**实验三 单元测试及JUnit的应用**

**一、实验目的**

1、掌握单元测试的基本理论和作用。

2、掌握典型单元测试工具 JUnit 的使用。

**二、实验类型**

验证加设计。

**三、实验内容**

请按照以下关于单元测试的讲解说明，完成两个测试任务（即“实验任务 1”和“实验任务 2”），并在有时间的情况下，阅读后面关于单元测试和 JUnit 高级特性的内容，进一步了解单元测试的理论和 JUnit测试的应用。

测试对于保证软件开发质量有着非常重要的作用，单元测试更是必不可。 JUnit 是一个非常强大的单元测试包，可以对一个或多个类的单个或多个方法进行测试，还可将不同的 TestCase 组合成 TestSuit，使测试任务自动化。 Eclipse 集成了 JUnit，可以非常方便地编写 TestCase。我们在编写大型程序的时候，需要编写成千上万个方法或函数，这些函数的功能可能很强大，但我们在程序中只用到该函数的一小部分功能，并且经过调试可以确定，这一小部分功能是正确的。但是，我们同时应该确保每一个函数都完全正确，因为如果我们今后如果对程序进行扩展，用到了某个函数的其他功能，而这个功能有 bug 的话，将会造成很多问题。所以说，每编写完一个函数之后，都应该对这个函数的方方面面进行测试，这样的测试我们称之为单元测试。 传统的编程方式，进行单元测试是一件很麻烦的事情，你要重新写另外一个程序，在该程序中调用你需要测试的方法，并且仔细观察运行结果，看看是否有错。正因为如此麻烦，所以程序员们编写单元测试的热情不是很高。而类似 JUnit 这样的单元测试包的推出，大大简化了进行单元测试所要做的工作。

**实验任务 1**：

请按以下说明完成一个基本的 JUnit4 测试实例。

File.java:

**public** **class** File {

/\*\*

\* **@param** args

\*/

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

// **TODO** Auto-generated method stub

}

**public** **int** abs(**int** n)

{

**if**(n>=0)

**return** 0;

**else**

**return** (-n);

}

}

FileTest:

**public** **class** FileTest {

**private** File file;

@Before

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

file = **new** File();

}

@After

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

}

@Test

**public** **void** testAbs() {

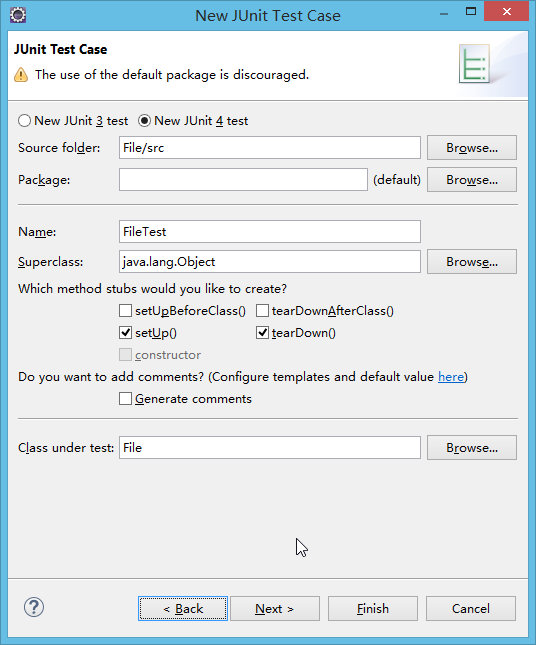
*assertEquals*(file.abs(14),14);

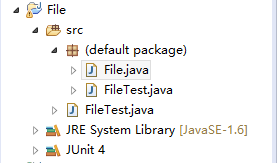
*assertEquals*(file.abs(-5),-5);

*assertEquals*(file.abs(0),0);

}

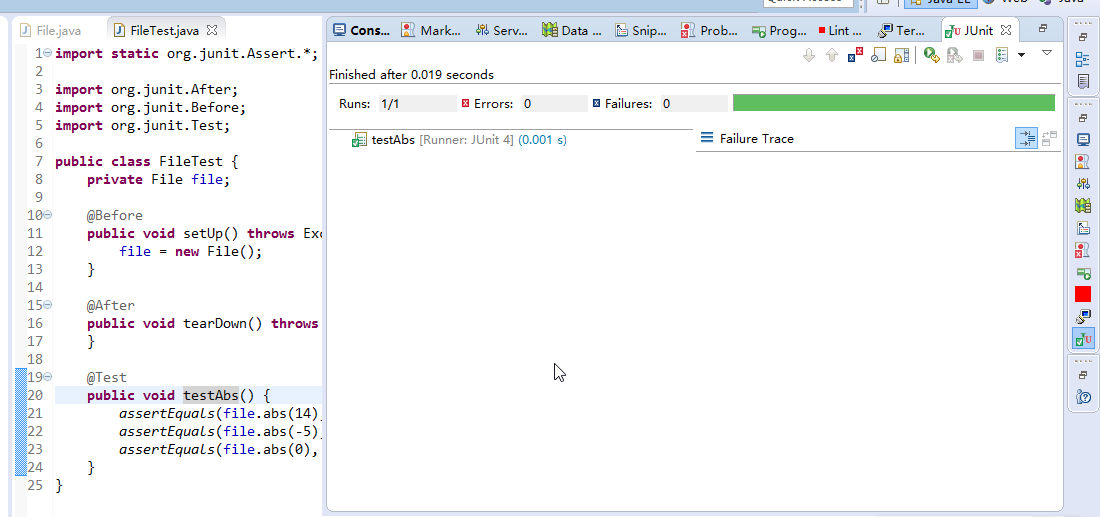
}



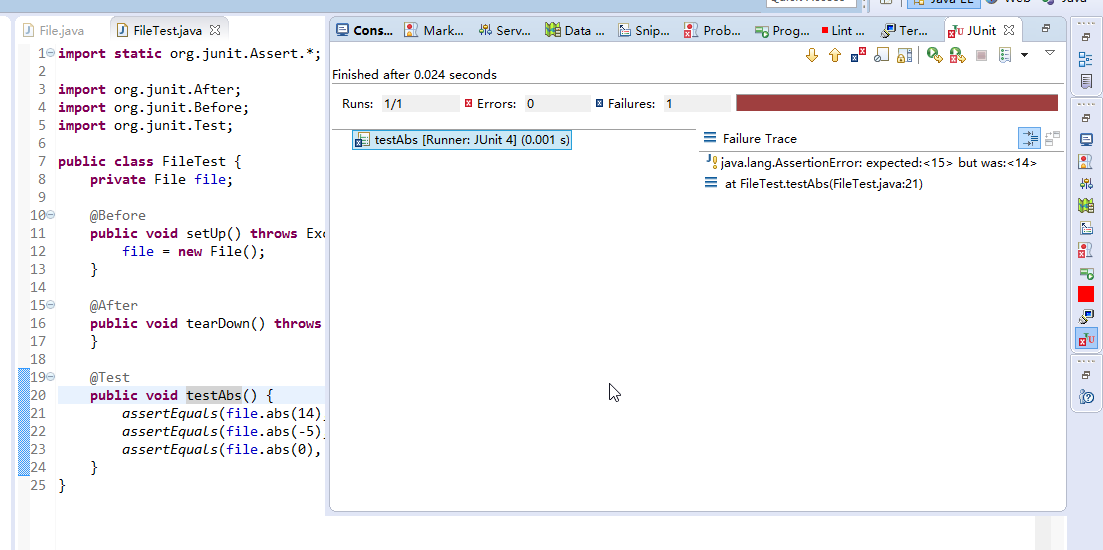


结果：

修改之前：

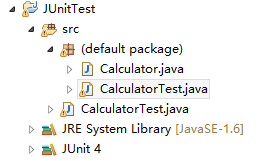


将代码修改后，测试出现错误:



**实验任务2：**

项目结构：

****

**Calculator.java**

**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** substract(**int** n) {

*result* = *result* - 1; // 故意的Bug，应该是 result =result-n

}

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

} // 假设此方法在项目完成过程中尚未写好

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

*result* = *result* / n;

}

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

*result* = n \* n;

}

**public** **void** squareRoot(**int** n) { // 求平方根

**for** (;;)

; // Bug : 死循环

}

**public** **void** clear() { // 将结果清零

*result* = 0;

}

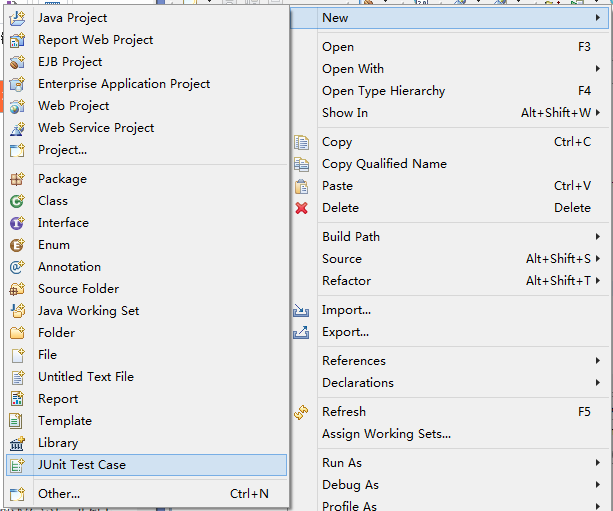
**public** **int** getResult() {

**return** *result*;

}

}

**新建JUnit类**



**CalculatorTest.java**

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

**import** org.junit.After;

**import** org.junit.Before;

**import** org.junit.Ignore;

**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** testSubstract() {

*calculator*.add(10);

*calculator*.substract(2);

*assertEquals*(8, *calculator*.getResult());

}

@Ignore("Multiply() Not yet implemented")

@Test

**public** **void** testMultiply() {

}

@Test

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

*calculator*.add(8);

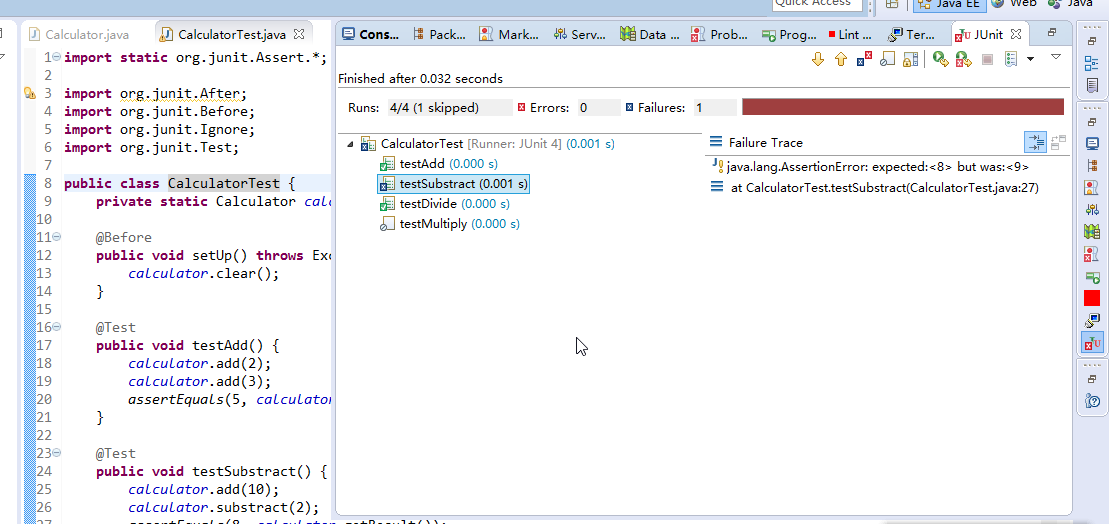
*calculator*.divide(2);

*assertEquals*(4, *calculator*.getResult());

}

}

**进行测试：**

****

**四、实验体会**

通过此次单元测试实验，对于Java程序中如何进行单元测试以及进行单元测试出现结果之后，结果如何进行分析都有了初步的了解，单元测试对于程序是一个重点。