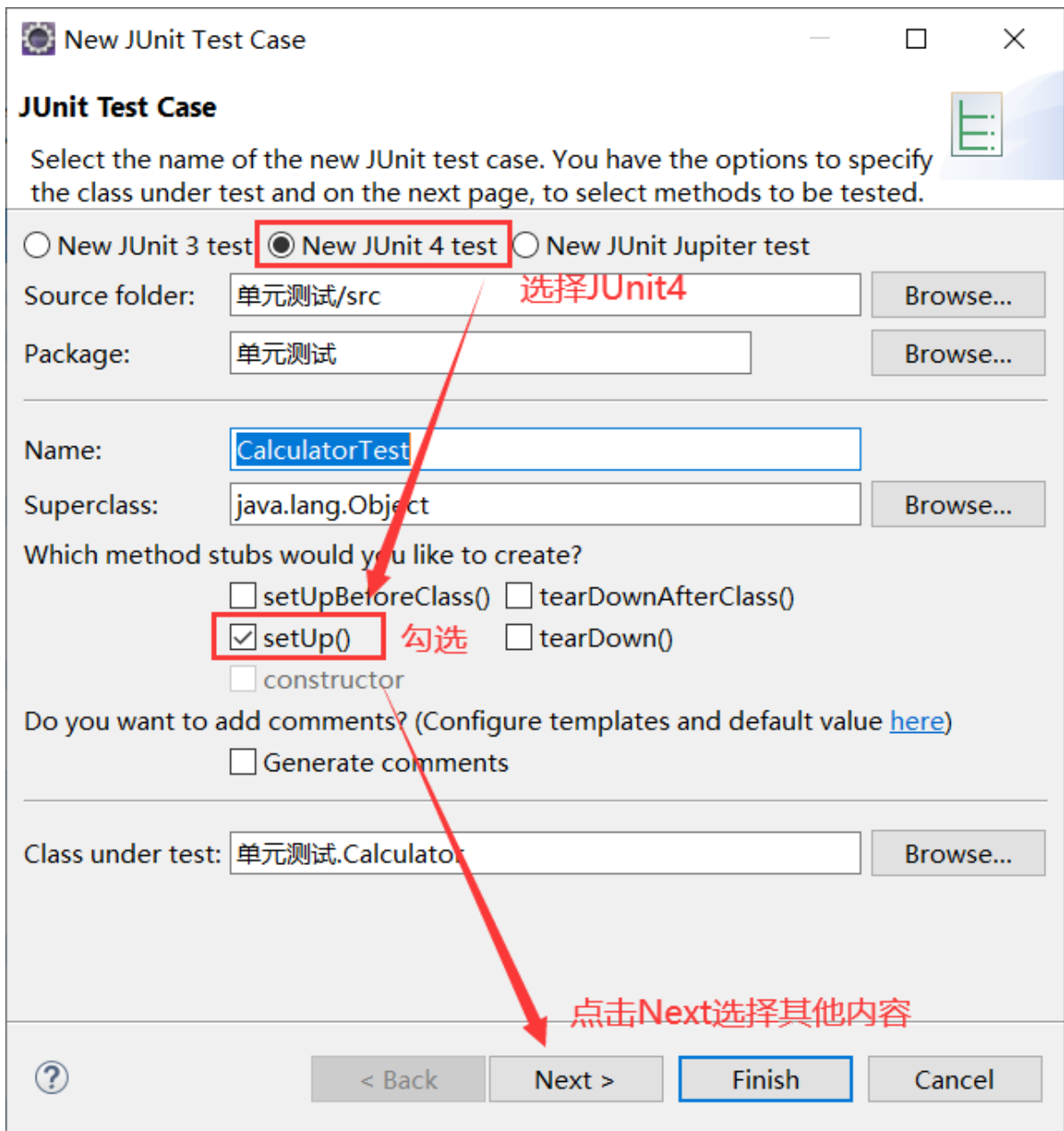
如果无法运行，可以将JUnit导入到Classpath中



### 3、生成JUnit框架

右键Calculator，点击JUnit Test Case



The image shows a 'New JUnit Test Case' dialog box. At the top, it says 'JUnit Test Case' and provides instructions: 'Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.' Below this, there are three radio buttons: 'New JUnit 3 test', 'New JUnit 4 test' (which is selected and highlighted with a red box), and 'New JUnit Jupiter test'. A red arrow points from this selection to the text '选择JUnit4'. The 'Source folder' is set to '单元测试/src' and the 'Package' is '单元测试', both with 'Browse...' buttons. The 'Name' field contains 'CalculatorTest'. The 'Superclass' is 'java.lang.Object' with a 'Browse...' button. Under 'Which method stubs would you like to create?', there are four checkboxes: 'setUpBeforeClass()', 'tearDownAfterClass()', 'setUp()' (which is checked and highlighted with a red box), and 'tearDown()'. A red arrow points from the 'setUp()' checkbox to the text '勾选'. Below this is a checkbox for 'constructor'. The question 'Do you want to add comments?' is followed by a checkbox for 'Generate comments' and a link 'here'. The 'Class under test' is '单元测试.Calculator' with a 'Browse...' button. At the bottom, there are four buttons: '< Back', 'Next >' (highlighted with a blue border), 'Finish', and 'Cancel'. A red arrow points from the 'Next >' button to the text '点击Next选择其他内容'.

**New JUnit Test Case**

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

☐ New JUnit 3 test ☒ **New JUnit 4 test** ☐ New JUnit Jupiter test

Source folder: 单元测试/src **选择JUnit4** Browse...

Package: 单元测试 Browse...

Name: CalculatorTest


Superclass: java.lang.Object Browse...

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()  
☒ **setUp()** **勾选** ☐ tearDown()  
☐ constructor

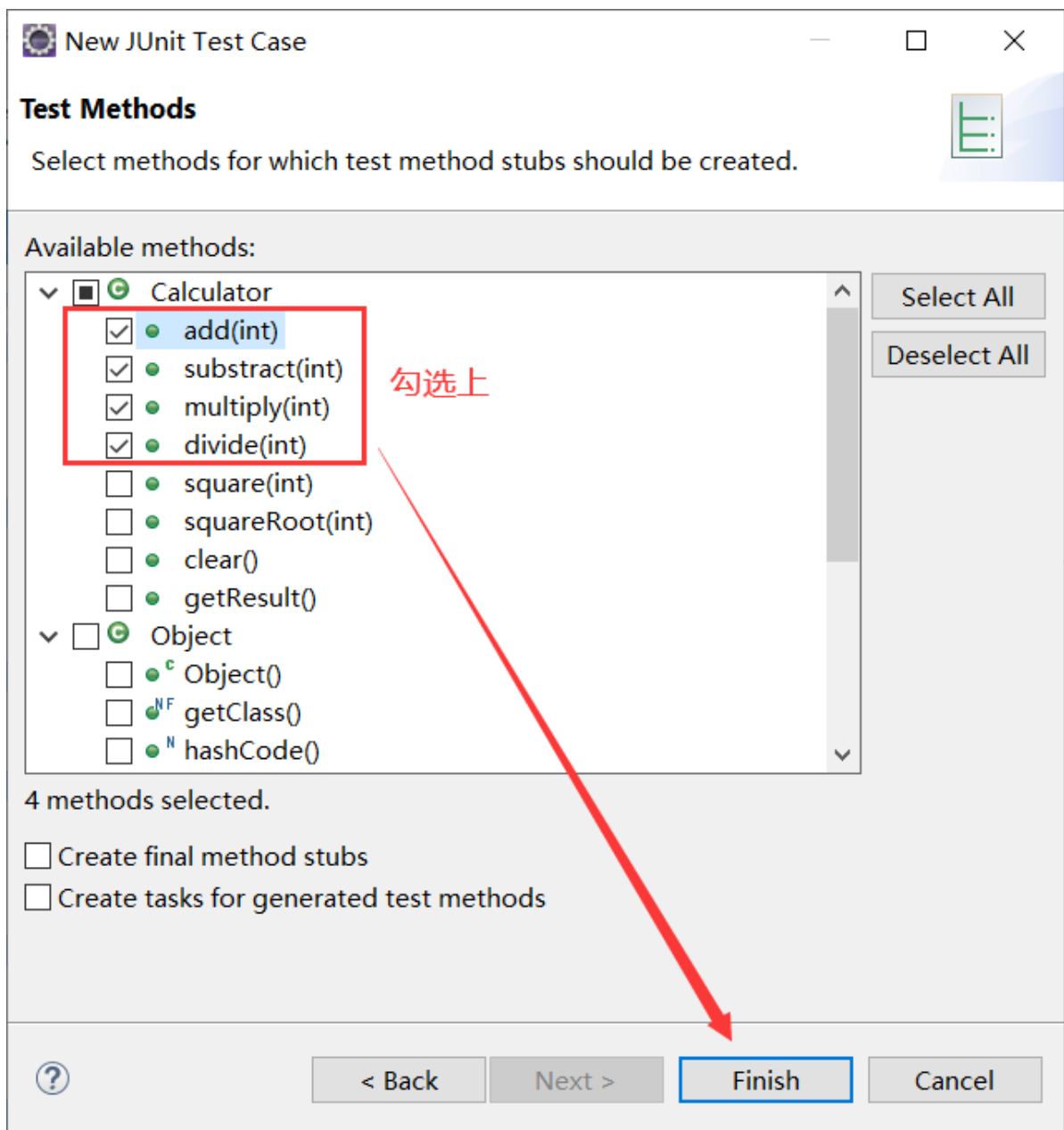
Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

Class under test: 单元测试.Calculator Browse...

 < Back **Next >** Finish Cancel

**点击Next选择其他内容**

勾选“加、减、乘、除”四个方法



add 加   subtract减   multiply 乘   divide除

系统会自动生成一个新的类CalculatorTest，里面包含一些空的测试用例。

现在需要将这些测试用例稍作修改即可使用。

完整的CalculatorTest代码如下：

```
package 单元测试;  
  
import static org.junit.Assert.*;  
  
import org.junit.Before;
```



```
import org.junit.Test;

public class CalculatorTest {
    private static Calculator calculator = new
calculator();

    @Before
    public void setUp() throws Exception {
        calculator.clear();
    }

    @Test
    public void testAdd() {
        calculator.add(2);
        calculator.add(3);
        assertEquals(5, calculator.getResult());
    }

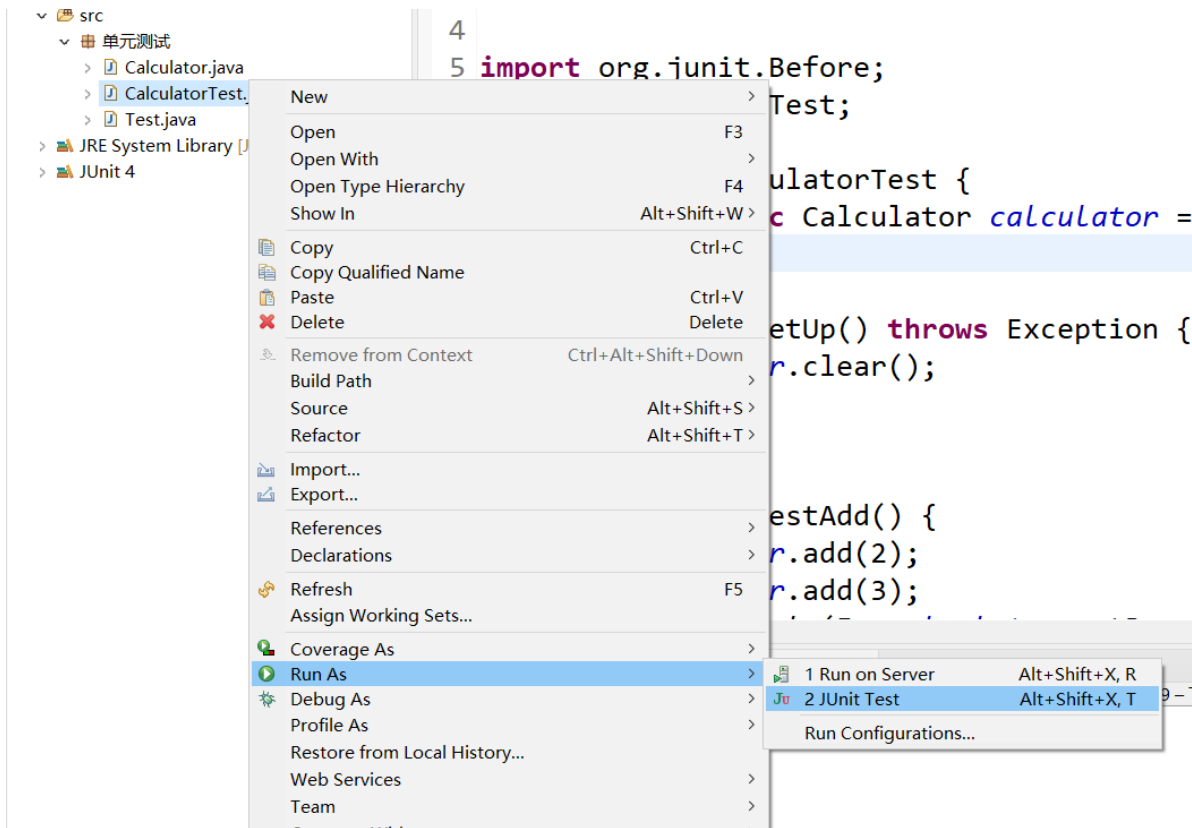
    @Test
    public void testSubtract() {
        calculator.add(10);
        calculator.subtract(2);
        assertEquals(8, calculator.getResult());
    }

    @Test
    public void testMultiply() {
    }

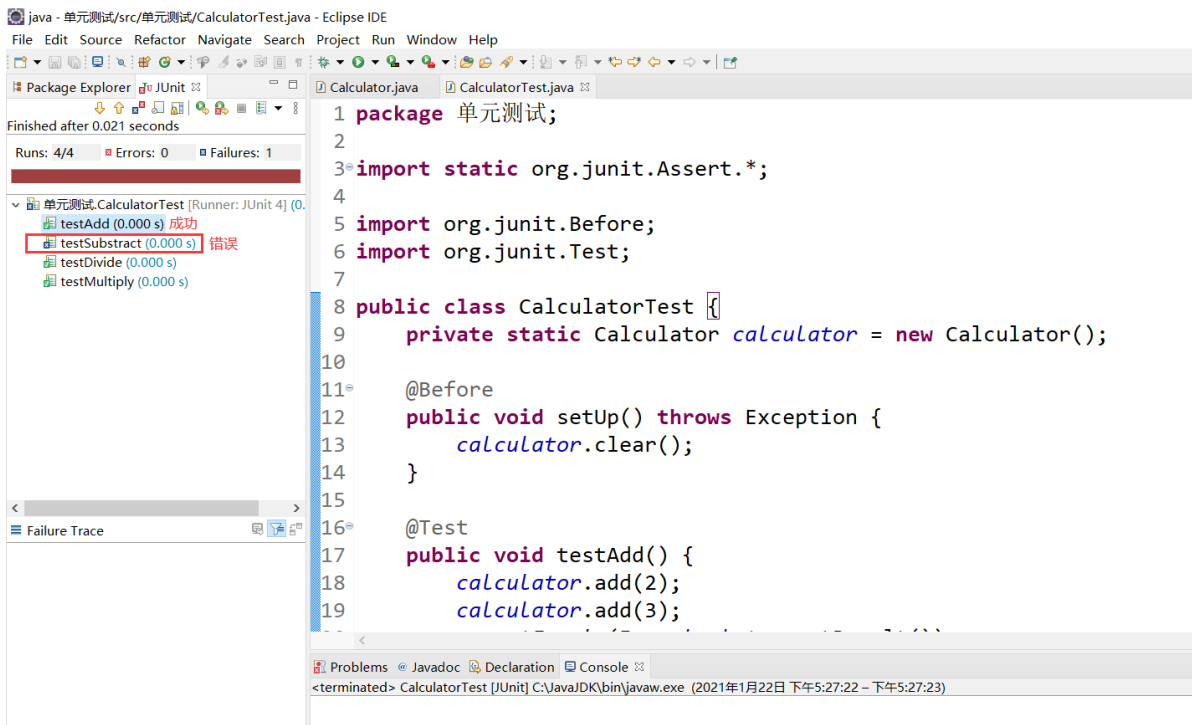
    @Test
    public void testDivide() {
        calculator.add(8);
        calculator.divide(2);
        assertEquals(4, calculator.getResult());
    }
}
```

## 4、运行测试代码

在CalculatorTest右键，点击Run As→JUnitTest运行测试案例



## 运行结果



进度条是红颜色表示发现错误，具体的测试结果在进度条上面有表示“共进行了4个测试，其中一个测试失败”

注意:

本次测试主要借鉴引用了CSDN中的内容

原文链接如下

<https://www.iteye.com/blog/avanry-669964>

后续主要学习建议参考上述链接

## 3、其他测试代码展示(源于网络)

### 案列一

新建测试类Tc\_Account

```
package 单元测试;

import org.junit.After;
import org.junit.Before;
import junit.framework.TestCase;
public class Tc_Account extends TestCase {

    public Tc_Account(String arg0)
    {
        super(arg0);
    }
    @Before
    public void setUp() throws Exception {
        super.setUp();
    }
    public void testDeposit(){
        Account account=new Account();
        assertEquals("Account should start with no
funds.",1,account.balance());
    }
}
```

```

        account.deposit(5);
        assertEquals("Account should reflect
deposit.", 7, account.balance());
    }

    public void testwithdraw(){
        Account account=new Account();
        account.deposit(5);
        account.withdraw(3);
        assertEquals("Account should reflect
withdraw.", 3, account.balance());
    }

    @After
    public void tearDown() throws Exception {
        super.tearDown();
    }
}

```

新建Account类，实现银行的余额、存款、取款：

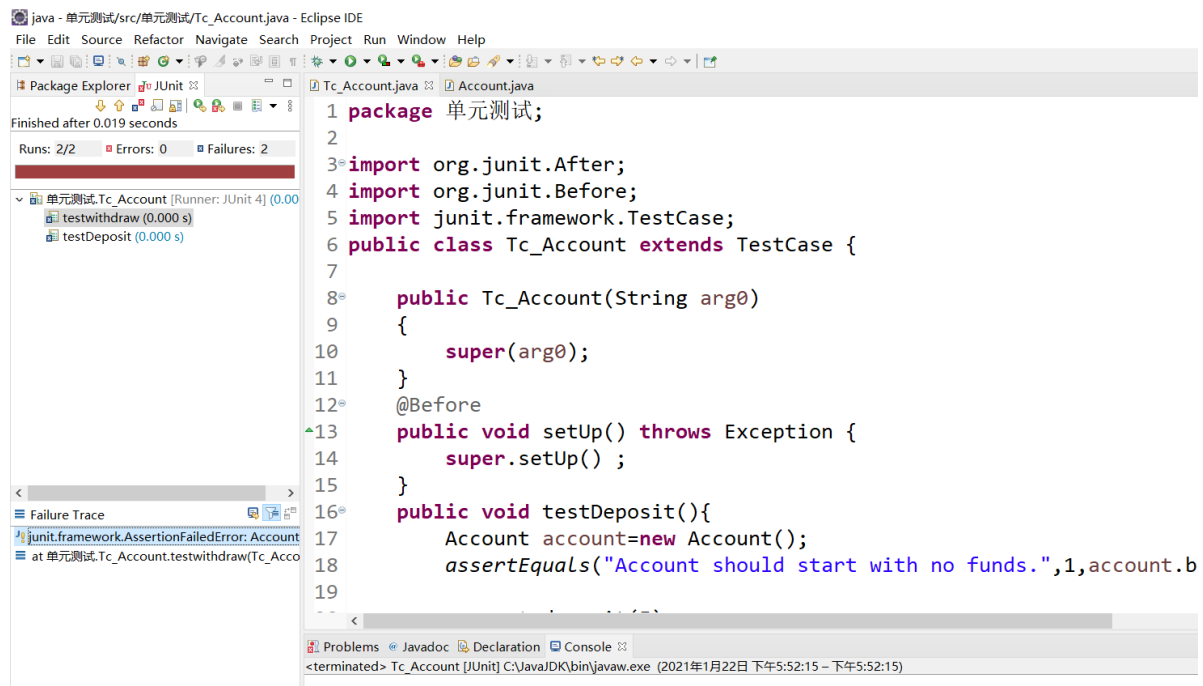
```

package 单元测试;

public class Account {
    protected int balance;
    public int balance(){
        return balance;
    }
    public void deposit(int amount){
        balance+=amount;
    }
    public void withdraw(int amount){
        balance-=amount;
    }
}

```

运行结果



## 调整原有代码

```
package 单元测试;

import org.junit.After;
import org.junit.Before;
import junit.framework.TestCase;
public class Tc_Account extends TestCase {

    public Tc_Account(String arg0)
    {
        super(arg0);
    }
    @Before
    public void setUp() throws Exception {
        super.setUp() ;
    }
    public void testDeposit(){
        Account account=new Account();
        assertEquals("Account should start with no
funds.",0,account.balance());

        account.deposit(5);
    }
}
```

```

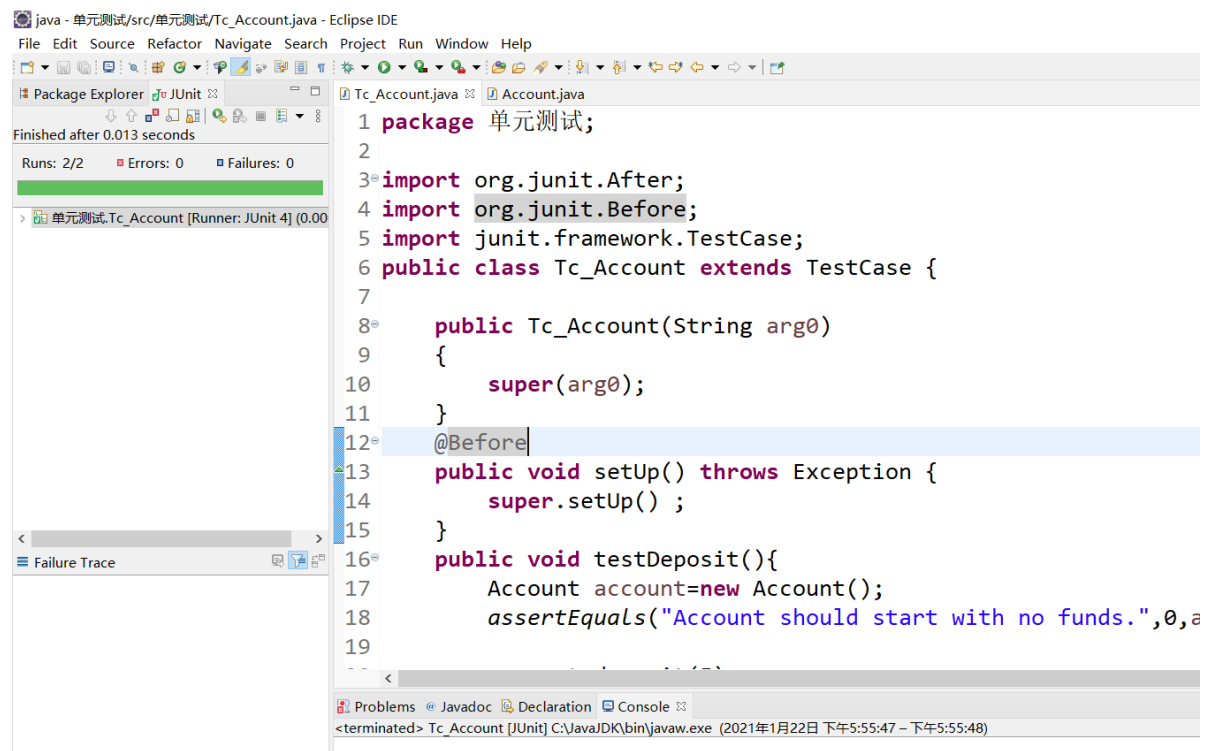
        assertEquals("Account should reflect
deposit.", 5, account.balance());
    }

    public void testwithdraw(){
        Account account=new Account();
        account.deposit(5);
        account.withdraw(3);
        assertEquals("Account should reflect
withdarw.", 2, account.balance());
    }

    @After
    public void tearDown() throws Exception {
        super.tearDown();
    }
}

```

运行结果，测试通过



# 案例二

## 测试类

```
package 单元测试;

public class Calculate {
    public int Add(int x,int y) {
        return x + y;
    }
}
```

## 编写测试用例

```
package 单元测试;

import static org.junit.Assert.*;

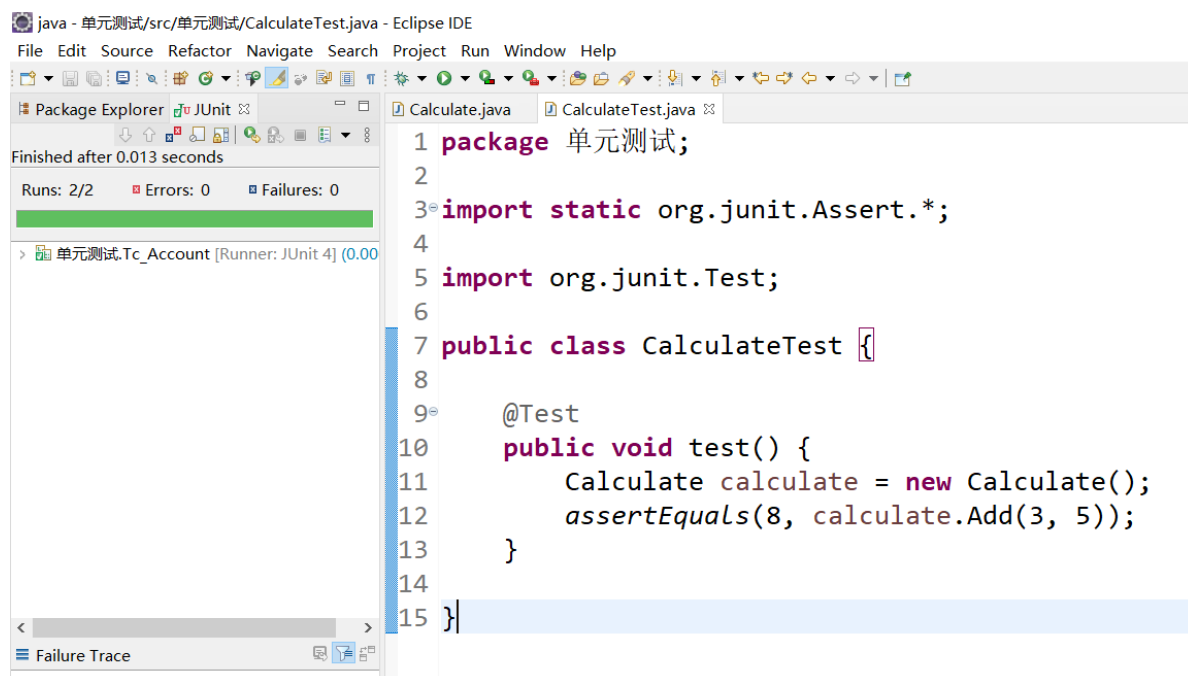
import org.junit.Test;

public class CalculateTest {

    @Test
    public void test() {
        Calculate calculate = new Calculate();
        assertEquals(8, calculate.Add(3, 5));
    }

}
```

## 运行结果



# 附录：

## eclipse快捷键

### 主要

Ctrl + 1 快速修复

Ctrl + D 删除当前行

Ctrl + Alt + ↓ 复制当前行到下一行(复制增加)

Ctrl + Alt + ↑ 复制当前行到上一行(复制增加)

Alt + ↓ 当前行和下面一行交互位置

Alt + ↑ 当前行和上面一行交互位置(同上)

Alt + ← 前一个编辑的页面

Alt + → 下一个编辑的页面(当然是针对上面那条来说了)

Alt + Enter 显示当前选择资源的属性

Shift + Enter 在当前行的下一行插入空行

Shift + Ctrl + Enter 在当前行插入空行

Ctrl + Q 定位到最后编辑的地方

Ctrl + L 定位在某行 【输入500，即快速锁定至第500行】

Ctrl + M 最大化当前的Edit或View (再按则反之)

Ctrl + / 注释当前行,再按则取消注释



Ctrl + O 快速显示 OutLine

Ctrl + T 快速显示当前类的继承结构

Ctrl + W 关闭当前Editor

Ctrl + K 参照选中的Word快速定位到下一个

Ctrl + E 快速显示当前Editor的下拉列表

Ctrl + / (小键盘) 折叠当前类中的所有代码

Ctrl + × (小键盘) 展开当前类中的所有代码

Ctrl + Shift + E 显示管理当前打开的所有的View的管理器(可以选择关闭,激活等操作)

Ctrl + J 正向增量查找

Ctrl + Shift + J 反向增量查找(和上条相同,只不过是后往前查)

Ctrl + Shift + F4 关闭所有打开的Editor

Ctrl + Shift + X 把当前选中的文本全部变为大写

Ctrl + Shift + Y 把当前选中的文本全部变为小写

Ctrl + Shift + F 格式化当前代码

Ctrl + Shift + P 定位到匹配的匹配符(譬如{ }) (从前面定位后面时,光标要在匹配符里面,后面到前面,则反之)

## 重构

Alt + Shift + R 重命名 (尤其是变量和类的重命名, 很便捷)

Alt + Shift + M 抽取方法 (这是重构里面最常用的方法之一)

Alt + Shift + C 修改函数结构(比较实用,有N个函数调用了这个方法,修改一次搞定)

Alt + Shift + L 抽取本地变量

Alt + Shift + F 把Class中的local变量变为field变量

Alt + Shift + I 合并变量

Alt + Shift + V 移动函数和变量

Alt + Shift + Z 撤销重构

## 编辑

全局 查找并替换 Ctrl + F  
全局 撤销 Ctrl + Z  
全局 复制 Ctrl + C  
全局 恢复上一个选择 Alt + Shift + ↓  
全局 剪切 Ctrl + X  
全局 快速修正 Ctrl + 1  
全局 内容辅助 Alt + / 【代码补全】  
全局 全部选中 Ctrl + A  
全局 删除选中代码 Delete  
全局 粘贴 Ctrl + V  
全局 重做 Ctrl + Y 【即Ctrl + Z 的逆向操作】

## 查看

全局放大 Ctrl + = 【可以放大或缩小代码】  
全局缩小 Ctrl + -

## 窗口

全局 激活编辑器 F12  
全局 切换编辑器 Ctrl + Shift + W  
全局 上一个编辑器 Ctrl + Shift + F6  
全局 上一个视图 Ctrl + Shift + F7  
全局 上一个透视图 Ctrl + Shift + F8  
全局 下一个编辑器 Ctrl + F6  
全局 下一个视图 Ctrl + F7  
全局 下一个透视图 Ctrl + F8  
文本编辑器 显示标尺上下文菜单 Ctrl + W  
全局 显示视图菜单 Ctrl + F10  
全局 显示系统菜单 Alt + -

# 导航

Java编辑器 打开结构 Ctrl + F3

全局 打开类型 Ctrl + Shift + T

全局 打开类型层次结构 F4

全局 打开声明 F3

全局 打开资源 Ctrl + Shift + R

全局 后退历史记录 Alt + ←

全局 前进历史记录 Alt + →

全局 上一个 Ctrl + ,

全局 下一个 Ctrl + .

Java编辑器 显示大纲 Ctrl + O

全局 在层次结构中打开类型 Ctrl + Shift + H

全局 转至匹配的括号 Ctrl + Shift + P

全局 转至上一个编辑位置 Ctrl + Q

Java编辑器 转至上一个成员 Ctrl + Shift + ↑

Java编辑器 转至下一个成员 Ctrl + Shift + ↓

文本编辑器 转至行 Ctrl + L

# 搜索

全局 出现在文件中 Ctrl + Shift + U

全局 打开搜索对话框 Ctrl + H

全局 工作区中的声明 Ctrl + G

全局 工作区中的引用 Ctrl + Shift + G

# 文本编辑

文本编辑器 改写切换 Insert

文本编辑器 上滚行 Ctrl + ↑

文本编辑器 下滚行 Ctrl + ↓

# 文件

全局 保存 Ctrl + S

全局 打印 Ctrl + P

全局 关闭 Ctrl + F4

全局 全部保存 Ctrl + Shift + S 【将已修改的所有java文件保存】

全局 全部关闭 Ctrl + Shift + F4

全局 属性 Alt + Enter

全局 新建 Ctrl + N

# 源代码

Java编辑器 格式化 Ctrl + Shift + F

Java编辑器 注释 Ctrl + / 【PS:再次点击即取消注释】

Java编辑器 组织导入 Ctrl + Shift + O 【自动导入所需要的jar包】