**软件工程中级实训——Junit学习笔记**：

1. **Junit设计理念与工作原理：**
   1. **概述**：JUnit是由 Erich Gamma 和 Kent Beck 编写的一个回归测试框架（regression testing framework）,用于帮助Java开发人员编写单元测试。所谓单元测试也就是白盒测试。单元测试在xp社区极为流行，作为测试驱动开发，junit是java开发使用最为广泛的框架。该框架也得到了绝大多数java IDE和其他工具（例如ant）的集成支持。同时，junit还有很多的第三方扩展和增强包可供使用。
   2. **基本概念：**

JUnit有几个基本概念：TestCase，TestSuite，TestFixtrue。

* + 1. **TestCase：**代表一个测试用例，每一个TestCase实例都对应一个测试，这个测试通过这个TestCase实例的名字标志，以便在测试结果中指明哪个测试出现了问题
    2. **TestSuite：**代表需要测试的一组测试用例。
    3. **TestFixtrue：**TestFixtrue代表一个测试环境。它用于组合一组测试用例，这组测试用例需要共同的测试运行环境。

1. **Junit的设计**

JUnit的核心是围绕命令模式和组合模式设计的，当然同时使用了模版方法模式，参数收集模式，适配器模式等。这只是简单介绍。

JUnit框架中有几个核心的接口和类。

1. **Test接口**

代表一个测试。它是框架的主接口有两个方法：  
int countTestCases();//返回所有测试用例的个数。  
void run(TestResult result);//运行一个测试，并且收集运行结果到TestResult

1. **TestCase类**

TestCase实现了Test接口，是框架提供的供我们继承的类，我们的所有的测试方法都需要在TestCase的子类中定义，并且符合特定的设计协议。

一个TestCase实例代表一个具体的测试实例，对应一个对某一方法或概念的测试。每个TestCase实例都有一个名字。一个TestCase类却定义了一个TestFixture。具体的说是我们自己定义的TestCase子类中可以定义很多的public 没有参数的 testxxx方法。运行时，每个testxxx都在自己的fixture中运行。每个运行的TestCase都有一个名字，如果不指定，一般是TestCase中定义的test方法的名字。

1. **TestSuite类**

和TestCase一样TestSuite也实现了Test接口。一个TestSuite可以包含一系列的TestCase。把testCase组装入TestSuite有几种方式：

**A：**通过将TestCase的Class参数传入TestSuite的构造函数，TestSuite会自动收集TestCase中所有的public的没有参数的testxxx方法加入TestSuite中。  
**B：**构造空的TestSuite后通过void addTest(Test test)方法添加测试。  
**C：**构造空的TestSuite后通过void addTestSuite(Class testClass) 方法添加测试集。

1. **TestResult类**

主要通过runProtected方法运行测试并收集所有运行结果。

1. **TestRunner类**

启动测试的主类，我们可以通过直接调用它运行测试用例，IDE和其他一些工具一般也通过这个接口集成JUnit。

1. **Assert类**

用于断言，TestCase继承自该类，我们的测试方法通过这些断言判断程序功能是否通过测试。

1. **TestListener接口**

测试运行监听器，通过事件机制处理测试中产生的事件，主要用于测试结果的收集。以上是框架的核心接口和类的介绍，通过上面的介绍我们很容易看出来Test，TestCase和TestSuite的设计采用了Composite模式。这样JUnit可以一次运行一个测试用例，也可以一次运行多个测试用例，TestRunner只关心Test接口，而对运行的是单个的TestCase还是同时运行多个TestCase并不在意。TestCase还使用了Template Method模式：

public void **run**()

{  
setUp();  
runTest();  
tearDown();  
}

protected void **runTest**() {}

protected void **setUp**() {}

protected void **tearDown**() {}

8. **JUnit同时使用了Command模式**，对于典型的Command模式一般有5种角色：

1)**命令角色（Command）**：声明执行操作的接口。有java接口或者抽象类来实现。  
2)**具体命令角色（Concrete Command**）：将一个接收者对象绑定于一个动作；调用接收者相应的操作，以实现命令角色声明的执行操作的接口。  
3)**客户角色（Client）**：创建一个具体命令对象（并可以设定它的接收者）。  
4)**请求者角色（Invoker）**：调用命令对象执行这个请求。  
5**)接收者角色（Receiver）**：知道如何实施与执行一个请求相关的操作。任何类都可能作为一个接收者。对于JUnit的设计，不能明显的区分出这5种角色，因为它的设计相对复杂，同时参杂了其他模式。

Test接口可以认为是命令模式中的命令角色Command接口，

void run (TestResult result)接口方法定义了需要执行的操作；TestCase可以看作是具体命令角色，但又不全是，因为我们还需要自己通过继承TestCase类定义测试方法，这样的每一个测试方法都回被包装在一个TestCase实例中。

TestResult可以看作请求者角色（Invoker），它会通过protected void run(final TestCase test) 运行测试并收集结果。我们自己写的Test方法可以认为是接收者角色（Receiver），因为我们的方法才具体执行这个命令。

TestRunner就是客户角色（Client），它通过TestResult result= createTestResult()构造TestResult，并通过suite.run(result)运行测试用例（suite是一个Test接口的具体实例，可以是TestCase也可以是TestSuite，但客户端不关心它是什么，这就是组合模式的好处。同时，suite.run（result）又调用result.run（test）

1. **使用Junit测试：**
   1. **快捷键的使用**

**方法一：**在待编写测试的 java 类源码块上按快捷键 Alt + Insert。选择 JUnit Test->JUnit 4。

**方法二：**在待编写测试的 java 类源码块上按快捷键 Ctrl + Shift + T.根据需要修改相关测试用例。

**方法三：**光标定位到待编写测试的 java 类源码块，选择 code->Generate，后面的步骤和方法 1 一样。

* 1. **Test中用到的注解解释：**

**@BeforeClass** 使用此注解的方法在测试类被调用之前执行

**@AfterClass** 使用此注解的方法在测试类被调用结束退出之前执行一个类中有多少个@Test注解方法，以下对应注解方法就被调用多少次

**@Before** 在每个@Test调用之前执行

**@After** 在每个@Test调用之后执行

**@Test** 使用此注解的方法为一个单元测试用例，一个测试类中可多次声明，每个注解为@Test只执行一次

**@Ignore** 暂不执行的测试用例，会被JUnit4忽略执行

**在IDEA中测试和使用的效果如下（在之后项目中的测试基本都采用了这种形式）：**

|  |
| --- |
|  |
|  |
|  |
|  |