实验课程名称：__软件工程基础实验_____

实验项目名称	单元测试			实验成绩	
实 验 者	蒋如星	专业班级	软件 1704	组　　别	
同 组 者	黄锐			实验日期	2019 年 4 月 28 日

第一部分：实验预习报告（包括实验目的、意义，实验基本原理与方法，主要仪器设备及耗材，实验方案与技术路线等）

一、实验目的

1) 掌握单元测试的方法；
2) 学习 JUnit 测试原理及框架；
3)掌握在 Eclipse 环境中加载 JUnit 及 JUnit 测试方法和过程。

二、实验内容与步骤

1. Eclipse 中 JUnit 的使用

Eclipse 集成了 JUnit，可以非常方便地编写 TestCase。Eclipse 自带了一个 JUnit 插件，不用安装就可以在项目中测试相关的类，并且可以调试测试用例和被测类。

（1）新建一个名为 JUnitTest 的项目，在其中编写一个 Calculator 类，这是一个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。这个类中我们故意保留 了一些 Bug 用于演示，这些 Bug 在注释中都有说明。

（2）将 JUnit4 单元测试包引入这个项目：在该项目上点右键，点“属性”

（3）在弹出的属性窗口中，首先在左边选择“JavaBuildPath”，然后到右上选择“Libraries” 标签，之后在最右边点击“AddLibrary…”按钮，然后在新弹出的对话框中选择 JUnit4 并点击确定，如上图所示，JUnit4 软件包就被包含进我们这个项目了。

（4）生成 JUnit 测试框架：在 Eclipse 的 PackageExplorer 中用右键点击该类弹出菜单，选 择“JUnit 测试用例”。在弹出的对话框中，进行相应的选择，点击“下一步”后，系统会自动列出你这个类中包含的方法，选择你要进行测试的方法。此例中，我们仅对“加、减、乘、除”四个方法进行测试。之 后 系 统 会 自 动 生 成 一 个 新 类 CalculatorTest，里面包含一些空的测试用例。你 只 需 要 将 这 些 测 试 用 例 稍 作 修 改 即 可 使用。

（5）运行测试代码：按照上述代码修改完毕后，我们在 CalculatorTest 类上点右键，选择“RunAs ——>JUnitTest”来运行我们的测试， 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 1 个测试被忽略，一个测试失败”。 2. 利用 JUnit 对“实验一”中的各个类，进行单元测试。

第二部分：实验过程记录（可加页）（包括实验原始数据记录，实验现象记录，实验过程发现的问题等）

生成的测试代码，分析：

1. 创建初始对象

```
public static Gol l=new Gol(); //待测试的类的对象
```

2. @BeforeClass，意味着在每一次测试前都将执行的方法

```
@BeforeClass
public static void setUpBeforeClass() throws Exception {
    for(int i=0;i<30;i++)      //将每个细胞的初始状态设定为死
    for(int j=0;j<30;j++)
        l.table[i][j]=false;
    for(int i=0;i<30;i++)     //将每个细胞临近活细胞的个数设定为0
    for(int j=0;j<30;j++)
            l.neighbors[i][j]=0;
}
```

3.测试GetNeighbors（）方法

```
@Test
public void testGetNeighbors1() {
    l.setTablecell(0, 0);
    l.setTablecell(1,1);
    l.setTablecell(2,2);
    l.getNeighbors();
     assertEquals(2,l.neighbors[1][1]);
}
```

4. 测试GetNeighbors（）方法，并使该方法的每一个判定都执行到，实现代码覆盖率100%

```
@Test
public void testGetNeighbors2() {
    l.setTablecell(12, 12);
    l.setTablecell(12,13);
    l.setTablecell(12,14);
    l.setTablecell(13,12);
    l.setTablecell(13,14);
    l.setTablecell(14,12);
    l.setTablecell(14,13);
    l.setTablecell(14,14);
    l.getNeighbors();
     assertEquals(8,l.neighbors[13][13]);  //白盒测试，覆盖率100%
}
```

5.测试testNextWorld()方法，活细胞个数为3，执行判定1

```
    @Test
    public void testNextWorld1() {
        l.setTablecell(9, 6);
        l.setTablecell(10,8);
        l.setTablecell(11,8);
        l.getNeighbors();
        l.nextWorld();
        assertEquals(true,l.table[10][7]);  //当临近活细胞个数为3时，变成活细胞
    }
```

6. 测试testNextWorld()方法，活细胞个数为1<2，执行判定2

```
    @Test
    public void testNextWorld2() {

        l.setTablecell(21,7);                //临近细胞的状态不包括自己
        l.setTablecell(22,8);
        l.getNeighbors();
        l.nextWorld();
        assertEquals(false,l.table[21][7]);  //当临近活细胞个数为1时，保持原状态

    }
```

7.测试testNextWorld()方法，活细胞个数为8>4，执行判定3

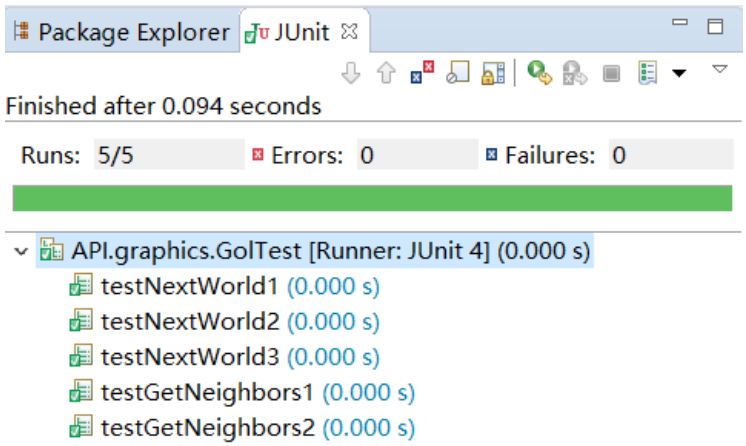
```
    @Test
    public void testNextWorld3() {
        l.setTablecell(24, 12);
        l.setTablecell(24,13);
        l.setTablecell(24,14);
        l.setTablecell(25,12);
        l.setTablecell(25,14);
        l.setTablecell(26,12);
        l.setTablecell(26,13);
        l.setTablecell(26,14);
        l.getNeighbors();
        l.nextWorld();
        assertEquals(false,l.table[25][13]);  //当临近活细胞个数为8>4时，此细胞致死
    }}
```

以上白盒测试，对 GetNeighbors（），testNextWorld()两个方法的代码覆盖率达到 100%

第三部分　结果与讨论（可加页）

一、实验结果分析（包括数据处理、实验现象分析、影响因素讨论、综合分析和结论等）

1.实验结果



2.实验中出现的问题

1) 在测试类中创建待测试类的对象(L)后,发现 l 直接使用成员属性报错，

```
19              l.table[i][j]=false;
private  boolean[][] table = new boolean[SIZE][SIZE];
```

原因是：原类中 table[][]二维数组定义成 private 类型，只能在原类中使用。

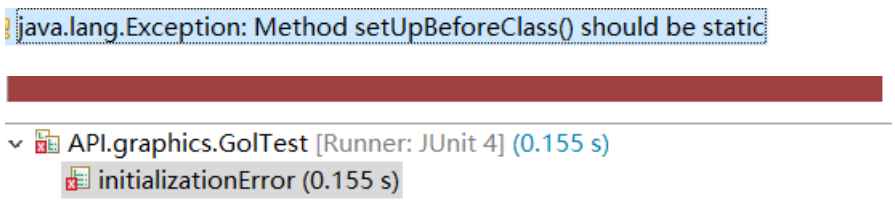
解决办法：将原类中的 private 类型改为缺省状态即可

2) @BeforeClass 是静态方法，不能在其中使用非静态变量

```
public static void setUpBeforeClass() throws Exception {
    for(int i=0;i<30;i++)
        for(int j=0;j<30;j++)
        l.table[i][j]=false;
    for(in
        fo
}
```

Cannot make a static reference to the non-static field l

解决方法：1.将@BeforeClass 改为非静态



却出现以上状态：setUpBeforeClass() should be static　说明测试前运行的@BeforeClass 方法必须是 static 静态状态，解决方案一不可行

方案 2：将最初初始化的类对象定义为静态：**public static Gol l=new Gol(); 成功运行**

3) 测试方法之间的影响

测试方法 GetNeighbors()

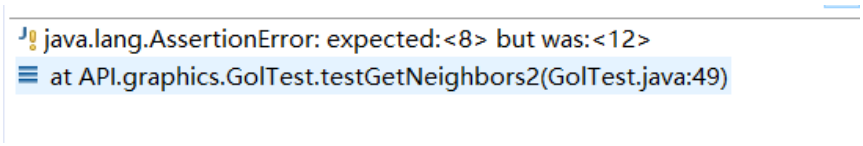
```
public void testGetNeighbors2() {
    l.setTablecell(12, 12);
    l.setTablecell(12,13);
    l.setTablecell(12,14);
    l.setTablecell(13,12);
    l.setTablecell(13,14);
    l.setTablecell(14,12);
    l.setTablecell(14,13);
    l.setTablecell(14,14);
    l.getNeighbors();
     assertEquals(8,l.neighbors[13][13]);  //白盒测试，覆盖率100%
    }
```

测试方法 NextWorld()

```
public void testNextWorld3() {
    l.setTablecell(12, 12);
    l.setTablecell(12,13);
    l.setTablecell(12,14);
    l.setTablecell(13,12);
    l.setTablecell(13,14);
    l.setTablecell(14,12);
    l.setTablecell(14,13);
    l.setTablecell(14,14);
    l.getNeighbors();
    l.nextWorld();
    assertEquals(false,l.table[13][13]);  //当临近活细胞个数为8>4时，此细胞致死

    }
```

学生在测试中发现，当设置的测试用例有重合部分时，先测试的用例会影响后测试的用例：



测试方法 testGetNeighbors()的结果 l.neighbors[13][13]理应是 8，但是测试出来显示错误，l.neighbors[13][13]为 12。此时说明测试时，先运行测试方法 testNextWorld3()，并且改变了细胞的状态，改变了 Neighbors[13][13]初始值，当再测试方法 testGetNeighbors2()时，导致 Neighbors[13][13]有 8 增大到了 12.

解决方法：不同的方法所用测试用例尽量不要重合

二、实验小结及体会

通过本次实验，我学会并掌握了 junit 测试框架的使用，能够加快代码开发进度，提高代码质量。同时本实验是由我们组内两人共同完成的，在实验中我们大大提高了开发的效率。

成绩评定表：

序号	评分项目	满分	实得分
1	**实验报告格式规范**	**2**	
2	**实验报告过程清晰，内容详实**	**4**	
3	**实验报告结果正确性**	**2**	
4	**实验分析与总结详尽**	**2**	
	总得分	**10**	