实验一 JUnit入门

1. 实验目的

初步了解和掌握单元测试工具JUnit。

1. 实验内容
2. 下载与安装
3. 下载和安装 JDK8和IntelliJ Idea（或Eclipse）

从网站<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>

下载安装Java SE 8

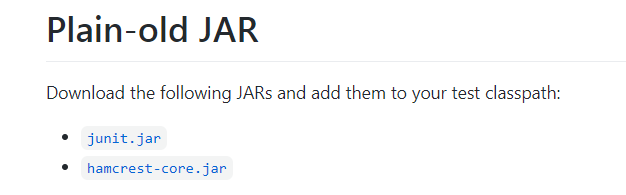
从网站<https://www.jetbrains.com/idea/>

下载安装IntelliJ Idea

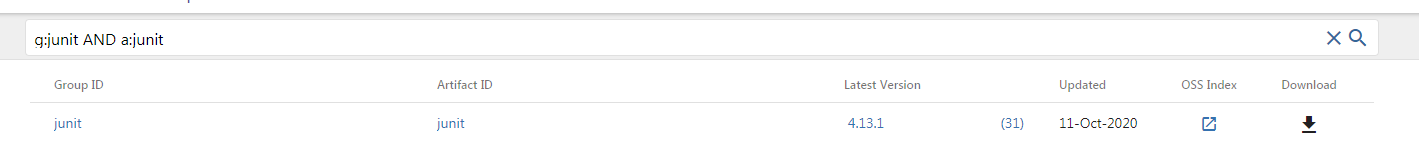
1. 下载JUnit4

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

junit.jar和hamcrest-core.jar



点击junit.jar，下载junit4.13.1.jar



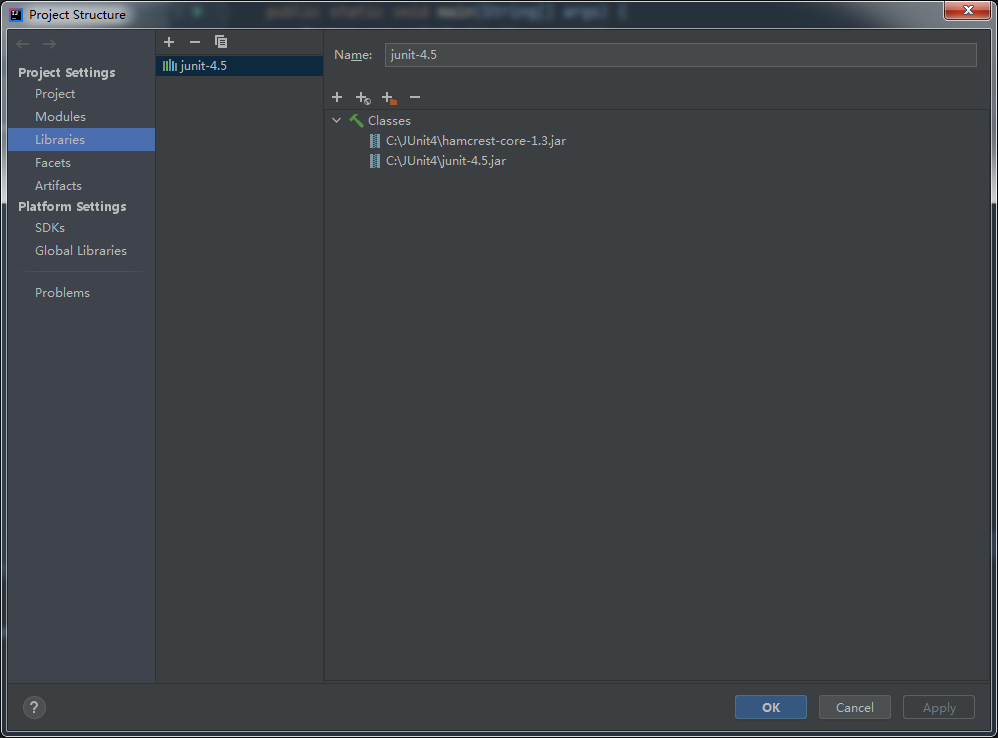
同样的方法下载hamcrest-core.jar

1. 将JUnit库导入工程

在IntelliJ Idea中新建工程

打开 File -> Project Structure （Ctrl + Shift + Alt + S）

单击 Libraries  -> "+" -> "Java" -> 选择下载的JUnit两个文件，点击OK



1. 测试安装是否成功

创建一个package，名为com.xjz.util

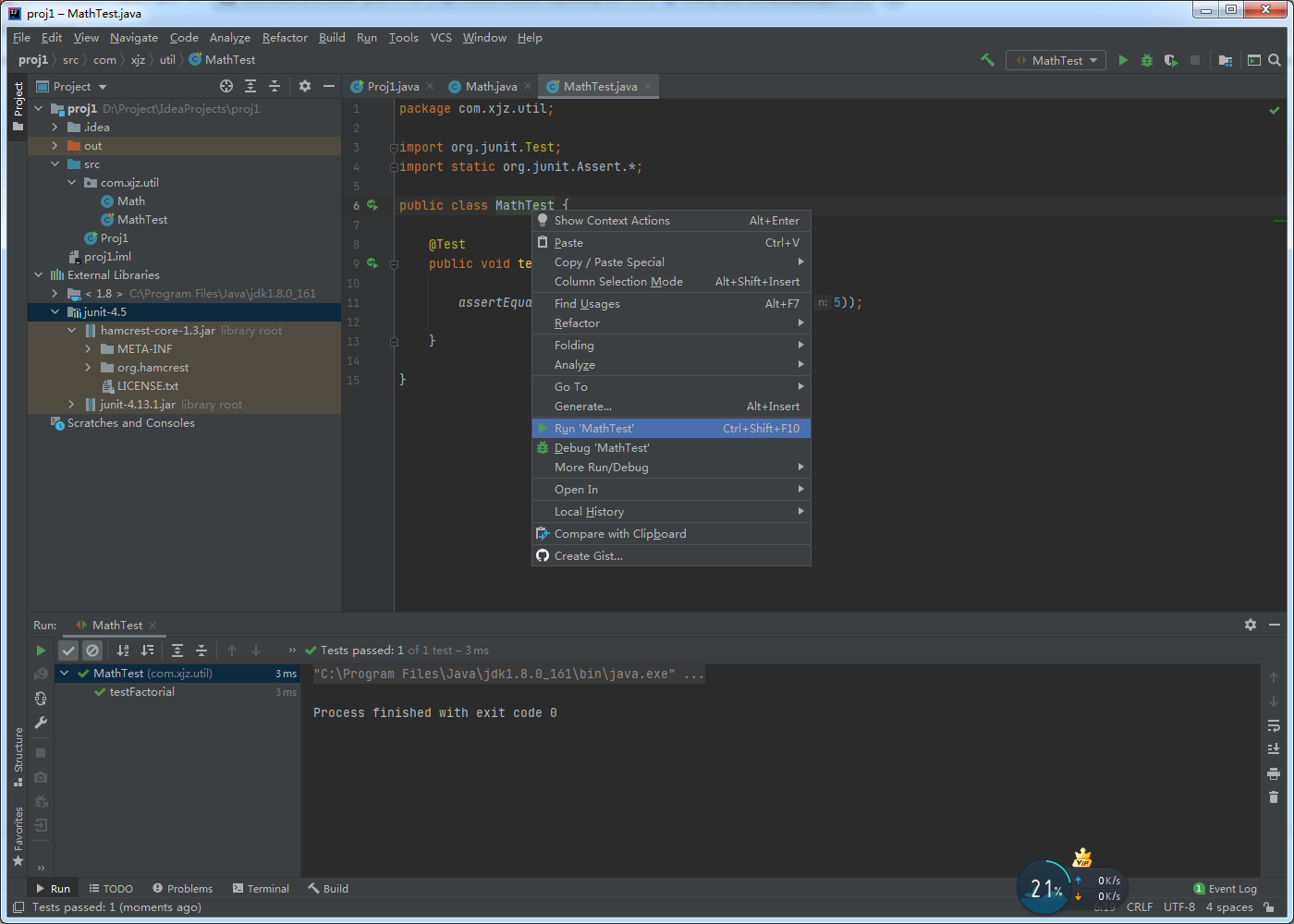
创建Math类，内容如下：

package com.xjz.util;  
  
public class Math {  
 public int factorial(int n) throws Exception {  
 if (n < 0) {  
 throw new Exception("负数没有阶乘");  
 } else if (n <= 1) {  
 return 1;  
 } else {  
 return n \* factorial(n - 1);  
 }  
 }  
}

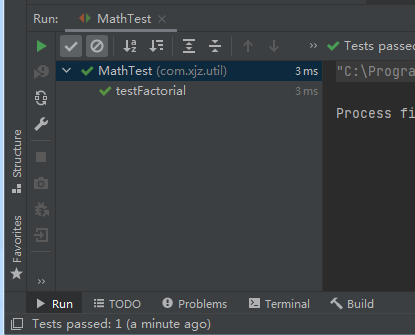
创建MathTest单元测试类，内容如下：

package com.xjz.util;  
  
import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class MathTest {  
  
 @Test  
 public void testFactorial() throws Exception {  
  
 *assertEquals*(120, new Math().factorial(5));  
  
 }  
  
}

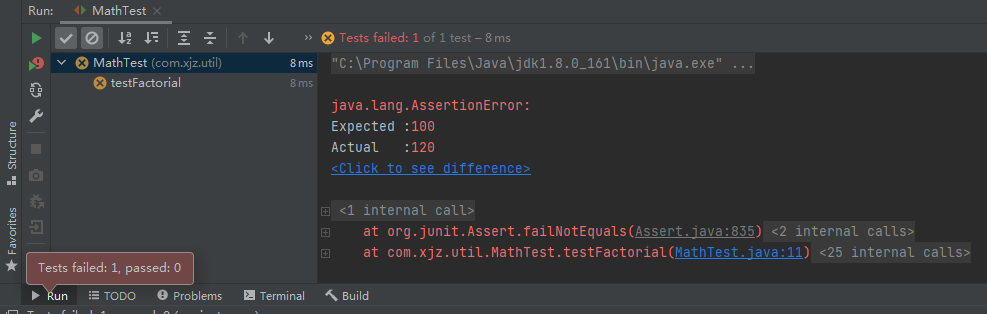
在代码面板MathTest上右键点击运行



运行结果为测试通过



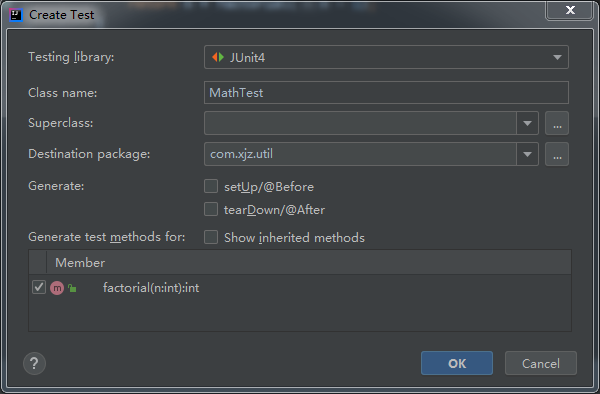
把MainTest中的*assertEquals*(120, new Math().factorial(5))的120改成100，显示测试不通过



注意：

第一，导入了org.junit.Test;和org.junit.Assert.\*;这两个包，注意后者是静态导入import static。  
第二，testFactorial是在要测试的方法名Factorial前加个test（这也是个好习惯）。  
第三，所有测试方法返回类型必须为void且无参数。  
第四，一个测试方法之所以是个测试方法是因为@Test这个注解。  
第五，assertEquals的作用是判断两个参数是否相等，例子中120是预期结果，new Math().factorial(5)是实际结果。单元测试通常不应该只比较一个值，要测试多几个特殊值，特别是临界值。例如Math().factorial(0)和Math().factorial(-1)等。

1. 自动生成测试类
2. 删掉MathTest类（project面板中右键选中MathTest类，Delete）
3. 在代码面板Math类上按Ctrl + Shift + T，点击Create New Test
4. 选择JUnit4，然后勾选要生成测试方法的方法。点击OK。



1. 在生成好的factorial()方法中添加想要的测试案例

*assertEquals*(120, new Math().factorial(5));

1. 运行MathTest
2. 执行顺序

JUnit4利用Annotation注解来定义测试规则。

以下几个常用的注解：

@Test：把一个方法标记为测试方法

@Before：每一个测试方法执行前自动调用一次

@After：每一个测试方法执行完自动调用一次

@BeforeClass：所有测试方法执行前执行一次，在测试类还没有实例化就已经被加载，所以用static修饰

@AfterClass：所有测试方法执行完执行一次，在测试类还没有实例化就已经被加载，所以用static修饰

可以测试一下，创建一个AnnotationTest类，包含两个测试方法test1和test2

package com.xjz.util;  
  
import org.junit.\*;  
import static org.junit.Assert.\*;  
  
public class AnnotationTest {  
 public AnnotationTest() {  
 System.*out*.println("构造方法");  
 }  
  
 @BeforeClass  
 public static void setUpBeforeClass() {  
 System.*out*.println("BeforeClass");  
 }  
  
 @AfterClass  
 public static void tearDownAfterClass() {  
 System.*out*.println("AfterClass");  
 }  
  
 @Before  
 public void setUp() {  
 System.*out*.println("Before");  
 }  
  
 @After  
 public void tearDown() {  
 System.*out*.println("After");  
 }  
  
 @Test  
 public void test1() {  
 System.*out*.println("test1");  
 }  
  
 @Test  
 public void test2() {  
 System.*out*.println("test2");  
 }  
}

执行结果如下：

BeforeClass

构造方法

Before

test1

After

构造方法

Before

test2

After

AfterClass

可以看到test1和test2调用前都调用了@Before的方法，调用后都调用了@After的方法。BeforeClass和AfterClass则总共调用了一次。

1. 请自己设计实现一个计算器Calculator类，可以实现计算器除图形界面和按钮触发以外的大部分功能，包括加、减、乘、除、清零和返回当前结果等功能，像计算器那样支持连续计算。每种运算对应一个方法。设计单元测试类CalculatorTest，对Calculator的各种运算方法进行测试（包括单独运算和四则运算等）。记录实验过程与结果。

提示：因为包含许多测试案例，需要灵活使用@Before和@After等功能才能比较方便地完成测试。浮点数assertequals()第三个参数可设置一个误差值。

1. 实验过程与结果

四、小结