

山东大学

SHANDONG UNIVERSITY



JUnit

实验报告

课程名称：软件质量保证与测试技术

姓名：武敬信

学号：202000800525

专业班级：21软件工程1班

授课教师：康钦马

2024 年 5 月 23 日

-
- 1 实验目的
 - 2 实验环境
 - 3 实验步骤
 - 3.1 JDK 与 Junit 3.81 的安装
 - 3.2 JUnit 的简单使用
 - 3.3 JUnit 的高级使用

1 实验目的

JUnit 是 Java 编程语言的单元测试框架，用于编写和可重复运行的自动化测试。通过本次实验，掌握 Junit 的简单和高级使用。

2 实验环境

虚拟机：Windows 10 x64

软件：JDK1.5 JUnit 3.8.1

CPU：AMD Ryzen 7 5800H

内存：8GB

3 实验步骤

3.1 JDK 与 Junit 3.81 的安装

安装 JDK 1.5，并配置环境变量。并测试是否安装成功。

```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS C:\Users\qr0w> javac
用法: javac <选项> <源文件>
其中, 可能的选项包括:
-g 生成所有调试信息
-g:none 不生成任何调试信息
-g:[lines,vars,source] 只生成某些调试信息
-nowarn 不生成任何警告
-verbose 输出有关编译器正在执行的操作的消息
-deprecation 输出使用已过时的 API 的源位置
-classpath <路径> 指定查找用户类文件的位置
-cp <路径> 指定查找用户类文件的位置
-sourcepath <路径> 指定查找输入源文件的位置
-bootclasspath <路径> 指定查找引导类文件的位置
-extdirs <目录> 覆盖安装的扩展目录的位置
-endorseddirs <目录> 覆盖签名的标准路径的位置
-d <目录> 指定存放生成的类文件的位置
-encoding <编码> 指定源文件使用的字符编码
-source <版本> 提供与指定版本的源兼容性
-target <版本> 生成特定 VM 版本的类文件
-version 版本信息
-help 输出标准选项的提要
-X 输出非标准选项的提要
-J<标志> 直接将 <标志> 传递给运行时系统

PS C:\Users\qr0w> java -version
java version "1.5.0_17"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_17-b04)
Java HotSpot(TM) 64-Bit Server VM (build 1.5.0_17-b04, mixed mode)
PS C:\Users\qr0w> cd C:\Users\qr0w\Desktop\JUnit
PS C:\Users\qr0w\Desktop\JUnit> javac .\Test1.java
PS C:\Users\qr0w\Desktop\JUnit> java .\Test1.java
Exception in thread "main" java.lang.NoClassDefFoundError: /\Test1/java
PS C:\Users\qr0w\Desktop\JUnit> java HelloWorld
HelloWorld
PS C:\Users\qr0w\Desktop\JUnit>
```

安装 JUnit 3.8.1, 并设置环境变量。在 `classpath` 中添加 JUnit 的路径。测试编译没有问题。

```
C:\Users\qr0w\Desktop\JUnit>javac Test1.java

C:\Users\qr0w\Desktop\JUnit>java HelloWorld
HelloWorld

C:\Users\qr0w\Desktop\JUnit>
```

3.2 JUnit 的简单使用

在这里, 以一个简单的计算器类为例, 代码如下:

```
import java.io.*;
import junit.framework.*;

public class Computer
{
    private int a;
    private int b;

    public Computer(int x, int v)
```

```

        a=x; b=y;
    }

    public int add()
    {
        return a+b;
    }

    public int minus()
    {
        return a-b;
    }

    public int multiply()
    {
        return a*b;
    }

    public int divide()
    {
        if(b!=0)
            return a/b;
        else
            return 0;
    }

    public static void main(String[] args)
    {
        Computer cpt1 = new Computer(1,1);
        System.out.println(cpt1.add());
        System.out.println(cpt1.minus());
        System.out.println(cpt1.multiply());
        System.out.println(cpt1.divide());
    }
}

```

将其编译完之后进行测试，结果如下：



```

C:\Users\qr0w\Desktop\JUnit>java Computer
2
0
1
1

```

对其他几组数据也进行测试，从中发现：如果参数是字母的时候，源文件就无法编译通过；当其参数类型不符时也会出错。

接下来，利用 JUnit 框架来测试这个计算器类。

首先，需要新建一个测试代码：

```

import junit.framework.*;

public class TestComputer extends TestCase
{
    ...
}

```

```

        super(name);
    }

    public void testadd()
    {
        assertEquals(3,new Computer(1,2).add());
    }
}

```

然后将其保存为 `TestComputer.java`，并和 `Computer.java` 放在同一个路径下，然后编译源程序。接着，执行以下命令 `java junit.textui.TestRunner TestComputer`

```

C:\Users\qr0w\Desktop\JUnit>java junit.textui.TestRunner TestComputer
.
Time: 0

OK (1 test)

```

也可以使用 `java junit.swingui.TestRunner TestComputer` 来使用图形界面测试。

此外，还可以加入其他运算的测试方法，代码如下：

```

import junit.framework.*;

public class TestComputer extends TestCase
{
    public TestComputer(String name)
    {
        super(name);
    }

    public void testadd()
    {
        assertEquals(3,new Computer(1,2).add());
        assertEquals(-2147483648, new Computer(2147483647,1).add());
    }

    public void testminus()
    {
        assertEquals(-1,new Computer(1,2).minus());
    }

    public void testmultiply()
    {
        assertEquals(4,new Computer(2,2).multiply());
    }

    public void testdivide()
    {
        assertEquals(0,new Computer(2,0).divide());
    }
}

```

3.3 JUnit 的高级使用

JUnit 提供了 `Setup` 和 `Tear-down` 方法，我们可以利用他们来修改代码。对之前计算器测试类进行修改，修改后的代码如下：

```
import junit.framework.*;

public class TestComputer extends TestCase
{
    private Computer a;
    private Computer b;
    private Computer c;
    private Computer d;

    public TestComputer(String name)
    {
        super(name);
    }

    public void setUp()
    {
        a = new Computer(1,2);
        b = new Computer(2147483647,1);
        c = new Computer(2,2);
        d = new Computer(2,0);
    }

    public void testadd()
    {
        assertEquals(3, a.add());
        assertEquals(-2147483648, b.add());
    }

    public void testminus()
    {
        assertEquals(-1, a.minus());
    }

    public void testmultiply()
    {
        assertEquals(4, c.multiply());
    }

    public void testdivide()
    {
        assertEquals(0, d.divide());
    }

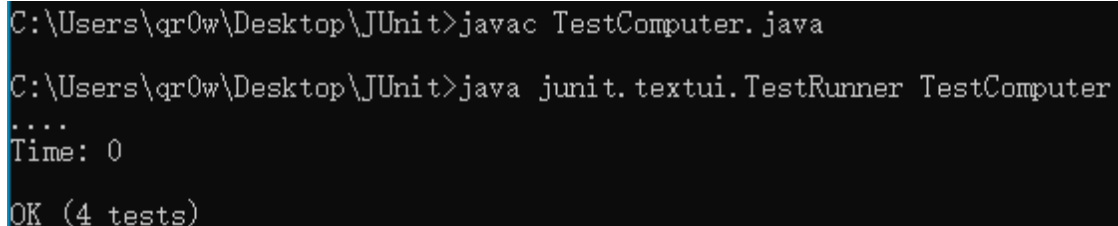
    public static void main(String[] args)
    {
        TestCase test1 = new TestComputer("testadd");
        TestCase test2 = new TestComputer("testminus");
        TestCase test3 = new TestComputer("testmultiply");
        TestCase test4 = new TestComputer("testdivide");
    }
}
```

```

        junit.textui.TestRunner.run(test2);
        junit.textui.TestRunner.run(test3);
        junit.textui.TestRunner.run(test4);
    }
}

```

执行测试的结果如下：



```

C:\Users\qr0w\Desktop\JUnit>javac TestComputer.java

C:\Users\qr0w\Desktop\JUnit>java junit.textui.TestRunner TestComputer
....
Time: 0
OK (4 tests)

```

利用这两个方法，可以有效的减少工作量。

此外，JUnit 还提供了 [集成模式](#)。JUnit 会自动运行所有以 `test` 开头的方法，可是如果执行一部分的话，就需要通过集成模式来解决。

我们需要在测试类中添加一个静态方法，代码是：`public static Test suite();`。可以将需要执行的测试方法放进去：

```

import junit.framework.*;

public class TestComputer extends TestCase
{
    private Computer a;
    private Computer b;
    private Computer c;
    private Computer d;

    public TestComputer(String name)
    {
        super(name);
    }

    @Override
    protected void setUp()
    {
        a = new Computer(1, 2);
        b = new Computer(2147483647, 1);
        c = new Computer(2, 2);
        d = new Computer(2, 0);
    }

    public void testadd()
    {
        assertEquals(3, a.add());
        assertEquals(-2147483648, b.add());
    }

    public void testminus()

```



```

}

public void testmultiply()
{
    assertEquals(4, c.multiply());
}

public void testdivide()
{
    assertEquals(0, d.divide());
}

public static Test suite()
{
    TestSuite suite = new TestSuite();
    suite.addTest(new TestComputer("testadd"));
    suite.addTest(new TestComputer("testminus"));
    return suite;
}

public static void main(String[] args)
{
    junit.textui.TestRunner.run(suite());
}
}

```

当然，也可以进行 **测试类** 的集成。比如，又有一个测试类 **TestComputerTwo**，现在要将 **TestComputer** 与 **TestComputerTwo** 一起集成到 **TC** 这个测试类中，则可以这么做：

```

import junit.framework.*;

public class TC extends TestCase
{
    public TC(String name)
    {
        super(name);
    }

    protected void setUp()
    {
        a = new Computer(1, 2);
        b = new Computer(2147483647, 1);
        c = new Computer(2, 2);
        d = new Computer(2, 0);
    }

    public static Test suite()
    {
        TestSuite suite = new TestSuite();
        suite.addTestSuite(TestComputerTwo.class);
        suite.addTest(TestComputer.suite);
        return suite;
    }
}

```

这样，这个类运行的时候就会执行 `TestComputerTwo` 下的所有测试方法，以及 `TestComputer` 类中 `suite()` 方法所包含的测试方法。