

- ❶ You're making an out-of-JVM call to the container, so you need to specify the JNDI properties.
- ❷ Perform a JNDI lookup to retrieve the PetstoreEJB home instance.

At this stage, you have the deployment working and the test case written. If you try running the tests by typing **ant test**, you'll discover that it still fails. You still need to adjust the JNDI names: You must match the JNDI names you use to look up objects with the JNDI names under which JBoss publishes the objects.

12.9.5 Fixing JNDI names

The JNDI names used in the application are defined in `JNDINames.java`, as shown in listing 12.23.

Listing 12.23 JNDI names used to look up the J2EE objects

```
package junitbook.ejb.util;

public abstract class JNDINames
{
    public static final String QUEUE_CONNECTION_FACTORY =
        "ConnectionFactory";
    public static final String QUEUE_ORDER =
        "queue/petstore/Order";
    public static final String ORDER_LOCALHOME =
        "ejb/petstore/Order";
    public static final String PETSTORE_HOME =
        "ejb/petstore/Petstore";
}
```

In order to publish the objects under these names, you need to define a JBoss-specific `jboss.xml` file (see listing 12.24) that matches the JNDI names defined in `JNDINames.java`.

Listing 12.24 `jboss.xml` with JNDI names matching `JNDINames.java`

```
<jboss>
  <enterprise-beans>

    <session>
      <ejb-name>Petstore</ejb-name>
      <jndi-name>ejb/petstore/Petstore</jndi-name>
      <resource-env-ref>
        <resource-env-ref-name>
          →   jms/queue/petstore/Order</resource-env-ref-name>
        <jndi-name>queue/petstore/Order</jndi-name>
      </resource-env-ref>
    </session>
  </enterprise-beans>
</jboss>
```

```

</session>

<entity>
  <ejb-name>Order</ejb-name>
  <local-jndi-name>ejb/petstore/Order</local-jndi-name>
</entity>

<message-driven>
  <ejb-name>OrderProcessor</ejb-name>
  <destination-jndi-name>
    → queue/petstore/Order</destination-jndi-name>
</message-driven>

</enterprise-beans>
</jboss>

```

You must also modify the `ejbjar` Ant target so that it picks the new JBoss-specific files you have added (changes are in bold):

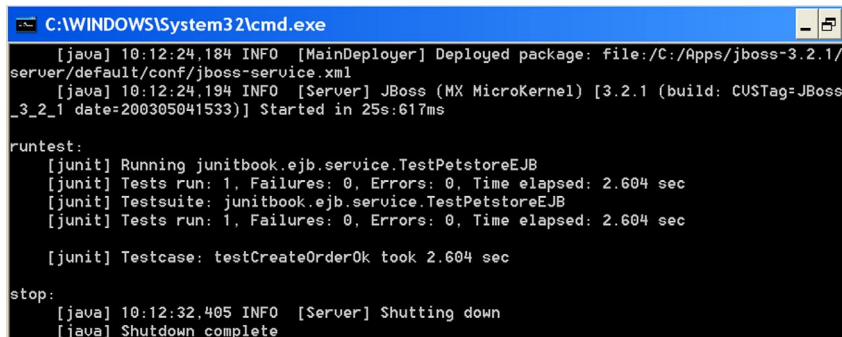
```

<target name="ejbjar" depends="compile">
  <jar destfile="${target.dir}/ejb.jar">
    <metainf dir="${conf.dir}">
      <include name="ejb-jar.xml"/>
      <include name="jbosscmp-jdbc.xml"/>
      <include name="jboss.xml"/>
    </metainf>
    <fileset dir="${target.classes.java.dir}"/>
  </jar>
</target>

```

12.9.6 Running the tests

At last, you're ready to run the tests. They should now execute fine. Typing `ant test` generates the results shown in figure 12.7.



```

C:\WINDOWS\System32\cmd.exe
[java] 10:12:24,184 INFO [MainDeployer] Deployed package: file:/C:/Apps/jboss-3.2.1/
server/default/conf/jboss-service.xml
[java] 10:12:24,194 INFO [Server] JBoss (MX MicroKernel) [3.2.1 (build: CUSTag=JBoss
_3_2_1 date=200305041533)] Started in 25s:617ms

runtest:
[junit] Running junitbook.ejb.service.TestPetstoreEJB
[junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 2.604 sec
[junit] Testsuite: junitbook.ejb.service.TestPetstoreEJB
[junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 2.604 sec

[junit] Testcase: testCreateOrderOk took 2.604 sec

stop:
[java] 10:12:32,405 INFO [Server] Shutting down
[java] Shutdown complete

```

Figure 12.7 Successful execution of pure JUnit tests calling the running container remotely

12.10 Using Cactus

Let's now use Cactus to run some EJB unit tests. Cactus has several advantages over a pure JUnit solution:

- Cactus lets you unit-test Enterprise Beans using local interfaces, because Cactus tests run inside the container. For example, you have not been able to perform integration unit tests for the Order CMP entity bean (which uses a local interface). We'll demonstrate how to do this using Cactus.
- The Ant scripts were a bit complex because you had to script the container's start and stop (JBoss, in this case). Cactus provides an Ant integration that simplifies this operation. In addition, this Cactus Ant task supports several containers out of the box (JBoss, Tomcat, Resin, Orion, WebLogic, and so on), making it easy to run the tests on any container.

At this time of this writing, Cactus doesn't yet provide EJB Redirectors you can use to directly write tests against EJBs as you have done for servlets, taglibs, and filters in previous chapters.³ Thus, you can't yet perform fine-grained integration tests, such as testing exceptions cases.

12.10.1 Writing an EJB unit test with Cactus

The current Cactus solution consists of transparently using the Cactus servlet Redirector so that the tests are executed within the context of the web container. For the pure JUnit solution, the tests perform a lookup on the EJB to unit-test and call its method to test. The difference is that this lookup is performed from the web container context and thus also works for local interfaces.

Let's demonstrate this on the OrderEJB CMP entity bean (listing 12.25).

Listing 12.25 OrderEJB unit test as a Cactus ServletTestCase

```
package junitbook.ejb.domain;

import java.util.Date;

import javax.naming.InitialContext;

import junit.framework.TestCase;
import junitbook.ejb.util.JNDINames;

public class TestOrderEJB extends ServletTestCase
```

³ The addition of EJB redirectors is scheduled for Cactus 1.6 or later (it is on the todo list: <http://jakarta.apache.org/cactus/participating/todo.html>).

```

{
    public void testEjbCreateOk() throws Exception
    {
        OrderLocalHome orderHome =
            (OrderLocalHome) new InitialContext().lookup(
                JNDINames.ORDER_LOCALHOME);

        Date date = new Date();
        String item = "item 1";

        OrderLocal order = orderHome.create(date, item);

        assertEquals(date, order.getOrderDate());
        assertEquals(item, order.getOrderItem());
        assertEquals(new Integer(date.hashCode()
            + item.hashCode()), order.getOrderId());
    }
}

```

❶ **Lookup
executed from
inside container**

Compared to listing 12.22, you don't need to set up JNDI properties before calling `new InitialContext()` (❶) because the test runs inside the container.

Let's now run this test using Ant.

12.10.2 Project directory structure

Figure 12.8 defines the directory structure for the Cactus tests. Compared to the directory structure from figure 12.4, you add the `conf/cactus/` directory, which contains configuration files for running Cactus tests, and the `src/test-cactus/` directory, which contains the Cactus unit tests. You also add an Ant build-`cactus.xml` buildfile that contains the Ant targets to automatically execute the Cactus tests using the Cactus/Ant integration. (For more details on the Cactus/Ant integration, refer to section 11.5.2 from chapter 11.)

You'll use the same Ant script introduced in listing 12.16 to package the application as an ear file. However, you need to add some targets to package the Cactus tests and execute them.

12.10.3 Packaging the Cactus tests

Because the Cactus tests execute in the web container, you need to package them in a war inside the ear. Fortunately, Cactus provides a `cactifywar` Ant task that makes the creation of this war easy:

```

<cactifywar version="2.3" destfile="${target.dir}/cactus.war"
    mergewebxml="${conf.dir}/cactus/web.xml">
    <classes dir="${target.classes.cactus.dir}"/>
</cactifywar>

```

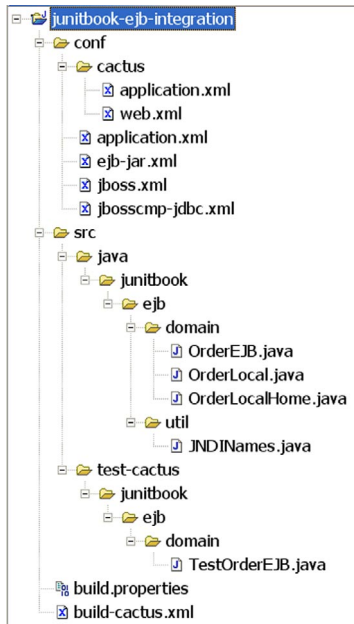


Figure 12.8
Full directory structure, including
configuration files and Ant build files
for Cactus tests

The version attribute specifies that you build a war for the Servlet API 2.3. Notice the `mergewebxml` attribute. This is needed because `TestOrderEJB` is called from a web context and is calling an EJB. Thus, as the J2EE specification mandates, you need an `<ejb-local-ref>` in the war `web.xml` file, as shown in listing 12.26.

Listing 12.26 `conf/cactus/web.xml` containing `ejb-local-ref` entry to call `OrderEJB`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <ejb-local-ref>
    <ejb-ref-name>Order</ejb-ref-name>
    <ejb-ref-type>Entity</ejb-ref-type>
    <local-home>junitbook.ejb.domain.OrderLocalHome</local-home>
    <local>junitbook.ejb.domain.OrderLocal</local>
    <ejb-link>Order</ejb-link>
  </ejb-local-ref>
</web-app>
```

The last packaging step is to package the war in the ear:

```
<ear update="true" destfile="${target.dir}/ejb.ear"
  appxml="${conf.dir}/cactus/application.xml">
  <fileset dir="${target.dir}">
    <include name="cactus.war"/>
  </fileset>
</ear>
```

You update the application ear by overwriting the application.xml deployment descriptor with the special Cactus one. This step is necessary in order to add the Cactus war in application.xml, as shown in listing 12.27. This step is required only because your production ear doesn't contain any war. Had it contained one, you could have reused this war by cactifying it. In that case, you wouldn't have needed to provide an additional application.xml file containing Cactus-specific definitions.

Listing 12.27 Special application.xml file containing the Cactus war definition

```
<?xml version="1.0"?>
<!DOCTYPE application PUBLIC
'-//Sun Microsystems, Inc.//DTD J2EE Application 1.2//EN'
'http://java.sun.com/j2ee/dtds/application_1_2.dtd'>
<application>
  <display-name>ejb</display-name>
  <description>EJB Chapter Sample Application</description>
  <module>
    <ejb>ejb.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>cactus.war</web-uri>
      <context-root>test</context-root>
    </web>
  </module>
</application>
```

The full Ant script is shown in listing 12.28.

Listing 12.28 Cactification of the application ear

```

<?xml version="1.0"?>

<project name="Ejb" default="test" basedir=".">
[...]
  <property name="src.cactus.dir"
    location="${src.dir}/test-cactus"/>
[...]
  <property name="target.classes.cactus.dir"
    location="${target.dir}/classes-test-cactus"/>
[...]
  <target name="compile.cactus" depends="compile">
    <mkdir dir="${target.classes.cactus.dir}"/>
    <javac destdir="${target.classes.cactus.dir}"
      srcdir="${src.cactus.dir}">
      <classpath>
        <pathelement location="${target.classes.java.dir}"/>
        <pathelement location="${cactus.jar}"/>
        <pathelement location="${j2ee.jar}"/>
      </classpath>
    </javac>
  </target>

  <target name="ear.cactify" depends="compile.cactus,ear">
    <taskdef resource="cactus.tasks">
      <classpath>
        <pathelement location="${cactus.ant.jar}"/>
        <pathelement location="${cactus.jar}"/>
        <pathelement location="${logging.jar}"/>
        <pathelement location="${aspectjrt.jar}"/>
        <pathelement location="${httpClient.jar}"/>
      </classpath>
    </taskdef>

    <cactifywar version="2.3" destfile="${target.dir}/cactus.war"
      mergewebxml="${conf.dir}/cactus/web.xml">
      <classes dir="${target.classes.cactus.dir}"/>
    </cactifywar>

    <ear update="true" destfile="${target.dir}/ejb.ear"
      appxml="${conf.dir}/cactus/application.xml">
      <fileset dir="${target.dir}">
        <include name="cactus.war"/>
      </fileset>
    </ear>

  </target>
</project>

```

12.10.4 Executing the Cactus tests

You execute the Cactus tests using the Cactus-provided `cactus` task. This is the nice part, compared to the pure JUnit approach from section 12.9, because the `cactus` task does everything for you: It deploys the ear, starts the container, executes the tests, and stops the container. The `cactus` task extends the `JUnit` Ant task and thus inherits from all its features. Listing 12.29 demonstrates how to use the `cactus` task to run the `TestOrderEJB` test.

Listing 12.29 Running Cactus tests automatically with the `cactus` task

```
<?xml version="1.0"?>

<project name="Ejb" default="test" basedir=". ">
[... ]
  <target name="test" depends="ear.cactify">

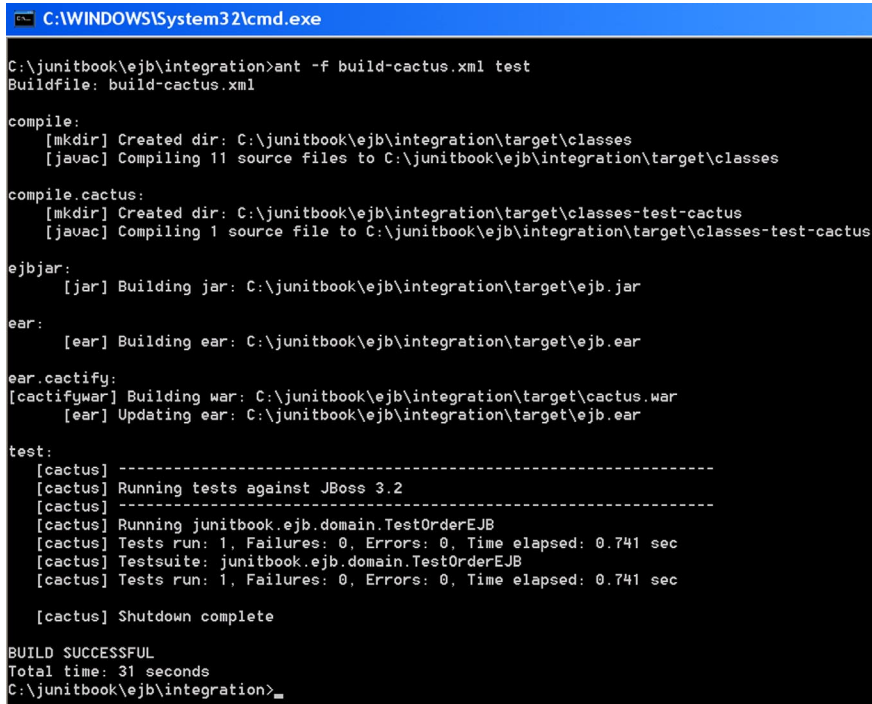
    <cactus earfile="${target.dir}/ejb.ear" fork="yes"
      printsummary="yes" haltonerror="true"
      haltonfailure="true">
      <containerset>
        <jboss3x dir="${cactus.home.jboss3x}"
          output="jbossresult.txt"/>
      </containerset>
      <formatter type="brief" usefile="false"/>
      <test name="junitbook.ejb.domain.TestOrderEJB"/>
      <classpath>
        <pathelement location="${target.classes.java.dir}"/>
        <pathelement location="${target.classes.cactus.dir}"/>
      </classpath>
    </cactus>

  </target>

</project>
```

**Run Cactus tests in
JBoss container**

Executing the tests by typing `ant -f build-cactus.xml test` yields the result shown in figure 12.9.



```

C:\WINDOWS\System32\cmd.exe

C:\junitbook\ejb\integration>ant -f build-cactus.xml test
Buildfile: build-cactus.xml

compile:
  [mkdir] Created dir: C:\junitbook\ejb\integration\target\classes
  [javac] Compiling 11 source files to C:\junitbook\ejb\integration\target\classes

compile.cactus:
  [mkdir] Created dir: C:\junitbook\ejb\integration\target\classes-test-cactus
  [javac] Compiling 1 source file to C:\junitbook\ejb\integration\target\classes-test-cactus

ejbjar:
  [jar] Building jar: C:\junitbook\ejb\integration\target\ejb.jar

ear:
  [ear] Building ear: C:\junitbook\ejb\integration\target\ejb.ear

ear.cactify:
  [cactifywar] Building war: C:\junitbook\ejb\integration\target\cactus.war
  [ear] Updating ear: C:\junitbook\ejb\integration\target\ejb.ear

test:
  [cactus] -----
  [cactus] Running tests against JBoss 3.2
  [cactus] -----
  [cactus] Running junitbook.ejb.domain.TestOrderEJB
  [cactus] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.741 sec
  [cactus] Testsuite: junitbook.ejb.domain.TestOrderEJB
  [cactus] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0.741 sec

  [cactus] Shutdown complete

BUILD SUCCESSFUL
Total time: 31 seconds
C:\junitbook\ejb\integration>_

```

Figure 12.9 Result of executing Cactus EJB tests with the Cactus/Ant integration

12.11 Summary

EJBs are complex and powerful beasts. Unit-testing doesn't have to be difficult. This chapter has demonstrated several techniques for handling EJB unit tests: mock objects for out-of-the-container testing of any kind of EJBs (session beans, entity beans, message-driven beans) and integration unit testing for testing Enterprise Beans when they run inside the container. We demonstrated integration unit testing with two tools: pure JUnit tests that call the EJBs remotely; and Cactus, which runs the tests from inside the container and lets you unit-test local interfaces.

When you're performing integration unit tests, writing the tests isn't even half the story. The hard part, which isn't specifically related to unit testing, is about automating the packaging of the application, its deployment and test execution, and the start/stop of containers. We've demonstrated several techniques using Ant, including using some Cactus custom-made Ant tasks that have helped in this endeavor.

The source code

This appendix covers

- Installing the book source code
- Software versions required
- Directory structure conventions

This appendix gives an overview of the book's source code, where to find it, how to install it, and how to run it. When we were writing, we decided to donate all of the book's source code to the Apache Software Foundation because we've used a lot of frameworks from there in the making of this book. Thus we have made our source code available as open source on Sourceforge at <http://junitbook.sourceforge.net/>.

We're also committed to maintaining this source code and fixing it if bugs are found, as a standard open source project. In addition, a Sourceforge forum has been set up for discussing the code at http://sourceforge.net/forum/forum.php?forum_id=291665.

A.1 Getting the source code

There are two possibilities for getting the source code on your local machine:

- Download a released version from http://sourceforge.net/project/show-files.php?group_id=68011 and unzip it somewhere on your hard drive.
- Use a CVS client and get the source from CVS HEAD. Getting the source from CVS is explained at http://sourceforge.net/cvs/?group_id=68011.

Either way, place the source code in a local directory named `junitbook/` (for example `c:\junitbook` on Windows or `/opt/junitbook` on UNIX).

A.2 Source code overview

Once you put the source code in the `junitbook/` directory, you should have the directory structure shown in figure A.1. Each directory represents the source code for a chapter of the book (except the `repository/` directory, which contains external jars required by the chapter projects). The mapping between chapter names and directory names is listed in table A.1.

Each directory maps directly to a project. A project is a way to regroup Java sources, test sources, configurations files, and so on under a single location. A project also has a build, which lets you perform various actions such as compiling the code, running the tests, and generating the Javadoc. We have used different build tools (Ant and Maven) for the different projects, as explained in the chapter matching each project.



Figure A.1
Directory structure for the source code, shown here in Windows Explorer. (Note that the directories are colored by the TortoiseCVS CVS client.)

Table A.1 Mappings between chapter names and source directory names

Chapter name	Directory name
Chapter 1: JUnit jumpstart	junitbook/jumpstart/
Chapter 2: Exploring JUnit	junitbook/exploring/
Chapter 3: Sampling JUnit	junitbook/sampling/
Chapter 4: Examining software tests	junitbook/examining/
Chapter 5: Automating JUnit	junitbook/automating/
Chapter 6: Coarse-grained testing with stubs	junitbook/coarse/
Chapter 7: Testing in isolation with mock objects	junitbook/fine/
Chapter 8: In-container testing with Cactus	junitbook/container/
Chapter 9: Unit-testing servlets and filters	junitbook/servlets/
Chapter 10: Unit-testing JSPs and taglibs	junitbook/pages/
Chapter 11: Unit-testing database applications	junitbook/database/
Chapter 12: Unit-testing EJBs	junitbook/ejb/

A.3 External libraries

You may have noticed a directory named `repository/` in figure A.1. It contains the different external libraries (jars) that all the other projects need in order to compile and run. As a convenience, we're making them readily available to you to prevent you from having to fish for them all over the Net.

The directory structure of `repository/` is of the format `<library name>/jars/<library name>-<version>.jar`, as shown in figure A.2.

NOTE We have chosen this directory layout because it is the one needed to make the `repository/` project a remote Maven artifact repository (see chapter 5 for a presentation of the Maven repositories). When you install Maven the first time, its remote repository is configured to point to `http://www.ibiblio.org/maven/`, which is the official Maven repository containing hundreds of open source jars. Maven supports having several remote repositories, so adding yours is as easy as adding the following in your `build.properties`.

On Windows:

```
maven.repo.remote =  
→ http://www.ibiblio.org/maven/,file://c:/junitbook  
→ repository/
```

On UNIX:

```
maven.repo.remote =  
→ http://www.ibiblio.org/maven/,file:///opt/junitbook  
→ repository/
```

With this configuration, Maven will look for any dependency on `ibiblio` first and then in the `junitbook` repository in your filesystem.

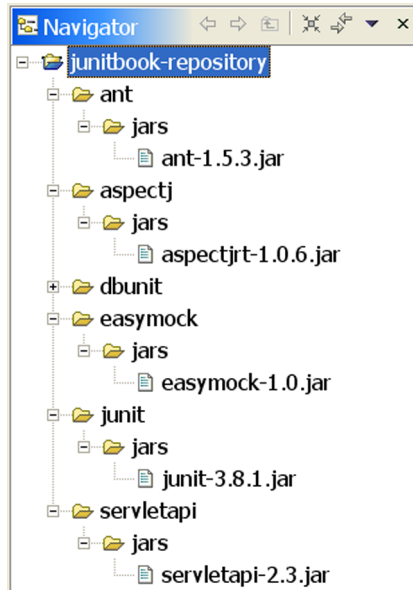


Figure A.2
Some jars from the repository/
directory, shown in the Eclipse
navigator view

A.4 Jar versions

Table A.2 lists the versions of all external jars and applications used in the projects. We recommend using these versions when you try the book examples.

Table A.2 External jar/application versions (sorted in alphabetical order)

External project name	Version	Project URL
Ant	1.5.3	http://ant.apache.org/
AspectJ	1.0.6	http://eclipse.org/aspectj/
Cactus	1.5	http://jakarta.apache.org/cactus/
Commons BeanUtils	1.6.1	http://jakarta.apache.org/commons/beanutils.html
Commons Collections	2.1	http://jakarta.apache.org/commons/collections.html
Commons HttpClient	2.0	http://jakarta.apache.org/commons/httpclient/
Commons Logging	1.0.3	http://jakarta.apache.org/commons/logging.html
DbUnit	1.5.5	http://dbunit.sourceforge.net/
EasyMock	1.0	http://easymock.org/
Eclipse	2.1	http://eclipse.org/

continued on next page

Table A.2 External jar/application versions (sorted in alphabetical order) *(continued)*

External project name	Version	Project URL
HttpUnit	1.5.3	http://httpunit.sourceforge.net/
Jakarta Taglibs / JSTL	1.0.2	http://jakarta.apache.org/taglibs/
JBoss	3.2.1	http://jboss.org/
Jetty	4.2.11	http://jetty.mortbay.org/
JUnit	3.8.1	http://junit.org/
Maven	1.0 beta 10	http://maven.apache.org/
MockObjects	0.09	http://www.mockobjects.com/
MockMaker plugin for Eclipse	1.12.0	http://www.mockmaker.org/
Servlet API	2.3	http://www.ibiblio.org/maven/servletapi/jars/
Tomcat	4.1.24	http://jakarta.apache.org/tomcat/

A.5 Directory structure conventions

For each project, we have followed the directory conventions listed in table A.3.

Table A.3 Directory structure conventions

Directory name	Explanation
<code><project name>/src/java</code>	Java runtime sources.
<code><project name>/src/test</code>	Java test sources.
<code><project name>/src/test-cactus</code>	Java Cactus test sources.
<code><project name>/src/webapp</code>	Web app resources (JSPs, <code>web.xml</code> , taglibs, and so on).
<code><project name>/conf</code>	Configuration files (if any).
<code><project name>/target</code>	Directory created by the build process (Ant or Maven) to store generated files and temporary files. It can be safely deleted, because it's re-created by the build.

Eclipse quick start

This appendix covers

- Installing Eclipse
- Setting up the book source code in Eclipse
- Running JUnit tests in Eclipse

In this appendix, you will learn how to install Eclipse (<http://eclipse.org/>) and how to run the book's source code from within the IDE. This appendix is meant as a quick start to get you up and running quickly with Eclipse and with the integrated JUnit.

B.1 Installing Eclipse

Installing Eclipse is very simple; the process consists of downloading Eclipse from <http://eclipse.org/> and then unzipping it to somewhere on your hard drive. We recommend downloading Eclipse 2.1 or greater. In the remainder of this appendix, we'll assume Eclipse is installed in `[ECLIPSE_HOME]` (for example, `c:\eclipse-2.1`).

B.2 Setting up Eclipse projects from the sources

The good news is that it's extremely easy to set up an Eclipse project, because we have provided the Eclipse project files with the book's source code distribution. Please refer to appendix A ("The source code") for directory structure organization and project names.

The first Eclipse project to import corresponds to the `junitbook/repository/` directory. It contains all the external libraries (jars) required by all the other projects. All the other Eclipse projects for this book depend upon this repository project for their classpath, which is why you need to import it first.

To import this project, select **File**→**Import** and then select **Existing Project into Workspace**. Point the Project Content to the `junitbook/repository/` directory on your hard disk, as shown in figure B.1.

Repeat the same process for all the projects you wish to see in your Eclipse workspace. If you import all the projects, you should end up with the workspace shown in figure B.2.

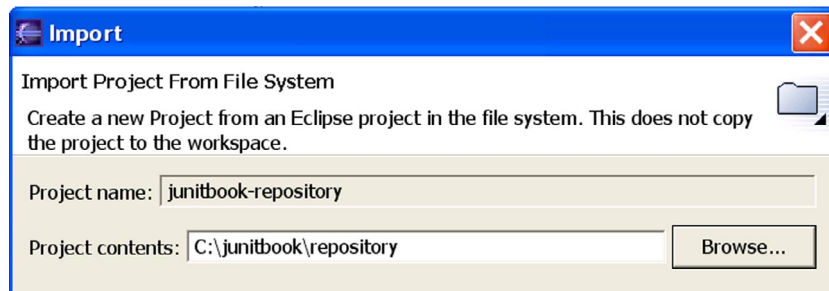


Figure B.1 Importing the `junitbook-repository` project into Eclipse

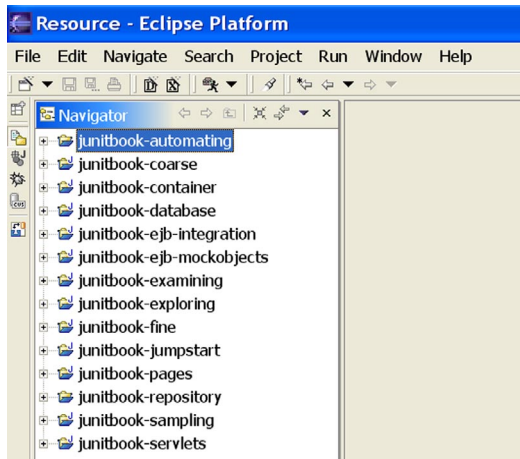




Figure B.2
Eclipse workspace when all the book projects have been imported

B.3 Running JUnit tests from Eclipse

To run a JUnit test in Eclipse, select the Java perspective () , click on the test class to execute, click the Run As icon arrow () , and select JUnit Test. Figure B.3 shows what you'll get if you run the TestAccount test case found in the junitbook-examining Eclipse project from chapter 4.

For full details on how to run JUnit tests from Eclipse, please see the integrated Eclipse Help: Click Help→Help Contents. Then, in the Help browser, select the

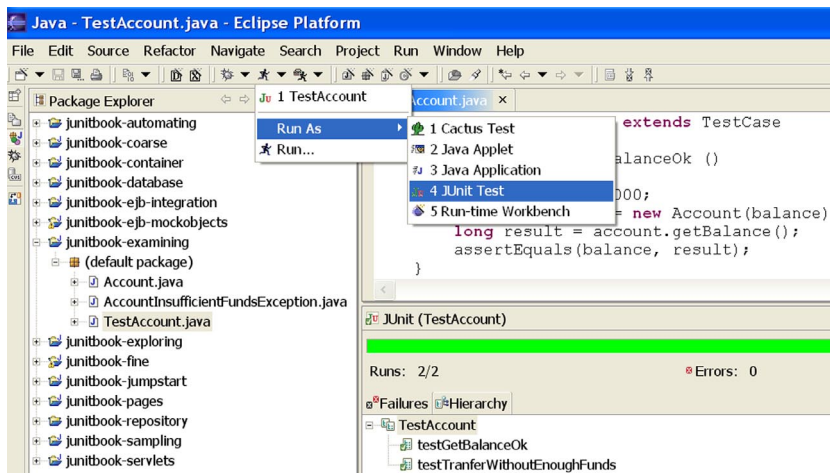


Figure B.3 Running the TestAccount test case in Eclipse using the built-in JUnit plugin

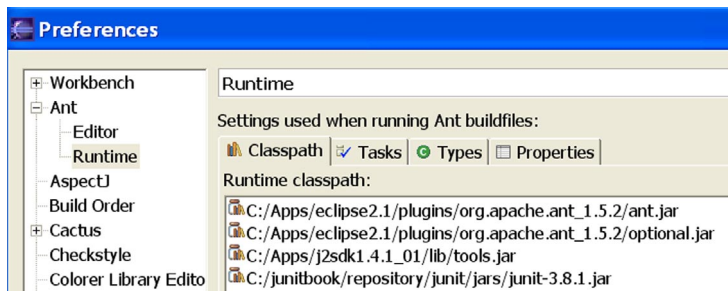



Figure B.4 Adding the JDK `tools.jar` and the JUnit jar to the Ant classpath in Eclipse


following topic: Java Development User Guide→Getting Started→Basic Tutorial→Writing and Running JUnit tests.

B.4 Running Ant scripts from Eclipse

Before running Ant scripts, make sure you've added the JDK `tools.jar` library to your Ant classpath (it's needed by the Ant `javac` task). In addition, you also need to add the JUnit jar to the Ant classpath. To do so, select **Window**→**Preferences**, choose **Ant**→**Runtime** in the Preferences dialog box, and add the jars as shown in figure B.4.

To execute a target from an Ant buildfile, first tell Eclipse to display the Ant view by clicking the **Window**→**Show View** menu entry and selecting **Ant**. Figure B.5 shows the result.

Then, click the  icon to add a buildfile to the Ant view. For example, add the `build.xml` file from the `junitbook-sampling` project. The Ant view now lists all the Ant targets it has found in the `build.xml` file, highlighting the default target (see figure B.6).

To execute a target, select it and click the  button. Figure B.7 shows the result of executing the `compile` target. Note that Eclipse captures the Ant output and displays it in the console view at the bottom right of the figure.

For full details on how to run Ant scripts from Eclipse, please see the integrated Eclipse Help: Click **Help**→**Help Contents**. Then, in the Help browser,



Figure B.5 Eclipse displays the Ant view.

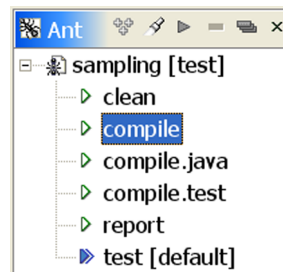


Figure B.6 The Ant view displays all the Ant targets found in `build.xml`.

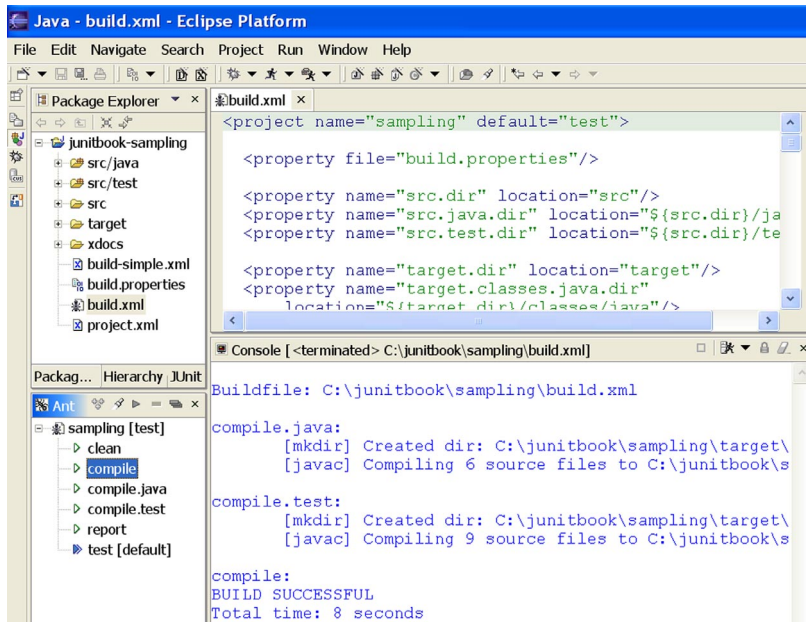


Figure B.7 Result of executing the compile Ant target for the junitbook-sampling project

select the following topic: Workbench User Guide→Getting Started→Ant & External Tools Tutorial→Eclipse Ant Basics.

B.5 Running Cactus tests from Eclipse

Executing a Cactus test involves several steps: packaging the application to run as a war file, deploying it to the container, starting the container, and launching the tests using a JUnit runner. Launching a JUnit runner is easy, as demonstrated by the previous section. However, the other steps are harder to perform. The Cactus project provides two solutions to help:

- A Jetty integration, which can be run from any IDE (including Eclipse). This integration is described in detail in chapter 8 (“In-container testing with Cactus”).
- A Cactus plugin for Eclipse, which lets you run Cactus tests within several containers (Resin, WebLogic, Tomcat, Orion, and so on). However, this plugin is still experimental (at the time of this writing). Full information about using the Eclipse plugin is available on the Cactus web site at <http://jakarta.apache.org/cactus/integration/eclipse/index.html>.

references

Bibliography

- Alur, Deepak, John Crupi, and Dan Malks. *Core J2EE Patterns: Best Practices and Design Strategies*. Upper Saddle River, NJ: Prentice Hall, 2001.
- Beck, Kent. *Smalltalk Best Practice Patterns*. Upper Saddle River, NJ: Prentice Hall, 1996.
- . *Extreme Programming Explained: Embrace Change*. Reading, MA: Addison-Wesley, 1999.
- . *Test Driven Development: By Example*. Boston: Addison-Wesley, 2003.
- Earles, John. “Frameworks! Make Room for Another Silver Bullet.” http://www.cbdh-q.com/PDFs/cbdhq_000301je_frameworks.pdf.
- Fowler, Martin. “The New Methodology.” <http://www.martinfowler.com/articles/newMethodology.html>.
- . *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley, 2003.
- . *Refactoring: Improving the Design of Existing Code*. Reading, MA: Addison-Wesley, 1999.
- Fowler, Martin, and Kendall Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Reading, MA: Addison-Wesley, 2000.
- Gamma, Erich, et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

- Hatcher, Erik, and Steve Loughran. *Java Development with Ant*. Greenwich, CT: Manning, 2003. <http://www.manning.com/hatcher/>.
- Jeffries, Ron. On the TestDrivenDevelopment mailing list: <http://groups.yahoo.com/group/testdrivendevelopment/message/3914>.
- Johnson, Ralph, and Brian Foote. “Designing Reusable Classes.” *Journal of Object-Oriented Programming* 1.5 (June/July 1988): 22–35. <http://www.laputan.org/drc/drc.html>.
- Marick, Brian. “How Many Bugs Do Regression Tests Find?” <http://www.testingcraft.com/regression-test-bugs.html>.
- Potapov, Roman. “The Origin of Murphy’s Law.” <http://www.geocities.com/murphylawsite/>.
- Rainsberger, J. B. “Refactoring: Replace Subclasses with Collaborators.” <http://www.diasparsoftware.com/articles/refactorings/replaceSubclassWithCollaborator.html>.
- Sisson, Derek. “Types of Tests.” <http://www.philosophie.com/testing/tests.html>.
- Walls, Craig, and Norman Richards. *XDoclet in Action*. Greenwich, CT: Manning, 2003. <http://books.manning.com/walls/>.

Software directory

The software packages listed here are covered by the main text. Appendix A also provides a detailed list of the software packages and versions used by the book’s source code.

Table R.1 Software directory

Name	Web site	Quick description
Ant	http://ant.apache.org/	Build tool
AspectJ	http://eclipse.org/aspectj/	AOP framework
Cactus	http://jakarta.apache.org/cactus/	J2EE unit-testing framework
Clover	http://www.thecortex.net/clover/	Test coverage tool
Commons BeanUtils	http://jakarta.apache.org/commons/beanutils/	Reflection and introspection utilities for working on JavaBeans

continued on next page

Table R.1 Software directory (continued)

Name	Web site	Quick description
Commons Collections	http://jakarta.apache.org/commons/collections.html	Complements the Java Collections API with other powerful data structures
Commons HttpClient	http://jakarta.apache.org/commons/httpclient/	HTTP client
Commons Logging	http://jakarta.apache.org/commons/logging.html	Logging façade to other logging systems
DbUnit	http://www.dbunit.org/	Database unit-testing framework
EasyMock	http://easymock.org/	Mock objects generation framework
Eclipse	http://www.eclipse.org/	Tools platform and Java IDE
HttpUnit	http://httpunit.sourceforge.net/	JUnit extension for testing web applications
JBoss	http://www.jboss.org/	J2EE container
Jester	http://jester.sourceforge.net/	Tool to verify quality of unit tests
Jetty	http://jetty.mortbay.org/	Servlet/JSP container
JMeter	http://jakarta.apache.org/jmeter/	Load-testing tool
JSTL	http://java.sun.com/products/jsp/jstl/	Standard JSP tag libraries
JUnit	http://junit.org/	Unit-testing framework
JUnitBook	http://sourceforge.net/projects/junitbook/	Source code for <i>JUnit in Action</i>
JUnitPerf	http://www.clarkware.com/software/JUnitPerf.html	JUnit extension for measuring performance and scalability
Maven	http://maven.apache.org/	Project comprehension build tool
MockObjects	http://www.mockobjects.com/	Mock-objects framework

continued on next page

Table R.1 Software directory *(continued)*

Name	Web site	Quick description
MockMaker plugin for Eclipse	http://www.mockmaker.org/	Static mock-objects generation framework
Taglibs	http://jakarta.apache.org/taglibs/	Jakarta's implementation of JSTL
Tomcat	http://jakarta.apache.org/tomcat/	Servlet/JSP container
xPetstore	http://xpetstore.sf.net/	Sample Petstore application

Software licenses

- The source code created for this book is provided under the Apache Software License (<http://apache.org/LICENSE>).
- JUnit is provided under the Common Public License (<http://oss.software.ibm.com/developerworks/oss/license-cpl.html>).

Numerics

302 response code (redirect) 73

A

abstract classes, testing 311

AbstractHttpHandler method
(Jetty) 128

AbstractHttpListener class
(Jetty) 128

acceptance testing 6, 71, 75–76

Account class 82, 141–145

account transfer testing 141

AccountManager interface
141–143

AccountService.transfer
method 141, 143–146

add method 7–8

addAccount method 144

addHandler method 42, 50,
60, 128

addTest method 23

addTestSuite method 24

Administration

application 188–189, 216,
240, 261

administrators 188

AdminServlet

Administration application
and 216

DataAccessManager and
244–247

refactoring 243–244

testing with Cactus 189–192

agile methodologies 68

Algol 68

alternate path of execution *See*
failures

Ant

book 93

build process management
with 92

build system choice with 315

build.properties file 94, 317

buildfiles 91, 93–94, 98,
101–102

cactification 266

cactifywar task 266–271, 329

cactus task 333

cactus testing with 265–274

Cactus/Ant integration
module 265–274, 329

configuration of 91

database integration testing
and 261

ear task 317

fileset element 100

installing 91–92, 98

Java application building
tool 91–92

javac task 94–96

JDBC query invoking with 91

junit jar 98

junit task 96–98, 100–101,
272, 322, 333

junitreport task 98–100

parallel task 321

project XML tag and 93

properties of 94

property elements 93

property task 94

retaining test results with 27

SQL task 266

targets 93, 101–102

taskdef element 273

tasks 93

WAR task 266

web site 91

AOP (Aspect-Oriented
Programming) 173

Apache test server 124

APIs (Application Programming
Interface)

contract defined 6

methods 81

testing of public 72, 78

verifying behavior of 171

application.xml file 315, 317–
318, 331

architecture patterns 41

Aspect Oriented Programming
(AOP) 173

AspectJ jar 177

aspectjrt.jar 270

assertEquals method 14, 31,
131, 134, 153, 219, 250

AssertionException 256

AssertionFailedError 159

assertSame 54

Avalon framework 149

B

BaseTestRunner class 21

BasicDynaClass class 219–220

- batchtest element 272
 - beanutils.RowSetDynaClass 247
 - Beck, Kent 5, 40, 70, 164, 166, 240
 - beginIsAuthenticatedNoSession method 181
 - best practices
 - business logics and mocks 144
 - Cactus tests, location of 176
 - code improvement with testing 60, 62
 - continuous regression testing 86
 - exception test readability 62
 - failure explanation in assert calls 51
 - packaging and directory locations of test classes 64
 - refactor long setups when using mock objects 254
 - refactor test setups and teardowns 299
 - refactoring and agile methodologies 68
 - test method naming 50
 - throw an exception for methods that aren't implemented 204
 - unit test one object at a time 48
 - unit tests and testMethods 53
 - use TDD to implement The Simplest Thing That Could Possibly Work 197
 - verification of test failure 192
 - what to test 56, 145
 - black box testing 78, 81
 - body tag 228
 - BodyContent class 230
 - BodyTagSupport class 229
 - bottlenecks, finding with profilers 74
 - branches, conditional 78
 - Brunner, John 89
 - build cactus.xml buildfile 329
 - build.properties file 111, 317
 - buildfiles (Ant) 91, 93–94, 97–98, 101–102
 - Bureau of Extreme Programming 45
 - business logic unit tests 144, 280
 - ByteArrayISO8859Writer class (Jetty) 128
- C**
-
- C2 Wiki 163
 - cactification 192, 267, 270–271, 331
 - Cactus
 - Ant integration 266–267, 328
 - DbUnit integration 271–273
 - defined 173, 267
 - directory structure for tests 329
 - EJB unit tests 328
 - FileRedirector 212
 - front ends 179
 - in-container testing with 166, 173–183
 - integration with Jetty 175–178
 - jars 177
 - JSP testing 217–218, 238, 331, 333
 - mock objects compared to 213, 216, 237, 278
 - running tests 174–175
 - setInitParameter method 211
 - task 273
 - taskdef 270
 - test runners 174–178
 - testing filters 208–213
 - testing life cycles 179–180, 230–233
 - testing taglibs 224–233, 237
 - testing under Jetty with Eclipse 177
 - testing under Maven 196
 - web sites 166, 267, 274
 - when to use 213–214, 216, 237, 278
 - XML and 329–331
 - cactus.jar 270
 - Cactus/Ant integration module 266–267
 - cactus-ant integration jar 267
 - Calculator class 6–7
 - CalculatorTest program 8–10
 - callView method 189, 198, 200–201, 242
 - Carroll, Lewis 282
 - Centipede 315
 - class factory refactoring 156
 - classloaders 10, 12
 - classpath 94, 97–98, 177, 270, 272
 - classpath element 96
 - clean build 92
 - CLEAN INSERT strategy 264
 - clean target 101
 - clearCache method 307
 - close calls 258
 - close method 159
 - Clover 79–80
 - coarse-grained testing 121
 - code issues 146
 - codesmell 62
 - collaboration tags, testing of 233
 - collecting parameters 25–27
 - Common Public License 5
 - CommonMockStatement 251
 - CommonPetstoreTestCase 305–306, 309
 - Commons-HttpClient jar 177
 - Commons-Logging jar 177
 - compile targets 95–96, 268
 - compile.cactustest target 269
 - compile.test target 319
 - condition task 321
 - conditional branches 78
 - Connection object 248–249, 258
 - ConnectionFactory class 156–157
 - Container Managed Persistence (CMP) 282–283
 - container-related code testing 167
 - continuous integration 179, 279–280
 - controller component 40–45
 - Controller interface 41–42
 - controller object 50
 - Controller project 116
 - countTestCases method 30
 - Craig, Philip 140
 - create method 307
 - createCommandResult method 201
 - createdb target 267–268
 - createOrder method 283, 286–287, 290, 293, 297, 303, 314

createOrderHelper
 method 290, 293
 cron job 99
 Cunningham, Ward 75
 custom taglibs 216
 customer
 always right 62

D

data access unit tests
 types of 241
 data.sql file 266
 data.xml 263–264, 270
 DataAccessManager class 262
 DataAccessManager
 interface 243, 247
 database
 access layer interface
 implementation 243
 access unit tests 241, 278
 connection pooling 241
 connectivity 29, 258, 260,
 263–264
 constraints 260
 features 242, 260
 in-memory 278
 integration testing 260–264,
 279–280
 queries 261
 referential integrity 260
 schema 266–267
 triggers 260
 unit-testing 240–242, 260–274
 DatabaseTestSetup 275–276
 DataSource
 implementation 262
 DbUnit
 adding jar 273
 Cactus integration 272
 database data presetting
 with 263
 database testing and 242,
 260–261
 web site 260
 dbunit.jar 270
 DefaultAccountManager
 class 147–149
 DefaultController 43, 45, 48,
 79, 91
 DefaultJMSUtil class 294

DefaultOrderUtil class 294
 definitions
 acceptance tests 6
 API contract 6
 component 167
 container 167
 domain object 47
 expectation 159
 fixture 29
 framework 5
 integration tests 6
 mock object 141
 refactor 52
 regression tests 90
 stub 121
 Test-Driven Development 81
 unit of work 5
 unit test 6
 dependency in software
 development 120
 deployment
 automatic 319
 descriptors 173, 238, 313,
 318, 331
 ears 333
 targets 319
 Design by Contract website 6
 design patterns
 collecting parameters 27
 command pattern 25
 composite pattern 25
 Factory 134
 Interfaces 164
 Inversion of Control 42, 149,
 155, 164
 MVC Model 2 199
 observer 28
 development cycle 82
 diagnostic tests 55
 directory structure 63, 105, 111,
 315, 329
 dist target 101
 doAfterBody method 229, 231
 document root 125–126
 doEndTag method 226, 228
 doGet method 201–204, 206,
 242, 246
 domain object 47–48
 doStartTag method 226–228
 doubles 7
 duplication eliminating 83

dyna beans 218–220
 DynaBean class 199
 DynaBeans 219, 225–226
 dynamic proxies 167
 Dynamic Proxy 204, 233, 238
 DynaMock
 EasyMock compared to 204
 testing session beans with 284
 writing mocks with 246
 DynaPropertiesTag class 225

E

ear files 315–317, 319–321, 329,
 331, 333
 EasyMock 167–170, 204
 Eclipse
 adding jars 177
 Cactus testing under Jetty
 with 177
 plug-ins 175, 234
 projects 113, 115, 175–177
 Quick Fixes 191
 test results retention with 27
 TestDefaultController, run-
 ning with 114
 web site 112
 EJB
 defining sample
 applications 282
 façade strategies 283–284
 home caching issue 306
 limitations of JUnit with 314
 local interfaces 328, 334
 Redirectors 328
 remote interface testing 314
 remote interfaces 314
 unit test writing with
 Cactus 328
 unit testing with Cactus 329
 ejbCreate method 311–312
 EJBException class 293
 ejb-jar.xml file 315
 ejb-local-ref element 330
 Electric XML 271
 eliminating duplication 83
 Emacs 112
 embedded servers 124
 endCallView method 221
 entity beans 282, 310
 EntityBean class 310

error conditions 56, 152
 error handling 26, 56, 152
 errorproperty attributes 322
 ErrorResponse class 44, 79
 Example TimedTest class 75
 exceptions 42, 55–56, 60
 execute method 248–250, 257–258, 261
 executeCommand method 189, 198, 201, 242
 executeQuery method 251
 exml.jar 271
 expectAndReturn method 293
 expectAndThrow method 307
 expectations 159, 163, 256
 Extreme Programming 5, 197

F

fail statements 61
 failureproperty attributes 322
 failures 51, 56–57, 132–133, 258
 field getters/setters and absolute classes 311
 fileset element 101
 filesystem 126
 FilterChain 209–210, 212
 FilterConfig 209, 212
 filters 188, 208, 314
 FilterTestCase 182
 findAccountForUser method 142–144, 147, 149
 findAncestorWithClass method 233
 fit framework
 web site 75
 Fixtures 50
 controller object created by
 default 58
 defined 29
 de-initializing with
 tearDown 32
 long and complex 254
 test case sharing of 53–54
 TestSetup 129, 134, 145, 275–276, 297–298
 fork attribute 97
 formatter element 272
 Fowler, Martin 4
 framework 4–5
 Freeman, Steve 140

functional testing 71–72, 76, 171, 261, 279, 283

G

Galileo 120
 Gamma, Erich 112
 Gang of Four 5
 Generate Getters and Setters feature 311
 getBalance method 82
 getColumnCount method 254–255
 getCommand method 189–191, 197–198, 242–243
 getConnection method 249, 258
 getContent method (WebClient) 122
 getHandler method 44, 47, 60, 79
 getInputStream method 136–137, 153
 getMetaData method 254–255
 getName method 55
 getOrder method 287
 getOrderHome method 287, 306
 getOrderId method 292–293
 getParameter method 206
 getParent method 233
 getRequest method 50
 getRequestDispatcher method 220
 Giraudoux, Jean 140
 green-bar tests 21

H

haltonerror attribute 97, 272
 haltonfailure attribute 96–97, 272
 Handler class (Jetty) 127–128
 Hashtable 144
 Heisenberg Uncertainty Principle 155
 Hollywood Principle 42
 hsqldb.jar 268
 HTML DOM 222

HTTP

clients 40
 connection mock objects 150
 connections 122–123, 150–151, 181
 cookies 181–182
 elements 40
 headers 40, 181–182, 222
 parameters 40, 216, 225
 protocols 157
 HTTP requests
 AdminServlet requirements 189
 doGet method entry point for 242
 functional unit tests using 73
 HTTPContext processing of 126
 HTTP-related parameters in 181
 interception by the security filter 216
 receipt by application 188
 using Cactus to add SQL command to 210
 web controller acceptance 40
 HTTP response 128, 182, 210, 212, 222, 225, 230
 HTTP sessions 40, 181, 225
 httpclient.jar 270
 HttpConnectionFactory class 157
 HttpContext 126
 HttpContext class 125–126, 129, 132–133
 HttpRequest class 128–129
 HttpResponse class 127, 133
 HttpServer class 125, 129–130, 132
 HttpServer class (Jetty) 125
 HttpServlet class 166, 168, 242, 245
 HttpServletRequest class 166, 168–170, 181, 205
 HttpServletResponse class 181, 220
 HttpSession class 166, 168–170, 181
 HttpSocketListener class (Jetty) 129

HttpUnit class 167, 217,
221, 225
 HttpURLConnection class 123,
133–134, 136–138, 153, 157
 HttpURLConnection
 interface 123
 Hypersonic SQL 260, 278,
315, 324

I

IDE (integrated development
 environments) 56, 112
 IEEE 5
 IllegalStateException 298
 incomplete mock object
 test 251
 in-container testing 166, 173,
178
 InitialContext class 285, 297–
298, 314
 inner classes 48
 InputStream class 159
 integrated development envi-
 ronments (IDE) 56, 112
 integration testing 6, 71–72,
133–134, 283
 integration unit testing
 comparison with logic and
 functional unit
 testing 76, 172
 database testing with 242,
 260–264, 279–280
 defined 76, 172–173
 EJBs and 313
 errors in 323–325
 execution time 179
 J2EE testing with 166
 mock objects approach com-
 pared to 179–180
 with Cactus and JUnit 334
 interactions between objects 81
 introspect method 253
 invalid URLs 132–133
 Inversion of Control
 (IOC) 148–149
 isAuthenticated method 166,
168–169, 173
 isolation testing 134, 140, 150,
164, 217
 It Works! return 127–128, 130

J

J2EE
 component unit testing
 166–167
 containers 260, 315
 integration issues, costs
 of 279
 jar proliferation 109
 Java Compiler (javac) 94–96
 Java IDE's 112
 Java Messaging Service
 (JMS) 283–284
 Java Naming and Directory
 Interface (JNDI) 284–285,
 297–307, 314
 Java Server Pages (JSP) 188–
189, 216–233
 Java Virtual Machine (JVM) 96,
320
 JAVA_HOME 92
 JavaBean 56
 javac (Java Compiler) 94–96
 Javadocs 14, 23, 30, 92, 102
 JBoss
 development 320
 Hypersonic SQL and 260,
 315, 324
 installing 317
 JNDI and 326
 version 3.2.1 273
 website 317
 jboss.xml file 326
 jboss3x element 273
 jbossresult.txt file 273
 JDBC 91, 120, 141, 145–147, 284
 JdbcDataAccessManager
 class 243–244, 247, 249–
 250, 261–262
 JEdit 112
 Jeffries, Ron 148
 Jester 80
 Jetty
 benefits of 124–125, 138, 175
 Cactus testing under, with
 Eclipse 177
 embedded server used as 125
 handler that returns 127
 Handlers 127–128
 JettySample class 125
 modularity 125
 NotFoundHandler 132
 opening a URL 125
 pros 125
 setting up stubs with 124–125
 starting and stopping 129
 starting from code 125–127
 website 124
 Jetty classes
 AbstractHttpHandler 127–
 129, 133
 ByteArrayISO8859Writer 128
 Handler 128–129
 HttpContext 125, 129
 HttpServer 125
 HttpServer class 129
 JettySample 125
 SocketListener 125, 129
 Jetty methods
 addHandler 128
 handle 128
 setContextPath 126
 setResourceBase 126
 setUpandtearDown 127
 JMeter 73
 JMS (Java Messaging
 Service) 283–284
 JMSException class 293
 JMSUtil class 287, 294–295, 303
 JMSUtil interface 295, 297
 JNDI (Java Naming and Direc-
 tory Interface) 284–285,
 297–307, 314
 JNDI API 188
 jndi.properties file 297
 JNDINames class 326
 JNDITestSetup 298
 Joyce, James 18
 JspRedirector class 224
 JSPs (JavaServer Pages) 216
 JspTagLifecycle class 228, 230
 JspTestCase class 182, 224–225,
 227, 230
 JSTL 219, 226
 JUnit
 Assert interface 30
 core classes 19–20
 core members 19
 design goals 15, 24, 30
 FAQ 56
 features 13
 IDEs and 112

- JUnit (*continued*)
 life cycle 37–38
 motto 20
 overview 5
 Test interface 23
- JUnit classes
 BaseTestRunner class 20, 22
 TestCase class 13, 18, 25, 28, 31
 TestFailure class 25
 TestListener interface 27
 TestResults class 25
 TestRunner class 18, 20, 180
 TestSuite class 18, 21–23, 131
- JUnit methods
 assertEquals method 13–14, 30, 126
- JUnit TestClass constructors
 version 3.8.1 and later 15
- junit.jar file 98
- JUnitEE website 314
- JUnitPerf website 74
- JUnitReport 99
- JVM (Java Virtual Machine) 32, 96, 320
-
- K**
-
- Kawasaki, Guy 66
-
- L**
-
- Log interface 147
- Log object 146
- Log4j 109
- LoggerFactory class 146
- logging.jar 270
- logic unit tests 76, 172, 241
- lookup method 287
-
- M**
-
- Mackinnon, Tim 140
- Marick, Brian 216
- matchAndReturn method 293, 307
- Maven
 artifacts 109
 cactification 192
 compared to Ant and Eclipse 90, 112
 configuring 103–105
 dependency handling 108–109
 directory structure 193, 212
 goal seeking 102, 109
 handling dependent jars 109
 HTML reports 196
 ID element 105
 IDEs and 112
 installing 103
 JUnit test with 109
 JUnit testing with 109
 maven-linkcheck-plugin 106
 PATH 103
 plugins 102–103, 105–106, 109, 192, 196
 portability 103
 project configuration files 222
 project description 104
 Project Object Module (POM) 104–108
 project.xml 108–109, 196
 reports element 108
 repositories 109
 running Cactus tests 192–193, 212–213
 url element 105
 version element 105
 website 102
 website generation with 105–108
 welcome page 106–107
 workflow 109
- maven site 109
- MAVEN_HOME 103
- maven-changelog-report 106
- MavenLJUnit test with 109
- mergewebxml attribute 330
- Message Driven Beans (MDB) 282–283, 307–310
- metadata 173, 218, 238, 313
- Method Factory refactoring 155
- Mock DataAccessManager 246
- mock objects
 as probes 159
 as Trojan horses 159
 benefits of 140
 best practices 254
 Cactus compared to 213, 216, 237, 278
 defined 141
 entity beans, testing with 310–312
 finding methods to mock 250
 HTTP connection and 150–159
 in-container testing with 166
 indirect calls, discovery of 253
 JNDI implementation strategy 297, 303, 309
 making generic 144
 making mocks generic 144
 message-driven bean testing 308
 mocking at a different level 254
 practical example 150
 pros and cons 144, 170–171
 real objects compared to 163
 servlet testing with 167–170
 session beans 284–285
 standard JDK APIs 163
 stubs compared to 120, 141, 144
 web site 163
 when to use 121, 138, 163, 213–214
 white box tests and 78, 81
- MockAccountManager
 class 143–144
- MockConfiguration class 149
- MockConnectionFactory
 class 157, 160
- MockHttpURLConnection
 class 153
- MockInputStream class 159
- MockLog class 149
- MockMaker 234, 237–238
- MockMultiRowResultSet
 class 252
- MockObject JDBC package 247
- MockObjects
 framework SQL package 280
 jar 112
 project web site 163
- MockSingleRowResultSet
 class 252
- MockURL class 150, 153
- monumental
 methodologies 197
- Murphy's Law 56

MVC Model 2 199

N

name property 42
Newton 25

O

objective standards 77
openConnection method 134
optional.jar file 98
Oracle 278
OrderEJB class 310–312
OrderFactory class 295
OrderProcessMDB 314
OrderProcessorMDB class 286
OrderUtil class 286, 303,
306–307
OrderUtil interfaces 294
Orion 171, 273

P

PageContext class 224, 234, 238
Pascal 68
performance testing 74
Petstore application 314–315
Petstore OrderEJB class 314
PetstoreEJB class 285, 289–291,
295, 303, 314
play-testing 4
POJO (Plain Old Java
Object) 212, 283
POM (project object
model) 104–108
pre-test state 50
printsummary attribute 97, 272
process method 42, 49
processRequest method 42, 44,
51, 57, 79
production environments, draw-
backs in testing 124
profilers 73
project directory structure 265
project object model
(POM) 104–108
project.xml file 104
PropertyResourceBundle
class 147
ProtocolException 136

proxy redirector 180
pushBody method 230

Q

QA teams 75

R

read-only data, factoring 275
red-bar tests 21, 58
redirect 302 response code 73
refactoring 244
best practice 60
class factory, used for 155–159
courage for 68
defined 52
easy method technique
for 152–155
extract hierarchy 62
extract method 62
making code unit-testable
with 69, 283
Method Factory 155
mock objects used as tech-
nique for 146–149
natural solution 245
PetstoreEJB 294
renaming 294
sample 58–59
setDataAccessManager 245
suite of unit tests, benefits
for 140
TDD two-step 83
referential integrity 260
reflection and introspection 10
regression testing 86, 90–91
report target 100
Request 209, 212
Request interface 41, 58
RequestDispatcher 208
RequestHandler class 42, 47,
49–50
RequestHandler interface 41
Resin 273
Response 209, 212
response code (302) redirect 73
Response interface 41, 44
results.jsp 218
ResultSet interface 248, 251
RowSetDynaClass 247

run method 30
RuntimeException class 44, 58,
60, 314

S

Sample servlet 166
Sampling project buildfiles
93, 96
scriptlets 189
Security error page 211
security filter 216
SecurityFilter class 208
SecurityFilter Filter 208
SELECT queries 216
SELECT statement 206, 208,
250
sendToJMSQueue method
286, 303
sendToJMSQueueHelper
method 290
sequence diagrams 33
Service Level Agreement
(SLA) 188
Servlet class 180
ServletConfig 181
ServletRedirector 179, 181
ServletRedirector class 180
ServletRequest 208
ServletResponse 208
servlets
API 188
containers 125, 189, 223, 273
Pet Store sample 283
remote web resources as 122
sample method for unit
testing 166
server side code that calls
EJBs 314
testing a method 166
unit testing 188
writing tests for using
Cactus 189
ServletTestCase class
Cactus and 180, 182, 217
extending with
TestAdminServlet 190,
200
presetting database data
with 261–262
ServletUnit 167

SessionBean class 290, 295
 setConnection method 250
 setContextPath method 125
 setDataAccessManager
 method 245–246
 setDoInput method 136
 setExpectedCloseCalls
 method 256
 setExpectedQueryString 256
 setInitialContextFactoryBuilder
 method 297–298
 setJMSUtil method 293, 297
 setOrderFactory method 297
 setOrderUtil method 293
 setPageContext method 224,
 227
 setParent method 227, 233
 setResourceBase method 125
 setUp methods 126–128, 130
 setURLStreamHandlerFactory
 method 134
 simulations, using tests to
 create 57, 133, 150
 SLA (Service Level
 Agreement) 188
 SocketListener class 125, 133
 software accidents 72
 software, testing 71
 sortHTMLTable tag 228
 SortHtmlTableTag class 228
 SQL
 commands 210
 queries 189, 208–211
 statements 208
 standard output 8
 start target 319
 stateless session beans 283
 Statement class 248
 static modifier 294
 stepwise refinement 68
 stop target 319
 stored procedures 242
 strategies/techniques for testing
 adjusting build failure
 criteria 79
 Cactus in-container 167,
 213–214
 choosing an appropriate
 260–264
 creating a component
 class 249

 creating a wrapper class 249
 database access layer 244
 exploring alternative 58
 façade 283–284
 factory class 285, 293,
 303, 312
 factory method 285, 289, 303,
 309, 312
 mock 297
 mock JNDI 285, 297–307,
 309, 312
 mock object 140–141, 151,
 213, 313
 stubbing 120–121, 124, 138
 stress/load testing 71–75, 261
 Strong, William 188
 Struts 40, 77, 199, 219
 StubURLConnection
 class 136
 StubURLStreamHandler
 class 134
 stubs
 choosing 124–125
 compared to mock
 objects 120, 138,
 141, 144
 creating sophisticated tests
 with 64
 how to use 121
 overview 120–121
 pros and cons 121
 replacing real code with 122
 sample use of 121–124
 stubbing connections
 134–138
 stubbing web server's
 resources 126–134
 when to use 121, 138
 white box testing and 81
 StubStreamHandlerFactory
 class 134
 subclasses, replacing with
 collaborators 156
 suite method 34
 Swing applications 283

T

tag life cycle testing
 method 227–228, 230–233
 taglibs (tag libraries) 188, 216

Body tag container life
 cycle 230
 EJB calling with 314
 unit testing with Cactus
 224–233
 unit testing with mock
 objects 233–237
 TagSupport class 225
 TDD (Test Driven Develop-
 ment)
 automatic documentation 69
 avoiding interface overdesign
 with 44
 best practice principle
 using 60
 core tenets 83
 defined 81
 design process 82
 effective adjustment to con-
 ventional development
 cycle 82
 elimination of refactoring
 code in unit testing
 enabling with 164
 initial step of 89
 reference book about 164
 Test First 189
 writing tests before writing
 code 189
 teams, working with 120, 140
 tearDown method 126–130
 terrific trio 18–19
 Test class, writing the 131
 Test Driven Development
 (TDD) *See* TDD (Test
 Driven Development)
 test runners 11–12, 21, 26–27,
 277
 TestableJdbcDataAccessMan-
 ager class 249
 TestableOrderEJB 311
 TestablePetstoreEJB class 291–
 292
 TestableWebClient class 154
 TestAccount 82
 TestAccount class 82
 TestAccountService 144
 TestAccountService class 144
 testAddHandler method 52
 TestAdminServlet 190, 200

- TestAdminServlet class 192, 196, 199, 219
- TestAdminServletDynaMock 246
- TestAdminServletMO class 205
- TestAll class 24
- TestCalculator class 13, 34
- TestCalculator program 18, 22, 25
- TestCalculator walk-through 33
- TestCallView 201
- TestCase class 61
- TestCases 131
 - collecting results from 25
 - defined 20
 - extending from JUnit 13
 - extracting from JUnit 13
 - fixtures and 29
 - for PetStoreEJB 295
 - grouping for
 - performance 277
 - how to group 22
 - objects, adding to Test Suites 18, 25
 - requirements for 29
 - running several at once 18
 - test interface and 23
 - TestSampleServletIntegration 180
 - typical 29
 - working with 28–32
 - writing the 299–303
- TestClass constructors version 3.8.1 16, 22
- testCreateOrderOk method 299
- testCreateOrderThrowsCreateException method 306
- testCreateOrderThrowsException method 306
- testCreateThrowsOrderException method 299
- TestCustomerAll 277
- TestCustomerAll class 277
- TestDefaultController class 46, 49–57, 101, 114
- Test-Driven Development (TDD)
 - failures and 192
- TestDynaPropertiesMO 234
- TestDynaPropertiesTag class 227
- TestExceptionHandler class 57
- testExecuteCloseConnectionOnException 258
- testExecuteOk method 256
- TestFailure class 25
- testGetBalanceOk method 82
- testGetContentOk class 153
- testGetContentOk method 154, 158, 161–162
- TestGetContentOkHandler class 127, 129
- testGetOrderHomeFromCache method 306
- test-infected programs 4, 45, 70
- testing
 - as first customer 62
 - challenges 120
 - components 167
 - database access 241
 - database logic 241
 - effectiveness 102
 - fine-grained 164, 173
 - in isolation 129, 150, 164
 - inside the container 167
 - live database 260
 - outside the container 167
 - performance turning 275
 - prime objective 214
 - strategies 121
- TestJdbcDataAccessManagerIC 261
- TestJdbcDataAccessManagerIC2 276
- TestJdbcDataAccessManagerMO1 class 250
- TestJdbcDataAccessManagerMO3 255
- TestOrderEJB 311, 328
- TestOrderProcessorMDB 309
- TestOrderUtil class 305
- TestPetStoreEJB 299
- TestPetstoreEJB class 292, 295, 325
- TestPetstoreEJB.java file 315
- testProcessRequest method 52, 54
- TestRequest class 57–58
 - constructor 59
- TestRequestHandler 57
- TestResponse class 49, 51, 54
- TestResult 25, 27
- TestRunner class 18, 20, 180
- TestRunners
 - Cactus and 174, 180
 - combining with a TestSuite 24
 - defined 19
 - defining 21
 - failures and 25–26
 - launching tests with 20–21
 - read-only database data and 277
 - running graphical and text 11
 - selecting 20
- tests
 - comparison of 172
 - composit and command patterns and 25
 - coverage 67, 79–80
 - fine-grained 164
 - folders 64
 - simulation cases 293
 - targets 93, 96, 101, 273, 321
 - types of 71–77, 172
- TestSampleServlet 168
- TestSampleServletIntegration class 173, 180
- TestSetup 128–131, 134, 145, 275–276, 297–299
- TestSetup class 129
- TestSortHtmlTableTag 230
- testSubtract method 23
- TestSuite class 18, 21–23, 131
- TestSuites
 - composing tests with 21–25
 - customizing 23–25
 - defined 19–20
 - TestCustomerAll 277
 - TestJdbcDataAccessManager 276
 - TestSetup and 128
 - TestWebClient1 131
- TestWebClient class 134, 137, 150, 158
- TestWebClient test case 122
- TestWebClient1 class 131
- TestWebClientSetup1 class 129, 131
- TestWebClientSkeleton class 126
- TestWebSetup1 class 132
- TextPad 112
- Tomcat 171, 192, 273

too simple to break 7, 228
tools for analysis and
 reporting 79
transfer method 144
transparency 123
trap doors 148, 151
trial and error 250
triggers 242
try/catch block 10, 61
try/finally block 258

U

UML 33
unit of work 5, 7
unit testing
 application life cycle 76
 as first-class users 146
 co-dependent tests, problems
 with 32
 core tenet of 7
 defined 6
 description of a typical 6
 drawbacks of 166
 flavors 76
 functional 76–77
 integration 76
 interaction 76
 logic 76
 need for 66–69

 old school 146
 running 20
 UNIX 91, 103
 updateAccount method 144
 URL interface unavailability 153
 URLConnection interface 123
 URLStreamHandler class 134
 URLStreamHandlerFactory
 class 134

V

verify method 256
version attribute 330

W

War plugin 222
war target 268
wars, creating 268
web application 222
web servers 124
WebClass methods
 getContent 150
WebClient
 refactored using
 ConnectionFactory 156
 refactored using
 MockConnectionFactory
 158

 testing for failure
 conditions 132–133
 verification for failure
 conditions 126
WebClient methods
 getContent 122, 150, 158
 getHttpConnection 154
 setHttpURLConnection 154
WebLogic 273
WebRequest class 224
WebResponse class 219, 224,
 232
white box testing 76, 78, 81
Windows 91, 103
Wirth, Nikolas 68
writing a failing test 83

X

XDoclet 282, 318
XP2000 140
xPetstore application
 web site 282
XSL stylesheet 99
xUnit 5

Y

YAGNI 68

JUnit IN ACTION

Vincent Massol with Ted Husted



Developers in the know are switching to a new testing strategy—unit testing—in which coding is interleaved with testing. This powerful approach results in better-designed software with fewer defects and faster delivery cycles. Unit testing is reputed to give developers a kind of “high”—whenever they take a new programming step, their confidence is boosted by the knowledge that every previous step has been confirmed to be correct.

JUnit in Action will get you coding the new way in a hurry. As inevitable errors are continually introduced into your code, you'll want to spot them as quickly as they arise. You can do this using unit tests, and using them often. Rich in real-life examples, this book is a discussion of practical testing techniques by a recognized expert. It shows you how to write automated tests, the advantages of testing a code segment in isolation from the rest of your code, and how to decide when an integration test is needed. It provides a valuable—and unique—discussion of how to test complete J2EE applications.

What's Inside

- Testing in isolation with mock objects
- In-container testing with Cactus
- Automated builds with Ant and Maven
- Testing from within Eclipse
- Unit testing
 - ◆ Java apps
 - ◆ Servlets
 - ◆ JSP
 - ◆ Taglibs
 - ◆ Filters
 - ◆ EJB
 - ◆ DB apps

Vincent Massol is the creator of the Jakarta Cactus testing framework and an active member of the Maven and MockObjects development teams. He is CTO of Pivolis, a specialist in agile offshore software development. Vince lives in the City of Light—Paris, France.

The *keep the bar green* frog logo is a trademark of Object Mentor, Inc. in the United States and other countries.

“... captures best practices for effective JUnit and in particular J2EE testing. Don't unit test your J2EE applications without it!”

—Erich Gamma, IBM OTI Labs
Co-author of JUnit

“Outstanding job... It rocks—a joy to read! I recommend it wholeheartedly.”

—Erik Hatcher, co-author of
Java Development with Ant

“Brings the mass of information out there under one coherent umbrella.”

—J. B. Rainsberger, leader in
the JUnit community, author

“Doesn't shy from tough cases ... Vince really stepped up, rather than side-stepping the real problems people face.”

—Scott Stirling, BEA

www.manning.com/massol



Author responds to reader questions



Ebook edition available

MANNING

\$39.95 US/\$59.95 Canada



ISBN 1-930110-99-5