

## Установка среды: JDK, Ant и JUnit

Выпуск 1, Неделя 4, Итерация 2



**Стив:** Привет, Рон. Дело движется. Мы пытаемся развернуть приложение, чтобы оно прошло приемочные испытания. Радж устраняет небольшие проблемы с сетью, которые мешают нам подключиться к серверу. В любом случае мы должны быть готовы к нашей встрече по поводу следующей итерации к понедельнику будущей недели.

(c) Visual Patterns, Inc.

В этой главе мы получим набор установленных инструментальных средств, которые позволят нам начать разрабатывать, создавать, проверять и развертывать код Java.

Одна из задач этой книги — быть быстрой в прочтении, а не избыточным справочником, полным устаревшей информации. Следовательно, я не собираюсь приводить инструкции по установке для различных инструментальных средств (например, Ant), описанных в этой главе. Вместо этого, я адресую вас к соответствующим Web-сайтам, потому что все затронутые здесь инструментальные средства снабжены вполне достаточными и современными инструкциями по установке.

## Что рассматривается в этой главе

В этой главе мы начинаем установку нашей среды разработки, а именно следующих основных инструментальных средств, обязательных для разработки Java в остальной части этой книги.

- Комплект разработки Java Development Kit (JDK) платформы Java Standard Edition (JSE). Поскольку данная книга о технологиях Java, это программное обеспечение необходимо для работы в первую очередь.
- Ant. Это, фактически, утилита для создания и развертывания приложений на базе Java.
- JUnit. Это просто среда проверки модулей, а также стандартный ныне способ проверки кода Java.
- Заставить все это работать вместе. И наконец, мы соберем все эти три технологии и опробуем их на примере простого модуля.

### Примечание

Полный код примеров, используемых в этой главе, находится внутри файла zip, доступного на Web-сайте книги<sup>1</sup>.

В последующих главах мы добавим к этой среде другие продукты, такие как базы данных, Web-серверы, IDE, библиотеки дескрипторов и др.

## Стандартная платформа и комплект разработки Java (JDK)

Поскольку мы занимаемся разработкой на Java, имеет смысл обзавестись необходимыми для этого инструментальными средствами (например, компилятором). Если на вашей машине установлена устаревшая версия JDK, не подходящая для JUnit и Ant, то вам следует загрузить его последнюю версию с Web-сайта `java.sun.com` и установить на своей машине так, чтобы команды типа `java` могли быть выполнены без указания пути.

После загрузки и установки Java, вы должны быть способны ввести команду `java -version`, чтобы проверить установку и удостовериться в соответствии версии JDK, как показано ниже.

---

<sup>1</sup> [http://www.samsublishing.com/content/images/0672328968/downloads/0672328968\\_AgileJavaDev\\_code.zip](http://www.samsublishing.com/content/images/0672328968/downloads/0672328968_AgileJavaDev_code.zip). — *Примеч. ред.*

```
C:\anil\rapidjava\timex>java -version
java version "1.5.0_06"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_06-b05)
Java HotSpot(TM) Client VM (build 1.5.0_06-b05, mixed mode, sharing)
```

## Структура каталога

Давайте вернемся к структуре каталога, описанной в предыдущей главе. Структура каталога представлена на рис. 4.1. Важно рассмотреть ее снова, прежде чем мы перейдем к обсуждению Ant. Поэтому обсудим здесь некоторые из наиболее важных подкаталогов.

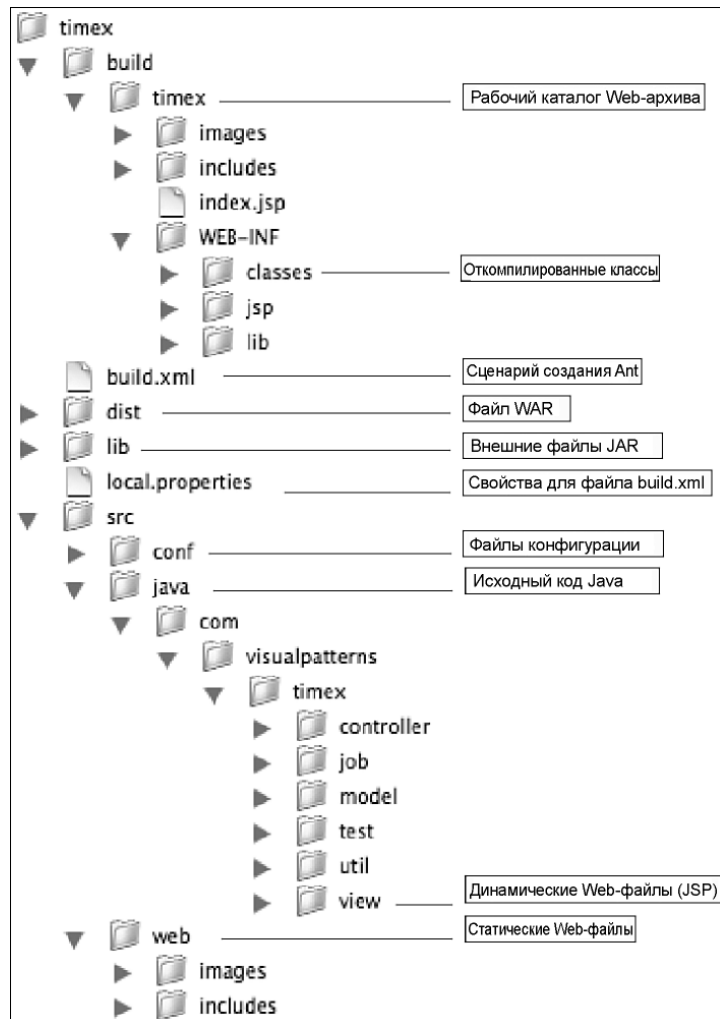


Рис. 4.1. Структура рабочего каталога для приложения Time Expression

- Каталог `src` будет содержать все разработанные нами файлы Java, HTML/Javascript, XML и другие исходные файлы.
- Каталог `build` будет содержать наши результаты построения (например, откомпилированные файлы, копии Web-файлов, библиотечных файлов и т.д.).
- Каталог `lib` будет содержать все внешние файлы JAR, необходимые для запуска нашего приложения.
- Каталог `dist` будет содержать наш Web-архив (файл `.war`) со всеми файлами, связанными с Web, включая откомпилированные файлы `.class`, библиотечные файлы `.jar` и др.

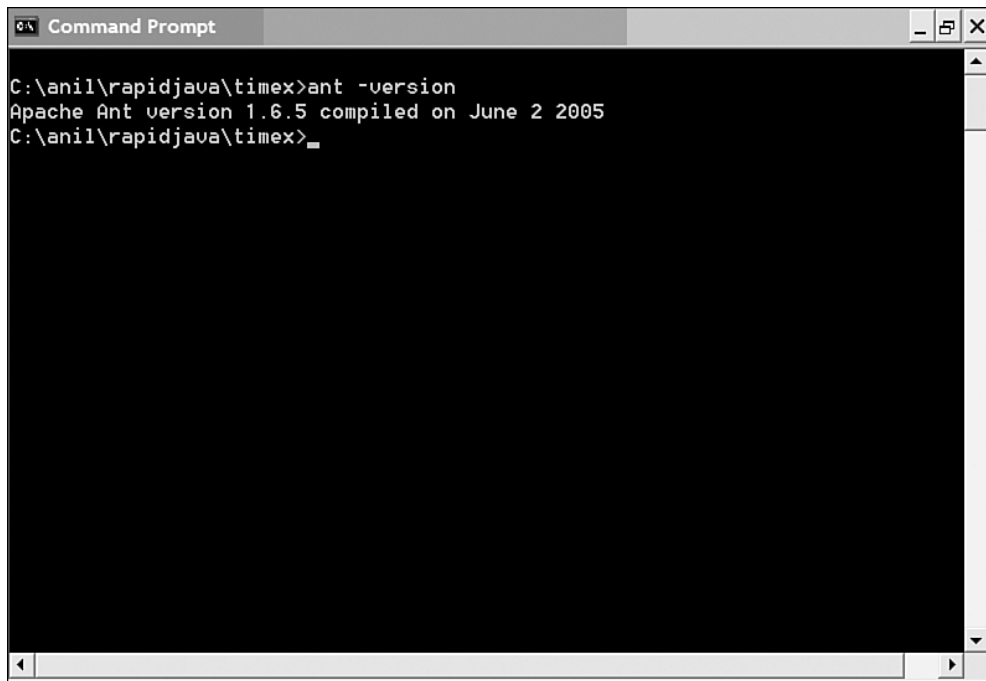
## Ant

Я вовсе не преувеличивал, когда утверждал, что Ant (`ant.apache.org`) является ныне, вероятно, одним из важнейших и наиболее популярных инструментов в мире Java! Следовательно, овладение этим инструментом — залог быстрой разработки Java. Поэтому нет ничего удивительного в том, что я рассматриваю этот инструмент непосредственно после JDK, поскольку считаю Ant наиболее важным инструментом и устанавливаю его сразу после Java.

Теперь вы, вероятно, понимаете важность роли Ant в разработке Java. В этой книге мы будем использовать его весьма интенсивно! Например, мы используем его для создания нашего приложения, развертывания и запуска разных программ Java, создания базы данных, выполнения проверок и т.д. Ant был первоначально разработан Джеймсом Дунканом Давидсоном (James Duncan Davidson) из отдела Open Source Program компании Sun Microsystems. Ant представляет собой независимый от платформы инструмент, который устраняет множество проблем и сложностей, присущих таким инструментальным средствам, как Unix `make`. Вместо того чтобы использовать команды оболочки, специфические для операционной системы, при определении различных задач Ant использует файлы XML. Ant — высоко расширяемый инструмент, главным образом, из-за наличия на рынке огромного количества встроенных и внешних дополнений (с открытым исходным кодом и коммерческих), которые делают его настолько мощным. Кроме того, вы можете легко написать свои собственные специальные расширения.

С учетом того, что Ant сам разработан на Java, он обладает переносимостью и, согласно Web-сайту Ant, был проверен на разных системах, включая Unix, Microsoft Windows, Mac OS X и др. Web-сайт `ant.apache.org` предоставляет вполне достаточную (и современную) информацию о том, как получить и установить Ant на вашей системе; если он еще не установлен, то сейчас самое время сделать это.

Когда Ant будет установлен успешно, вы должны быть способны выполнить команду `ant` без указания полного пути. Иными словами, команда `ant` должна быть указана в вашей переменной окружения `path`, поскольку в остальной части нашей книги мы будем обращаться к ней без полного пути. Например, если вы введете в командной строке `ant -version`, то результат должен быть похож на рис. 4.2.



```

C:\anil\rapidjava\timex>ant -version
Apache Ant version 1.6.5 compiled on June 2 2005
C:\anil\rapidjava\timex>_

```

Рис. 4.2. Проверка установки ant при помощи команды `ant -version`

## Создание простого файла Ant

По умолчанию Ant ожидает, что файл `build.xml` расположен в текущем каталоге, если вы не сопроводите команду `ant` какими-либо аргументами. Давайте вспомним упоминавшийся ранее в этой книге файл `build.xml`, содержащий следующий крошечный сценарий Ant. Он предоставляет целевой объект, который выполняет команду `echo`:

```

<?xml version="1.0"?>

<project name="HelloTest" default="printmessage">
  <target name="printmessage">
    <echo message="Hello, world!"/>
  </target>
</project>

```

Например, если мы сохраним этот код XML в файле по имени `build.xml`, расположим его в том же каталоге, что и Ant, а затем введем команду `ant`, то результат будет выглядеть следующим образом:

```

> ant
Buildfile: build.xml
printmessage:
[echo] Hello, world!
BUILD SUCCESSFUL
Total time: 0 seconds

```

## Создание исчерпывающего файла Ant

Примеры “Hello, world” хороши, но давайте перейдем к созданию исчерпывающего сценария Ant для нашего типового приложения.

### Примечание

Загружаемый код этой книги демонстрирует полный сценарий для нашего типового приложения, включая файл `build.xml` и файл `local.properties`, используемый файлом `build.xml` для загрузки некоторых внешних свойств. Оба эти файла следует поместить в каталог верхнего уровня, `timex/`. Обратите внимание на то, что использование файла `local.properties` демонстрирует удобный способ управления конфигурацией в различных случаях (разработка, проверка, опробование, работа) за счет применения разных файлов свойств.

### Концепция Ant

Прежде чем поэтапно исследовать наш файл `build.xml`, сделаем обзор некоторых фундаментальных концепций Ant.

К ключевым концепциям Ant относятся проект, свойства, целевые объекты, задачи и элементы. *Свойства* (property) — это переменные, которые вы можете устанавливать для сеанса Ant. *Целевые объекты* (target) содержат блоки кода XML, которые запускаются на выполнение в форме задач. *Задачи* (task) — это фактические действия, такие как встроенные задачи `javac`. Задачи в свою очередь могут содержать *элементы* (element), например, `dirset` или `fileset`.

### Поэтапное исследование

Теперь сделаем обзор ключевых объектов нашего файла `build.xml`, однако сначала рассмотрим графическое представление этого файла, представленное на рис. 4.3. Вы можете также вернуться к рис. 4.1, прежде чем начать это исследование, поскольку сценарий построения Ant жестко привязан к данной структуре рабочего каталога.

Первый элемент XML, который должен присутствовать в файле Ant, — это элемент `project`, как показано ниже:

```
<project name="timex" basedir="." default="build">
```

Следующие строки кода, по существу, устанавливают внутренние переменные (свойства) для сценария. Большинство этих свойств связано с различными используемыми входными и выходными каталогами, как показано в приведенном ниже отрывке кода (обратите внимание на то, как мы можем использовать внутренние переменные, заключенные в фигурные скобки и предваренные знаком доллара; например, `${dist.dir}`):

```
<property name="appname" value="timex" />
<property name="lib.dir" value="lib" />
<property name="war.dir" value="build/timex" />
<property name="war.file" value="${dist.dir}/${appname}.war" />
<property name="webinf.dir" value="${war.dir}/WEB-INF" />
```

После установки свойств сценарий устанавливает путь к классу, который используется другими задачами в файле. *Путь к классу* (classpath), по существу, включает два набора файлов: все файлы `.jar` в нашем каталоге `lib/` и файлы классов в каталоге `build/timex/WEB-INF/classes/`, как показано ниже:

```

<path id="master-classpath"
      description="Master CLASSPATH for this script">
  <fileset dir="${lib.dir}">
    <include name="*.jar" />
  </fileset>
  <pathelement location="build/timex/WEB-INF/classes/" />
</path>

```



Рис. 4.3. Иерархическое представление нашего файла Ant build.xml

Следующий в нашем сценарии целевой объект, `init`, гарантирует, что внутри каталога `build/` будут созданы некоторые выходные каталоги, используемые для других задач (для этого используется атрибут `depends` других целевых объектов):

```

<target name="init" description="Setup for build script">
  <mkdir dir="${class.dir}" />
  <mkdir dir="${libs.dir}" />
  <mkdir dir="${jsp.dir}" />
</target>

```

Наши целевые объекты `updateweb`, `updatelib`, `deleteconfig` и `updateconfig` просто копируют или удаляют файлы, связанные с Web и библиотеками в каталоге назначения.

Следующий интересный целевой объект, `compile`, компилирует файлы `.java` каталога `src/java/` в файлы `.class` каталога `build/timex/WEB-INF/classes/`, как показано ниже:

```

<target name="compile"
        description="Compiles .java files to WAR directory">
  <javac srcdir="${src.dir}" destdir="${class.dir}" debug="true"
        failonerror="true" classpathref="master-classpath" />
</target>

```

Наш целевой объект `dist` создает файл WAR и развертывает его по месту, указанному внутренней переменной `${war.file}` (т.е. `dist/timex.war`). Интересный момент, который следует отметить, обсуждая данный целевой объект, — это использование задачи `war` и элемента `fileset` (известного как тип Ant). Задача `war` создает файл `.war`, а тип `fileset` применяется для определения индивидуального файла или группы файлов (используя наборы схем `include` (включить) и `exclude` (исключить)). Примеры задачи `war` и элемента `fileset` приведены ниже:

```
<war destfile="${war.file}" webxml="${src.dir}/conf/web.xml">
  <fileset dir="${war.dir}">
    <include name="**/*.*" />
    <exclude name="**/web.xml" />
    <exclude name="**/test/*.class" />
    <exclude name="**/*mock*.jar" />
  </fileset>
</war>
```

Используются и другие целевые объекты, включая `deploy`, `clean` и `test`. Целевой объект `deploy` копирует файл `.war` в каталог назначения (мы используем его для развертывания в каталог `webapps` сервера Apache Tomcat). Целевой объект `clean` удаляет файлы из каталога назначения. Целевой объект `test` мы используем в главе позже.

### Категории задач Ant

Ниже приведены некоторые из задач, использованных в нашем файле `build.xml`:

- задачи архивации, такие как `war`;
- задачи компиляции (т.е. `javac`);
- файловые задачи, такие как `copy`, `delete`, `move` и др.;
- разнообразные задачи, такие как `echo`;
- задачи свойств по установке внутренних переменных.

Другие, встроенные, задачи относятся к следующим категориям:

- задачи аудита и покрытия (`audit/coverage`);
- задачи развертывания;
- задачи документирования;
- задачи выполнения;
- задачи для почты;
- задачи препроцессора;
- задачи свойств;
- отдаленные задачи.

Как я упомянул ранее, мы используем Ant для создания и развертывания нашего Web-архива (файла `.war`). В следующих главах мы продолжим использовать командную строку для работы с Ant. Однако когда в главе 8, “Феномен Eclipse!”, перейдем к рассмотрению Eclipse, придется использовать Ant внутри Eclipse (рис. 4.4), что существенно упростит редактирование и запуск (того же файла Ant `build.xml`)!



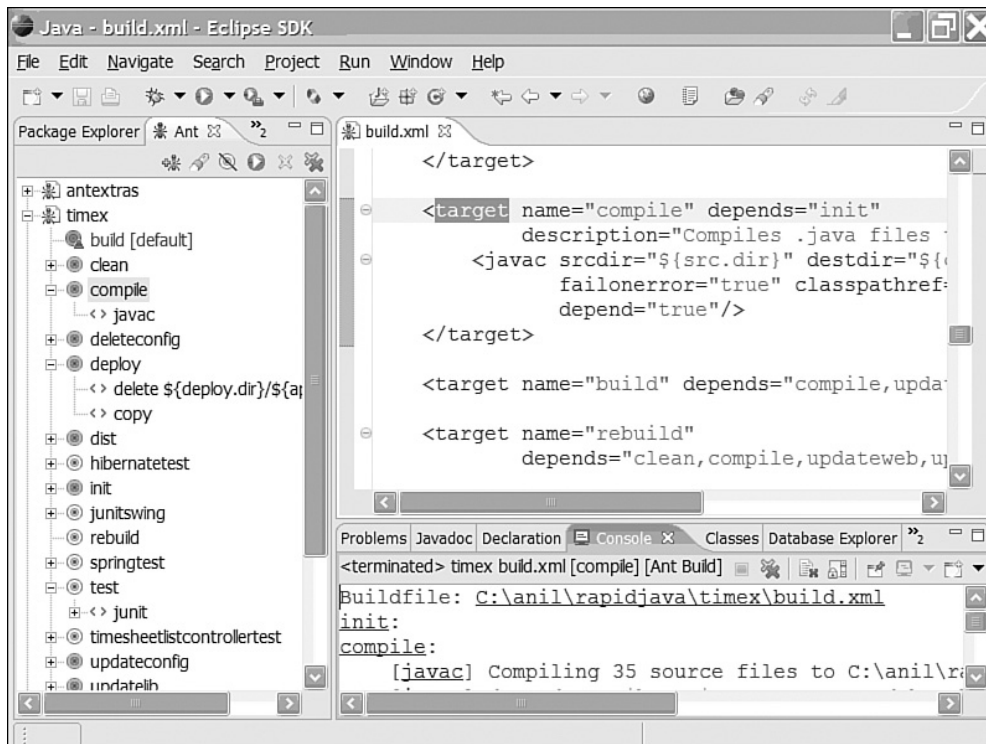


Рис. 4.4. Вид Ant внутри Eclipse

Несколько задач рассмотрим также в главе 10, “Кроме основ”. Напомню, поскольку Ant посвящены целые книги, я, как вы уже догадались, не буду описывать эту технологию особенно глубоко. Идея этой книги заключается в быстром чтении. Если вы желаете исследовать Ant далее, то можете найти вполне достаточно информации в сети и в печати (то же относится ко всем остальным технологиям, рассматриваемым в этой книге).

## JUnit

Среда JUnit, первоначально описанная в книге *Gang of Four, Design Patterns* Эрихом Гамма (Erich Gamma) и Кентом Бекон (являющимся также автором экстремального программирования), представляет собой проверочную среду выполнения Java с открытым исходным кодом, общепринятую для проверки модулей кода Java. Ее можно разгрузить с Web-сайта [junit.org](http://junit.org); этот Web-сайт предоставляет не только инструкцию по установке, но и статьи по проверке модулей и множество советов по разработке предварительных проверок.

*Проектирование методом проверки* (Test-Driven Design, TDD) — это термин, введенный Кентом Бекон для описания подхода, позволяющего улучшить проект, упростить код (меньше операторов `print` и `debug`, а также сценариев проверки) и повысить его

эффективность. Поскольку в этой книге мы будем использовать этот подход, создавая предварительные проверки, JUnit имеет смысл рассмотреть непосредственно после разделов по JDK и Ant.

### Автономные пускатели JUnit

Инструкции по установке JUnit, как обычно, находятся на Web-сайте этого продукта, `junit.org`. *Пускатель* (runner) проверок JUnit (главный класс Java) поставляется двух видов: текстовая версия и графическая. Графическая версия доступна в двух вариантах: на базе Java Swing (рекомендуемая) и устаревшая на базе AWT.

После того как JUnit установлен правильно, вы должны быть способны ввести приведенную ниже команду (в каталоге, где установлен JUnit; например, `C:\junit3.8.1`) и запустить версию Swing пользовательского интерфейса JUnit, показанного на рис. 4.5:

```
java -cp junit.jar junit.swingui.TestRunner
```

### JUnit в SDK Eclipse

В следующих главах мы продолжим использовать автономный пускатель проверок JUnit для работы с JUnit, как описано здесь. Но когда далее в книге я представлю Eclipse, мы перейдем к использованию JUnit внутри Eclipse (как показано на рис. 4.6), что позволяет осуществлять отладку и проверку JUnit значительно удобнее. Но иногда возникает необходимость запуска на сервере *пакетной проверки* (batch test) с использованием задачи Ant `junit` или проверки отдельного класса вне IDE при помощи одного из встроенных пускателей JUnit.

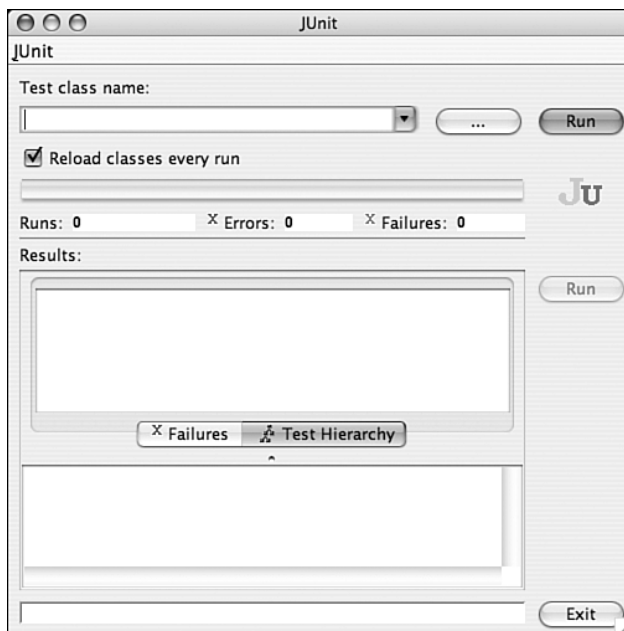


Рис. 4.5. Пускатель проверок JUnit на базе Java Swing

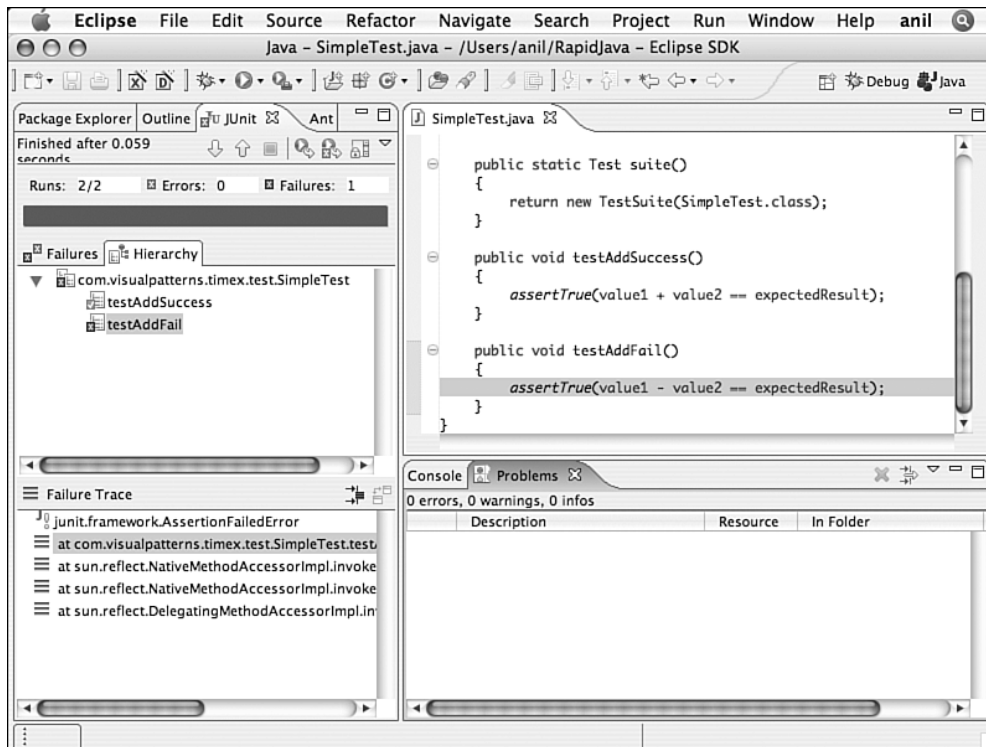


Рис. 4.6. Вид JUnit внутри Eclipse

## Как заставить все это работать вместе

Теперь, когда JDK, Ant и JUnit установлены правильно, можем написать пример проверки JUnit и опробовать ее на практике.

Независимо от используемой разновидности JUnit, можем либо передать ему имя проверяемого класса<sup>2</sup>, либо ввести его в UI пускателя. Например, если хотим записать очень простой случай проверки, например для проверки факта  $2 + 3 = 5$ , должны будем предпринять следующее:

- разработать проверочный класс JUnit, например `SimpleTest.java`;
- запустить класс в JUnit, используя один из его пускателей.

### Файл `SimpleTest.java`

Файлы кода этой книги (доступные на Web-сайте) содержат файл `SimpleTest.java` полностью. Этот код довольно прост. Имеются два испытательных метода: `testAddSuccess` и `testAddFail`, как показано далее:

```
public void testAddSuccess()
{
```

<sup>2</sup> Автор, вероятно, имел в виду командную строку. — *Примеч. ред.*

```

    assertTrue(value1 + value2 == expectedResult);
}

public void testAddFail()
{
    assertTrue(value1 - value2 == expectedResult);
}

```

Метод `testAddSuccess` засвидетельствует успех, а метод `testAddFail` будет свидетельствовать о неудаче (потому что 2 минус 3 не равно 5). Успех или неудача определены методом JUnit `assert`, который в случае негативного результата проверки передает исключение.

## Методы JUnit `assert`

В предыдущем примере был продемонстрирован метод JUnit `assertTrue`; JUnit предоставляет также несколько других разновидностей методов `assert`, приведенных ниже:

- `assertEquals`
- `assertFalse`
- `assertNotNull`
- `assertNotSame`
- `assertNull`
- `assertSame`
- `assertTrue`

## Запуск примера `SimpleTest`

Чтобы опробовать код примера `SimpleTest`, необходимо создать файл `SimpleTest.java`, скомпилировать его, а затем попытаться запустить. Поэтому давайте создадим файл `SimpleTest.java` в нашем каталоге `src/java/com/visualpatterns/timex/test/`. Затем, чтобы скомпилировать исходный код нашего проверочного модуля, просто введем команду **ant** в каталоге `timex/`.

Теперь давайте попробуем запустить наш пример `SimpleTest` (из каталога, уровнем выше `timex/`) при помощи проверочного пускателя JUnit, как показано здесь:

```

C:\anil\rapidjava\timex>java
↳ -cp \junit3.8.1\junit.jar;build\timex\WEB-INF\classes
↳ junit.textui.TestRunner com.visualpatterns.timex.test.SimpleTest

```

Мы должны увидеть нечто подобное тому, что представлено на рис. 4.7.

Мы могли бы также запустить класс `SimpleTest.class` в пускателе JUnit Swing, как показано ниже (результат выполнения этой команды показан на рис. 4.8).

```

java -cp \junit3.8.1\junit.jar;build\timex\WEB-INF\classes
↳ junit.swingui.TestRunner
com.visualpatterns.timex.test.SimpleTest

```

На самом деле стандартное сообщение об ошибке JUnit `junit.framework.AssertionFailedError`, которое вы видите в обоих случаях, — это хороший знак, поскольку наш метод проверки `testAddFail` потерпел неудачу.

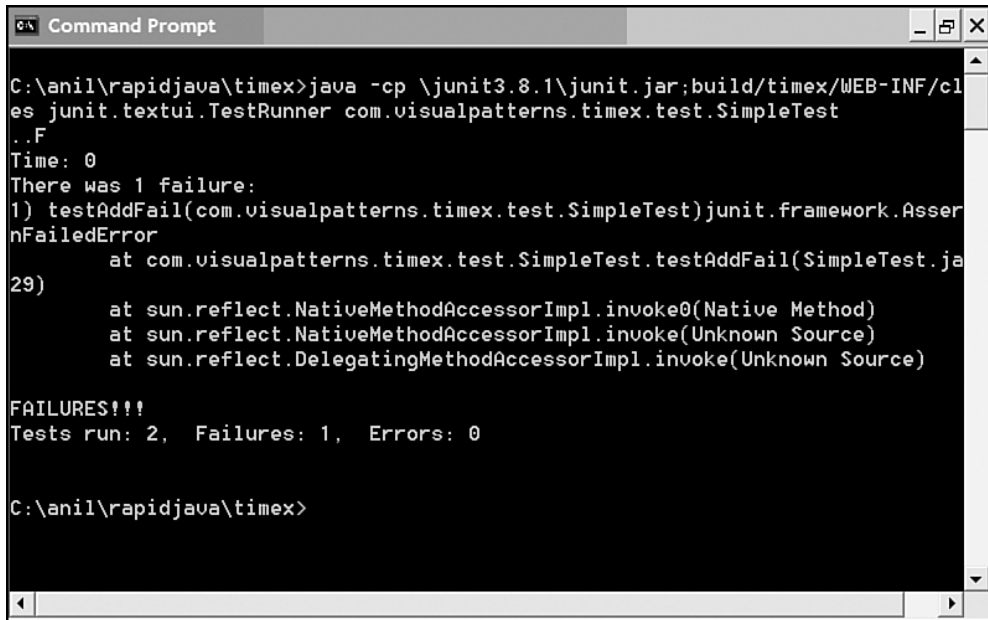


Рис. 4.7. Запуск примера SimpleTest в текстовом пускателе



Рис. 4.8. Запуск приложения SimpleTest в пускателе JUnit на базе Swing

## Запуск проверки JUnit в пакете

Существует еще один способ запуска JUnit, в качестве задачи Ant. Впоследствии мы рассмотрим, как это сделать.

Сначала скопируем файл `junit.jar` из каталога установки JUnit в каталог `<место-ant>/lib`; например, на операционной системе Microsoft Windows введите в командной строке что-нибудь вроде `copy \junit3.8.1\junit.jar\apache-ant-1.6.5\lib`. Это позволит нам использовать задачу Ant `junit`.

Затем необходимо скопировать тот же файл `junit.jar` в наш каталог `timex/lib`; это также поможет в компиляции с использованием Ant.

Теперь вернемся к нашему файлу `build.xml`. Этот файл содержит целевой объект по имени `test`, который использует задачу `junit`, как показано в следующем отрывке:

```
<target name="test" depends="compile">
  <junit printsummary="true" showoutput="yes" filtertrace="false">
    <classpath refid="master-classpath"/>
    <batchtest fork="yes">
      <formatter type="plain"/>
      <fileset dir="${class.dir}">
```

Если ввести команду `ant test`, то результаты пакетного теста будут помещены в файл по имени `TEST-com.visualpatterns.timex.test.SimpleTest.txt`, расположенный в текущем каталоге. Его отрывок приведен ниже:

```
Testsuite: com.visualpatterns.timex.test.SimpleTest
Tests run: 2, Failures: 1, Errors: 0, Time elapsed: 0.021 sec

Testcase: testAddSuccess took 0.004 sec
Testcase: testAddFail took 0.003 sec
FAILED
```

Это просто замечательно, поскольку вы можете применить несколько методов проверки внутри каждого подкласса примера `TestCase`. Кроме того, вы можете включать индивидуальные комплекты внутри других комплектов (без ограничений). Например, можно создать класс по имени `AllTests`, который вызывает комплекты всех других проверочных классов в пакете проверки.

## Разработка с предварительной проверкой и рефакторинг

*Разработка методом проверки* (Test-Driven Development – TDD) вынесла концепцию *разработки с предварительной проверкой* (test first design) на первый план. Этот подход имеет несколько преимуществ, а следовательно, мы будем писать в этой книге *предварительные проверки* (tests first) всякий раз, когда это можно.

Хотя написание предварительной проверки занимает не много времени, зачастую, когда сроки поджимают, вы задаетесь вопросом, а стоит ли его тратить на проверки. Но я нахожу этот способ программирования удобным (если владеешь им хорошо), в частности, потому, что он помогает мне обдумывать конструкцию классов. Кроме того, если вы не располагаете достаточным временем на проверку модулей и устранение дефектов, обнаруженных в ходе проверок функций и проверок пользователя, обнаружите, что этот стиль работы способен существенно сэкономить время.

Запись предварительных проверок обеспечивает несколько преимуществ. Например, это гарантирует, что вы пишете только функциональный код, который будет фактически использоваться (данное утверждение основано на предположении, что вы пишете код для удовлетворения проверок модулей, которые в свою очередь написаны на основании приемочных испытаний, определенных ранее на базе бизнес-требований). Далее, если ваш код прошел проверку модуля и приемочные испытания, эту часть кода можно считать законченной. Кроме того, это поможет вам лучше спроектировать классы, поскольку при записи предварительной проверки вы заранее определяете, как будут фактически использоваться ваши классы и методы. И наконец, предварительная проверка может помочь вам в повторном использовании, поскольку можно быстро проверить по модулям JUnit код, *подвергшийся рефакторингу* (refactored), и удостовериться, что он работает, как и исходная версия (с учетом отсутствия или незначительных изменений внешнего интерфейса, как определено на [refactoring.com](http://refactoring.com)).

Хотя проверка модуля — это только часть корпоративной проверки, разработчики должны выполнять ее всегда. К другим проверкам относятся: проверка функций, *входной контроль пользователя* (User Acceptance Testing — UAT), проверка интеграции системы (называемая также проверкой интерфейса), проверка в предельных режимах и проверка производительности, а также многие другие.

Мы используем JUnit для реализации наших приемочных испытаний. Ниже приведены примеры файлов, демонстрирующих наше соглашение об именовании для проверочных классов:

- `test/TimesheetListControllerTest.java`
- `test/TimesheetManagerTest.java`
- `test/ReminderEmailTest.java`

Я решил хранить наши проверочные классы JUnit в отдельном проверочном пакете (`г.е. com.visualpatterns.timex.test`), поскольку, на мой взгляд, это упрощает проектирование. Но я также видел, что другие разработчики хранят проверочные классы JUnit в том же каталоге, что и проверяемый код. В данном случае, это был бы наш пакет контроллера `TimesheetListControllerTest.java`.

#### **Личное мнение: установка среды заранее все же необходима**

После почти двух десятилетий разработки программного обеспечения, я поражен, как изменились некоторые из фундаментальных концепций. Несмотря на существенное изменение технологий, основные концепции, такие как установка среды проектирования, разработка, отладка и так далее, остаются неизменными. Одно, я уяснил за последние годы, — это то, что установка среды почти всегда подразумевает немного большее, чем ожидают люди или предусматривает план. Поэтому установка среды заранее жизненно необходима. На основании моего личного опыта, я предпочитаю рекомендовать своим клиентам относительно установки среды следующее. Во-первых, сначала получите минимально достаточную, но полностью функциональную среду (структура каталогов и сценарии создания). Это должно быть согласовано со всеми участниками группы разработчиков программного обеспечения. Во-вторых, получите простую работоспособную демонстрационную версию (например, пользовательский интерфейс для обращения к базе данных). В-третьих, с учетом того, что установка среды продлится дольше, чем ожидалось, заранее зарезервируйте для нее достаточно времени. Это одна из причин, почему многие проекты Agile используют нулевую итерацию или нулевой цикл (Джим Хайсмит (Jim Highsmith)), поскольку демонстрация и установка среды не являются чем-нибудь материальным, с точки зрения клиента; они нужны, скорее, разработчикам для работы и проверок.

## Резюме

В этой главе мы начали установку нашей среды разработки со следующими базовыми инструментальными средствами, обязательными для разработки Java:

- комплект разработчика Java (JDK);
- Ant;
- JUnit.

Однако в следующих главах нам предстоит установить еще несколько систем. Например, необходимо будет установить базу данных, Web-сервер, Eclipse и т.д. Но мы займемся каждой из этих технологий по мере необходимости.

## Рекомендуемые ресурсы

Дополнительная информация по темам, обсуждаемым в этой главе, содержится на следующих Web-сайтах:

- Встроенные задачи Ant <http://ant.apache.org/manual/taskoverview.html>
- Ant <http://ant.apache.org/>
- Статья Continuous Integration <http://www.martinfowler.com/articles/continuousIntegration.html>
- EasyMock <http://www.easymock.org/>
- Внешние задачи Ant <http://ant.apache.org/external.html>
- Среда для интегрированной проверки (FIT) <http://fit.c2.com/>
- Полный жизненный цикл объектно-ориентированной проверки <http://www.ambyssoft.com/essays/floot.html>
- Справочник по Java <http://java.sun.com/docs/books/tutorial/>
- JDK <http://java.sun.com>
- Проверочная среда JUnit <http://junit.org>
- JUnit <http://junit.org>
- Maven от Apache <http://maven.apache.org/>
- Ложные объекты <http://mockobjects.com>
- Разработка методом проверки (TDD) <http://www.agiledata.org/essays/tdd.html>
- Критерии предварительной проверки <http://www.xprogramming.com/xpmag/testFirstGuidelines.htm>