

第六讲：JUNIT 测试工具

LIANYU
THE SCHOOL OF SOFTWARE AND MICROELECTRONICS
PEKING UNIVERSITY
NO.24 JINYUAN RD, BEIJING 102600

提 纲

1

引言

2

使用Junit (简单例子,安装与运行,常见问题)

3

JUnit设计 (设计目标,设计内容)

4

模仿对象测试

5

DbUnit

6

JUnit 4.0

7

总结

引言

- JUnit是一个开源的Java编程语言的单元测试框架，最初由 Erich Gamma 和 Kent Beck 编写的。
- Junit在代码驱动单元测试框架家族里无疑是最为成功的一例。
 - 在“最佳Java性能监视/测试工具”类别里，多次获得“JavaWorld 编者精品奖（Editors' Choice Awards）”。

引言（续）

- Junit框架让我们继承**TestCase**类，用**Java**来编写自动执行、自动验证的测试。这些测试在**Junit**中称作“测试用例”。
- **JUnit**提供一个机制能够把相关测试用例组合到一起，称之为“测试套件（**test suite**）”。
- **JUnit**还提供了一个“运行器”来执行一个测试套件。如果有测试失败了，这个测试运行器就报告出来；如果没有失败，就会显示“OK”。

引言（续）

- 测试驱动开发是一种编程的风格，这种风格大致的意义时：
 - 先编写测试代码，再编写产品代码本身，另外还需要在编写代码的时候执行重构。
 - 这样的产品代码测试覆盖率高，容易修改，容易扩展，并且容易理解。

引言（续）

- 简要概述一下使用JUnit的益处：
 - 提高开发速度：测试是以自动化方式执行的，提升了测试代码的执行效率。
 - 提高软件代码质量：它使用小版本发布至集成，便于开发人员除错。同时引入重构概念，让代码更干净和富有弹性。
 - 提升系统的可信赖度：它是回归测试的一种。支持修复或更正后的“再测试”，可确保代码的正确性。

引言（续）

- 介绍JUnit的使用，框架设计和相关的“模仿对象（Mock Objects）”，以及数据库的单元测试工具DbUnit.
- 介绍JUnit是用JUnit 3.8版本，在结束之前简要介绍JUnit 4。

提纲

- 引言
- ➡ ■ 使用JUnit
 - 一个简单的例子
 - JUnit安装与运行
 - JUnit常见问题
- JUnit设计
 - 设计目标
 - 设计内容
- 模仿对象测试
- DbUnit
- JUnit 4.0

使用JUnit

- 用JUnit编写一个简单的测试，可以有三个简单步骤：
 - 创建TestCase类的一个子类。
 - 编写若干测试用例，每个测试用例写成如下格式的子类方法，注意JUnit对于测试用例的命名法是“test”+ <TestCaseName>测试用例的名字。

```
public void test<TestCaseName> () { ...}
```
 - 编写一个测试套件方法用来把第2步里写的测试用例加入到测试套件中。

```
public static Test suite() { ...}
```
- 然后编译上述子类以及被测构件，用JUnit提供的运行器TestRunner运行测试。

使用JUNIT（续）

- 一个简单的JUnit的测试例子。这个简单测试是要：
 - （1）验证Java标准类库的Math类里max（）方法；
 - （2）测试被零除的结果。

```
1.  public class SimpleTest extends TestCase {
2.      public void testMax() {
3.          int x=Math.max(5,10);
4.          assertTrue(x>=5 && x>=10);
5.      }
6.      public void testDivideByZero() {
7.          int zero= 0;
8.          int result= 8/zero;
9.      }
10.     public static Test suite() {
11.         return new TestSuite (SimpleTest.class);
12.     }
13. }
```

JUNIT安装与运行

- 要安装和使用JUnit是很容易的，只需三个步骤：
 - （1）下载JUnit软件；
 - （2）将JUnit包解开，放到文件系统中；
 - （3）运行JUnit测试时，将JUnit中的所有*.jar文件放到类路径里。

JUNIT安装与运行（续）

■ 下载JUnit:

- 目前，下载JUnit的最好地方是URL:<http://www.junit.org>。在此，会有一个下载链接，链到此产品的最新版本。点击下载链接，将这个软件下载到用户机器的文件系统中。
- 注：URL被链接到sourceForge.net

JUNIT安装与运行 （续）

■ 解包JUnit:

文件/目录	描述
junit.jar	JUnit框架结构、扩展和测试运行器的二进制发布
src.jar	JUnit的源代码，包括一个 Ant 的buildfile文件
junit	是个目录，内有JUnit自带的用JUnit编写的测试示例程序
javadoc	JUnit完整的API文档
doc	一些文档和文章，包括 “Test Infected: Programmers Love Writing Tests”和其它一些资料，可以帮助我们入门。

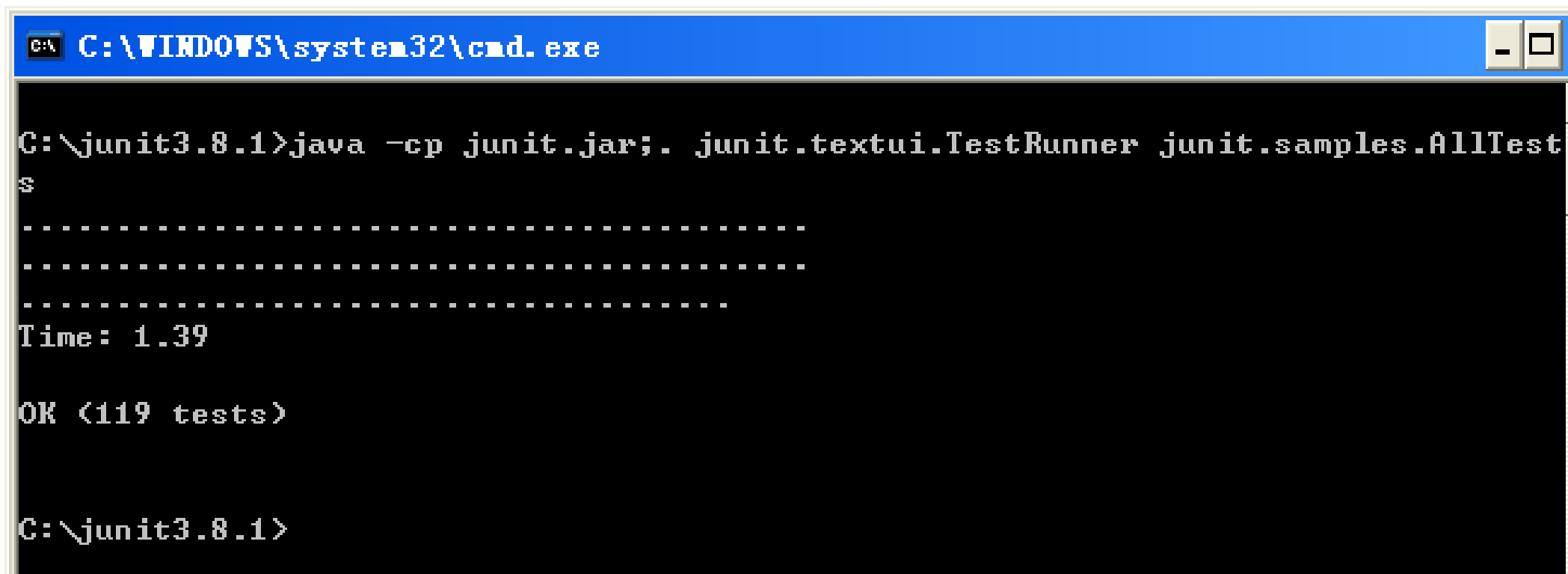
JUNIT安装与运行（续）

■ 检验安装JUnit:

- 要检验安装是否正确，可以通过执行JUnit发布中自带的、用JUnit编写的测试示例程序。
- 要执行这些程序，按照以下步骤进行：
 - 打开一个有命令行提示的窗口。
 - 转到含有JUnit的目录下（Windows系统的D:\junit3.8；或Linux系统的/opt/junit3.8或任何安装时选定的地方）。
 - 执行下列命令：

```
>java -classpath junit.jar;. Junit.textui.TestRunner junit.samples.AllTests
```

JUNIT运行截图



```
C:\WINDOWS\system32\cmd.exe

C:\junit3.8.1>java -cp junit.jar;. junit.textui.TestRunner junit.samples.AllTest
s
.....
.....
.....
Time: 1.39

OK (119 tests)

C:\junit3.8.1>
```


JUNIT安装与运行（续）

- **运行JUnit测试:**

- JUnit框架提供两种运行测试的方式：文本式和图形用户界面式。
- 在运行JUnit的测试程序之前，需要编译。编译的命令如下：

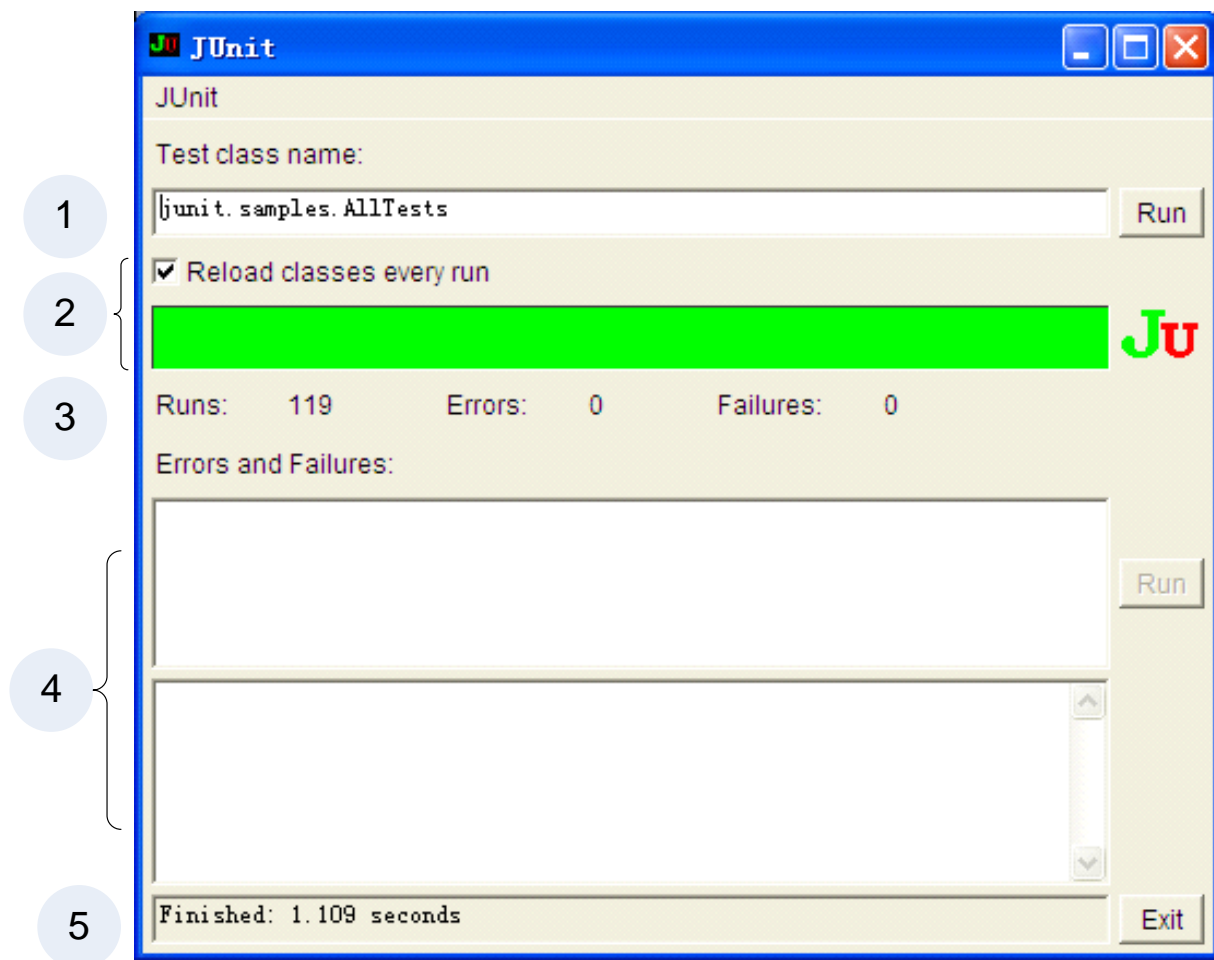
```
>javac -classpath junit.jar;<your_classpath> -d <destination_directory> <your_source_files>
```

JUNIT安装与运行（续）

- JUnit框架提供图形用户界面式运行测试有两个版本：**AWT**（**Abstract Window Toolkit**）版本和**Swing**版本。
- 使用**AWT**版本时输入以下命令来启动**JUnit**运行器：

```
>java -cp junit.jar;<your_classpath> junit.awtui.TestRunner
```

JUNIT安装与运行（续）

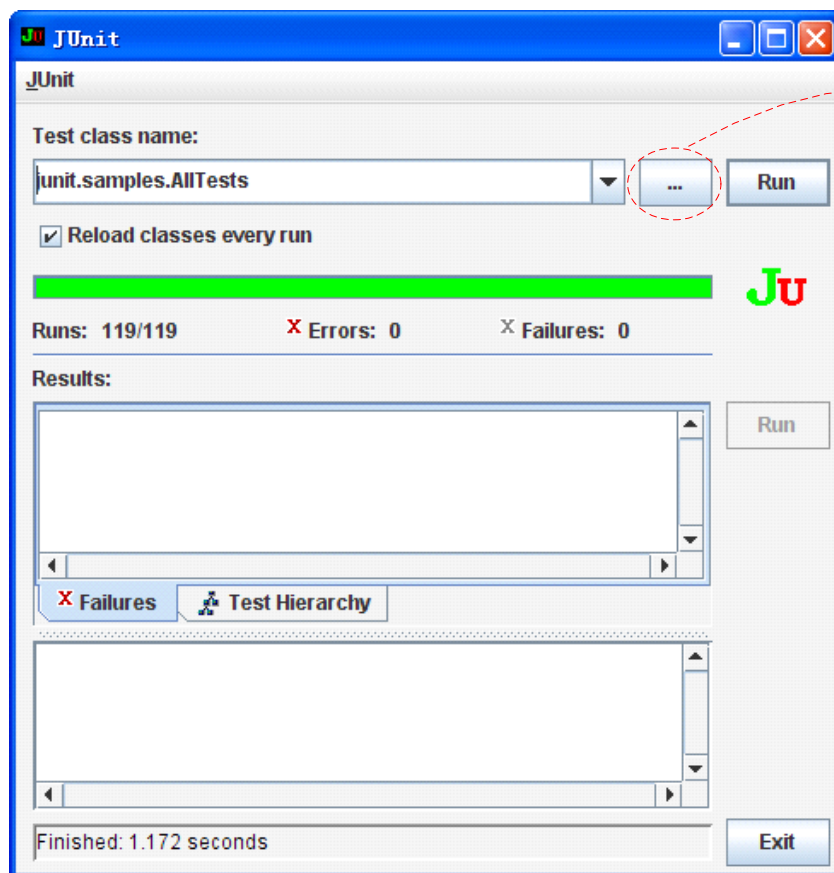


JUNIT安装与运行（续）

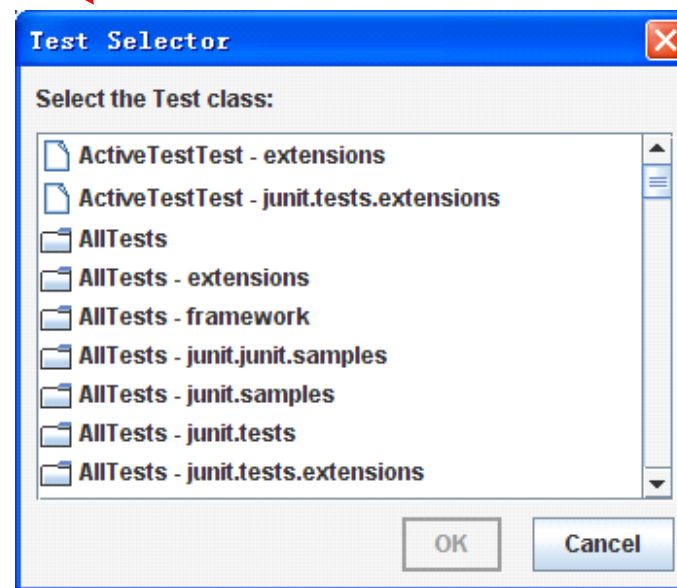
```
>java -cp junit.jar;<your_classpath> junit.swingui.TestRunner
```

- 使用Swing版本时输入以下命令来启动JUnit运行器：

JUNIT安装与运行（续）

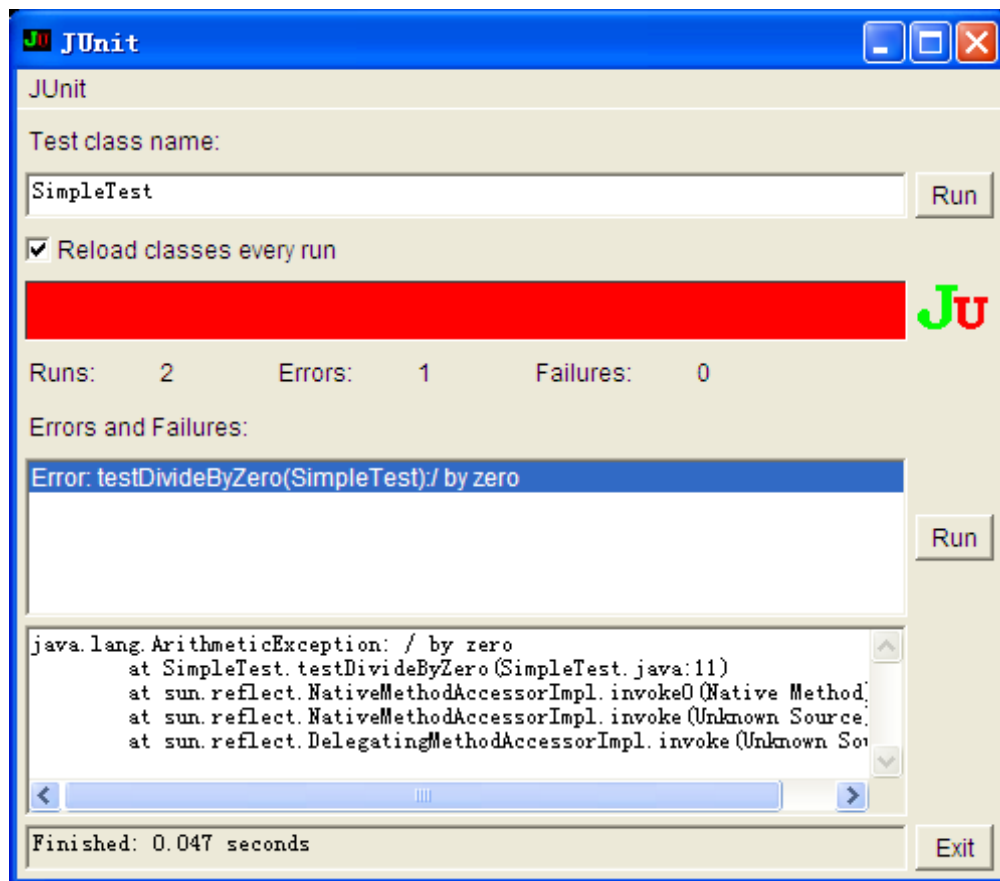


click



JUNIT安装与运行（续）

横条显示
“红”色



1

2

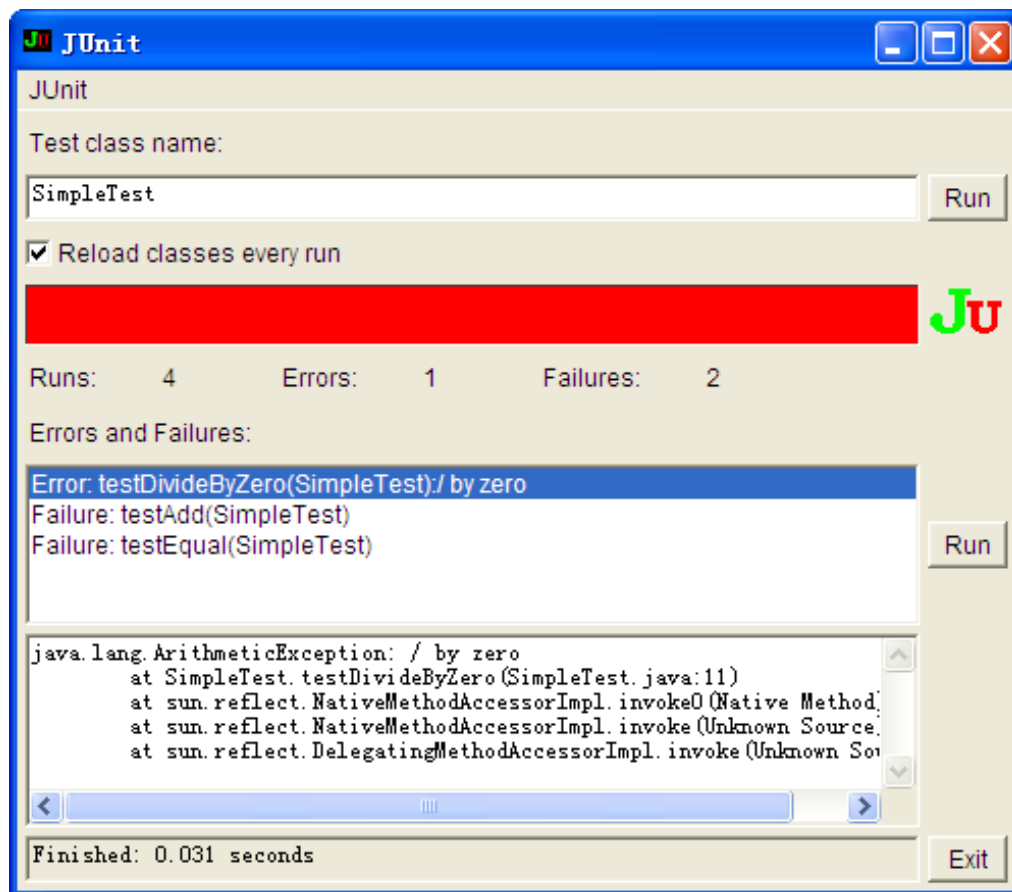
3

JUNIT安装与运行（续）

- 现在让我们在SimpleTest里再加2个测试方法testAdd()和testEqual(),

```
1.    public void testAdd() {  
2.        double result= 2 + 3;  
3.        assertTrue(result == 6);  
4.    }  
5.    public void testEqual() {  
6.        double a = 6, b = 7;  
7.        assertTrue( a == b);  
8.    }
```

JUNIT安装与运行（续）



JUNIT的断言

■ JUnit的断言

- 要理解JUnit是怎样决定一个测试到底是通过了还是失败了，我们需要了解断言的方法是怎样表示一个断言已经失败了。
- 要让JUnit测试能够自行验证，就必须对于对象的状态做出断言。
 - 当JUnit断言失败了，断言的方法就会抛出一个异常，来表示这个断言失败了。
 - 更准确地说，当断言的方法失败的时候，断言的方法会抛出一个错误：`AssertionFailedError`。

```
1.      static public void assertTrue(boolean condition) {  
2.          if (!condition)  
3.              throw new AssertionFailedError();  
4.      }
```

The `AssertionFailedError` is not meant to be caught by the client (a testing method inside a `TestCase`) but inside the Template Method `TestCase.run()`.

```
public void run(TestResult result) {  
    result.startTest(this);  
    setUp();  
    try {  
        runTest();  
    }  
    catch (AssertionFailedError e) { //1  
        result.addFailure(this, e);  
    }  
    catch (Throwable e) { // 2  
        result.addError(this, e);  
    }  
    finally {  
        tearDown();  
    }  
}
```

TESTRESULT

- **TestResult** 中的各方法来收集这些错误，如下所示：

```
public synchronized void addError(Test test, Throwable t) {  
    fErrors.addElement(new TestFailure(test, t));  
}  
  
public synchronized void addFailure(Test test, Throwable t) {  
    fFailures.addElement(new TestFailure(test, t));  
}
```

JUNIT的断言（续）

- 当一个断言失败了，加上一个简短的消息来通知这个失败的一些属性甚至是失败的原因，可能会带来比较好的效果。
- JUnit框架里，一个断言的方法的第一个参数作为可选项，接受一个“String”类型的参数，包含一些消息，在断言失败的时候显示出来。对于`assertTrue()`有2种形式：

```
static public void assertTrue(boolean condition)
static public void assertTrue(String message, boolean condition)
```

JUNIT的断言（续）

- JUnit框架里，和`assertTrue()`相对应的断言的方法有`assertFalse()`，也有2种形式：

```
static public void assertFalse(boolean condition)  
static public void assertFalse(String message, boolean condition)
```

失败与错误

- 失败与错误：JUnit框架区别失败与错误。
 - “失败”是指当断言失败时，而“错误”是指当某种其它异常发生时，这种异常还没被测出也没被预料到。
 - 这种区别是微妙的也是有用的。出现“失败”的断言通常表示产品代码中有问题，而出现“错误”却表示测试本身或周围的环境存在着问题。

失败与错误（续）

- 失败与错误：JUnit框架区别失败与错误。（续）
 - “错误”是指，比如指望不正当的异常对象或者不正确地想要调用一个空实例的方法（抛出“`NullPointerException`”异常），或者对于数组的操作超出数组的范围（抛出“`ArrayIndexOutOfBoundsException`”异常）；也可能是磁盘已经满了或者网络连接不通，或者一个文件找不到。
 - JUnit并不把这些算作产品代码的缺陷，而是采取一种“举手投降”方式来表达：“有些不对劲了。我不能分辨这个测试是否通过。请解决这个问题后在重新测试一次。”

失败与错误（续）

- 失败与错误：JUnit框架区别失败与错误。（续）
 - 有时候，JUnit框架的“失败”被称为“预期的失败条件”；而“错误”被称为“不曾预料到的失败条件”。
 - 然而“错误”，即“不曾预料到的失败条件”，在运行时候（**runtime**）将引起异常。
- 注意：当我们得到一个测试运行结果的报告时，如果报告里既有“失败”又有“错误”时，建议先调查“错误”，解决了错误问题以后再重新运行这个测试。

测试套件

■ 测试套件

```
public static Test suite() {  
    return new TestSuite (SimpleTest.class);  
}
```

上面的代码可以用下列代码代替：

```
public static Test suite() {  
    TestSuite suite = new TestSuite();  
    suite.addTest(new TestSuite ("testMax"));  
    suite.addTest(new TestSuite ("testDividedByZero"));  
    return suite;  
}
```

测试套件（续）

- 其实，不定义一个**TestSuite**，JUnit测试运行器将自动生成一个缺省**TestSuite**。
- 这个缺省**TestSuite**扫描测试类里所有以“**test**”方法，在内部为每个“**testXXX**”方法生成一个**TestCase**实例，并把被调用的方法名作为参数转入**TestCase**的构造函数。

测试套件（续）

- 现在的情况是这样，一个测试类里有很多测试方法，我们并不想运行所有这些方法，而只是其中一些。

```
1. public static Test suite() {  
2.     TestSuite suite= new TestSuite();  
3.     suite.addTest(new SimpleTest ("testDivideByZero")  
4.                 {protected void runTest(){testDivideByZero();}});  
5.     return suite;  
6. }
```

- 把testDividedByZero()加入TestSuite是通过使用Java的匿名内部类（anonymous inner classes）方式实现的。

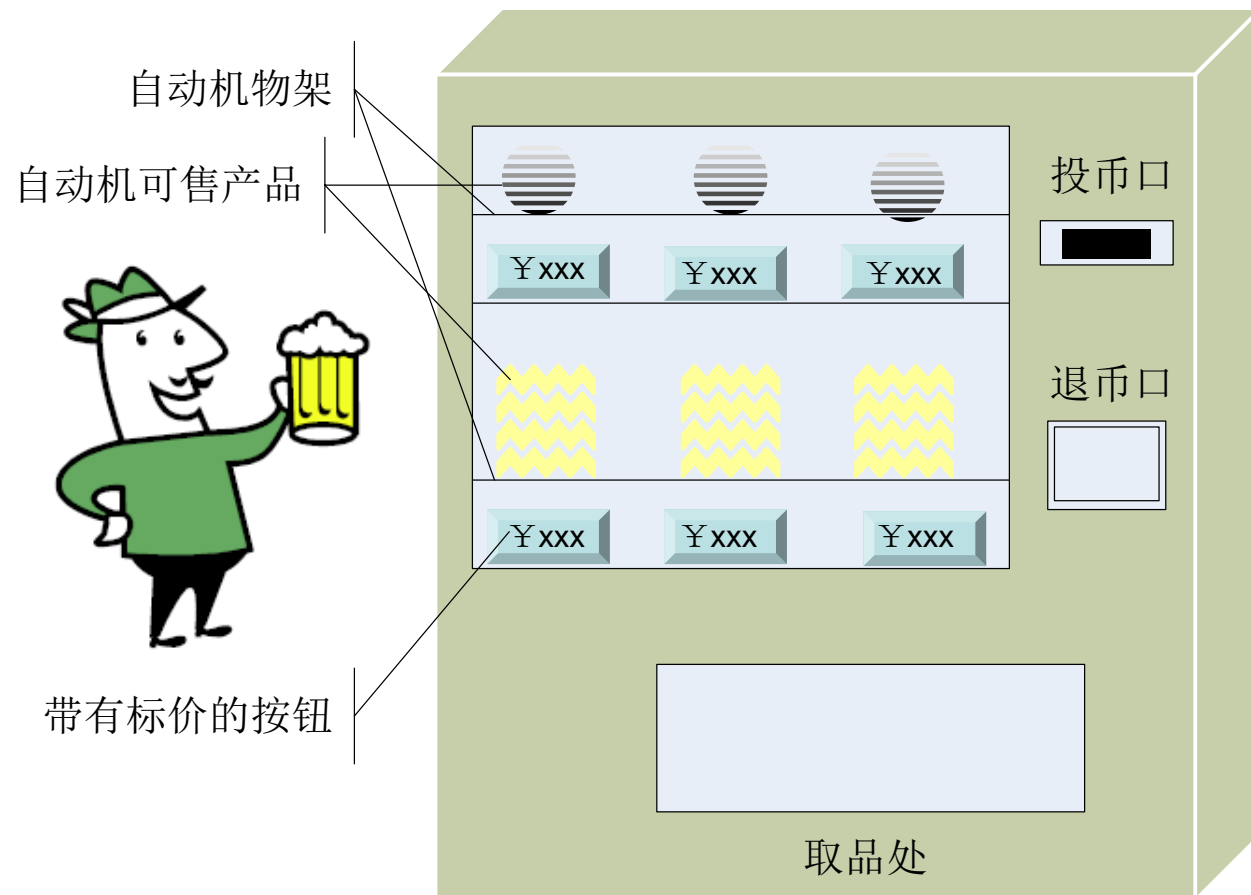
测试置具

■ 测试置具

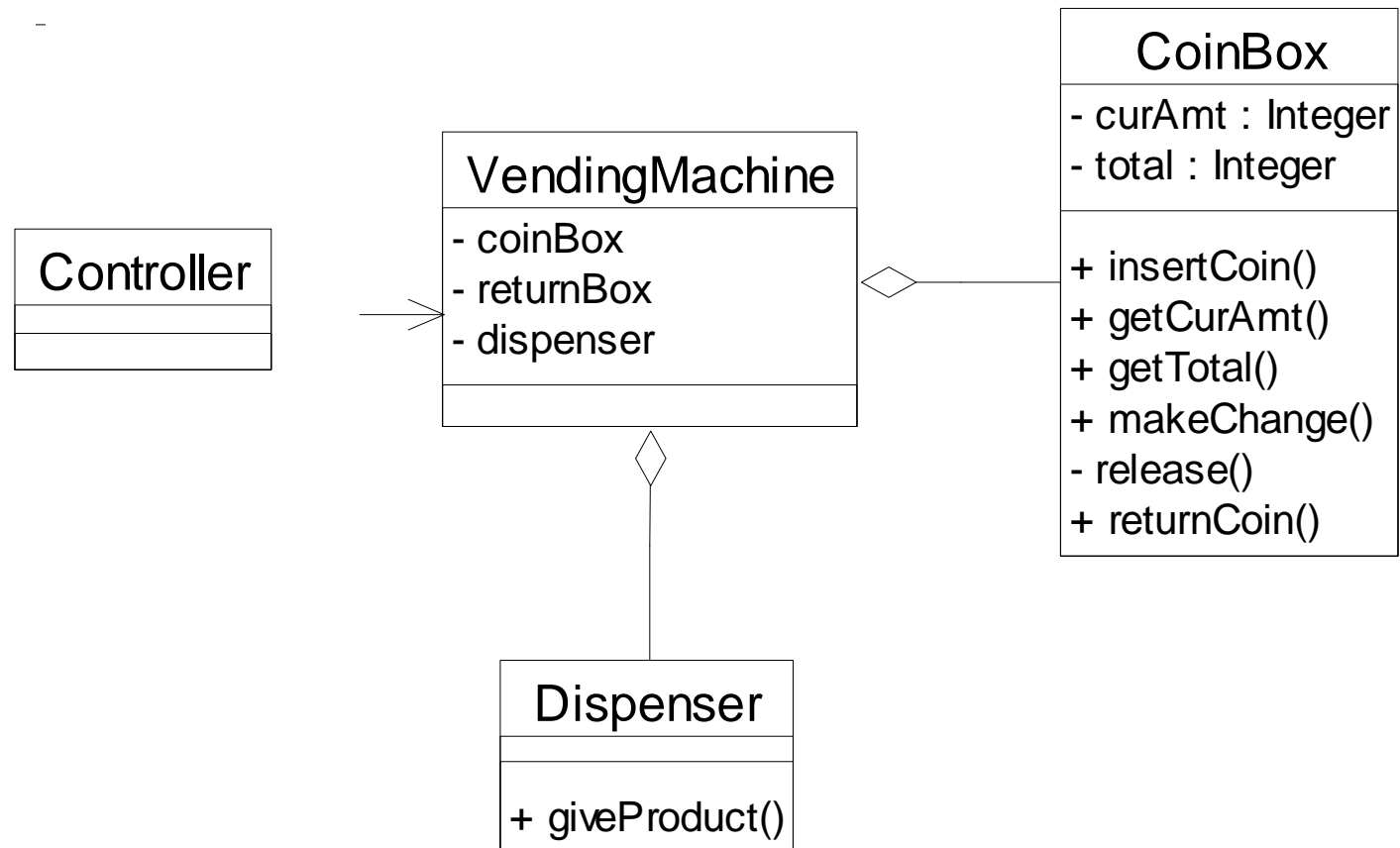
- 常常有这样的情况，我们写的几个测试都用到同一个或同一组对象。
 - 如果在每个测试里各自写这样的对象，将造成代码冗余，不利于维护。
 - 把这样的代码分离出来单独写，让所有的测试都可以利用这些对象代码。
 - 在JUnit框架里，把这些对象称为“测试置具”（**test fixture**）。

```
public class VectorTest extends TestCase {  
    protected Vector fEmpty, fFull; //fixtures 1  
  
    // setup the fixtures  
    protected void setUp() { 2  
        fEmpty= new Vector();  
        fFull= new Vector();  
        fFull.addElement(new Integer(1));  
        fFull.addElement(new Integer(2));  
        fFull.addElement(new Integer(3));  
    }  
    public void testCapacity() { 3  
        int size= fFull.size();  
        for (int i= 0; i < 100; i++)  
            fFull.addElement(new Integer(i));  
        assertTrue(fFull.size() == 100+size);  
    }  
    public void testContains() { 4  
        assertTrue(fFull.contains(new Integer(1)));  
        assertTrue(!fEmpty.contains(new Integer(1)));  
    }  
}
```

- 利用自动售货机（**Vending Machines**）可以销售多种产品，如冷饮料、热咖啡、糖果、小吃、甚至速冻食品等，可以容易地安装在城市的街道上、学校、工厂里，给人们的生活带来方便。



THE CLASS DIAGRAM OF THE VENDING MACHING



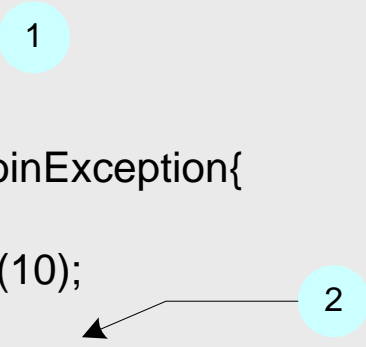
```

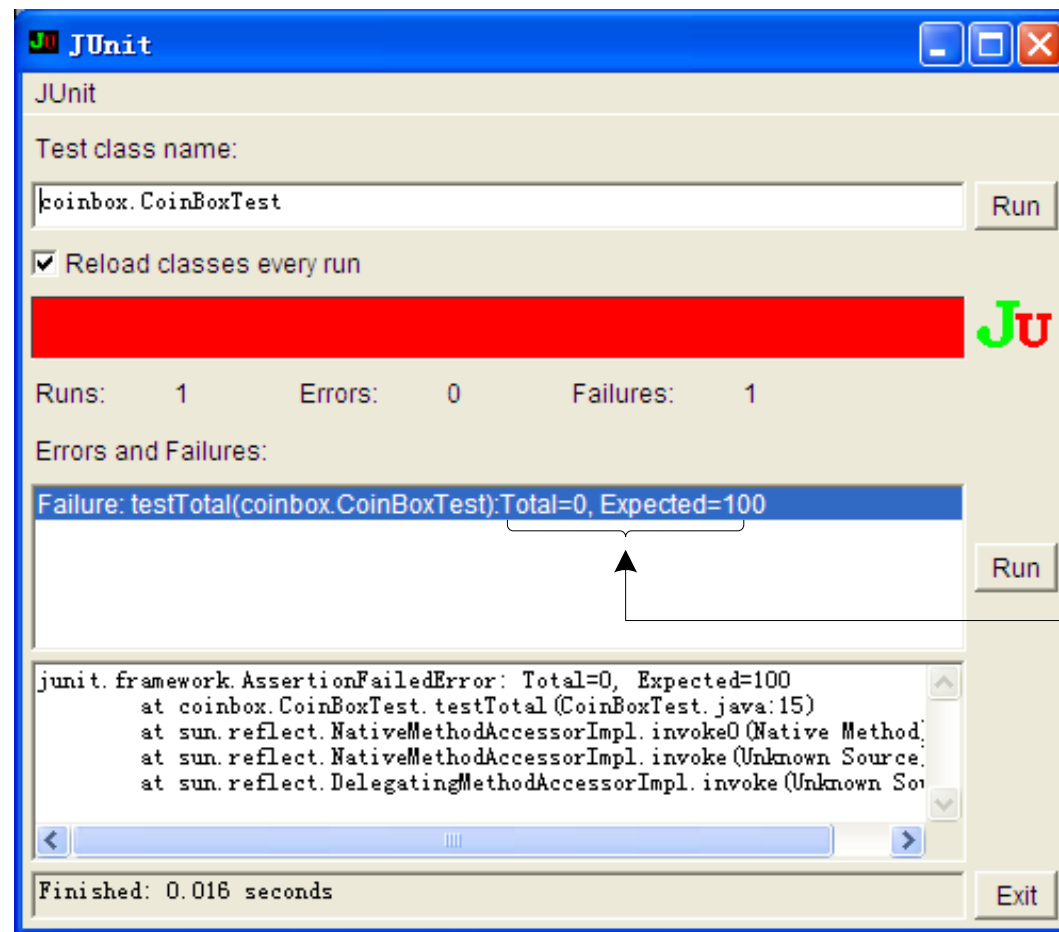
package coinbox;
import java.io.*;
public class CoinBox{
    private int curAmt, total;
    public void insertCoin (int amt) throws InvalidCoinException {
        if (amt!=5&&amt!=10&&amt!=25) {
            throw new InvalidCoinException();
        }
        this.curAmt += amt;
        System.out.println ("Current Amount is "+curAmt);
    }
    public int getCurAmt () {
        return curAmt;
    }
    public int getTotal () {
        return total;
    }
    public void makeChange (int price ) throws InvalidPriceException{
        if (price<=0){
            throw new InvalidPriceException();
        }
        release(curAmt-price);
        curAmt = 0;
        total += price;
    }
    private void release (int amt){
        System.out.println ("Release "+amt);
    }
    public void returnCoin (){
        release (curAmt);
        curAmt=0;
    }
}

```



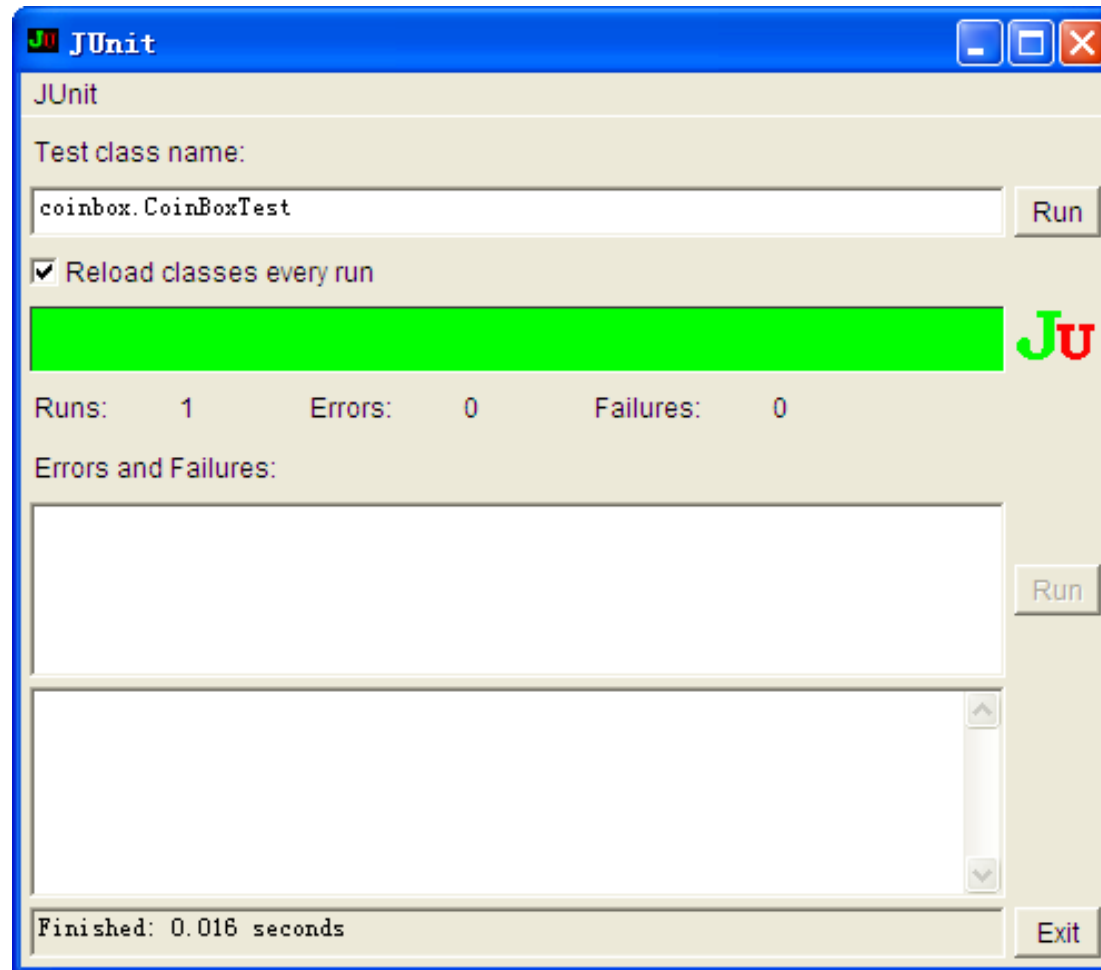
```
import junit.framework.*;
public class CoinBoxTest extends TestCase{
    private CoinBox coinBox;
    public CoinBoxTest(String testCaseName){
        super(testCaseName);
    }
    public void setUp() {
        coinBox=new CoinBox();
    }
    public void testTotal() throws InvalidCoinException{
        for(int i=1; i<=10; i++){
            coinBox.insertCoin(10);
            int total=coinBox.getTotal();
            assertTrue("Total="+total+" Expected=100", total==100);
        }
    }
}
```





“String”参数部分
输出的消息

```
import junit.framework.*;
public class CoinBoxTest extends TestCase{
    private CoinBox coinBox;
    public CoinBoxTest(String testCaseName){
        super(testCaseName);
    }
    public void setUp() {
        coinBox=new CoinBox();
    }
    public void testCurAmt() throws InvalidCoinException,InvalidPriceException{
        for(int i=1; i<=10; i++)
            coinBox.insertCoin(10);
        int curAmt=coinBox.getCurAmt();
        assertTrue("curAmt="+curAmt+"Expected=100", curAmt==100);
    }
}
```



提纲

- 引言
- 使用JUnit
 - 一个简单的例子
 - JUnit安装与运行
 - JUnit常见问题
- ➔ ■ JUnit设计
 - 设计目标
 - 设计内容
- 模仿对象测试
- DbUnit
- JUnit 4.0

JUNIT设计

- 三个类
 - TestCase
 - TestSuite
 - TestResult

```

public abstract class TestCase implements Test {
    private final String fName;
    public TestCase(String name) {
        fName= name;
    }
    public void run(TestResult result) {
        result.startTest(this);
        setUp();
        runTest();
        tearDown();
    }
    protected void runTest() {
    }
    protected void setUp() {
    }
    protected void tearDown() {
    }
    public TestResult run() {
        TestResult result= createResult();
        run(result);
        return result;
    }
    protected TestResult createResult() {
        return new TestResult();
    }
    ..... //omit other methods
}

```

JUnit的TestCase抽象类

```

public class TestResult extends Object {
    protected int fRunTests;
    public TestResult() {
        fRunTests= 0;
    }
    public synchronized void startTest(Test test) {
        fRunTests++;
    }
    ..... //omit other methods
}

```

JUnit的TestResult类

```

public void run(TestResult result) {
    result.startTest(this);
    setUp();
    try {
        runTest();
    }
    catch (AssertionFailedError e) {
        result.addFailure(this, e);
    }
    catch (Throwable e) {
        result.addError(this, e);
    }
    finally {
        tearDown();
    }
}

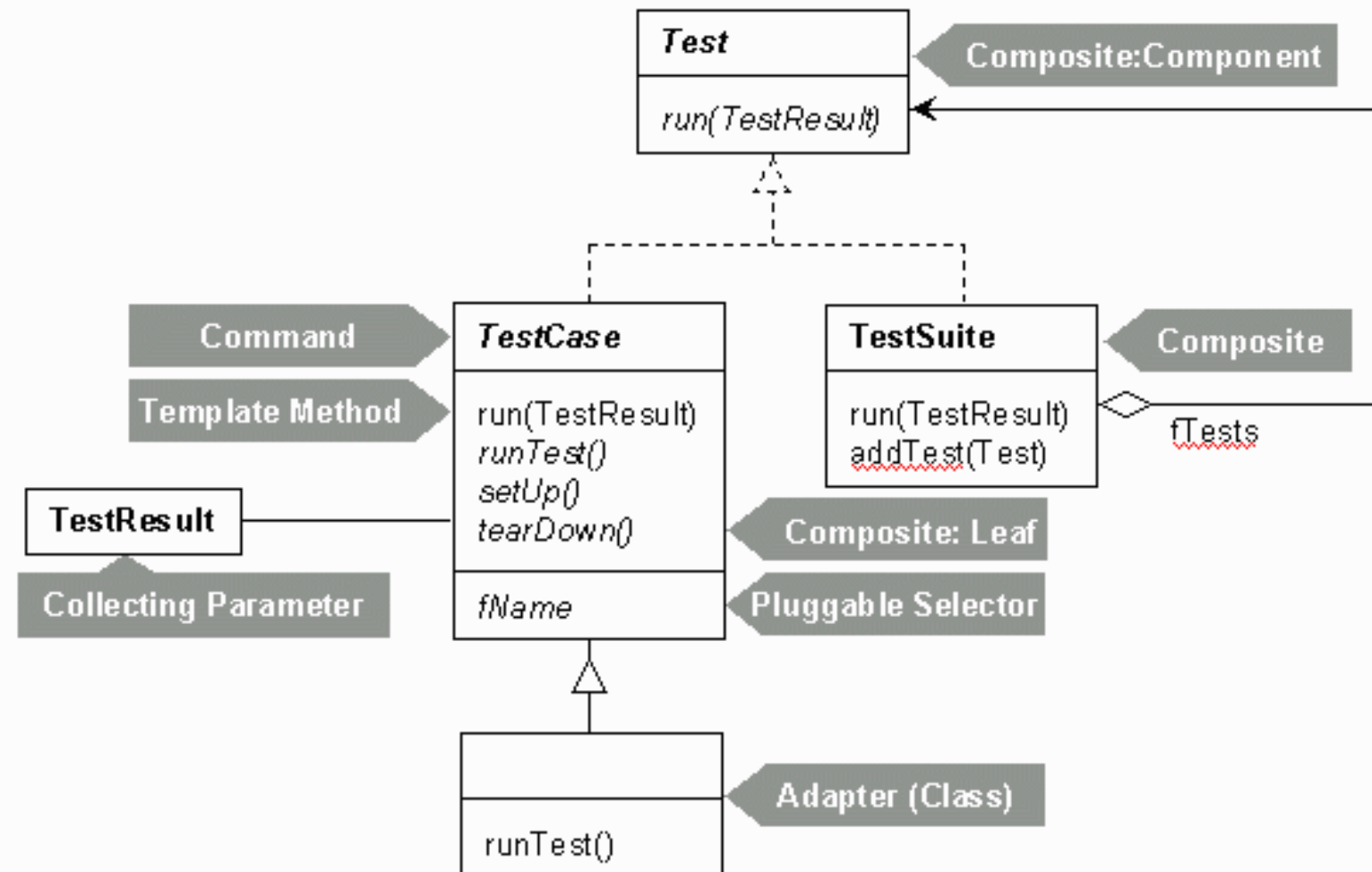
```

TestCase的run(TestResult result)方法


```
public class TestSuite implements Test {  
    private Vector fTests= new Vector();  
    public void run(TestResult result) {  
        for(Enumeration e= fTests.elements(); e.hasMoreElements();) {  
            Test test= (Test)e.nextElement();  
            test.run(result);  
        }  
    }  
    public void addTest(Test test) {  
        fTests.addElement(test);  
    }  
    ..... //omit other methods  
}
```

JUnit的TestSuit类

JUNIT PATTERNS SUMMARY



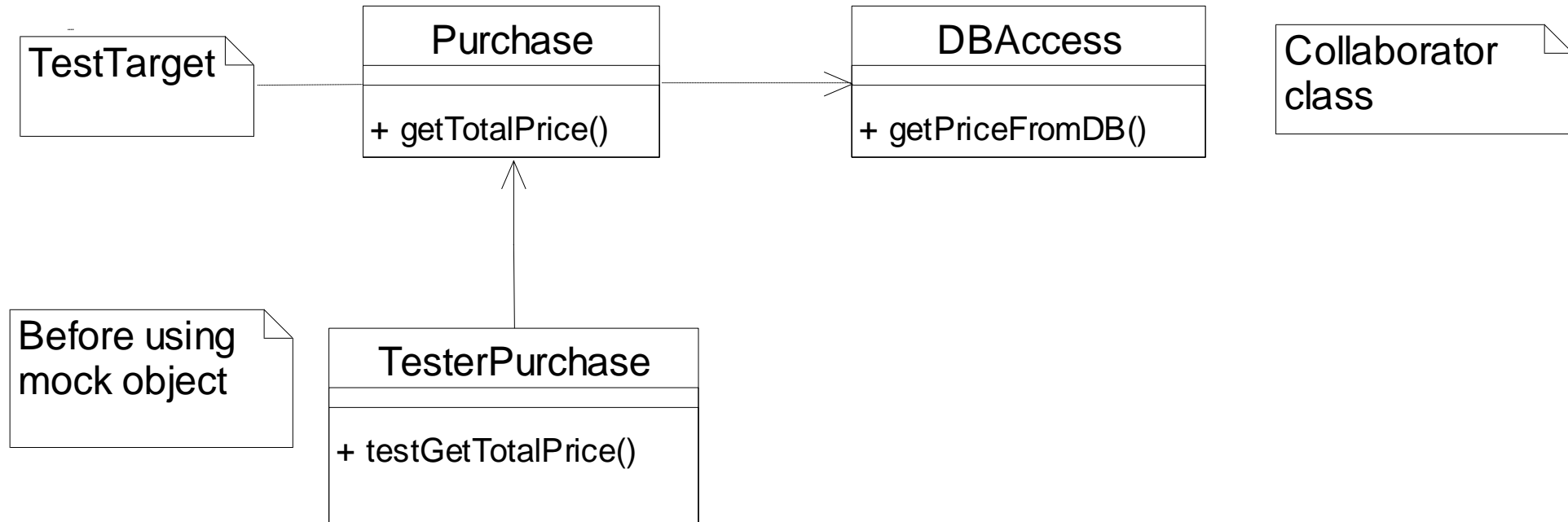
提纲

- 引言
- 使用JUnit
 - 一个简单的例子
 - JUnit安装与运行
 - JUnit常见问题
- JUnit设计
 - 设计目标
 - 设计内容
- ➡ ■ 模仿对象测试
- DbUnit
- JUnit 4.0

模仿对象测试

- 当被测目标是一个简单的对象，即对象中的方法不调用其它对象的方法，撰写基于JUnit的单元测试很简单；当我们准备测试一个依赖于其他对象的类时，就需要构造出这些合作者的实例。

THE CLASS DIAGRAM

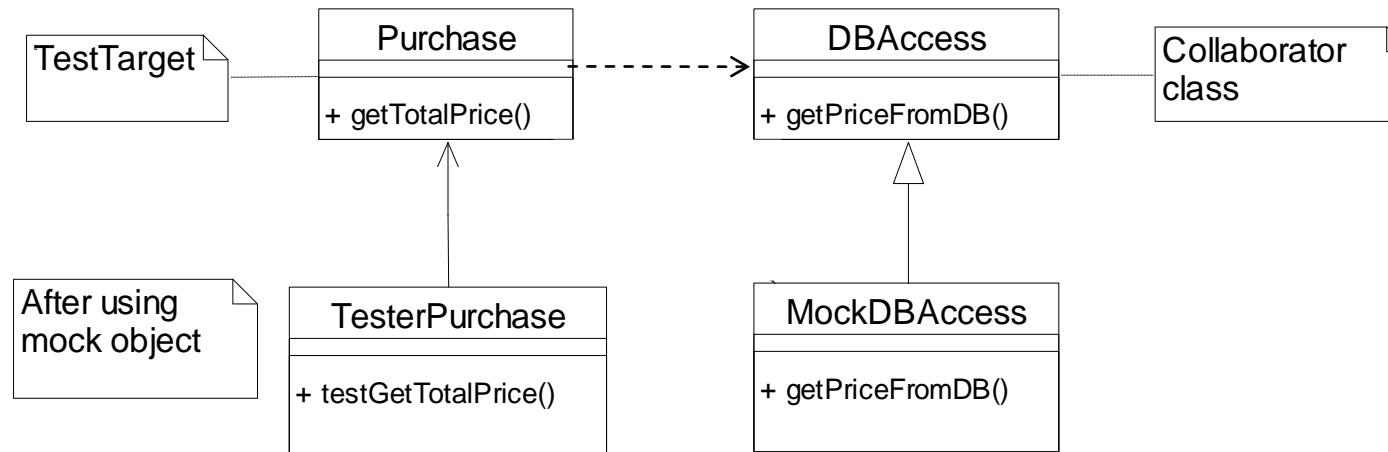


```
class Purchase {  
    ...  
    public double getTotalPrice(DBAccess dbAccess) {  
        double price = dbAccess.getPriceFromDB();  
        return price;  
    }  
    ...  
}
```

```
class PurchaseWithRealDBAccessTest extends TestCase {  
  
    public void testGetTotalPrice {  
        Purchase purchase = new Purchase() ;  
        DBAccess dbAccess = new DBAccess();  
        assertTrue(180, purchase.getTotalPrice(dbAccess));  
    }  
}
```

Using “real” object

THE CLASS DIAGRAM



```
class MockDBAccess extends DBAccess {  
    public double getPriceFromDB() {  
        double price = 180;  
        return price;  
    }  
}
```

```
class PurchaseWithMockDBAccessTest extends TestCase {  
  
    public void testGetTotalPrice {  
        Purchase purchase = new Purchase();  
        MockDBAccess mockDBAccess = new MockDBAccess();  
        assertTrue(180, purchase.getTotalPrice(mockDBAccess));  
    }  
}
```

Using Mock Object,
instead of “real” object

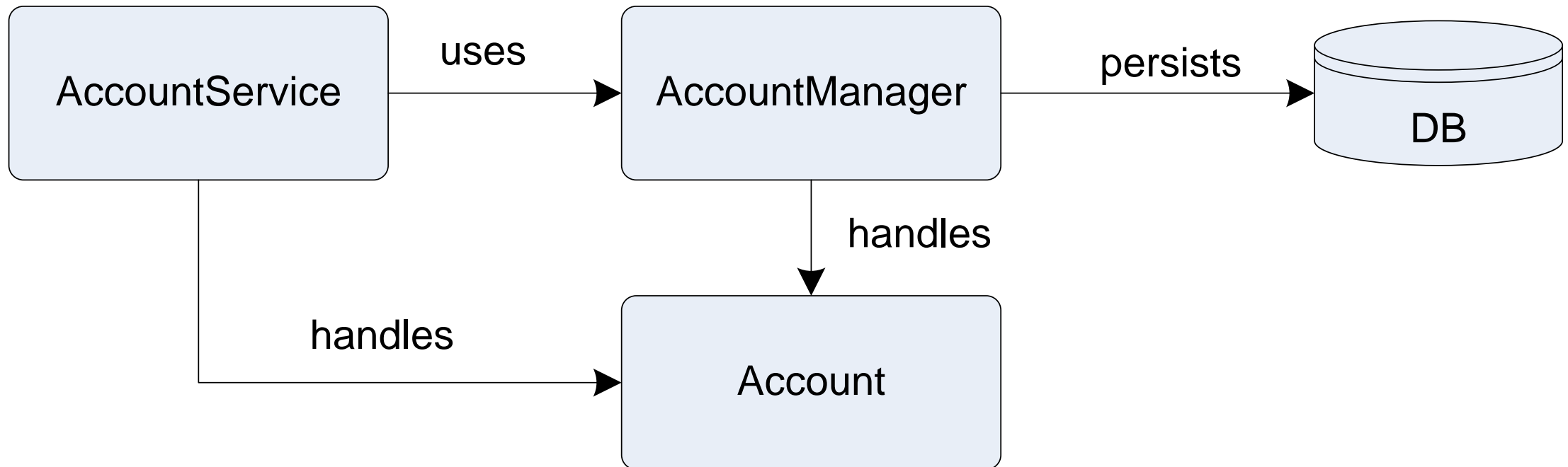
EASYMOCK 和JMOCK

- EasyMock provides Mock Objects for interfaces in JUnit tests by generating them on the fly using Java's proxy mechanism.
 - Due to EasyMock's unique style of recording expectations, most refactorings will not affect the Mock Objects.
 - So EasyMock is a perfect fit for Test-Driven Development.
- EasyMock is open source software available under the terms of the MIT license. Development and downloads are hosted on [SourceForge](#).

EASYMOCK 和JMOCK

- JMock is a library that supports test-driven development of Java code with mock objects.
- Mock objects help you design and test the interactions between the objects in your programs.
- The JMock library:
 - makes it quick and easy to define mock objects, so you don't break the rhythm of programming.
 - lets you precisely specify the interactions between your objects, reducing the brittleness of your tests.
 - works well with the autocompletion and refactoring features of your IDE
 - plugs into your favorite test framework
 - is easy to extend.

DIAGRAM OF THE ACCOUNT SERVICE



EASYSMOCK 和JMOCK

- Class Under test
- Mock Object
- JUnit with Mock Object

- EasyMock
- JMock

STEPS TO USE EASYMOCK AND JMOCK

- 1. 创建：创建Mock Object的实例
- 2. 训练：设置Mock Object中的状态和期望值
- 3. 测试：将Mock Object作为参数来调用被测对象
- 4. 验证：验证Mock Object中的一致性和被测对象的返回值或状态

EASYMOCK 和JMOCK

- easyMock的Mock机制是基于状态，首先是录制状态，记录下来待测的方法和参数，返回值等，然后切换为回复状态。
- 而JMock没有切换这一步，直接将参数返回值用一句话写出来。
- 具体对比请参看后面的表格

COMPARE BETWEEN EASYMOCK AND JMOCK

	EasyMock	jMock
通过接口模拟	是	是
控制方法有效次数	是	是
定制参数匹配	是	是
不需要状态转换	否	是
具体类模拟	是	是
具体类可有构造参数	是	否
接口统一	否	是
条件代码在一行中完成	否	是
支持其他参数规则，如not	否	是
自验证 verify()	是	是

MOCK OBJECT AND REFACTORING

- Refactoring with Mock Object

提纲

- 引言
- 使用JUnit
 - 一个简单的例子
 - JUnit安装与运行
 - JUnit常见问题
- JUnit设计
 - 设计目标
 - 设计内容
- 模仿对象测试
- ➡ ■ DbUnit
- JUnit 4.0

DBUNIT

- Writing unit and component tests for objects with external dependencies, such as databases or other objects, can prove arduous, as those dependencies may hinder isolation.
- Ultimately, effective white-box tests isolate an object by controlling outside dependencies, so as to manipulate its state or associated behavior.

DBUNIT

- Utilizing mock objects or stubs is one strategy for controlling outside dependencies. Stubbing out associated database access classes, such as those found in JDBC, can be highly effective;
- however, the mock object solution may not be possible in application frameworks where the underlying database access objects may be hidden, such as those utilizing EJBs with container-managed persistence (CMP) or Java Data Objects (JDO).

DBUNIT

- The open source DbUnit framework, created by Manuel Laflamme, provides an elegant solution for controlling a database dependency within applications by allowing developers to manage the state of a database throughout a test.
- With DbUnit, a database can be seeded with a desired data set before a test; moreover, at the completion of the test, the database can be placed back into its pre-test state.

DBUNIT

- Automated tests are a critical facet of most successful software projects.
- DbUnit allows developers to create test cases that control the state of a database during their life cycles;
 - consequently, those test cases are easily automatable, as they do not require manual intervention between tests; nor do they entail manual interpretation of results.

DBUNIT

- The DbUnit framework provides a base abstract test-case class, which extends JUnit's TestCase and is called DatabaseTestCase.
- Think of this class as a template pattern for which one must provide implementations of two hook methods: getConnection() and getDataSet().

```
protected IDatabaseConnection getConnection()  
throws Exception {  
  
    Class driverClass =  
        Class.forName("org.gjt.mm.mysql.Driver");  
  
    Connection jdbcConnection =  
        DriverManager.getConnection(  
            "jdbc:mysql://127.0.0.1/hr", "hr", "hr");  
  
    return new DatabaseConnection(jdbcConnection);  
}
```

```
protected IDataset getDataSet() throws Exception {  
    return new FlatXmlDataSet(  
        new FileInputStream("hr-seed.xml"));  
}
```

DBUNIT

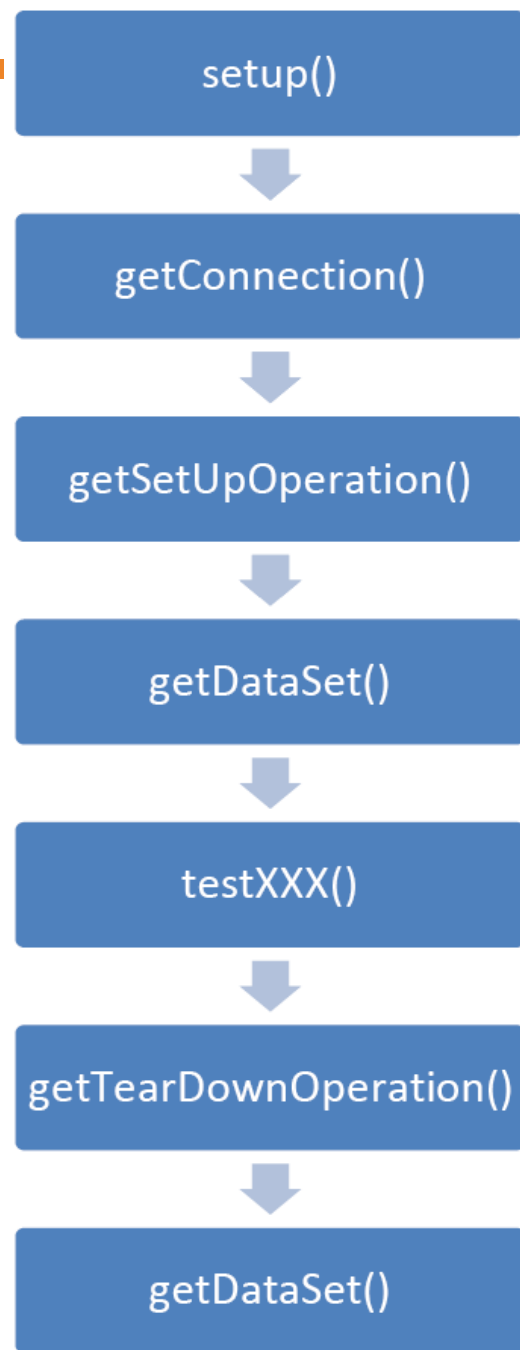
- With those two methods defined, DbUnit can function with default behavior; however, the DatabaseTestCase class provides two fixture methods that control the state of the database before and after a test: `getSetUpOperation()` and `getTearDownOperation()`.
 - An effective strategy is to have the `getSetUpOperation()` perform a REFRESH operation, which updates the desired database with the data found in the seed file.
 - Consequently, the `getTearDownOperation()` performs a NONE operation.


```
protected DatabaseOperation getSetUpOperation()
    throws Exception {
    return DatabaseOperation.REFRESH;
}

protected DatabaseOperation getTearDownOperation()
    throws Exception {
    return DatabaseOperation.NONE;
}
```

- Another effective approach is to have the `getSetUpOperation()` method perform a `CLEAN_INSERT`, which deletes all data found in tables specified in the seed file and then inserts the file's data. This tactic provides precision control of a database.

每个测试用例时是遵循以下步骤



CODE EXAMPLE

- In a J2EE human resources application, we would like to automate a series of test cases for a Session Façade that handles employee creation, retrieval, updating, and deletion.
- The remote interface contains the following business methods (the throws clauses are removed for brevity's sake):



```
public void
```

```
    createEmployee( EmployeeValueObject emplVo )
```

```
public EmployeeValueObject
```

```
    getEmployeeBySocialSecNum( String ssn )
```

```
public void
```

```
    updateEmployee( EmployeeValueObject emplVo )
```

```
public void
```

```
    deleteEmployee( EmployeeValueObject emplVo )
```

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<dataset>
```

DbUnit seed file, named employee-hr-seed.xml

```
  <EMPLOYEE employee_uid='1'
```

```
    start_date='2001-01-01'
```

```
    first_name='Drew' ssn='333-29-9999'
```

```
    last_name='Smith' />
```

```
  <EMPLOYEE employee_uid='2'
```

```
    start_date='2002-04-04'
```

```
    first_name='Nick' ssn='222-90-1111'
```

```
    last_name='Marquiss' />
```

```
  <EMPLOYEE employee_uid='3'
```

```
    start_date='2003-06-03'
```

```
    first_name='Jose' ssn='111-67-2222'
```

```
    last_name='Whitson' />
```

```
</dataset>
```

DBUNIT

- The test suite, `EmployeeSessionFacadeTest`, will
 - extend DbUnit's `DatabaseTestCase` and
 - provide implementations for both the `getConnection()` and `getDataSet()` methods,
 - where the `getConnection()` method obtains a connection to the same database instance the EJB container is utilizing, and
 - the `getDataSet()` method reads in the above `employee-hr-seed.xml` file.
- The test methods are quite simple, as DbUnit handles the complex database lifecycle tasks for us.
 - To test the `getEmployeeBySocialSecNum()` method, simply pass in a social security number from the seed file, such as "333-29-9999."

```
public void testFindBySSN() throws Exception{

    EmployeeFacade facade = //obtain somehow

    EmployeeValueObject vo =
        facade.getEmployeeBySocialSecNum("333-29-9999");

    TestCase.assertNotNull("vo shouldn't be null", vo);
    TestCase.assertEquals("should be Drew",
        "Drew", vo.getFirstName());
    TestCase.assertEquals("should be Smith",
        "Smith", vo.getLastName());
}
```

```
public void testEmployeeCreate() throws Exception{  
    EmployeeValueObject empVo =  
        new EmployeeValueObject();  
    empVo.setFirstName("Noah");  
    empVo.setLastName("Awan");  
    empVo.setSSN("564-55-5555");  
  
    EmployeeFacade empFacade = //obtain from somewhere  
    empFacade.createEmployee(empVo);  
  
    //perform a find by ssn to ensure existence  
  
}
```



```
public void testUpdateEmployee() throws Exception{
    EmployeeFacade facade = //obtain façade
    EmployeeValueObject vo =
        facade.getEmployeeBySocialSecNum("111-67-2222");
    TestCase.assertNotNull("vo was not null", vo);
    TestCase.assertEquals("first name should be Jose",
        "Jose", vo.getFirstName());
    vo.setFirstName("Ramon");
    facade.updateEmployee(vo);
    EmployeeValueObject newVo =
        facade.getEmployeeBySocialSecNum("111-67-2222");
    TestCase.assertNotNull("vo was null", newVo);
    TestCase.assertEquals("name should be Ramon",
        "Ramon", newVo.getFirstName());
}
```

```
public void testDeleteEmployee() throws Exception{
    EmployeeFacade facade = //obtain façade
    EmployeeValueObject vo =
        facade.getEmployeeBySocialSecNum("222-90-1111");
    TestCase.assertNotNull("vo was null", vo);
    facade.deleteEmployee(vo);
    try{
        EmployeeValueObject newVo =
            facade.getEmployeeBySocialSecNum("222-90-1111");
        TestCase.fail("returned removed employee");
    }catch(Exception e){
        //ignore
    }
}
```

DBUNIT – CREATE DATA SET

- Create data set
 - Create partial data set using `queryDataSet`.
 - Create full data set.
- Assert actual database table match expected table using `assertEquals()` method.

提纲

- 引言
- 使用JUnit
 - 一个简单的例子
 - JUnit安装与运行
 - JUnit常见问题
- JUnit设计
 - 设计目标
 - 设计内容
- 模仿对象测试
- DbUnit
- JUnit 4.0



JUNIT4

- 取消用反射机制来命名定位测试，而采用JDK的注释机制。方法命名更加多样！

```
@Test public void additionTest() {  
    int z = x + y;  
    assertEquals(2, z);  
}
```

JUNIT4

- 不需要继承**TestCase**类，只需要使用JDK5.0中的**static import**。

```
import static org.junit.Assert.assertEquals;
```

```
...
```

```
@Test public void addition() {
```

```
    int z = x + y;
```

```
    assertEquals(2, z);
```

```
}
```

JUNIT4

- 新的setUp()和tearDown()。
- 在JUnit4中不再使用setUp()和 tearDown()来完成测试前后的准备和清除工作，而是采用：
 - @Before,@After： 标识需要在每个测试方法前后需要进行的操作。
 - @BeforeClass,@AfterClass： 标识需要在测试类前后进行的操作。

JUNIT4

■ 异常测试 old Junit

```
try {  
    int n = 2 / 0;  
    fail("Divided by zero!");  
} catch (ArithmeticException success){  
    assertNotNull(success.getMessage());  
}
```

- 不能保证代码覆盖率，总有代码测不到。

■ 异常测试 Junit4

```
@Test(expected=ArithmeticE  
xception.class)  
public void divideByZero() {  
    int n = 2 / 0;  
}
```

- 如果该异常没有抛出（或者抛出了一个不同的异常），那么测试就将失败。

JUNIT4

- 如果需要忽略某个测试的时候，可以采用这个标识@Ignore，当使用文本界面时，会输出一个“I”（代表 ignore）。
- 标识@Test(timeout=500) 可以帮助解决单元测试的一些性能问题。
- Assert判断方法精简，并添加了：

```
public static void assertEquals(Object[] expected, Object[] actual)
```

```
public static void assertEquals(String message, Object[] expected,  
                                Object[] actual)
```

- 为数组的比较提供了方便。

REFERENCES

- Elliotte Harold,JUnit 4 抢先看,
- <http://www-128.ibm.com/developerworks/cn/java/j-junit4.html>,2005-10-13
- JUnit 4.0 in 10 minutes, <http://www.gunjandoshi.com/>,2005-06-10

总结

JUnit 3.8.x

Mock object

DBUnit

JUnit 4.x