Java Workshop

Table of contents

1. About

1.1. Welcome to the Java Workshop

The programs in this book have been included for instructional value, the author does not offer any warranties or representations in respect of their fitness for a particular purpose, nor does the publisher accept any liability for any loss or damage arising from their use.

1.1.1. About The Book

1.1.1.1. Introduction

The original idea behind the Java Workshop was to introduce some of the practicalities faced when developing software using the Java Platform. As such, the original book focused on developing software using the Java Platform and not so much on the tools or the practical side of things. In revising the text, I decided to shift the focus somewhat to a new direction, the tools that a Java Developer should have in his / her toolbox.

It has often been said that the right tools for the job can make the job a lot easier. This is even more so when one considers the practice of developing software. When one uses the right tools, suddenly the task at hand becomes easier and quicker to accomplish creating a marked improvement in productivity. This book therefore focuses on the practical aspects of developing in Java specifically looking at common tasks and at the tools that make accomplishing these tasks easier and faster.

This is not a book that teaches Java. Instead it is a book that showcases typical tools found in a Java developer's toolbox. If you are looking for a book that teaches Java you should look elsewhere and once you've found such a book, you should couple it with this one, I guarantee the result will be worth it.

1.1.1.2. An "Open Source Book"?

The idea behind this is far from original. Credit must be given to Bruce Eckel's **Thinking in Java**, which is freely available on the web and also available in printed format. While I used a similar concept, the Java Workshop differs fundamentally in that it may be redistributed and modified, much like open source software. For those that are a little finicky about details, the license is available here: License.

1.1.1.3. Who The Book Is For

This book is for anyone interested in Java. Whether that interest is purely in an academic

sense or truly a passion for the language, I hope that everyone can gain something from this book.

The target audience is specifically students who are learning Java for the first time. I remember when I first learnt Java, many of the trivialities were far from trivial and even the simple task of setting an environment variable was no small feat. I hope that if anything, this book will help students to accomplish these and many more trivial tasks and that they will not seem like small miracles.

The goal of this book is to introduce students to the practicalities of the Java language and provide them with enough references to pursue the topic further. In this regard, each topic is presented in a simplistic manner and a topic does not cover all there is, especially those topics that are extremely broad like XML. Rather a brief introduction is given and then a list of suitable references for further information.

1.1.2. Overview

Ironically my vision for the first edition was to showcase a number of tools for Java developers, however the book also included a large amount of theoretical content which was contrary to the idea of an introduction to the practicalities of programming with the Java platform; this second edition is much closer to a manifestation of that original vision than the first edition was.

I focused each chapter around a certain piece of software that I have found exceedingly useful, a tool that I consider to be an essential item in each developer's toolbox. This means that each chapter focuses around performing a specific task.

1.1.2.1. Topics Presented In The Book

Installing The Java Development Kit

Installing the Java Development Kit (JDK) is not difficult but it can be tricky for people that have never done it before. One of the most important learning experiences novice programmers can have is to install the JDK an set the PATH variable. Not only is it an enriching experience but also the satisfaction gained from accomplishing this task is monumental. I hope that the steps outlined in this theme will guide many to successfully install Java on their computers.

Using The Eclipse Integrated Development Environment

With all the IDEs available, many students may wish to become familiar with one of them and use it to write their programs with. The choice of which IDE to use is not an easy one,

yet Eclipse has become the industry standard IDE surpassing many of its competitors. This chapter will look at installing and using this invaluable tool that every Java Developer should have.

Creating and Using Java Packages

Understanding Java Packages is key when one wishes to organize one's source code and this chapter will look at how to create and use Java Packages.

Using the Javadoc Tool to Produce an Application Programmer Interface

This chapter will focus on the importance of comments and the correct way to write Javadoc comments and will show how to use the Javadoc tool to produce an API.

Using the Jar Tool to Create Libraries and Executable Java Archive Files

Java Archives or jar files have become the most popular form of packaging and distributing Java software. It is thus imperative for students to understand what a jar file actually is, how to create one, view the contents and extract the contents as well as package software in an executable jar file.

Using Apache Ant to Simplify the Build Process

Small projects can quite easily be compiled from the command-line. Medium to large projects can become tedious to compile and package and therefore a build tool of some sort can make the process easier. In this regard, Apache Ant is simply the best build tool available. This chapter will discuss Ant and how to use it.

Using JUnit Framework to Perform Unit Testing

Unit testing is perhaps one of the most important aspects of software development. This chapter looks at how unit testing can be performed using the JUnit package.

Using Apache Log4J to Perform Logging

Logging is a very important way of debugging code as it provides a mechanism to see what a program is doing at every step of the way. This chapter looks at logging using the Apache Log4J package.

Processing XML with Apache Xerces

XML is used in a number of places and it is useful to know how to process it. This chapter will look at how to process XML using the Xerces parser as well as perform transformations

using Apache Xalan.

1.1.3. Software and Versions Used

Keeping up to date with rapidly changing technologies is difficult at best. Below is the list of software and versions used during the writing of this book, by the time you read this there will no doubt be newer versions of some if not all the software listed below.

Tool	URL	Chapter	Description
JDK 1.5.0	java.sun.com (http://java.sun.com)	1, 3-5	The Java 2 Standard Edition SDK
Eclipse 3.0	eclipse.org (http://www.eclipse.org)	2	The Eclipse IDE
Apache Ant 1.6.5	ant.apache.org (http://ant.apache.org)	6	The Apache Ant build tool
JUnit 3.2	junit.sourceforge.net (http://junit.sourceforge.net)	7	JUnit unit testing framework
Apache Log4J 1.2.11	logging.apache.org (http://logging.apache.org)	8	Apache Log4J logging framework
Apache Xerces 2.7.1	xml.apache.org (http://xml.apache.org)	9	Apache Xerces XML parser
Apache Xalan 2.7.0	xml.apache.org (http://xml.apache.org)	9	Apache Xalan XML Transformation tool

1.1.4. Conventions Used

Italic is used for emphasis.

`Constant width` is used for all code fragments, paths and filenames.

`Constant width italic` is used for general placeholders that indicate some item is replaced for an actual value.

`Constant width bold` is used for emphasis in a code fragment.

1.1.5. Reach Out

I would like your feedback. After you have had a chance to use this book please take a moment to e-mail me with your evaluation. I'd like to know what you liked, what was useful and more importantly what wasn't of use. Although I will read all mails I recieve, I cannot

guarantee a reply.

`tcmiller@users.sourceforge.net`

Despite numerous efforts to ensure this book is error free, some errors have most likely slipped through. Even still, some of the material in this book will change or no longer be applicable over time. Please let me know of any errors or omissions in this text.

Trevor C. Miller

Pretoria, South Africa, January 5, 2006

1.1.6. Acknowledgements

This book would not have been possible if it were not for the aid and assistance of a number of people. First I would like to thank the people of Sun Microsystems, the Apache Software Foundation, IBM, and of course the Fedora Linux Community. All the software used to write this book came from these organizations that have all produced open source software of such a high standard that it often exceeds that of commercial software. The book was written on Windows using custom software that I wrote to produce the XML content. This was then transformed into static HTML using Apache Forrest. The book was also tested on Linux using Fedora Core.

I wish to thank all those wonderful people that I've had the pleasure tutoring and teaching, whether it was Java, C, C++ or Delphi. These people truly inspired me by persevering, and in those times they even challenged me by asking awkward questions that I at times didn't have an answer for. While I was teaching programming to these people, they in return taught me much; about teaching, about people and most importantly, about myself. To them I am ever grateful.

Lastly, writing this book wouldn't have been possible if it weren't for the music of Leftfield, Michelle Branch and 30 Seconds To Mars.

1.2. Download the Java Workshop

1.2.1. Editions

You can get the latest edition here (http://sourceforge.net/project/showfiles.php?group_id=157002) .

1.2.2. Extracting The Archives

1.2.2.1. Windows

It is recommended that you download the zip distribution archive. Windows XP has a native built-in Zip tool that allows you to extract zip archives. Other versions can use the Jar tool bundled with the JDK to extract the contents of the archive. Tools like Winzip and ZipGenius can extract the zip archive as well as the tarball.

1.2.2.2. Linux / Unix

Most Linux and Unix distributions have the `tar` utility so it is recommended that you download the tarball. If you do not have `tar` you may have `unzip` in which case it is advisable to get the zip archive. If you do not have any of these utilities you can use the Jar tool bundled with the JDK to extract the contents of the zip archive.

1.3. Java Workshop Licence

Licensed under the Academic Free License version 2.1

```
This Academic Free License (the "License") applies to any original work
of authorship (the "Original Work") whose owner (the "Licensor") has
placed the following notice immediately following the copyright notice
for the Original Work:

Licensed under the Academic Free License version 2.1

1) Grant of Copyright License. Licensor hereby grants You a world-wide,
   royalty-free, non-exclusive, perpetual, sublicenseable license to do
   the following:

A. to reproduce the Original Work in copies;
B. to prepare derivative works ("Derivative Works") based upon the Original
   Work;
C. to distribute copies of the Original Work and Derivative Works to the
   public;
D. to perform the Original Work publicly; and
E. to display the Original Work publicly.

2) Grant of Patent License. Licensor hereby grants You a world-wide,
   royalty-free, non-exclusive, perpetual, sublicenseable license, under
   patent claims owned or controlled by the Licensor that are embodied in
   the Original Work as furnished by the Licensor, to make, use, sell and
   offer for sale the Original Work and Derivative Works.

3) Grant of Source Code License. The term "Source Code" means the preferred
   form of the Original Work for making modifications to it and all
available
   documentation describing how to modify the Original Work. Licensor
hereby
```

```
   agrees to provide a machine-readable copy of the Source Code of the
   Original Work along with each copy of the Original Work that Licensor
   distributes. Licensor reserves the right to satisfy this obligation by
   placing a machine-readable copy of the Source Code in an information
   repository reasonably calculated to permit inexpensive and convenient
   access by You for as long as Licensor continues to distribute the
Original
   Work, and by publishing the address of that information repository in a
   notice immediately following the copyright notice that applies to the
   Original Work.

4) Exclusions From License Grant. Neither the names of Licensor, nor the
   names of any contributors to the Original Work, nor any of their
   trademarks or service marks, may be used to endorse or promote products
   derived from this Original Work without express prior written permission
   of the Licensor. Nothing in this License shall be deemed to grant any
rights
   to trademarks, copyrights, patents, trade secrets or any other
intellectual
   property of Licensor except as expressly stated herein. No patent
license is
   granted to make, use, sell or offer to sell embodiments of any patent
claims
   other than the licensed claims defined in Section 2. No right is granted
to
   the trademarks of Licensor even if such marks are included in the
Original
   Work. Nothing in this License shall be interpreted to prohibit Licensor
from
   licensing under different terms from this License any Original Work that
   Licensor otherwise would have a right to license.

5) This section intentionally omitted.

6) Attribution Rights. You must retain, in the Source Code of any
Derivative
   Works that You create, all copyright, patent or trademark notices from
the
   Source Code of the Original Work, as well as any notices of licensing
and
   any descriptive text identified therein as an "Attribution Notice." You
must
   cause the Source Code for any Derivative Works that You create to carry
a
   prominent Attribution Notice reasonably calculated to inform recipients
that
   You have modified the Original Work.

7) Warranty of Provenance and Disclaimer of Warranty. Licensor warrants
that
   the copyright in and to the Original Work and the patent rights granted
   herein by Licensor are owned by the Licensor or are sublicensed to You
   under the terms of this License with the permission of the
contributor(s)
```

```
   of those copyrights and patent rights. Except as expressly stated in the
   immediately proceeding sentence, the Original Work is provided under
this
   License on an "AS IS" BASIS and WITHOUT WARRANTY, either express or
implied,
   including, without limitation, the warranties of NON-INFRINGEMENT,
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS
TO
   THE QUALITY OF THE ORIGINAL WORK IS WITH YOU. This DISCLAIMER OF
WARRANTY
   constitutes an essential part of this License. No license to Original
   Work is granted hereunder except under this disclaimer.

8) Limitation of Liability. Under no circumstances and under no legal
theory,
   whether in tort (including negligence), contract, or otherwise, shall
the
   Licensor be liable to any person for any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or the use of the Original Work including,
   without limitation, damages for loss of goodwill, work stoppage,
   computer failure or malfunction, or any and all other commercial
   damages or losses. This limitation of liability shall not apply to
   liability for death or personal injury resulting from Licensor's
   negligence to the extent applicable law prohibits such limitation.
   Some jurisdictions do not allow the exclusion or limitation of
   incidental or consequential damages, so this exclusion and limitation
   may not apply to You.

9) Acceptance and Termination. If You distribute copies of the Original
   Work or a Derivative Work, You must make a reasonable effort under the
   circumstances to obtain the express assent of recipients to the terms
   of this License. Nothing else but this License (or another written
   agreement between Licensor and You) grants You permission to create
   Derivative Works based upon the Original Work or to exercise any of
   the rights granted in Section 1 herein, and any attempt to do so except
   under the terms of this License (or another written agreement between
   Licensor and You) is expressly prohibited by U.S. copyright law, the
   equivalent laws of other countries, and by international treaty.
   Therefore, by exercising any of the rights granted to You in Section
   1 herein, You indicate Your acceptance of this License and all of its
   terms and conditions.

10) Termination for Patent Action. This License shall terminate
    automatically and You may no longer exercise any of the rights granted
    to You by this License as of the date You commence an action, including
    a cross-claim or counterclaim, against Licensor or any licensee
alleging
    that the Original Work infringes a patent. This termination provision
    shall not apply for an action alleging patent infringement by
combinations
    of the Original Work with other software or hardware.

11) Jurisdiction, Venue and Governing Law. Any action or suit relating to
```

```
this
    License may be brought only in the courts of a jurisdiction wherein the
    Licensor resides or in which Licensor conducts its primary business,
and
    under the laws of that jurisdiction excluding its conflict-of-law
    provisions. The application of the United Nations Convention on
Contracts
    for the International Sale of Goods is expressly excluded. Any use of
the
    Original Work outside the scope of this License or after its
termination
    shall be subject to the requirements and penalties of the U.S.
Copyright
    Act, 17 U.S.C. § 101 et seq., the equivalent laws of other countries,
    and international treaty. This section shall survive the termination of
    this License.

12) Attorneys Fees. In any action to enforce the terms of this License or
    seeking damages relating thereto, the prevailing party shall be
entitled
    to recover its costs and expenses, including, without limitation,
    reasonable attorneys' fees and costs incurred in connection with such
    action, including any appeal of such action. This section shall survive
    the termination of this License.

13) Miscellaneous. This License represents the complete agreement
concerning
    the subject matter hereof. If any provision of this License is held to
be
    unenforceable, such provision shall be reformed only to the extent
    necessary to make it enforceable.

14) Definition of "You" in This License. "You" throughout this License,
    whether in upper or lower case, means an individual or a legal entity
    exercising rights under, and complying with all of the terms of, this
    License. For legal entities, "You" includes any entity that controls,
    is controlled by, or is under common control with you. For purposes of
    this definition, "control" means (i) the power, direct or indirect, to
    cause the direction or management of such entity, whether by contract
or
    otherwise, or (ii) ownership of fifty percent (50%) or more of the
    outstanding shares, or (iii) beneficial ownership of such entity.

15) Right to Use. You may use the Original Work in all ways not otherwise
    restricted or conditioned by this License or by law, and Licensor
    promises not to interfere with or be responsible for such uses by You.

This license is Copyright (C) 2003-2004 Lawrence E. Rosen. All rights
reserved. Permission is hereby granted to copy and distribute this
license without modification. This license may not be modified without
the express written permission of its copyright owner.
```

2. Chapters

2.1. Installing Java

2.1.1. Jargon

API
Application Programming Interface, Set of related classes and methods that provide certain functionality. The API represents the parts of a class exposed through access modifiers to code written by other programmers.
Compiler
A computer program that translates a high-level programming language into machine language. The program fed into the compiler is called the source program; the generated machine language program is called the object program.
Computer
An electronic device that has the ability to store, retrieve, and process data, and can be programmed with instructions that it remembers. The physical parts that make up a computer (the central processing unit, input, output, and memory) are called hardware. Programs that tell a computer what to do are called software.
Environment Variable
A variable that specifies how an operating system or another program runs, or the devices that the operating system recognizes.
JDK
Java Development Kit, A software package that contains the minimal set of tools needed to write, compile, debug, and run Java applets and applications.
Java
Java is an object-oriented programming language developed initially by James Gosling and colleagues at Sun Microsystems. The language, initially called Oak (named after the oak trees outside Gosling's office), was intended to replace C++, although the feature set better resembles that of Objective C. Java should not be confused with JavaScript, which shares only the name and a similar C-like syntax. Sun Microsystems currently maintains and updates Java regularly.
Platform
In computing, a platform describes some sort of framework, either in hardware or software, which allows software to run. Typical platforms include a computer's architecture, operating system, or programming languages and their runtime libraries.
Program
A set of intangible instructions that tells a computer how to operate. Computer

programs are also known as software.
SDK
Software Development Kit A set of programs that allows software developers to create products to run on a particular platform or to work with an API.
Software
A computer program that helps the user accomplish a specific task; for example, a word processing program. Application programs should be distinguished from system programs, which control the computer.

2.1.2. Introduction

In order to develop software using the Java language you will need to install a suitable compiler and interpreter on your computer. This will enable you to compile and run Java programs that you write.

This is not a complex process but varies depending on which platform you use. The platform will dictate the method you use to set variables as different platforms use different methods for achieving this task. Windows itself has different ways depending on which version of Windows you use. Linux and Unix based platforms also have numerous different ways of performing the same task.

Once you have installed Java, you may wish to consult some references. There are a number of excellent references available to Java programmers; some of these include Sun Microsystems's Java Page and the Java API.

You can get the JDK @ java.sun.com (http://java.sun.com)

2.1.3. How to install the JDK

For this installation, the latest version of Java was used at the time of writing, Java 1.5.0, but previous and later versions should work just fine.

2.1.3.1. Microsoft Windows 95

Installing the JDK

1. Run the setup program `jdk-1_5_0-windows-i586.exe`

   Running the setup program.

2. Click "Continue" on the warning message.

   Warning message.

3. Click "Accept Terms in License Agreement" then click on the Next button.

Licence Agreement.

4. Select the components to install. If you are low on free space you can leave out the Demos and Source Code.

Components to Install

5. The wizard will now install the JDK files on your system. This may take a while.

Installing the files

6. Click the Next button to install the J2SE Runtime Environment

Install the Java Runtime Environment

7. In the next dialog select the browsers to integrate the JRE with, I recommend all. Click the Next button.
8. The installation should be complete! Click the Finish button.

Installation complete

Setting the Path Variable

We need to set the path variable so that the system can find the Java compiler.

1. Open the Windows Explorer and navigate to the `C:\` drive. Right-click on the file named `Autoexec.bat` and select edit from the menu.

Edit the autoexec.bat file

Warning:

`Autoexect.bat` may not be visible if it is set as a hidden file. You should set folder options to display hidden files by selecting Options from the Folder menu.

2. The file may be empty, but it will most likely contain some entries. You should add the following entries at the bottom of the file, after the last entries:

```
set JAVA_HOME=C:\Program Files\Java\jdk1.5.0
set PATH="%PATH%;%JAVA_HOME%\bin"
```

Note:

If you changed the installation location of java to some other location, you will have to change the location here to.

3. Save the file, close the text editor and restart your computer.

2.1.3.2. Microsoft Windows 98

Installing the JDK

1. Run the setup program `jdk-1_5_0-windows-i586.exe`

Running the setup program.

2. If you use Windows 98 First Edition and get the warning message, click "Continue" on the warning message.

Warning message.

3. Click "Accept Terms in License Agreement" then click on the Next button.

Licence Agreement.

4. Select the components to install. If you are low on free space you can leave out the Demos and Source Code.

Components to Install

5. The wizard will now install the JDK files on your system. This may take a while.

Installing the files

6. Click the Next button to install the J2SE Runtime Environment

Install the Java Runtime Environment

7. In the next dialog select the browsers to integrate the JRE with, I recommend all. Click the Next button.
8. The installation should be complete! Click the Finish button.

Installation complete

Setting the Path Variable

We need to set the path variable so that the system can find the Java compiler.

1. Open the Windows Explorer and navigate to the `C:\` drive. Right-click on the file named `Autoexec.bat` and select edit from the menu.

Edit the autoexec.bat file

Warning:

`Autoexect.bat` may not be visible if it is set as a hidden file. You should set folder options to display hidden files by selecting Options from the Folder menu.

2. The file may be empty, but it will most likely contain some entries. You should add the following entries at the bottom of the file, after the last entries:

```
set JAVA_HOME=C:\Program Files\Java\jdk1.5.0
set PATH="%PATH%;%JAVA_HOME%\bin"
```

Note:
If you changed the installation location of java to some other location, you will have to change the location here to.

3. Save the file, close the text editor and restart your computer.

2.1.3.3. Microsoft Windows 2000

Installing the JDK

Note:
You should be logged in as Administrator or have Administrative privileges to perform this installation. If not, you should get your System Administrator to do the install for you.

1. Run the `jdk-1_5_0-windows-i586.exe` setup program.
2. Select Accept with the Terms in the License Agreement and click the Next button.

Accept the terms in the licence

3. Select the components you wish to install. If you are low on disk space you can leave out the Demos and Source Code. Click the Next button to continue.

Select components to install

4. The installer will now copy files to your computer, this may take a while.

Copying files to your computer

5. Install the Public Java Runtime Environment by clicking the Next button. You should not have to edit any settings here.

Install the runtime environment

6. Select which browsers to integrate the runtime environment with. I recommend all.

Select browsers to integrate the runtime with

7. That should take care of the installation. The JDK is now successfully installed on your system.

Installation completed.

Setting the Path Variable

We need to set the path variable so that the system can find the Java compiler.

Note:

We will create the variables in the global system space so that any user on the system can use the java tools. If you don't want this, you can create the variables in the Administrator's user space, but then only the administrator will be able to use the java tools. This might be what you want if there is only one user on the computer.

1. Right click on the My Computer icon on the desktop and select Properties from the pop up menu. Click the Environment Variables button.

My Computer Properties

2. Under the System Variables section click the New button to add a new variable.

Environment Variables

3. Add a variable named `JAVA_HOME` with the value `C:\Program Files\Java\jdk1.5.0` and click the Ok button.

The Java Home Variable

Note:
If you installed Java to a different location, you should change the value here to reflect this.

4. Lastly edit the variable named `PATH`, add a semi colon (`;`) after the last entry in the value field and then add `C:\Program Files\Java\jdk1.5.0\bin`

The PATH variable

2.1.3.4. Microsoft Windows XP

Installing the JDK

Note:
You should be logged in as Administrator or have Administrative privileges to perform this installation. If not, you should get your System Administrator to do the install for you.

1. Run the `jdk-1_5_0-windows-i586.exe` setup program.

Run the setup program

2. Select Accept with the Terms in the License Agreement and click the Next button.

Accept the terms in the licence

3. Select the components you wish to install. If you are low on disk space you can leave out the Demos and Source Code. Click the Next button to continue.

Select components to install

4. The installer will now copy files to your computer, this may take a while.

Copying files to your computer

5. Install the Public Java Runtime Environment by clicking the Next button. You should not have to edit any settings here.

Install the runtime environment

6. Select which browsers to integrate the runtime environment with. I recommend all.

Select browsers to integrate the runtime with

7. That should take care of the installation. The JDK is now successfully installed on your system.

Installation completed.

Setting the Path Variable

We need to set the path variable so that the system can find the Java compiler.

> **Note:**
> We will create the variables in the global system space so that any user on the system can use the java tools. If you don't want this, you can create the variables in the Administrator's user space, but then only the administrator will be able to use the java tools. This might be what you want if there is only one user on the computer.

1. Right click on the My Computer icon on the desktop and select Properties from the pop up menu. Click the Environment Variables button.

My Computer Properties

2. Under the System Variables section click the New button to add a new variable.

Environment Variables

3. Add a variable named `JAVA_HOME` with the value `C:\Program Files\Java\jdk1.5.0` and click the Ok button.

The Java Home Variable

> **Note:**
> If you installed Java to a different location, you should change the value here to reflect this.

4. Lastly edit the variable named `PATH`, add a semi colon (`;`) after the last entry in the value field and then add `C:\Program Files\Java\jdk1.5.0\bin`

The PATH variable

2.1.3.5. Linux (Red Hat, Fedora Core & Mandrake)

Installing the JDK

> **Note:**
> This how-to assumes you have root privileges and that you are to install Java system wide so that all users can use the Java tools.

1. Start a shell. Copy the Linux installer to the `/usr/local` directory.

```
cp jdk-1_5_0-linux-i586.bin /usr/local
```

2. Now change into the `/usr/local` directory and run the installer.

```
cd /usr/local
chmod +x jdk-1_5_0-linux-i586.bin
./jdk-1_5_0-linux-i586.bin
```

3. Read through the license agreement and if you agree type `yes`. This will start the installer.
4. If everything goes well you should end up with output that ends with:

```
Done.
```

Setting the Path Variable

1. Create the file named `java.sh` in the directory named `/etc/profile.d` using your favourite text editor. I used Vi.

```
cd /
touch /etc/profile.d/java.sh
vi /etc/profile.d/java.sh
```

2. Add the following to the file.

```
export JAVA_HOME=/usr/local/jdk1.5.0
export PATH=${PATH}:${JAVA_HOME}/bin
```

3. Make the script executable.

```
chmod +x /etc/profile.d/java.sh
```

4. Log out and then log in again for the changes to take effect.

2.1.3.6. Sun Solaris

The JDK should be installed and setup by default. For information on upgrading go to Sun's Website (http://www.sun.com) and browse through the Solaris documentation.

2.1.3.7. MacOS

Most newer versions of MacOS have the JDK installed and setup by default. For information on upgrading check out the MacOS documentation.

2.1.4. Compiling and Running Java Programs

2.1.4.1. Compiling Java Programs

We will now test our installation by compiling and running a simple Java Program.

Microsoft Windows

1. Open the Notepad text editor `Start > All Programs > Accessories > Notepad` and type the following program into it:

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

2. Save the program to the `C:\` drive in a file named `HelloWorld.java`. Make sure to select "All Files" in the Type field of the Save As dialog box.
3. To compile the program, open the Dos Prompt `Start > All Programs > Accessories > Command Prompt` and type:

```
cd\
javac HelloWorld.java
```

   If you don't get any errors then everything went well. If you get errors, check firstly that you saved the program on the `C:\` drive and that it is named `HelloWorld.java`.

Linux

1. Start your favourite text editor and type the following program in it, then save it in a file named `HelloWorld.java` in your home directory.

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

2. To compile the program, start a shell and use the Java Compiler:

```
cd ~
javac HelloWorld.java
```

   If you don't get any errors then everything went well. If you get errors, check firstly that you saved the program in your home directory and that it is named `HelloWorld.java`.

Solaris

1. Start a text editor, either `vi` in the command line or Gedit. Type the following program in

it then save it in a file name `HelloWorld.java` in your home directory.

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

2. To compile the program, start a shell and use the Java Compiler:

```
cd ~
javac HelloWorld.java
```

If you don't get any errors then everything went well. If you get errors, check firstly that you saved the program in your home directory and that it is named `HelloWorld.java`.

2.1.4.2. Running Java Programs

1. To run the program, type:

```
java HelloWorld
```

you should get the following output:

```
Hello World!
```

2.1.4.3. Common Errors and Solutions

1. `Exception in thread "main" java.lang.NoSuchMethodError: main`

   You must define a main method as: `public static void main(String[] args) {...}`

2. `HelloWorld.java:3: illegal character: \8220`
   `System.out.println(öHelloWorld!ö);`

   Do not type your code in a word processor application that uses smart quotes!!

3. `HelloWorld.class Exception in thread "main"`
   `java.lang.NoClassDefFoundError: HelloWorld/class`

   The correct way to call a program is `java HelloWorld` without the ".java" or ".class"

2.1.5. Valuable Resources

2.1.5.1. The Java Application Programmer Interface

The Java API is an Application Programmer Interface that describes in great detail the core Java Foundation Classes that make up the Java Platform. It highlights which packages make up the Platform, which classes belong to which packages and what methods and fields are available in each class. It is an invaluable tool to any Java Programmer.

The Java API

2.1.5.2. Sun's Java Page

Sun Microsystems's Java page can be accessed via the Internet at http://java.sun.com and it provides a download page where you can download all of the Java Technology that you want. It includes documentation: API's, Tutorials, Short Courses and White Papers. It often has industry news and updates and is also an invaluable resource to any Java programmer.

Sun't Java Page

2.1.6. Summary

In order to compile and run Java programs you need to install the J2SE SDK and set the PATH environment variable. Setting an environment variable is dependent on the underlying platform.

Compiling java programs is done using the "javac" command and running them with the "java" command.

The Java API and Sun's Java page are exceptional resources for any Java programmer.

2.1.7. Exercise Set

2.1.7.1. Multiple Choice

1. The JDK is available for which platforms?
    1. Windows
    2. Linux
    3. MacOS
    4. Sun Solaris
    5. All of the above
2. The JDK contains:
    1. Tools to develop software for the Java Platform
    2. A Java Runtime Environment
    3. The core Java API
    4. The Java Source Code
    5. All of the above
3. Installing the JDK requires:
    1. Extracting the JDK archive.
    2. Depends on the platform.

3. Running the installation binary.
4. Setting the path and classpath variables.
5. 1 and 4.

4. The Java API:
    1. Explains which packages form the core of Java Programming Language
    2. Which classes are located in which package.
    3. Which methods a class contains.
    4. The fields that the class contains.
    5. All of the above.
5. Sun's java page:
    1. Contains the Java Tutorial.
    2. Contains all the Java Software.
    3. Is not worth visiting.
    4. 1 and 2.
    5. 2 and 3.

2.1.7.2. Exercises

1. Install the JDK on your own computer.
2. Compile and Run the Hello World program.
3. Browse through the Java API.
4. Browse the Java page at Sun Microsystems's site: java.sun.com (http://java.sun.com)
5. Download or browse the Java Tutorial at: http://java.sun.com/docs/books/tutorial/index.html

2.1.7.3. Programming Exercises

1. Go read through the Java Tutorial at Sun's Java page and then try the following programs:
    1. Change the Hello World program so that it prints out your name.
    2. Write a program to display the first twenty powers of two.
    3. Write a program that generates a temperature conversion chart between Fahrenheit and Celsius.
    4. Write a program to display the first twenty Fibonacci numbers.
    5. Write a program that reads a number and displays it's square root.

2.2. The Eclipse IDE

2.2.1. Jargon

Command
An instruction given to the computer, by means of a keyboard, punch card, mouse, voice command, or other method.

Console
The command prompt window that rece ves output from a `System.out.print` call.
Drive
A device that spins disks or tapes in order to read and write data; for example, a hard drive, floppy drive, CD-ROM drive, or tape drive.
Error
A mistake made by a software developer that introduces a defect in a computer program.
Generate
The automated creation of source code.
IDE
Integrated Development Environment, an IDE combines the editor, compiler and other useful tools in the same software package. Its advantage is that when a program with syntax errors is compiled, the programmer sees the error messages and the original program at the same time -- this makes debugging much easier.
Refactor
Refactoring is the process of rewriting written material to improve its readability or structure, with the explicit purpose of keeping its meaning or behavior.
Syntax
The rules by which the words in a program are combined to form commands to a computer.
Syntax Highlight
Syntax highlighting is a feature of some text editors that displays text, especially source code, in different colors and fonts according to the category of terms. This feature eases writing in a structured language such as a programming language or a markup language as both structures and syntax errors are visually distinct.
Zip
To zip a file is to compress it into an archive so that it occupies less disk space.

2.2.2. Introduction

Often newcomer's to programming will find editing source files in a text editor to be tedious. Seasoned developers hardly ever use a text editor either. An Integrated Development Environment (IDE) provides the functionality to edit source code, compile it and run the source code all from within a single application.

Eclipse is an excellent open source IDE for Java. It is developed by IBM and written in Java itself. Eclipse provides a wide range of capabilities and features; we will look at this useful IDE in this chapter.

You can get Eclipse @ www.eclipse.org (http://www.eclipse.org)

2.2.3. Installing The Eclipse SDK

The first thing we need to do is to install Eclipse.

> **Note:**
> Eclipse is a memory intensive application may take a while to load, depending on how much memory your system has.

2.2.3.1. Installing on Windows

1. Extract the eclipse archive to the `C:\` drive using a program like Winzip or ZipGenius. You should end up with a folder in the C: drive named `eclipse`.

   Extract the archive
2. The eclipse program is in the `eclipse` folder and is a file named `eclipse.exe`. You can double click this file to run the program. You might want to create a shortcut to this file on your desktop or in your start menu.

   Extract the archive

2.2.3.2. Installing on Linux

> **Note:**
> You should be logged in as root to perform this installation.

1. Copy the Eclipse SDK archive to the `/usr/local` directory.

   ```
   cp eclipse-SDK-3.0-linux-gtk.zip /usr/local/
   ```
2. Change into the `/usr/local` directory and extract the archive.

   ```
   cd /usr/local/
   unzip eclipse-SDK-3.0-linux-gtk.zip
   ```
3. Edit the file named `java.sh` in the `/etc/profile.d` directory and add the following to it.

   > **Note:**
   > If you installed java according to chapter 1 you should have a file named `java.sh` in the `/etc/profile.d` directory. If not, now would be an opportune time to create it.

   ```
   export ECLIPSE_HOME=/usr/local/eclipse
   ```
4. Now add `ECLIPSE_HOME` to the `PATH` variable, save the file and exit your editor.

   ```
   PATH=${PATH}:${ECLIPSE_HOME}
   ```

   Log out and then log in again for the changes to take effect.

5. Run Eclipse: in a terminal type
```
eclipse
```
6. In the dialog that asks you to select a workspace click the OK button. After eclipse has started, you will get a welcome screen.

2.2.4. Creating a Hello World Application

The best way to learn about Eclipse and its features is to start using it; we will therefore create a simple Hello World application using it.

2.2.4.1. Create a New Java Project

1. Start eclipse with the command:
```
eclipse
```
2. Select a workspace from the dialog and ensure to check the box that makes it the default workspace.
3. In the welcome screen, click on the "Workbench" icon.

01

4. From the File menu, select New and then Project.
```
File > New > Project
```
5. In the New Project Dialog, click Java Project and then click the Next button.

02

6. In the New Java Project Dialog enter `Hello World` as the Project name and click the Finish button.

03

7. A dialog will pop up asking you to confirm the switch to the Java Perspective. Check the checkbox that reads `Remember my decision` and then click the Yes button.

04

8. In the Java Perspective, in the left side of the screen is the Package Explorer. In it you will see the Project named `Hello World`. Make Sure it is selected.

05

Create a New Class

1. Select File from the menu and click New. Select Class.
```
File > New > Class
```
2. In the New Java Class dialog enter `HelloWorld` as the name of the class, ensure that the `public static void main(String[] args)` checkbox is selected and then click the Finish button.

06

3. You'll notice in the source code editor that the class has already been created for you.

07

Add Source Code

1. In the main method, type

   ```
   System.
   ```

   You'll notice a popup, prompting you to select a member of the `System` class. Select the `out` member.
2. Append the following to the line:

   ```
   .println("Hello World!")
   ```
3. You'll notice a red squiggly line under the closing bracket. This indicates that there is an error in your code. Add a semi colon (`;`) and it will disappear.

08

Save the Code

1. Select Save from the File menu to save the changes to the source code. You'll notice that Eclipse compiles source code automatically, every time that you save it.

   ```
   File > Save
   ```

Run the Program

1. Right Click on the `HelloWorld.java` file in the Package Explorer and select Run followed by Java Application from the pop up menu.

   ```
   Run > Java Application
   ```
2. You'll notice the output in the console.

09

2.2.5. Some Advanced Features

2.2.5.1. Configuring Some Preferences

We will now change a few of the default preferences.

1. Select Preferences from the Window menu.

   ```
   Window > Preferences
   ```
2. Click the arrow next to `Workbench` to expand the sub categories and select `Colors and Fonts`. In the right pane expand `Java` and then `Java Editor Text Font.` Click the Change button, and change the font according to your preferences.

10

3. In the left pane, expand the `Java` category and then the `Code Style` sub category.

Select `Code Templates`. In the right pane select `Comments` and then `Types`. Click the Edit button and change the comment to your preferences.

11

4. Now select `Code` and then the `New Java files` sub category. Click the Edit button and change the comment according to your preferences.

12

5. In the left pane select `Editor` and select the `Show line numbers` check box.

13

6. Select the `Syntax` tab in the right pane and change the color of the keywords from mauve to blue. Click the Apply button.

14

7. Click the Ok button. You'll notice that the changes have taken effect in the source code editor.

15

2.2.5.2. Create a New Project

We will now create a project outside of eclipse and then import it into eclipse.

1. In your home directory, create some directories:

```
mkdir ~/test
mkdir ~/test/src
mkdir ~/test/classes
```

2. In Eclipse, create a new project by selecting New from the File menu and then select Project.

```
File > New > Project
```

3. In the dialog, ensure that `Java Project` is selected and click the Next button.
4. In the next dialog, use `Test` as the project name. Then click the radio button that reads `Create project at external location`. Click the browse button and select the `test` directory in your home directory. Be sure to click the Next button.

16

5. In the source folders list, select `Test` and click the remove button. Click the Add Folder button and select the `src` directory. Click the OK button.

17

6. Click the browse button to change the Default output folder and select the `classes` directory. Click the Ok button.

18

7. The resulting changes should be as follows:

19

8. Click the Finish button.

2.2.5.3. Create Some Packages

You'll notice the `Test` project in the Package Explorer. We will define a package for our project.

1. In the package explorer expand the `Test` project.
2. Right click on the `src` folder in the package explorer and select New and then Package. Add `test` as the package name and click the Finish button.

20

3. You will now see the empty package in the Package Explorer.

2.2.5.4. Create a New Class

We will create a simple class in the package we have just created..

1. Right click on the `Test` package. Create a new class named `Cat` by selecting New and then Class from the pop up menu.

21

2. Add two variables, one named `age` of type `int` and one named `name` of type `String`.

22

3. Add a simple constructor to initialize the variables.

23

2.2.5.5. Generate Getters and Setters

Now we will use eclipse to generate getter and setter methods for our class.

1. Select `Generate Getters and Setters` from the `Source` menu.

```
Source > Generate Getters and Setters
```

2. In the dialog click the Select All button followed by the Ok button.

24

3. You'll notice that the get and set methods were created.

25

4. Don't forget to save the file.

2.2.5.6. Refactoring

We would like to change the name of the package from test and have all changes reflected. First we will create a main program that will use our class.

1. Create a new package named `com.acme.test.main`
2. Create a program called `Main` in the new package.

26

3. Add the following code to the main method.

```
Cat kitty = new Cat()
```

4. You will again notice some squiggly red lines. In the gutter next to the line numbers you will see a hint icon. Click on it.

27

5. In the pop up hint you will be prompted to import the Cat class. Do so, by clicking on the suggestion. You will notice that an import statement has been added to your code, namely: `import test.Cat`. Save the class.

28

6. Right click on the test package in the Package Explorer and select refactor from the menu. Select Rename from the sub menu.

```
Refactor > Rename
```

7. Rename the package to `com.acme.test.pets` .

29

8. If you look at the main class, you'll notice that the import statement has changed to match the new package name, ie: `import com.acme.test.pets.Cat!`

30

2.2.6. Installing a Plug-In

Eclipse is designed in such a way that third party developers can create plug-ins for it. One such plug-in allows you to edit XML files: it provides syntax highlighting for XML and code completion.

Installing a plug-in in Eclipse is a simple and painless process. All plug-ins are distributed in zip compressed archives, much like eclipse. All you have to do is extract the archive in the correct directory within Eclipse and it should work out of the box.

For this installation, we're using the XML buddy plug-in form here (http://www.xmlbuddy.com) .

1. If you open an XML file in Eclipse you will notice that Eclipse uses the default text editor for XML files. This allows you to edit the files but it doesn't provide features common to most IDE's. In this case, installing a plug-in will help.

31

2. Close Eclipse before proceeding.
3. Extract the archive to location of your choice. This results in a folder named

`com.objfac.xmleditor_`*version*.
4. Move this folder into a folder named `plugins` within the `eclipse` folder.

32

5. Start Eclipse and open an XML document, you should see that it is now syntax highlighted.

33

Note:
Installing a plug-in that is not for your version of Eclipse will not work. You should make sure you get the plug-in that is specifically for your version of Eclipse.

2.2.7. Summary

Eclipse is an advanced and feature rich IDE for the Java language. It is simple to install and easy to learn to use. We will be using this valuable tool in the next few chapters in this book.

2.2.8. Exercise Set

2.2.8.1. Multiple Choice

1. Eclipse is written in which language?
    1. Pascal
    2. C
    3. C++
    4. Java
    5. Delphi
2. Eclipse is:
    1. Proprietary Software
    2. Commercial Software
    3. Open Source Software
    4. Shareware
    5. Freeware
3. Which is not a feature of Eclipse
    1. Eclipse allows easy refactoring of code.
    2. Eclipse is available for many platforms.
    3. Eclipse has Apache Ant integrated into it.
    4. Eclipse requires you to compile source code after it is saved.
    5. Eclipse has JUnit integrated into it.
4. Eclipse allows you to:

    1. Generate new classes and interfaces quickly and easily.
    2. Use third-party plug-ins.
    3. Generate getters and setters.
    4. Refactor code.
    5. All of the above.
5. Eclipse is available for:
    1. Windows
    2. Linux
    3. Sun Solaris
    4. 1 and 2.
    5. 2 and 3.

2.2.8.2. Exercises

1. Install the Eclipse on your own computer.
2. Create the Hello World program described in this chapter.
3. Import your existing projects into Eclipse.
4. Configure the Preferences of Eclipse according to your taste.
5. Install the plug-in as described in this chapter.

2.2.8.3. Programming Exercises

1. Complete the following programs using Eclipse:
    1. A program that calculates how much is left of a bank loan. Input parameters are initial loan amount, interest rate, payment per month and current month.
    2. A program to calculate the average of a list of numbers.
    3. A program that calculates a student's grade from grades, each counting an equal percentage.

2.3. Java Packages

2.3.1. Jargon

Attribute
A single piece of information that represents a property present in all instances of a class. An attribute is often modeled as a variable in a class.
Byte
The amount of memory space used to store one ASCII character, which is usually 8 bits. A bit is the smallest unit of information a computer can hold; short for binary digit, the value can be either one or zero.
Component
A piece of software with a clear function that can be isolated and replaced by

another component with equivalent functionality.

Function
A method that performs some form of processing in order to return a result. For example, a function to calculate the sum of two integers.

Input
Something put into a system or expended in its operation to achieve output or a result.

Library
A library is a collection of subprograms used to develop software. Libraries are distinguished from executables in that they are not independent programs.

Namespace
A scoping construct to subdivide the set of names and their visibility within a system. In many languages, this is tied to other computational constructs, eg, classes, procedures, modules, packages. A mechanism used to resolve naming conflicts.

Performance
A major factor in determining the overall productivity of a system, performance is primarily tied to availability, throughput and response time.

Statement
An entity in a programming language which is typically the smallest indivisible unit of execution.

Sub-class
A class that is an extension of another class and inherits public and protected variables and methods from the other class. Also known as a derived class.

2.3.2. Introduction

In order to provide namespace management Java provides the mechanism of creating Java Packages. A Java Package can be thought of as a collection of Java source files, usually grouped by functionality. In order to utilize Java fully you should understand how packages work.

2.3.3. Java Packages

2.3.3.1. What is a Package?

One of the many concerns that programmers face today is trying to ensure that their source code does not conflict with the source code of other programmers. A typical example is the case when two programmers define two distinct classes that have the same name. Suppose you decide to write a `List` class that keeps a sorted list of objects. This would inherently

conflict with the `List` class in the Java API that is used to display a list of items in a Graphical User Interface. Java has a simple solution to this problem that is known as namespace management. Each programmer defines their own namespace and place their code within that namespace, thus two classes that have the exact same name are now distinguishable since they occur in different name spaces. Namespaces are called packages in Java.

Another important reason for using packages is that it provides programmers with greater control over their source code. It is typical to have a few thousand source files in medium to large scale applications, and trying to maintain them would be difficult at best, if not impossible. However, separating these source files into packages makes it much easier to manage the source code. What usually occurs is that related classes are grouped into a single package, for example, all the user interface classes of an application will be grouped into a package.

Access protection is another benefit of using packages. Suppose a group of different applications utilize the same set of source code, it would make sense to separate this source code and maintain it as a separate library that each application uses. In such a library, there is a public interface and a private interface. The public interface is those classes and methods that are accessible to the application using the library, while the private interface is not accessible to the application. Using a package makes it easier to define which classes form part of the public interface and which are part of the private interface. This makes it easy to control access to certain code sections.

Some of the benefits of using packages are:

It shows that the classes and interfaces in the package are related.
Often a group of classes and interfaces are related according to functionality so naturally they should be grouped into a package. An excellent example is the `java.io` package which groups a set of classes that all perform input and output functions.

You know where to find the classes you want if they're in a specific package.
If you are looking for a specific class and you know what functionality it provides you will naturally be able to find it in the right package. If you are looking for an `InputStreamReader` you will find it in the input and output package, ie: `java.io`

The names of your classes and interfaces won't be in conflict with those of other programmers.
If you decide to implement your own String class and you place it in a package it will be distinguishable from the `java.lang.String` class that comes with the Java API.

You can restrict the access to your classes.
When you only want your code or program to access certain code, for example in a library, then using packages makes access control to source code much easier.

2.3.3.2. Creating Packages

Creating a Java Package is relatively simple but may seem a bit confusing for people who have not done it before. There are two fundamental requirements for a package to exist:

- The package must contain one or more classes or interfaces. This implies that a package cannot be empty.
- The classes or interfaces that are in the package must have their source files in the same directory structure as the name of the package.

Naming Packages

Since packages are used to create namespaces that are used to prevent the problem of name conflicts, namespaces must be unique. This means that package names must be unique. Naming a package is therefore a very important aspect of creating a package.

Often organizations that are involved in software development will use their organi ation's domain name, since domain names by definition are unique. The domain name is inverted to make it easier to read. Good examples can be found from looking at the source code of the Apache Software Foundation, whose domain name is `www.apache.org`. All their code is placed into packages that have names beginning with `org.apache`.

Since the Apache Software Foundation has a number of sub-projects, each with its own web address, the project name is also used in the package name. The Ant project can be found at `http://ant.apache.org`, and it should be no surprise that their code is in the packages with names beginning with `org.apache.ant`.

When naming your package you should take your inverted domain name, if you have one, and add the project name to it. This will ensure that you have a unique package name, since it is highly unlikely that your organization is working on two projects that have the exact same name. As an example: `com.mycompany.myproject`

Some companies have decided to drop the top level domain (com, org, net, ...) from their package names for the sake of brevity, this is still perfectly acceptable: `mycompany.mypackage`

Declaring Package Members

The first thing to do when creating a package is to decide which classes and interfaces should

belong to that package. Once you have decided that, you need to specify this by adding a package declaration to the source file of each class and interface that is a part of the package.

The declaration is simple, you use the `package` keyword followed by the package name. This declaration comes right at the top of the source file, before any `import` statements.

```
/*
 * File comments...
 */

package com.mycompany.myproject;

import java.util.*;

class MyClass {

}
```

Source and Class File Organization

One of the requirements for a package is that the source file should be contained in a directory structure that resembles the name of the package. This is simply creating a set of directories with the names given in the package name.

Take the package name and split it up according to where the periods (`.`) occur, for example, `com.mycompany.myproject` can easily be split up into three components, namely: com, mycompany and myproject. These will be the three directories that you create. The first directory will be the `com` directory and within this directory you will create a sub directory namely the `mycompany` directory and within that directory a sub directory, namely the `myproject` directory.

Finally, all the source files that are part of the package will go into the `myproject` directory.

It is typical in many java project to have the source code go into a `src` directory and when it is compiled, the resulting byte code goes into a `classes` directory. The package directories will therefore go into the `src` directory. As an example, consider a class named `MyClass` in the package named `com.mycompany.myproject` and all source code in the `src` directory. The following picture depicts this situation clearly:

Source Code Organization

Compiling The Package

At first it may seem like a nightmare trying to compile the package, especially having the resulting class files occur in the `classes` directory, but it can be achieved using a single

command. You don't even need all the package directories under the `classes` directory since they will be created in the compilation.

To try this out, create a `test` directory and all the sub directories required:

```
mkdir test
mkdir test\src
mkdir test\classes
mkdir test\src\com
mkdir test\src\com\mycompany
mkdir test\src\com\mycompany\myproject
```

Now create the `MyClass` source file named `MyClass.java` in the `test\src\com\mycompany\myproject` directory.

```
/* MyClass.java
 * Author: Trevor Miller
 */
package com.mycompany.myproject;

public class MyClass {

}
```

Change into the `test` directory and compile the package:

```
cd test
javac -d ./classes/ ./src/com/mycompany/myproject/*.java
```

If you take a look in the `classes` directory, you will find the exact same directory structure as in the `src` directory. You'll also find a `MyClass.class` file in the `test\classes\com\mycompany\mypackage` directory. This is good as it allows you to keep your source and class files separate and provides greater control over your files.

Source and Class Organization

2.3.3.3. Using Packages

Once you have created your package you may want to use it in your program or you may wish to use another programmer's package or the packages in the Java API. There are three ways of using the resources in a package; inline package member declarations, importing only the package member that you want and importing the entire package.

Inline Member Declarations

In this approach you simply declare the package member you wish to use with its fully qualified package name. As an example, suppose you wish to use the `Vector` class in the `java.util` package. You can easily declare a vector using `java.util.Vector vector;`

When you initialize the object by calling its constructor, you will again have to use the fully qualified package name.

```
class Test {
   java.util.Vector vector;

   Test() {
      vector = new java.util.Vector();
   }
}
```

Importing a Single Package Member

Using the inline declaration works well if you only use it a few times in your source code, but it can become a pain to type the fully qualified package name if you use it often. Instead you want to do this once and only once, and from then on, simply use the member's name wherever you need it.

This is easily achieved using an import statement in which the `import` keyword is followed by the fully qualified name of the member you wish to use. Consider the example given previously, but revised to use this method:

```
import java.util.Vector;
class Test {
   Vector vector;

   Test() {
      vector = new Vector();
   }
}
```

Importing an Entire Package

It may be that you use a number of members from a package and end up with a large number of import statements. As an example consider the following:

```
import java.util.Vector;
import java.util.LinkedList;
import java.util.Hashtable
import java.util.Stack
import java.util.Set

class Test {
   ...
}
```

This can become messy quite quickly, and you would like to get away from this. The answer is to simply import the entire package, again using the import statement but instead of declaring the member you use a wildcard symbol to indicate that everything should be imported. The wildcard symbol is an asterisk (`*`), using this method you can replace all the import statements of the previous example with simply one import statement.

```
import java.util.*;

class Test {
   ...
}
```

As you can see, this is much cleaner but it does have its limitations. If the package is very large and you import everything in the package, this will impact the performance of your program. You should consider carefully whether greater clarity in your code and ease of writing the code outweighs the addition of some overhead to the performance of your application.

2.3.4. Summary

Packages are a great way to organize your source code and split source code from byte code (class files). Understanding how packages work is vital to writing quality software.

2.3.5. Exercise Set

2.3.5.1. Multiple Choice

1. Namespaces are used to:
    1. Control access to classes.
    2. Resolve naming conflicts among programmers.
    3. Bundle a set of classes together.
2. Which is not a benefit of using packages?
    1. Finding the classes you want is difficult.
    2. Access protection.
    3. Separation of source and class files.
3. To create a package named "za.co.nephila.maculata" you would use which statement?
    1. `package maculata`
    2. `package nephila.maculata`
    3. `package za.co.nephila.maculata`
4. To import the ChatServer member of the Maculata package (za.co.nephila.maculata) you would use:
    1. `import maculata.*;`
    2. `import za.co.nepihla.maculata.*;`
    3. `import za.co.nephila.maculata.ChatServer;`
5. The classes of the Maculata package (za.co.nephila.maculata) will be in which directory?
    1. `/Maculata/classes`
    2. `/Maculata/classes/za/co/nephila/maculata`

3. `/Maculata/src/za/co/nephila/maculata`

2.3.5.2. Exercises

1. Take a look at the Java API and see how the packages are structured.

2.3.5.3. Programming Exercises

1. Complete the following:
   1. Create a package named `org.shapes`.
   2. Create some classes in the package representing some common geometric shapes like Square, Triangle, Circle and so on.
   3. Finally compile the package as discussed in this chapter.

2.4. Javadoc Comments

2.4.1. Jargon

Deprecated
An API item that is considered obsolete and on its way out, usually in favor of something better. Usually, though the item may have been originally included as part of an API, the use of it is no longer advised, and slowly support for the item is phased out.

Documentation
Instructions that come with a software program, which may include paper or electronic manuals, README files, and online help.

Fatal Error
An error that causes a program to stop executing. See Error and Application Failure.

HTML
Hypertext Markup Language, The language used to create World Wide Web pages, with hyperlinks and markup for text formatting.

Internet
A network of computer networks which operates world-wide using a common set of communications protocols.

Member
Class members are items that belong to that class, usually methods and variables and also nested classes.

Options
Alternatives or choices, often refers to settings or preferences in a program that may be set according to the users preference or taste.

Tag

A tag is a marker embedded in a document that indicates the purpose or function of the element. Each element has a beginning tag and an end tag.

Throw

In Java terms, an exception is said to be thrown if the exception is raised. A method may throw an exception if an error occurs in its processing.

Visibility

The accessibility of methods and instance variables to other classes and packages, through the use of access modifiers: public, protected, package or private.

2.4.2. Introduction

In this chapter we will look at the importance of commenting source code and discuss Java's own method of performing source code documentation using the Javadoc Tool that comes bundles with the Java Development Kit.

The Javadoc Tool uses simple Java comments with a number of meta tags to provide meta information about the source code. It then parses these comments and uses this information to produce an API that can be used by developers to see the functionality of the source code. It is important to understand how to write these comments in order to produce a good API.

We will also look at how to run the Javadoc Tool to produce the API and look at some of its more advanced options like sending output to a specific directory, showing private and protected class members and what text should occur in the Window Title of the browser.

You can get the Javadoc Tool @ [bundled with your JDK, if you have the JDK installed you have the javadoc tool].

2.4.3. Commenting In Java

Commenting in Java can take two forms, typical comments that can be found in numerous other programming languages like C/C++; namely single line comments and multiline comments. Then there are Javadoc comments that are unique to the Java language. In this section we will discuss why commenting is needed and look at how to provide the standard single line and multi-line comments in Java.

2.4.3.1. The Need for Comments

There are a number of reasons for including comments in source coe, even though many newer software development methodologies refrain from their use, comments do have their place in source code. Some of the reasons for using comments are:

- Often comments are added at the start of each source file to give a description of what

source the file contains and copyright information.
- The fields and methods of a class are often given brief comments that describe their purpose and what they do.
- Complex code is often commented heavily to make it clearer and easier to understand.

The goal in commenting code is to make it possible for the reader to understand what the code does and how it achieves that. There are generally two schools of thought on this, the first says that the more comments the better, however the other school of thought believes that more comments just clutter the code and make it difficult to read.

Commenting code is therefore a fine balance, you should only provide comments where absolutely necessary and keep them brief. If code is too complex to understand you should consider revising the code to make it clearer or simpler, unless it is impossible to do so or you have very good reasons not to do so.

2.4.3.2. Comments in Java

Single Line Comments

Single line comments are used to add a very brief comment within some code, often a long or complex method. They begin with a double forward slash (`//`) and end with the end of line or carriage return. As an example consider:

```
private static String name = "Guys"; //The name to print
```

Multi Line Comments

If a comment is going to span across more than one line then a multi-line comment should be used. These are often useful for providing more in-depth information. They start with a forward slash followed by an asterisk (`/*`) and end with an asterisk followed by a forward slash (`*/`). Consider:

```
/* Getter method provides public access in read only fashion.
   This function returns the port number. */
int getPort() { ... }
```

2.4.4. Javadoc Comments

Javadoc Comments are specific to the Java language and provide a means for a programmer to fully document his / her source code as well as providing a means to generate an Application Programmer Interface (API) for the code using the javadoc tool that is bundled with the JDK. These comments have a special format which we will discuss in this section

and then in the following section we will look at how to use the javadoc tool to generate an API.

2.4.4.1. The Format of Javadoc Comments

A Javadoc comment precedes any class, interface, method or field declaration and is similar to a multi-line comment except that it starts with a forward slash followed by two atserisks (`/**`). The basic format is a description followed by any number of predefined tags. The entrie comment is indented to align with the source code directly beneath it and it may contain any valid HTML. Generally paragraphs should be separated or designated with the `<p>` tag. As an example consider:

```
/**
 * A Container is an object that contains other objects.
 * @author Trevor Miller
 * @version 1.2
 * @since 0.3
 */
public abstract class Container {

    /**
     * Create an empty container.
     */
    protected Container() { }

    /**
     * Return the number of elements contained in this container.
     * @return The number of objects contained
     */
    public abstract int count();

    /**
     * Clear all elements from this container.
     * This removes all contained objects.
     */
    public abstract void clear();

    /**
     * Accept the given visitor to visit all objects contained.
     * @param visitor The visitor to accept
     */
    public abstract void accept(final Visitor visitor);

    /**
     * Return an iterator over all objects conatined.
     * @return An iterator over all objects
     */
    public abstract Iterator iterator();

    /**
```

```
     * Determine whether this container is empty or not.
     * @return <CODE>true</CODE> if the container is empty:
     * <CODE>count == 0</CODE>, <CODE>false</CODE>
     * otherwise
     */
    public boolean isEmpty() {
        return (this.count() == 0);
    }

    /**
     * Determine whether this container is full.
     * @return <CODE>true</CODE> if conatiner is full,
     * <CODE>false</CODE> otherwise
     */
    public boolean isFull() {
        return false;
    }

}
```

We will now discuss the descriptions of a Javadoc comment first before looking at the different tags and their uses.

2.4.4.2. Descriptions

The description should give a concise summary of the item being commented. It should be written in simple and clear English using correct spelling and grammar. Punctuation is required. There are some important style guidelines to bear in mind:

The first sentence

The first sentence of the description is the most important part of the entire description. It should be a short and concise summary of the item being commented. This is due to the fact that the Javadoc tool copies the first sentence to the appropriate class or package summary page, which implies that the first sentence should be compact and can stand on its own.

Take a look at the example above again and you'll see that the first sentence is a brief descriptive summary of each item.

The use of the <code> tag

The use of the <code> tag is greatly encouraged and should be used for all Java keywords, names and code samples. you'll notice this in the comments of the last two methods of the class in the example above.

Omission of parenthesis

When referring to a method that has no parameters or a method which has multiple forms (method overloading) it is acceptable and even encouraged to simply omit the parenthesis. Consider the following example:

```
The <code>add</code> method inserts items into the vector.
```

This is the correct way of doing it as opposed to the incorrect way in the next example:

```
The <code>add()</code> method inserts items into the vector.
```

Method descriptions begin with a verb

A method usually defines a certain behaviour or operation; because of this it usually signals an action that is best described by a verb.

```
Determine whether this container is empty or not.
```

As opposed to:

```
This method is used to determine whether this container is empty or not.
```

Avoid abbreviation

One final word on style guidelines is to avoid the use of abbreviation at all costs as this renders comments unclear. Instead of using an abbreviation you should use its expanded form. This applies to all abbreviations.

```
This is also known as...
```

Instead of using:

```
AKA ...
```

2.4.4.3. Javadoc Tags

The Javadoc tags are used to provide important or essential meta information about the code. Consider the `@author` tag in the class comment for the example given above, it gives important information as to who the author of the code is. Each tag has a specific format which we will now look at.

Author Tag

Form: `@author name`

Used Where: Interface and Class comments.

Used For: Giving the names of the authors of the source code. You should use the full name of the author or "unascribed" when the author is unknown. Authors are listed in chronological order, with the creator of the class or interface being listed first.

Since Tag

Form: `@since version`

Used Where: Interface and Class comments.

Used For: Indicates the version of the source code that this item was introduced. It is usually just a version umber but may also contain a specific date.

Version Tag

Form: `@version description`

Used Where: Interface and Class comments.

Used For: Describes the current version number of the source code. This is often simply a version number including only the major and minor number and not build number. Some instances also include a date.

Deprecated tag

Form: `@deprecated`

Used Where: Interface, class and method comments.

Used For: Used to indicated that an item is a member of the deprecated API. Deprecated items should not be used and are merely included for backwards compatibility.

Parameter Tag

Form: `@param name description`

Used Where: Method comments.

Used For: Describes a method parameter. The name should be the formal parameter name. the description should be a brief one line description of the parameter.

Return Tag

Form: `@return description`

Used Where: Method comments.

Used For: Describe the return value from a method with the exception of void methods and con tructors.

Exception Tag

Form: `@throws exception description`

Used Where: Method comments.

Used For: Indicates any exceptions that the method might throw and the possible reasons for this exception occurring.

See Class Tag

Form: `@see classname`

Used Where: Any item being commented.

Used For: If another class may help provide clarity this tag may be used to provide a link to that class.

See Class Member Tag

Form: `@see classname#member`

Used Where: Any item being commented.

Used For: If another class's members may provide additional clarity this tag can be used to link to that class's member.

General Order of Tags

The general order in which the tags occur is as follows:

- @author
- @version
- @param
- @return
- @throws

- @see
- @since
- @deprecated

Ordering Multiple Tags

There are three tags that may occur more than once, thay are:

- @author
- @param
- @throws

As mentioned above, the author tag should be listed in chronological order, with the creator of the class or interface listed first. This implies that the last person to work on the source code will have their name appended to the bottom of the list of author tags.

A method may have numerous parameters. In this case, the param tags should be defined in the exact same order as the parameters are declared in the method declaration.

A method may throw numerous exceptions, in such a case it is customary to list the exceptions in alphabetical order, although in some cases they may be listed according to severity, the most severe exception is listed first.

Tag Summary

Image: Tag Summary

2.4.5. The Javadoc Tool

The Javadoc tool comes bundled with the Java JDK and is used to produce an API similar to the Java API. It parses a set of Java source files gathering the information contained within the actual source code as well as the Javadoc comments and uses tis to produce a set of HTML pages documenting the classes, interfaces, methods and fields.

2.4.5.1. Running The Javadoc Tool

The Javadoc tool is run from the command line much like the java compiler. To invoke it you simply use the `javadoc` command and pass a number of command line arguments to the program.

The format for running the tool is:

```
javadoc [options] [packagenames] [sourcefiles] [classnames] [@files]
```

At its most simplest invocation you would call the Javadoc program supplying a Java source

file:

```
javadoc MyClass.java
```

Alternatively you could give a list of files or specify all Java source code using the wild-card (*) symbol:

```
javadoc *.java
```

This will produce the HTML output in the same directory as the source code. This might not be what you want and you should invest some time learning about the tool's more advanced options.

2.4.5.2. Advanced Options

Specifying an Overview File

The default Javadoc page usually displays a list of packages and should also include an overview of the software the API is for. This overview is generated using an overview file.

The contents of the Overview file is standard HTML and is simply copied to the appropriate page by the Javadoc tool. All you need to do is create the HTML page and put in anything you want as the overview. When you run the Javadoc tool, you specify the overview file using the `-overview` option:

```
javadoc -overview overview.html MyClass.java
```

An example overview is:

```
<HTML>
<BODY>
My Overview
</BODY>
</HTML>
```

Specifying Visibility

By default, the Javadoc tool will document both public and protected members of an API. Sometimes you might want to show the private members as well or only show public members. In order to do this you can tell the Javadoc tool which member accessibility level to document.

There are actually four types of visibility that you can specify:

- **public** - Will naturally only document public members, private and protected members are not shown.
- **protected** - Documents both public and protected members and not private members. This is the default.
- **package** - Similar as protected but also shows package classes and members.
- **private** - Displays all members whether they are private or not.

An example of using this method:

```
javadoc -private MyClass.java
```

Specifying a Source Path

Often your source code will be in a `src` directory, especially if you have used packages as explained in the previous chapter. In order to tell the Javadoc tool where to find the source code you need to use the `-sourcepath` option.

The interesting thing about this option is that you need to speify the actual package name if using packages. So assume we are using the example in the previous chapter where the `MyClass.java` file is in the `src/com/mycompany/myproject` directory then you can use:

```
javadoc -sourcepath ./src com.mycompany.myproject
```

Specifying an Output Directory

Until this point, all the output from the Javadoc tool has gone into the current directory. You might want it to go into an alternative directory. Usually the api is found in the `docs/api` directory. To specify this we use the `-d` option:

```
javadoc -d ./docs/api -sourcepath ./src com.mycompany.myproject
```

The resulting output will be in the `docs/api` directory. you'll notice that these directories are automatically created for you, you do not need to create them.

Author & Version Tags

By default, Javadoc does nothing with the author and version tags. You need to explicitly specify that it should use these in the generated API. Doing so is simple:

```
javadoc -author -version -d ./docs/api -sourcepath ./src
com.mycompany.myproject
```

Generating a Package Summary

You may wish to also have a package summary generated. This is nothing more than a simple HTML file much like the overview file except that it is placed within the package directory; for example, if you use the package `com.mycompany.myproject`, then the package overview will be placed within the `myproject` directory. The file must be named `package.html`

You do not need to specify to Javadoc that it should use a package overview, it will automatically detect the file and use it if it exists.

An example:

```
<HTML>
<BODY>
My Package Overview
</BODY>
</HTML>
```

2.4.5.3. Example Output

As an example, I'm going to use all the options in the previous section to generate an API and show the resulting output. The Javadoc command used:

```
javadoc -author -version -private -overview overview.html -d ./docs/api
-sourcepath ./src com.mycompany.myproject
```

The resulting output is:

Image: Javadoc Output

2.4.6. Summary

Javadoc comments can be used to document all java source code. Comments follow a standard format consisting of a description followed by block tags. The first sentence of the description should be clear and concise as it is used in the summary of the API item.

The Javadoc tool can be used to parse doc comments in source code and generate an API from them. The Javadoc tool is flexible and powerful enough to document any number of source files and packages.

2.4.7. Exercise Set

2.4.7.1. Multiple Choice

1. Multi line comments begin and end with curly braces "{" and "}".
    1. true
    2. false
2. Doc comments precede:
    1. Classes
    2. Interfaces
    3. Methods
    4. Fields
    5. All the above
3. Doc comments can include HTML tags.
    1. true
    2. false
4. The first sentence should be?
    1. Concise
    2. Clear
    3. Brief
    4. A summary
    5. All the above
5. Abbreviation is allowed in doc comments.
    1. true
    2. false
6. Which tag cannot be used multiple times?
    1. @author
    2. @param
    3. @throws
    4. @returns
7. The default visibility option of Javadoc is?
    1. Public
    2. Protected
    3. Package
    4. Private
8. The output directory of Javadoc can be changed using?
    1. -d
    2. -directory
    3. -output
9. To run Javadoc on a set of files in a directory named "src", the command that can be used

is?

1. `javadoc *.java`
2. `javadoc -sourcepath ./src *.java`
3. `javadoc ./src/*.java`

10. To run Javadoc on two packages named "nephila.maculata" and "nephila.cruentata" in the directories g:\packages\Maculata and g:\packages\Cruentata you would use:

    1. `javadoc -d ./docs/api -sourcepath ./src nephila`
    2. `javadoc -d ./docs/api -sourcepath g:\packages nephila.maculata nephila.cruentata`
    3. `javadoc -d ./docs/api -sourcepath g:\packages\Maculata\src;g:\packages\Cruentata\src nephila.maculata nephila.cruentata`

2.4.7.2. Exercises

1. Find out about some other options to the Javadoc tool in the documentation provided with the API.

2.4.7.3. Programming Exercises

1. Find some source code that you have written, preferably a set of classes in a package and add Javadoc comments to this source.
2. Now run the Javadoc tool on this source code to generate an API. If there are any errors, fix them and then run Javadoc again.

2.5. Java Archive (Jar) Files

2.5.1. Jargon

Algorithm
A step-by-step problem-solving procedure, especially an established, recursive computational procedure for solving a problem in a finite number of steps.

Archive
A file that contains a group of files and possibly directories which must be extracted in order to use them. An archive may optionally be compressed which would require it to be decompressed before the files can be extracted.

Compression
The temporary coding of data in a way that saves storage space or transmission time. Most text files can be compressed to about half their normal size. Graphics can be compressed to 10 percent of their original size.

Digital Signing
An attempt to mimic the offline act of a person applying their signature to a paper

document.

Enumeration
An object that assists in iterating over a set of objects.

Portability
A measure of system independence; portable programs can be moved to a new system by recompiling without having to make any other changes.

Sealed
A Sealed Jar guarantees that all classes in a package come from the same code source.

Security
Techniques and practices that preserve the integrity of computer systems, and digital library services and collections.

Stream
A continuous flow of data, usually digitally encoded, designed to be processed sequentially. Also called a bitstream.

Versioning
Versioning is a mechanism that keeps track of all changes in content and code and allows any change to be 'rolled back' to any previous version. This also means that a deleted file can be recovered to its last saved state.

2.5.2. Introduction

Jar files provide a way to package software written in Java and to easily distribute that software. In this chapter we will look at what Jar files are, how to use them and how to use the Jar API to access Jar files directly from within our programs.

2.5.3. What Are Jar Files?

2.5.3.1. A Compressed Java Archive File

A Java Archive File, commonly referred to as a Jar File is nothing more than a compressed archive file. The file has two important characteristics:

- The archive does not have to be compressed but if it is, the Zip compression algorithm is used.
- The archive's filename must have the `.jar` extension.

Archives are used to bundle a number of files and directories into one single file, as such Java Archives or Jar Files are no different. Typically, Jar files are used to distribute Java software, these Jar files contains the actual class files of the programs archived in them.

Some benefits of using Jar files are:

- *Security*, A jar file can be digitally signed enabling users to verify the signature and then grant the program security privileges.
- *Decreased download time*, since the archive is compressed it takes less time to download than it would to download each individual file.
- *Package Versioning*, A Jar file may contain vendor and version information about the files it contains.
- *Portability*, all Java Runtime Environments know how to handle Jar files.

2.5.3.2. Creating and Viewing a Jar File

The Jar Tool comes bundled with the Java Development Kit, so if you have a JDK installed you will have the Jar tool. The Jar tool is run via the command line, this means you will use an MS Dos Prompt in Windows and a Shell or Console in Linux / Unix. The command for running the Jar tool is `jar`, you simply issue this command in the command line.

The basic command for creating a Jar file is:

```
jar cf jar-filename   input-files
```

The `c` option indicates that you wish to create a jar and the `f` option indicates that output should be sent to a file, `jar-filename` is the name of the resulting Jar file and `input-files` are the files you wish to include.

As an example, consider a program in the `project` directory. The program consists of a class file and some images in an `images` directory:

example

You would then run the following command to create the jar file from within the `project` directory.

```
jar cvf project.jar MyClass.class images
```

The `v` option is for verbose output. The output would be as follows:

```
added manifest
adding: MyClass.class(in = 462) (out= 291)(deflated 37%)
adding: images/(in = 0) (out= 0)(stored 0%)
adding: images/logo.gif(in = 4178) (out= 4001)(deflated 4%)
```

The command to view the contents of a jar file is as follows:

```
jar tf jar-filename
```

The `t` option is to print a table of contents and the `f` option indicates that your are reading from a file, the `jar-filename` is the filename of the jar you wish to use. If we use the jar file we created earlier:

```
jar tf project.jar
```

The output is as follows:

```
META-INF/
META-INF/MANIFEST.MF
MyClass.class
images/
images/logo.gif
```

The output indicates that out class file and the images directory were included in the jar file. You will also notice that the Jar tool, automatically created and added the manifest file for us. We will find out more about this file in the next section.

2.5.3.3. Extracting Files from a Jar File

If we wish to extract a file from the Jar file, we use the following command:

```
jar xf jar-filename  archived-files
```

The `x` option indicates we wish to extract files, the `f` option indicates from a jar file, the `jar-filename` is the filename of the jar file to extract from and the `archived-files` are the files to extract. In our example we can do the following:

```
jar xf project.jar MyClass.class images/logo.gif
```

This will produce no output on the command line, but will extract the class file and the image file. We can also extract all files by simply not specifying which files to extract:

```
jar xf project.jar
```

If you look in the `project` directory you will notice a directory named `META-INF` which contains a file named `MANIFEST.MF`.

2.5.4. Packaging Software in a Jar File

2.5.4.1. The Manifest File

We said earlier that the Jar file automatically adds a manifest file to the jar file, the file is always in a directory named `META-INF` and is named `MANIFEST.MF`; these are, like Java, case sensitive. This file contins information about the jar file and the files it contains. If you open the manifest file that was extracted in the previous section you will see it contains the following:

```
Manifest-Version: 1.0
Created-By: 1.5.0 (Sun Microsystems Inc.)
```

Jar files are used in a wide variety of situations and therefore need to support a wide range of functionality like digital signing, version control, package sealing, and others. By tailoring the meta information contained in the manifest file, this Jar file is enabled to be used in these situations.

An example of modifying the manifest file is package sealing, which means that all classes defined in a package must be archived in the same Jar file. To do this you need to add a Name and Sealed header to the manifest file, the name contains the package name and the sealed header indicates whether it is sealed or not.

```
Name: MyCompany/MyProject/MyPackage/
Sealed: true
```

You would add this information to a text file, suppose it is named seal.txt, you can add this information to the manifest using the `m` option.

```
jar cmf seal.txt myJar.jar MyCompany
```

Two other important manifest headers are the Main-Class and Class-Path, which are used to bundle software into a Jar file so that the software can be run from within the Jar file. This is known as an executable Jar file.

2.5.4.2. Packaging the Program

Assume your program is name `Main.java` which compiles to `Main.class`, and that it requires a jar file named `jutil.jar` in the `lib` directory. You can get the source code here. You will then add the following to a text file named `headers.txt`:

```
Main-Class: Main
Class-Path: ./lib/jutil.jar
```

Make sure the headers.txt has a blank new line at the end of it. You can use the following command to create the Jar file.

```
jar cmf headers.txt project.jar Main.class
```

2.5.4.3. Running the Program

You will use the Java tool to run the program packaged inside the Jar file. Simply issue the command:

```
java -jar project.jar
```

This should run the program which will display a blank Swing Window.

Swing Window

2.5.5. Using The Jar API

Java comes with an API for working with Zip archives as well as Jar files. In this section we'll look at how to use these APIs.

2.5.5.1. Printing the Contents of a Jar File

The first thing that we'll want to do is to open the jar file. This is easily done by creating a new `java.util.jar.JarFile` instance.

```
try {
    JarFile jarFile = new JarFile("test.jar");
} catch (Exception e) {
    e.printStackTrace();
}
```

The next thing is to enumerate through the jar entries. We get an `Enumeration` that contains `java.util.zip.ZipEntry` instances. We then simply print out the name of each entry.

```
Enumeration entries = jarFile.entries();
while (entries.hasMoreElements()) {
    ZipEntry entry = (ZipEntry) entries.nextElement();
        System.out.println(entry.getName());
}
```

The last thing we need to do is to close the Jar file.

```
jarFile.close();
```

Running this program will print out all the entries in the jar file named `test.jar`.

2.5.5.2. Extracting a File from a Jar File

Extracting a file from a Jar file is a little more tricky since we have to use some IO classes to do the dirty work for use. We can create the Jar file as we did in the previous section and we can get the entry by its name. We will use the jar entry to get an input stream to it as follows:

```
JarFile jarFile = new JarFile("test.jar");
InputStream in = jarFile.getInputStream(
    jarFile.getEntry("META-INF/MANIFEST.MF"));
```

We then create an output stream to a file.

```
OutputStream out = new FileOutputStream("manifest.mf");
```

We then copy the data from the input stream into the output stream.

```
int c;
while ((c = in.read()) != -1)
    out.write(c);
```

Finally we close the streams and the Jar file.

```
in.close();
out.close();
jarFile.close();
```

Running this program will output the Jar file's manifest to a file named `manifest.mf`.

2.5.6. Summary

A Java Archive File, commonly referred to as a Jar File is nothing more than a compressed archive using the Zip compression algorithm. Jar files are used to package Java programs and makes them easy to distribute. We can also access Jar files from within our Java programs.

2.5.7. Exercise Set

2.5.7.1. Multiple Choice

1. A Jar File is a compressed archive file.
    1. True
    2. False

2. The Java Archive must be compressed.
    1. True
    2. False
3. Benefits of using Jar Files are:
    1. Security
    2. Decreased download time
    3. Package Versioning
    4. Portability
    5. All the Above
4. Which command is used to create a jar file?
    1. java -jar project.jar
    2. jar tf jar-filename
    3. jar cf jar-filename input-files
    4. jar xf jar-filename archived-files
    5. jar cmf seal.txt myJar.jar MyCompany
5. Which command is used to view the contents of a Jar file?
    1. java -jar project.jar
    2. jar tf jar-filename
    3. jar cf jar-filename input-files
    4. jar xf jar-filename archived-files
    5. jar cmf seal.txt myJar.jar MyCompany
6. Which command is used to add information to the Manifest file when creating the Jar file?
    1. java -jar project.jar
    2. jar tf jar-filename
    3. jar cf jar-filename input-files
    4. jar xf jar-filename archived-files
    5. jar cmf seal.txt myJar.jar MyCompany
7. Which command is used to run a program bundled in a Jar file?
    1. java -jar project.jar
    2. jar tf jar-filename
    3. jar cf jar-filename input-files
    4. jar xf jar-filename archived-files
    5. jar cmf seal.txt myJar.jar MyCompany
8. Which command is used to extract the contents of a Jar file?
    1. java -jar project.jar
    2. jar tf jar-filename
    3. jar cf jar-filename input-files

4. jar xf jar-filename archived-files
5. jar cmf seal.txt myJar.jar MyCompany

9. The class containing the main method in a program is named net.sourceforge.jinit.Main. What entry will go in the manifest file?
    1. Main-Class: Main
    2. Class-Path: net/sourceforge/jinit
    3. Main-Class: net.sourceforge.jinit.Main
    4. Class-Path: net.sourceforge.jinit.Main
10. The `JarFile#getEntry` method retuns a?
    1. JarEntry
    2. ZipEntry

2.5.7.2. Exercises

1. Find out about the other options to the `jar` command.

2.5.7.3. Programming Exercises

1. Complete the following:
    1. Write a Java program to extract the contents of a jar file.
    2. Write a program to create a jar file. The contents of the jar file are passed as a command line parameter to the program.

2.6. Apache Ant

2.6.1. Jargon

Automated
Acting or operating in a manner essentially independent of external influence or control.

Binaries
An application binary is the compiled form of a program, the actual machine code that is executed by a system in order to run the program.

Build
The process of compiling and integrating all components of a piece of software, incorporating all changes made since the last build.

Embedded
An object, such as a graphic, which has been placed in a document from another file.

Executable
A program that a computer can directly execute. See Binaries.

Markup

Syntactically delimited characters added to the data of a document to represent its structure. There are four different kinds of markup: descriptive markup (tags), references, markup declarations, and processing instructions.

Property
A variable in the form `name=value`.

Script
A file containing operating system commands that are processed in a batch method, one at a time, until complete.

Standalone
A program, function, files or system that operates on its own and has no dependencies on other programs, functions, files or systems.

Task
A sequence of instructions treated as a basic unit of work.

2.6.2. Introduction

Medium to large sized applications can benefit from an automated build process. Apache Ant is an excellent build tool specifically for the Java platform and we'll look at it in this chapter.

You can get Apache Ant @ ant.apache.org (http://ant.apache.org) .

2.6.3. XML

In order to use Apache Ant you will need to understand a little about XML first. We will briefly introduce XML in this section and look at Ant in the next.

2.6.3.1. What Is XML?

XML is an Acronym derived from *eXtensible Markup Language* which is a simplified version of SGML designed to describe data and focus on what data is. This may seem a little confusing especially if you don't understand what a markup language is. If you are familiar with HTML you'll know what a markup language is, but it is important to understand the difference between HTML and XML.

HTML is a very common markup language used on the web today, but XML and HTML differ considerably. XML was designed to carry data and provide a description of that data without caring less about how the data should be presented. HTML, on the other hand, is all about the presentation of data. HTML is about displaying information while XML is about describing information.

From this it should become clear that XML was not designed to replace HTML, in fact the two go hand in hand and complement one another. There are two important principles that

stem from the definition of XML given above:

- XML is not a predefined language like HTML, instead you get to define your own tags.
- XML can be validated using a DTD or XML Schema.

2.6.3.2. Defining an XML Document

Markup consists of text which is the content of a document along with the markup tags embedded within the text. Most markup languages like HTML have a predefined set of tags, however XML is different. The author of an XML document gets to define the set of tags that may be used for that document.

Example XML Document

As an example consider a memo defined using XML:

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

<!DOCTYPE memo
    SYSTEM "memo.dtd">

<memo>
    <to>John</to>
    <subject>Re: Progress Meeting</subject>
    <message>Don't forget that we must book the boardroom
        well in advance or we'll have to have the meeting in
        one of our offices which will turn out to be
        <emphasis>very cramped</emphasis>.
        See you later!
    </message>
    <from>Trevor</from>
</memo>
```

Angle braces (< and >) are used to define XML tags. All XML tags begin with the start brace (<) followed by the tag and end with the closing brace (>). The tags define the structure of the document and add extra information called meta information about the document while the text between the tags is the actual content of the document.

The first line in any XML document must be the XML declaration which announces that this document is an XML document. The declaration is as follows:

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

In the example above, the second tag defines the document type, this is called a document type declaration. This is optional and declares which DTD is used to validate the XML document. The general form of such a type declaration is:

```
<!DOCTYPE root-element uri-of-dtd>
```

The root element is the first XML tag in the document. An XML document contains only one root element which contains all the other elements. In the example above, the root element is `<memo>`.

2.6.3.3. Elements and Attributes

Elements are the actual building blocks of XML. They are XML tags which may contain content or other XML elements. In the example above some elements are: `to`, `subject`, `message` along with the root element `memo`. An element may be empty in which case a closing tag is not required:

```
<empty></empty>
```

which can be written as:

```
<empty_noclosing/>
```

Element names may be anything the author wishes and can be any length provided they start with a letter or underscore. It is usually a god idea to name the tags something relevant rather than something random like signs of the zodiac.

XML elements may provide extra information through the use of attributes. Attributes can describe details about an element or a property of the element. If we take the example above, we can make the memo have a high priority by giving it a simple attribute:

```
<memo priority="high">
...
</memo>
```

Attributes are simple `name=value` pairs with the value in single (`' '`) or double (`" "`) quotes. An element may only contain many attributes but a single attribute may not be used more than once for example:

```
<team player="joe" player="john">
```

the above should be resolved into a number of separate elements:

```
<team>
        <player>joe</player>
        <player>john</player>
```

```
</team>
```

2.6.4. Ant Jumpstart

In this section we'll get straight into using Apache Ant by discussing how to install it, how it works and finally how to run Ant.

2.6.4.1. Installing Ant

Installing Apache Ant is a simple two step process:

1. Extract the binary archive to a folder of your choice.
2. Set the appropriate environment variables.

I like to extract Ant to `C:\ant` on Windows and `/usr/local/ant` on Linux.

In order to set the variables you should follow the methods described in Chapter 1 for your operating system. The variables that need to be set are:

- Add ant's `bin` directory to the `PATH` variable.
- Add `ANT_HOME` variable to point to the location where ant is installed.
- Ensure that `JAVA_HOME` is set correctly.

These variables may look something like:

```
set ANT_HOME=c:\ant
set JAVA_HOME=c:\j2sdk1.4.2
set PATH=%PATH%;%ANT_HOME%\bin
```

for Windows and

```
export ANT_HOME=/usr/local/ant
export JAVA_HOME=/usr/local/j2sdk1.4.2
export PATH=${PATH}:${ANT_HOME}/bin
```

for Linux.

2.6.4.2. Project Directory Structures

How you structure your directories will depend on the type of project you are working on, however, the basic pattern appears to be the same for all projects and you should use it as a starting point and make deviations accordingly. There are usually five important directories within a project directory:

```
/project
    /bin
```

```
    /build
    /docs
    /lib
    /src
```

The directories and their purposes are:

- `bin` - Project binaries and scripts go in here.
- `build` - Used during the build process to store temporary class files and so on. It gets emptied when Ant does a clean.
- `docs` - Project documentation will go here along with the generated API.
- `lib` - Any libraries that the project uses go in this directory.
- `src` - All the project's source code will go in this directory.

2.6.4.3. The Build Process

The build process follows a number of simple steps to achieve the desired end result. These steps are:

Clean
During this step the `build` directory is emptied and all temporary files and directories are removed. After this step is performed, only the source code, program binaries and jar files remain.

Compile
In this step, the source code is compiled and the resulting class files are placed into the build directory. Any file the project needs are also placed into the build directory.

Jar
The class files are archived into the project's jar file.

Docs
The documentation is generated and the API generated using the Javadoc tool.

Distributions
Usually the build includes generating source and binary distributions of the project.

2.6.4.4. The Ant Buildfile

The build file used by Ant is written in XML, saved as "build.xml" and contains one project element. The project element contains a number of elements; the most common ones are property and target elements. The target elements specify a specific target that can be run and will usually consist of one or more built-in tasks.

Project

The project element is the root element of the build file and can contain three attributes: name, basedir and default. Of the three only the default attribute is required. the name defines the name of the project, the basedir defines the base directory of the project and the default attribute sets the default target of the project.

```
<project name="Chapter6" basedir="." default="dist">
```

Properties

A project can specify a set of properties to be used in the build file. These can be set using the property task, a property has a name and a value; both will be attributes of this task.

Properties may be used in task attributes by placing the property name between a "${" and "}", for example, suppose there is a property named `build.dir` and has a value `build` then it could be used in an attribute as "${build.dir}/classes" assuming that the "classes" directory is in the "build" directory. This would then be resolved to: `build/classes`.

```
<property name="build.dir" value="build" />
<property name="build.dest" value="${build.dir}/classes" />
```

Targets

Targets specify which task or tasks need to be run in order to accomplish a certain activity. A target can be dependent on another target, say you have two targets: one to compile and another to build a jar. You cannot build a jar without first compiling and thus the jar target is dependent on the compile target.

A target element can have the following attributes:

- name
- depends
- if
- unless
- description

The name is the name of the target and can be called to run via the command-line when Ant is run. Names should be short but descriptive of what the target does. An initialization target can be called "init" and "compile" is usually used for a target that compiles the source.

The "depends" attribute specifies which other targets must execute before this one. This is a comma-separated list. You should not make any assumptions about the order that targets get called, if you have the following:

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="C,B,A"/>
```

and you run target D, the order of execution is not C, B, A, D but rather A, B, C and finally D.

The "if" attribute specifies the name of a property that must be set in order for this target to be executed. If the property is not set, the target is not executed.

"Unless" specifies the name of a property that must not be set in order for this target to execute, if the property is set then the target is not executed.

"Description" provides a short description of the target's function and will be printed out in help messages.

The only attribute that is required is "name" although it is a good idea to include "description" as well.

```
<target name="init">
        <tstamp/>
        <mkdir dir="${build}"/>
</target>
```

An example build file is:

```
<project name="Chapter6" basedir="." default="dist">

        <!-- Remove temporary files and folders. -->
        <target name="clean">
                <delete dir="build" />
        </target>

        <!-- Compile the source code and place resulting class
                files in the build directory. -->
        <target name="compile" depends="clean">
                <mkdir dir="build"/>
                <javac srcdir="src" destdir="build" />
        </target>

        <!-- Create a jar file named chapter6.jar from the
                class files in the build directory and use the file
                named manifest as a manifest file in the jar. -->
        <target name="jar" depends="compile">
                <jar destfile="bin/chapter6.jar" basedir="build"
                        manifest="manifest" />
        </target>
```

```
        <!-- Create the javadoc API in the docs directory. -->
        <target name="docs">
                <mkdir dir="docs"/>
                <javadoc sourcepath="src" destdir="docs"
                        packagenames="com.mycompany.*"
                        overview="overview.html" />
        </target>

        <!-- Create a distribution directory named chapter6 in
                the build directory then copy the build, overview
                and manifest file as well as the bin, src and docs
                directory to it. Finally create a zip file
                containing this folder. -->
        <target name="dist" depends="jar, docs">
                <delete dir="build" />
                <mkdir dir="build" />
                <mkdir dir="build/chapter6" />
                <mkdir dir="build/chapter6/src" />
                <mkdir dir="build/chapter6/docs" />
                <mkdir dir="build/chapter6/bin" />
                <copy todir="build/chapter6" file="build.xml" />
                <copy todir="build/chapter6" file="manifest" />
                <copy todir="build/chapter6" file="overview.html" />
                <copy todir="build/chapter6/src">
                        <fileset dir="src" />
                </copy>
                <copy todir="build/chapter6/bin">
                        <fileset dir="bin" />
                </copy>
                <copy todir="build/chapter6/docs">
                        <fileset dir="docs" />
                </copy>
                <zip destfile="chapter6.zip" basedir="build" />
                <delete dir="build" />
        </target>

</project>
```

2.6.4.5. Running Ant

In order to run Ant, you'll need to have a buildfile defined within the project directory. For this purpose, you can use the buildfile in the example given above. You will also need the source code in the correct package structure from the previous two chapters, which you can get here: chapter6.zip

If you downloaded the example code and extracted the archive, you can then run Ant within the `chapter6` directory from a command line using the command: `ant`. this will produce output similar to:

```
Buildfile: build.xml
```

```
clean:

compile:
    [mkdir] Created dir: /home/trevor/chapter6/build
    [javac] Compiling 1 source file to /home/trevor/chapter6/build

jar:
      [jar] Building jar: /home/trevor/chapter6/bin/chapter6.jar

docs:
    [mkdir] Created dir: /home/trevor/chapter6/docs
  [javadoc] Generating Javadoc
  [javadoc] Javadoc execution
  [javadoc] Loading source files for package com.mycompany.myproject...
  [javadoc] Constructing Javadoc information...
  [javadoc] Standard Doclet version 1.5.0
  [javadoc] Building tree for all the packages and classes...
  [javadoc] Building index for all the packages and classes...
  [javadoc] Building index for all classes...

dist:
   [delete] Deleting directory /home/trevor/chapter6/build
    [mkdir] Created dir: /home/trevor/chapter6/build
    [mkdir] Created dir: /home/trevor/chapter6/build/chapter6
    [mkdir] Created dir: /home/trevor/chapter6/build/chapter6/src
    [mkdir] Created dir: /home/trevor/chapter6/build/chapter6/docs
    [mkdir] Created dir: /home/trevor/chapter6/build/chapter6/bin
     [copy] Copying 1 file to /home/trevor/chapter6/build/chapter6
     [copy] Copying 1 file to /home/trevor/chapter6/build/chapter6
     [copy] Copying 1 file to /home/trevor/chapter6/build/chapter6
     [copy] Copying 2 files to /home/trevor/chapter6/build/chapter6/src
     [copy] Copying 3 files to /home/trevor/chapter6/build/chapter6/bin
     [copy] Copying 16 files to /home/trevor/chapter6/build/chapter6/docs
      [zip] Building zip: /home/trevor/chapter6/chapter6.zip
   [delete] Deleting directory /home/trevor/chapter6/build

BUILD SUCCESSFUL
Total time: 13 seconds
```

This is how to run a standard build, however, Ant has a number of options that can be passed via command line arguments that make it more flexible and powerful.

Determining the Version of Ant

You may wish to find out which version of Ant is installed on a System, especially if you are running Ant on a system that you did not install it on and you require features from a specific version of Ant. This can be achieved using the `-version` command line argument:

```
ant -version
```

which will produce output similar to:

```
Apache Ant version 1.6.2 compiled on July 16 2004
```

Running a Specific Target

You may wish to run only one or two specific targets instead of the default target. This is achieved by adding the target names as command line arguments to the Ant program: `ant target1 target2 ...`, as an example, suppose we only want to run the compile target:

```
ant compile
```

An interesting thing occurs if you did not write the buildfile yourself, how do you know which targets there are? You could simply just open the buildfile in a text editor and check but this may be tedious. Instead, you can let ant tell you by using the `-projecthelp` command line argument:

```
ant -projecthelp
```

which, for the example source code will give you

```
Buildfile: build.xml

Main targets:

Other targets:

 clean
 compile
 dist
 docs
 jar
Default target: dist
```

Using Logging

You may wish to tell Ant how much information to print on the console when it is running a build. There are three options here: quiet, verbose and debug. Quiet will print almost nothing to the console, while verbose prints a large amount of information. Debug prints the most and is only useful for debugging purposes. To specify which you pass either a `-q` for quiet, a `-v` for verbose or a `-d` for debug. As an example, we'll use quiet:

```
ant -q
```

produces the following output

```
BUILD SUCCESSFUL
Total time: 8 seconds
```

If the amount of output is large, you can also send it to a log file instead of the console using the `-l` option. Consider:

```
ant -l log.txt
```

which produces

```
Buildfile: build.xml
```

in the console.

Specifying An Alternate Buildfile

There may come a time when a project has more than one buildfile, both cannot be named `build.xml` which is the default that Ant looks for. In this case you can tell Ant to use the other file that may be named something like `build2.xml` using the `-buildfile` option:

```
ant -buildfile build2.xml
```

2.6.5. Ant Tasks

In this section we will briefly look at some of the most widely used Ant Tasks. It is useful to understand core types before looking at the tasks that depend on them.

2.6.5.1. Ant Types

Pattern Sets

A pattern set is used to define a special pattern that may be used in file sets or directory sets. It includes patterns which either match or do not, those that match get included in the set while those that do not match are excluded. It is also possible to explicitly exclude items from set using a pattern.

A pattern usually consists of alpha numeric characters along with the wildcard character (`*`). A typical example is all files ending with the `.java` extension:

```
**/*.java
```

As another example, consider matching all names that have the string `Test` in them:

```
**/*Test*
```

Patterns are used as values in attributes for the `include` and `exclude` elements. Suppose we wish to include all source files and exclude any source file that is a test, we can use the patterns given above in a pattern set:

```
<patternset id="non.test.sources">
  <include name="**/*.java"/>
  <exclude name="**/*Test*"/>
</patternset>
```

Directory Sets

Directory sets are used to select a set of directories based on a pattern set, starting in a base directory specified in the `dirset` element as an attribute. Suppose we wish to select the classes directory from the build directory but not the images directory:

```
<dirset dir="build">
  <include name="**/classes"/>
  <exclude name="**/images"/>
</dirset>
```

File Sets

File sets are used to define a set of files. This is easily done by first selecting the directory in which the files occur and then using includes and excludes to limit the files. A typical example is for defining a set of source files:

```
<fileset dir="src">
  <include name="**/*.java" />
</fileset>
```

The `include` and `exclude` elements can also be set as attributes of the `fileset` element. Case sensitivity can be set using the `casesensitive` attribute of the `fileset` element:

```
<fileset dir="src" casesensitive="yes">
  <include name="**/*.java"/>
  <exclude name="**/*Test*"/>
</fileset>
```

Path Structures

You can quite easily use the fileset to define a classpath for the compilation of source code. Suppose your project uses jar files from another project and these jars are stored in the `lib` directory. You can define a classpath as follows:

```
<classpath id="classpath">
    <fileset dir="lib">
        <include name="**/*.jar"/>
    </fileset>
</classpath>
```

2.6.5.2. Core Tasks

In this section, I'll briefly showcase some of the most common tasks used with Ant. For detailed information you should consult the Ant Manual.

Archive Tasks

Ant has a number of tasks that aid in working with compressed archive files, you can easily create as well as extract these archives in a number of formats.

zip, unzip

The zip and unzip tasks can be used to create and extract archives in the zip format.

```
<zip destfile="archive.zip" basedir="src" />
<unzip src="archive.zip" dest="build" />
```

gzip, gunzip

Similar to the zip and unzip tasks, with the exception that these tasks use the GZIP compression algorithm. It should be noted that the gzip task works on a single file so if you wish to compress a bunch of files you will need to use this task in conjunction with the `tar` task.

```
<gzip destfile="archive.gzip" src="afile.txt" />
<gunzip src="archive.gzip" dest="build" />
```

bzip2, bunzip2

Exactly like the gzip and gunzip tasks with the only exception being the use of the BZIP2 compression algorithm.

```
<bzip2 destfile="archive.bzip2" src="afile.txt" />
```

```
<bunzip2 src="archive.bzip2" dest="build" />
```

tar, untar

The tar and untar task is used to create and extract archives using the tar format. It should be understood that this is not a compression utility only an archive utility. You can create an archive using tar and then compress it using gzip or bzip.

```
<tar destfile="archive.tar" basedir="src" />
<untar src="archive.tar" dest="build" />
```

Java Tasks

It should come as no surprise that Ant allows you to run the tools that come with the JDK.

javac

This task is used to compile source files. The srcdir attribute specifies the directory containing the source files and the destdir is the directory where resulting classfiles should be put.

```
<javac srcdir="src"
    destdir="build"
    classpath="xyz.jar"
/>
```

javadoc

You can use this task to generate an API using the javadoc tool.

```
<javadoc packagenames="com.mycompany.myproject.*"
    sourcepath="src"
    defaultexcludes="yes"
    destdir="docs/api"
    author="true"
    version="true"
    use="true"
    windowtitle="MyCompany MyProject API"
/>
```

jar

You can create Java Archives (jar files) with this task.

```
<jar destfile="app.jar"
    basedir="build/classes"
    excludes="**/Test.class"
/>
```

File Tasks

It would be impossible for a build tool not to allow you to manipulate files. These are some of the most important tasks.

mkdir

This one creates a directory.

```
<mkdir dir="build"/>
```

copy

Copy files and directories.

```
<!-- Copy a single file -->
<copy file="myfile.txt" tofile="mycopy.txt"/>

<!-- Copy a single file to a directory -->
<copy file="myfile.txt" todir="../some/other/dir"/>

<!-- Copy a directory to another directory -->
<copy todir="../new/dir">
    <fileset dir="src_dir"/>
</copy>
```

move

Move files and directories.

```
<!-- move single file -->
<move file="file.orig" tofile="file.moved"/>

<!-- move file to directory -->
<move file="file.orig" todir="dir/to/move/to"/>

<!-- move directory to new directory -->
<move todir="new/dir/to/move/to">
    <fileset dir="src/dir"/>
</move>
```

delete

Delete files and directories.

```
<delete file="/lib/ant.jar"/>
<delete dir="lib"/>
```

Execution Tasks

You may wish to run a program outside of Ant, this can easily be done.

exec

Use this task to run a program.

```
<exec executable="notepad.exe" os="Windows 2000" />
```

java

You can also run a java program using the java task.

```
<java classname="com.mycompany.myproject.Main">
    <classpath>
        <pathelement location="lib/xyz.jar"/>
    </classpath>
</java>
```

2.6.5.3. Defining Your Own Task

It is quite possible that you want Ant to do something and there is no task for it, in such a case you can quite easily write your own Ant task to do the work. It is very simple, and there are only a few steps to the process, you can find out how to do this by reading the Ant Manual.

2.6.6. Summary

XML is a powerful markup language for describing and structuring data. It consists of tags enclosed in angle braces (< and >), tags consist of elements and attributes. It is easy to see why it is used for ant build scripts.

Ant is used as a build tool to perform the build process. A build process consists of compiling the source code, creating a jar archive of the resulting class files, creating or generating documentation, creating distributions and finally cleaning any temporary files used in the build process.

The Ant build file is defined in XML and contains a number of targets. Each target performs a number of tasks. These tasks can be used to manipulate files and directories, archives, run programs and run the tools provided by the JDK.

2.6.7. Exercise Set

2.6.7.1. Multiple Choice

1. Common binaries and scripts will usually be in which directory?

    1. `src`
    2. `build`
    3. `lib`
    4. `bin`
    5. `doc`
2. Ant is:
    1. An insect with six legs.
    2. An integrated development environment.
    3. A java build tool.
3. XML:
    1. Focuses on how data is to be displayed.
    2. Is an extensible markup language.
    3. Consists of a pre-defined set of tags.
4. A valid XML declaration would be:
    1. `<xml version="1.0">`
    2. `<?xml standalone="true" ?>`
    3. `<?xml version="1.0" standalone="yes" encoding="UTF-8" ?>`
5. XML Elements may have attributes that describe the element.
    1. True
    2. False
6. An Ant build file contains a project element as the root element.
    1. True
    2. False
7. Project elements contain
    1. Targets
    2. Properties
    3. Targets and Properties
    4. Targets or Properties
    5. All the above
8. The `init` target will most likely use which task?
    1. `mkdir`
    2. `copy`
    3. `delete`
9. The copy task can be used to:
    1. Copy a single file to another file
    2. Copy a file to another directory
    3. Copy a directory to another directory
    4. Copy a set of files to another directory

5. All the above

10. You can set the classpath in an Ant build file.
    1. True
    2. False

2.6.7.2. Exercises

1. Read through the Ant manual to discover some more tasks and various options to the tasks discussed here.

2.6.7.3. Programming Exercises

1. Complete the following:
    1. Download Ant from the Apache Software Foundation and install it.
    2. Write some source code that is part of a package. Now write an Ant build file to compile, jar and javadoc the package.

2.7. Unit Testing

2.7.1. Jargon

Automated
Acting or operating in a manner essentially independent of external influence or control.

Constructor
An object instance is created by calling the constructor of a class. A constructor is method-like block of code that is called when the object is created. Typically constructors initialize data members and acquire resources the abject may require.

Framework
A Skeletal software component that performs functions required by a system and which is incorporated into the design of such systems.

Functionality
The capabilities or behaviors of a program, application, or system; the total set of its features.

Install
To load and configure a piece of software on a computer.

Logical Unit
A group of code blocks or statements that together form a logical cohesive whole.

Test Case
A set of conditions or variables under which a tester will determine if a

requirement upon an application is partially or fully satisfied. It may take many test cases to determine that a requirement is fully satisfied.

Test Driven Development

Test-driven development (TDD) is a programming technique heavily emphasized in Extreme Programming. Essentially the technique involves writing your tests first then implementing the code to make them pass.

Test Suite

A test suite is a set of related tests, usually pertaining to a group of features or software component.

Unit Test

A unit test is a method of testing the correctness of a particular module of source code.

2.7.2. Introduction

Testing code is an important aspect of software development as it ensures that the code is correct, does what is intended and does not contain any errors or bugs.

Unit testing allows the developer to test every logical unit of code. This has one considerable benefit, code can easily be changed without fear since the unit tests will quickly determine if the code changes have broken the code. We will look at unit testing in this chapter using the JUnit framework.

When combined with a source version control system, this is a very powerfu combination, since if the changes break the code, the changes can be reverted, but if the changes are good they can be included in the code base. This process is known as refactoring.

You can get JUnit @ junit.sourceforge.net (http://junit.sourceforge.net)

2.7.3. Test Driven Programming

In this section we will briefly look at where Unit Testing comes from, why it is important and how it fits into the big picture of software development. I hope you will find this section informative even though it is a brief overview.

2.7.3.1. Extreme Programming

eXtreme Programming commonly referred to as XP is a process for developing software that was created in order to provide for the trend in which many software development projects would overrun their schedule and budgets. Developing software is not an easy task and one requires a development methodology that takes numerous factors into account.

With XP, there is constant interaction with the client of the software, this means that the

questions the developers have are answered quickly, the client also gets to see the progress of the project. Frequent small releases keep the client satisfied.

There are twelve principles that XP focuses on, but for many the core principles are:

Simplicity
Simplicity is what the development team should strive for. Find the simplest working solution to the problem.

Unit Testing
Using unit testing, the development team can determine if their solution is a working solution. It is no good to have a solution that doesn't solve the problem, testing is used to ensure the code does what is intended.

Refactoring
Refactoring is the process of making changes to the code and its design in an effort to improve or simplify it. If you have a solution but it isn't the simplest solution you need to refactor it.

2.7.3.2. What is Unit Testing?

It is common sense that all code developed by a software developer should be tested, this is generally understood. We can further this and say that every logical unit of code should be tested. And it would be good if these tests are automated and even better if they can be incorporated into the build process. This is unit testing, automated testing of each logical unit of code.

Each class in object-oriented programming is a logical unit of code, we therefore need to test each and every class's code. A class consists of variables and methods. All variables should be declared as private and used by the methods of the class, any public or unused variables should be refactored. All private and protected methods should be used by the public methods of the class, if not the unused methods should be refactored. Given this, by testing the class's public methods it should be clear that we in effect test all the class's code.

Needless to say, a unit test tests the public methods of a class to verify that it is correct and does what is intended.

2.7.3.3. The Test, Code, Design Cycle

Traditional software development practices focus on first designing the code, then writing the code and finally testing it. in XP, the opposite is true, we first write the test, then write the code and finally we consider its design and refactor if necessary.

Writing the test first may seem like a ridiculous idea especially for code that hasn't been written yet, but I urge you to give it a try. By writing the test first, you are carefully

considering what needs to go into the code, what functionality of the code should be tested.

This is known as test driven development. Unit testing in this case actually drives the functionality of the code and you end up writing only sufficient code to run the tests and no more. This means you don't write any unused code and all the code you write has a test for it.

Once the test is written you can then write the code to make the test compile and run. The first time you run the tests you may get errors, this is not a bad thing as it shows you how the code is constantly changing to get the tests to run and succeed.

After the tests are run you should first attempt to fix any failures if there are any and finaly look at the code and consider if it needs to be refactored. While looking at the code ask yourself: Is it the simplest working solution? If not, you need to refactor it.

2.7.4. Writing Unit Tests

In this section we'll look at the code that goes into a unit test and what is involved in actually writing the tests. In the next section I'll show you how to run the tests. As an example consider writing a unit test for the `java.util.Vector` class. Although the code for Vector has already been written, instead of the test first approach, we can still use it as an example.

2.7.4.1. Test Cases

For every class we will write a test case. In our example we will only write one test case since we plan to only test the Vector class. It should come as no surprise that a test case is a class that inherits from (subclasses) `junit.framework.TestCase`.

Every test case has a single constructor that takes a string parameter and passes this to the constructor of the `junit.framework.TestCase` class using super constructing. Assume the test we're writing has a class name `VectorTest`, the constructor is as follows:

```
public VectorTest(String name) {
    super(name);
}
```

Every test case also has a `suite` method that is used by the test framework to set up the test suite and run the tests. In our test case this method is as follows:

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

If you need to initialize resources or free resources in the test case you can override the `setUp` and `tearDown` methods of the `junit.framework.TestCase` class. In our test case we want to init alize a variable named `vector` and set it to `null` when we're finished with it, as follows:

```
protected void setUp() {
    vector = new Vector();
}

protected void tearDown() {
    vector = null;
}
```

2.7.4.2. The Test Methods

Finally we get to the actual testing of the class. Every test method has a method signature: `public void` and the name must begin with `test`. The names should also be informative since they are used in the output when a test fails. Consider a method to test adding an object to the vector:

```
public void testAddObject() {
    Integer i = new Integer(7);
    vector.add(i);
    assertTrue(vector.size() == 1);
}
```

Note the last line of this method body contains an `assertTrue` statement. This is used by the test framework to verify that the expression is indeed true. If it is not true, the assertion fails causing the test to fail. Consider another test, to remove an object from the vector and verify that this is the case:

```
public void testRemoveObject() {
    Integer j = new Integer(3);
    vector.add(j);
    vector.remove(j);
    assertTrue(!vector.contains(j));
}
```

We can continue to add test methods to test every public method of the Vector class. It is important to note that the setUp and tearDown methods are called for each individual test, so every test method starts with an empty vector. This implies that the `Integer` object `i` added in the `testAddObject` cannot be removed in the `testRemoveObject` method, which is why we add a new object before removing it.

2.7.5. Running JUnit With Ant

One of the aspects of unit testing is that the test should be automated and preferably part of the build process. To get this we will incorporate JUnit into Ant and have ant run the tests with our build.

2.7.5.1. Install JUnit in Ant

In order to run JUnit with ant you need to install JUnit in Ant. This is easily done. First extract the JUnit archive to a location of your choice. The resulting folder will be named `junit`*version* and will contain a jar file named `junit.jar`.

Copy this jar file (junit.jar) to the `lib` directory in your ant directory. The ant directory is the directory you specified as `ANT_HOME` when you installed ant.

2.7.5.2. The Test Targets

How you structure your source code was discussed in the previous chapter, but we need to revise this structure a bit to incorporate unit tests into the code base. We want to keep our code base clean so the last thing we want to do is to mix the actual source code of the project with the unit test source code.

A good way to achieve this is to have a `java` and `test` directory within the `src` directory. The java directory will contain our java code and the test directory will contain our unit tests. I also like to keep all the tests in a package named `test` but some people prefer to use their inverted domain name as discussed in the chapter on java packages: `com.mycompany.myproject.test`

The structure can be seen in the image below.

Directory Structure

Compile Tests Target

Once you have the correct directory structure its time to add the required targets to the ant build file. The first thing we need to do is compile our tests before we can run them. Our tests also depend on the application's code being compiled so our target will depend on the `compile` target. It is important to add the application's compiled classes and the junit jar to the classpath:

```
<target name="compile.tests" depends="compile"
        description="o Compile unit test cases.">
    <mkdir dir="build"/>
```

```
    <javac srcdir="src/test"
        destdir="build"
        debug="true"
        deprecation="false"
        optimize="false">
        <classpath>
            <pathelement location="build"/>
            <pathelement location="lib/junit.jar"/>
        </classpath>
    </javac>
</target>
```

Run Tests Target

To run the tests we use the `junit` task with a nested `batchtest` task that runs all files with a "Test" in the filename. For more information on the tasks and their options see the Ant manual.

```
<target name="test" depends="compile.tests"
        description="o Run all unit test cases.">
    <junit printsummary="no" haltonfailure="yes" showoutput="yes">
        <classpath>
            <pathelement location="build"/>
            <pathelement location="lib/junit.jar"/>
        </classpath>
        <formatter type="brief" />
        <batchtest fork="yes" todir="build">
            <fileset dir="src/test">
                <include name="**/*Test*.java"/>
            </fileset>
            <formatter type="brief" usefile="false" />
        </batchtest>
    </junit>
</target>
```

2.7.5.3. Running The Tests

To run the test you simply type ant test in the command line from within the project root directory. This will compile the unit tests and run them.

In our example application, the output is:

```
Buildfile: F:\chapter7\build.xml
compile:
     [echo] Compiled.
compile.tests:
test:
    [junit] Testsuite: test.VectorTest
    [junit] Tests run: 2, Failures: 0, Errors: 0, Time elapsed: 0 sec
```

```
BUILD SUCCESSFUL
Total time: 1 second
```

You can get the source code from this chapter here: chapter7.zip

2.7.6. Summary

Unit Testing provides an excellent way in which to test all the code in your application. It is easily incorporated into the build process using Apache Ant and facilitates refactoring of the code. JUnit is the unit test framework for the java platform.

2.7.7. Exercise Set

2.7.7.1. Multiple Choice

1. Testing code ensures
    1. The code is correct
    2. Does what is intended
    3. Does not contain any errors
    4. All the above
    5. None of the above
2. Unit testing allows the developer to test every logical unit of code
    1. True
    2. False
3. The development team should strive for:
    1. Simplicity
    2. Refactoring
    3. Unit Testing
4. Unit Tests should be
    1. Automated
    2. Integrated into the build process
    3. Run by Ant
    4. All the above
    5. None of the above
5. Testing a class's public methods is sufficient in testing the class.
    1. True
    2. False
6. The process is
    1. Design - Code - Test
    2. Code - Test - Design

3. Test - Code - Design
4. Code - Design - Test
5. Design - Code - Test

7. A Test Case should subclass `junit.framework.TestCase`.
   1. True
   2. False
8. The suite method returns
   1. `junit.framework.Suite`
   2. `junit.framework.Test`
   3. `junit.framework.TestSuite`
9. Initialising resourecs is done in the ... method?
   1. constrcutor
   2. `setUp`
   3. `tearDown`
10. `assertTrue` determines if an expression is true.
    1. True
    2. False

2.7.7.2. Exercises

1. The `assertTrue` method is one of numerous assertion methods in the JUnit framework. Find out what the others are.

2.7.7.3. Programming Exercises

1. Complete the following:
   1. Complete the `VectorTest` class. You can find out what other public methods the `Vector` class has by checking the API.
   2. Add another test case to the example in this chapter to test the `java.util.Properties` class.
   3. Write some unit tests for the project you are currently working on.
   4. On your next project, try the test - code - design cycle.

2.8. Performing Logging

2.8.1. Jargon

Component
A piece of software with a clear function that can be isolated and replaced by another component with equivalent functionality.
Configuration

The way that a program or computer is set up, various settings are used in the configuration.

Debugging
The process of locating errors in source code whether logical or syntactic and fixing them often through the use of a debugger.

Fatal Error
An error that causes a program to stop executing. See Error and Application Failure.

File
A block of information in the form of bytes, stored together on a computer or external digital storage medium, and given a name. A file may be a program, a document, a database, or some other collection of bytes.

Framework
A Skeletal software component that performs functions required by a system and which is incorporated into the design of such systems.

Logger
A component with sole responsibility of handling the logging operations in a logging framework.

Logging
The process of storing information about events that occurred in an application, used for debugging purposes.

Message
Any information sent as a component interacts with another.

Output
Information that has been manipulated by the central processing unit (CPU) of the computer, and displayed either on the video monitor or rendered on paper or film as hard copy, or saved on disk in a digital format.

2.8.2. Introduction

Logging is a useful technique for debugging medium to large scale applications. It is also useful when traditional debuggers are ineffective.

Typically most novice programmers simply use `System.out.println` to check that a program reaches a section of code or to print out the value of a variable. This is fine for a small application or a snippet of code, but is not the preferred way for medium and large scale applications, or for professional commercial applications.

A logging framework is used to replace all the `System.out.println` statements and to send the output to a different target, instead of the console, we can print that information to a file or any other destination imaginable.

You can get Log4J @ logging.apache.org (http://logging.apache.org)

2.8.3. What Is Logging?

2.8.3.1. Logging As Debugging

Logging is a technique that is used to assist in the debugging process and may be the only way to perform debugging in cases where debuggers are not available or useful such as distributed applications. Logging equips the developer with a detailed context for application failures and should not be confused with testing. Testing provides quality assurance and confidence in the application, logging and testing are complimentary.

Many people may argue that log statements pollute source code, decrease the legibility of the code and increase its size. Another argument against logging is the fact that it impacts performance of the application. This means that the speed of a logging framework is particularly important.

Using Log4J, it is possible to enable or disable logging at run-time without having to recompile the application binary, the framework is designed so that the log statements can remain in the code without incurring a heavy performance cost.

2.8.3.2. How Does Logging Work?

The application developer decides to generate a message that needs to be logged whenever something important in the application occurs, for example: the click of a button. The developer then decides where this message should go, its *target*, typically a file, and the developer may control the message's *format*.

An important aspect is the ability to assign different priority levels to the message and further to only log messages with a certain priority level. This affords the developer a large amount of control over how logging gets performed, for example, messages of a certain priority can be logged to a file and other messages with a different priority to another *target*.

The basic idea is that a `Logger` is responsible for actually performing the logging in an application by handling the log operations, the `Appender` is responsible for appending the message to its *target* and controlling that output, and the `Layout` is responsible for formatting the logging message.

2.8.4. Five Core Components

Log4J makes use of five core components and in order to understand how Log4J works we need to understand how these components work and interact with one another.

These components are `Level`s, `LoggingEvent`s, `Layout`s, `Appender`s and `Logger`s. We will look at how each of them works and how they cooperate to provide the core functionality of the logging framework.

2.8.4.1. Levels

Every event that is logged has an assigned priority level. Fatal errors would naturally be more severe than a simple warning and thus would have a higher priority level. In this regard, each level has an integer value assigned to it so that it can be compared to other levels.

There are five levels by default, namely (lowest priority to highest): `DEBUG`, `INFO`, `WARN`, `ERROR` and `FATAL`. Users can quite easily add more levels as they see fit. Using these levels it is possible to only log events of a certain priority level, say for example: error, which results in only that level and any level with a higher priority being logged. This means that all error and fatal messages will be logged, since fatal has a higher priority than error while all debug, info and warn messages will be ignored.

Two special priority levels exist, these are `ALL` and `OFF`, all is used to log all messages of any priority level, even user defined ones while off is used to ignore all messages regardless of their priority level. In essence, during debugging, all should be used while production code should use off.

2.8.4.2. Logging Events

Logging events represent the actual event that gets logged and an instance is created whenever the Log4J suite makes an affirmative decision to log something. Logging events contain vital information such as the time it was created, the message itself, any exception if an exception is raised and an associated priority level.

The `LoggingEvent` is passed around to the various Log4J components that use it and the information contained in the logging event is used to format and print the information to the desired target so that it can be analyzed at a later stage.

LoggingEvents are not usually used by application developers but rather only internally by Log4J.

2.8.4.3. Layouts

Layouts provide the functionality of formatting logging events into any number of appropriate formats. The layout takes the `LoggingEvent` and formats it appropriately into a string so that it can be written to the target by the Appender. `Layout` is abstract and subclassed by a number of concrete layout classes that provide an implementation of the

`format(LoggingEvent)` method.

A layout may also provide a header and footer as is the case in an HTML layout where an HTML document usually contains a header and a footer. These are returned by the `getHeader` and `getFooter` methods respectively. The default implementation returns empty strings for these. A layout can also choose to ignore any exceptions that are contained in the `LoggingEvent`, this can be determined from the `ignoresException` method which returns `true` by default. Naturally any subclasses that handle exceptions will override this method to return `false` such as in the XML layout.

The layouts provided by the Log4J package are:

HTMLLayout

`HTMLLayout` prints out log events in an HTML table. It returns appropriate header and footers to complete the HTML document.

Example output:

```
<tr>
<td>0</td>
<td title="main thread">main</td>
<td title="Level"><font color="#339933">DEBUG</font></td>
<td title="root category">root</td>
<td title="Message">Debug Message</td>
</tr>
```

PatternLayout

`PatternLayout` formats a logging event according to a conversion pattern that is similar to the conversion pattern of the `sprintf` function in C. The conversion pattern consists of literal text and conversion specifiers.

Each conversion specifier starts with a percent sign (%) and is followed by optional format modifiers and a conversion character. The conversion character specifies the type of data, e.g. category, priority, date, thread name. The format modifiers control such things as field width, padding, left and right justification. The following is a simple example: **`%d [%5p] %m%n`**

Example output:

```
12/20/2005 4:53:33 PM [DEBUG] Debug message
```

SimpleLayout

`SimpleLayout` provides basic formatting, the output from this layout is the priority level of the log message followed by a dash (-) and the log message itself. It returns the default implementation for all other methods defined in the `Layout` class, thus empty header and footer strings and it ignores exceptions.

Example output:

```
DEBUG - Button Clicked
```

TTCLayout

`TTCCLayout` consists of the time, thread, priority level and diagnostic context information.

Example output:

```
262 [main] DEBUG org.apache.log4j.examples.SortAlgo.OUTER i=1 - Outer loop.
```

XMLLayout

The output of the `XMLLayout` consists of a series of `log4j:event` elements as defined in the `log4j.dtd`. It does not output a complete well-formed XML file. The output is designed to be included as an external entity in a separate file to form a correct XML file.

Example output:

```
<log4j:event logger="root"
   timestamp="1135964385140"
   level="DEBUG"
   thread="main">
<log4j:message>
   <![CDATA[Debug Message]]>
</log4j:message>
</log4j:event>
```

2.8.4.4. Appenders

An `Appender` is responsible for "appending" a log message to a specific *target*. Each appender specifies its own `Layout` (the formatting it will use) as well as a threshold which determines the level priority that it will log. Some appenders inherently require a layout but others might not.

The two most commonly used appenders are `ConsoleAppender` and `FileAppender`. Typically you will simply create an Appender instance and then add it to the logger instance you are using. You can add multiple appenders to a logger instance so that it may send log messages to multiple *target*s.

2.8.4.5. Loggers

Loggers are named entities with their names being case sensistive. Loggers are part of a hierarchy following a simple rule: A logger may be an ancestor of another if its name followed by a period (`.`) is the prefix of the descendant logger. A logger is the ancestor of a descendant if there are no ancestors between it and the descendant logger. As an example consider the name: `com.mycompany.myproject`. "`com`" is a **parent** of `com.mycompany` and an **ancestor** of `com.myproject.mycompany`.

A root logger exists and obeys two fundamental rules: 1) it always exists and 2) it cannot be retrieved by name. In order to access the root logger you use the static method `Logger.getRootLogger`. All other loggers are retrieved by name using `Logger.getLogger(`*name*`)`.

Loggers may be assigned a priority level, if a given logger is not assigned a priority level then it inherits from the closest ancestor with a priority level. A log request with a given assigned level say ***S*** in a logger with level ***T*** is enabled if and only if S is greater or equal to T: ***S***>=***T***.

2.8.5. Writing Log Statements

2.8.5.1. Installing and Using Log4J

Log4J is distributed in archive format using either zip or gzip compression. Once you have downloaded the archive of your choice you should extract it to the location of your choice. This will result in a folder named `logging-log4j-`*version*.

You will find a jar file named `log4j-`*version*`.jar`in the `dist/lib` directory in the log4j directory. You need to copy this archive to your project's lib directory and update your classpath as appropriate.

2.8.5.2. Configuring Log4J

If you try to run the following code without configuring Log4J you will get an error.

```
Logger logger = Logger.getRootLogger();
logger.debug("Debug Message");
```

The error will read:

```
log4j:WARN No appenders could be found for logger (root).
log4j:WARN Please initialize the log4j system properly.
```

In order to prevent this you need to configure Log4J properly.

Hard Coded Configuration

You can perform a simple configuration on the root logger manually in the code of your application. Suppose you only wish to use the root logger using a simple layout and a file appender, your application's main method will be as follows:

```
Layout layout = new SimpleLayout();
FileAppender appender= new FileAppender(layout, "app.log");
Logger logger = Logger.getRootLogger();
logger.addAppender(appender);
```

Basic Configuration

You can use the BasicConfigurator to do a basic configuration which logs to the ConsoleAppender using TTCCLayout.

```
BasicConfigurator.configure();
Logger logger = Logger.getRootLogger();
logger.debug("Debug Message");
```

Output on the console will be:

```
0 [main] DEBUG root  - Debug Message
```

Properties File Configuration

Log4J can be configured through a java properties file. The file has the same format as java properties, that is `name=value` pairs, one on a line. The file should ideally be named log4j.properties and on the classpath, but you can name it anything and then load it in your application.

Controlling Log4J's Internal Logging

Log4J's internal logging is controlled through the log4j.debug directive. Those who are interested to see what Log4J is doing can set this value to `true`. This results in a file named `log4j.log`. If you do not wish to have internal logging then set this value to `false`.

```
log4j.debug=false
```

Configuring the Root Logger

The next thing to do is to set the root logger by specifying the priority level and an appender's name. The format is `log4j.rootLogger=LEVEL, AppenderName`. The **LEVEL** is any one of `DEBUG`, `INFO`, `WARN`, `ERROR` or `FATAL` and the **AppenderName** is any alpha-numeric string of no more than 255 characters. As an example, consider setting the root logger to debug and using a file appender:

```
log4j.rootLogger=DEBUG, fileAppender
```

Configuring the Appender

The first step is to tell Log4J which appender you wish to use by giving the appender class's name.

```
log4j.appender.fileAppender=org.apache.log4j.FileAppender
```

Some appender's require additional information, these are set using directives that match the appender's setter methods. The FileAppender class has a setFile method that can be used as follows:

```
log4j.appender.fileAppender.File=app.log
```

Setting the Layout

Some appenders require a layout. This is easily set using the layout directive and specifying the layout class to use.

```
log4j.appender.fileAppender.layout=org.apache.log4j.SimpleLayout
```

As a complete example, consider the following configuration file:

```
log4j.rootLogger=DEBUG, consoleAppender
log4j.appender.consoleAppender=org.apache.log4j.ConsoleAppender
log4j.appender.consoleAppender.layout=org.apache.log4j.SimpleLayout

log4j.logger.test.app=ERROR, fileAppender
log4j.appender.fileAppender=org.apache.log4j.FileAppender
log4j.appender.fileAppender.File=app.log
log4j.appender.fileAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.fileAppender.layout.ConversionPattern=%d [%-5p] %c - %m%n
```

Loading the File in Your Application

To load the properties from your application you would simply place the properties file on

the classpath and Log4J should find it. If it isn't in the classpath you can manually load it from your application.

You can simply pass the filename to the `PropertyConfigurator`'s `configure` method.

```
PropertyConfigurator.configure("log4j.properties");
```

Or you can simply create a `java.util.Properties` object and load the properties into that then pass the `Properties` instance to the `PropertyConfigurator`'s `configure` method.

```
Properties props = new Properties();
props.load(new FileInputStream("log4j.properties"));
PropertyConfigurator.configure(props);
```

2.8.5.3. Log Statements

Using the given properties file from the example above, we can write log statements to one of the two loggers, either the root logger or the application logger which is set to the ERROR level.

```
Logger rootLogger = Logger.getRootLogger();
rootLogger.debug("Debug message.");

Logger appLogger = Logger.getLogger("test.app");
appLogger.error("An error message.");
```

This results in the following output on the console:

```
DEBUG - Debug message.
ERROR - An error message.
```

and the following in the `app.log` file.

```
2006-01-02 10:11:20,000 [ERROR] test.app - An error message.
```

2.8.6. Summary

Log4J is a powerful and flexible logging framework for the Java platform. Logging is a technique that enables debugging and is especially useful in situations where traditional debuggers are ineffective like distributed systems.

2.8.7. Exercise Set

2.8.7.1. Multiple Choice

1. Logging is a technique for debugging.
    1. True
    2. False
2. Logging is useful when debuggers are ineffective such as distributed applications.
    1. True
    2. False
3. Arguments against logging include:
    1. Log statements pollute source code
    2. Decreases the legibility of the code
    3. Increase the size of the code base
    4. Impacts performance of the application
    5. All the above
4. The target of a logging message is
    1. The format of the message
    2. Its destination
    3. A file
5. Which is not a default prioirty level?
    1. ALL
    2. DEBUG
    3. ERROR
    4. EXCEPTION
    5. WARN
6. Which is not a core components
    1. Level
    2. LoggingEvent
    3. Layout
    4. WriterAppender
    5. Logger
7. To install Log4J
    1. Extract the archive
    2. Include the jar file in your project's code base
    3. Update your classpath
    4. All the above
    5. None of the above

8. Log4J is configured automatically.
    1. True
    2. False
9. Log4J can be configured through
    1. Hard Coded Configuration
    2. Basic Configuration
    3. Properties File Configuration
    4. All the above
    5. None of the above
10. To turn of log4j's internal loggin you would use
    1. log4j.debug=true
    2. log4j.debug=false

2.8.7.2. Exercises

1. Read through the Log4J short manual to discover more wbou the different components in the Log4J framework.

2.8.7.3. Programming Exercises

1. Complete the following:
    1. Add logging to one of your projects using the Log4J framework.

2.9. XML Processing

2.9.1. Jargon

Child Node
A child node or descendant node is a node in a tree data structure that is linked to by a parent node.

Element
A block in a markup document like a tag in an HTML document.

Parse
Parsing is the process of splitting up a continuous stream of characters (read from a file or keyboard input, for example) into meaningful tokens, and then building a parse tree from those tokens.

Serialize
Encode a data structure as a sequence of bytes.

Stream
A continuous flow of data, usually digitally encoded, designed to be processed sequentially. Also called a bitstream.

String
A group or sequence of characters.
Stylesheet
An ASCII text document that is attached to an SGML or XML encoded document and that contains instructions to specify the formatting and display of the encoded document, or to transform it into another format.
Template
A set of pre-designed formats for text and graphics on which new pages and webs can be based.
Text
Data consisting of a sequence of characters, as opposed to binary numbers, images, graphics commands, executable programs, and the like.
Transformation
Transformation is a form of conversion in which a file is converted into a file format with a comparable structure (eg from XML to XML, or from SGML to HTML). Usually, this form of conversion can be carried out very well.

2.9.2. Introduction

We were briefly introduced to XML in chapter 6. In this chapter we are going to look at working with XML in our programs, in effect we will be writing XML (creating), reading XML (parsing) and performing transformations on XML (from XML to HTML).

In this chapter we will be looking at two tools instead of one like previous chapters. We will first look at the Xerces XML parser and then look at Xalan, a tool that performs XML transformations.

You can get Xerces and Xalan @ xml.apache.org (http://xml.apache.org) . Note that we are using Xerces-J for Java.

2.9.3. Writing XML

The first thing we may wish to do is to actually create an XML document. In this section we will look at some ways in which to accomplish this.

2.9.3.1. Three Ways To Write XML

There are actually three ways in which to create an XML document. The first is the easiest, we simply type an XML document into our favourite text editor and then save the document with the `.xml` extension. The second way is to actually hardcode the XML into our program and the last method is to actually create a Document object and then serialize it.

Consider the example given in chapter 6 of a memo. It is printed here again:

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

<!DOCTYPE memo
    SYSTEM "memo.dtd">

<memo>
    <to>John</to>
    <subject>Re: Progress Meeting</subject>
    <message>Don't forget that we must book the boardroom
        well in advance or we'll have to have the meeting in
        one of our offices which will turn out to be
        <emphasis>very cramped</emphasis>.
        See you later!
    </message>
    <from>Trevor</from>
</memo>
```

You can quite easily copy the text, paste it into your text editor and save it. This is the easiest way in which to create an XML document. But what if we want to create a document from within our program?

2.9.3.2. Programmatically

Programmatically creating an XML document is almost as easy as creating one in a text editor. For starters we can simply print out the document to the console:

```
System.out.println("<?xml version=\"1.0\" ?>");
System.out.println("");
System.out.println("<!DOCTYPE memo SYSTEM \"memo.dtd\">");
System.out.println("");
System.out.println("<memo>");
System.out.println("    <to>John</to>");
System.out.println("    <subject>Re: Progress Meeting</subject>");
System.out.println("    <message>Don't forget that we must book the
boardroom");
System.out.println("        well in advance or we'll have to have the
meeting in");
System.out.println("        one of our offices which will turn out to be");
System.out.println("        <emphasis>very cramped</emphasis>.");
System.out.println("        See you later!");
System.out.println("    </message>");
System.out.println("    <from>Trevor</from>");
System.out.println("</memo>");
```

This will print out the document to the console. What if we wish to send that to a file? This is done quite easily using Java's IO system.

```
try {
    PrintWriter out = new PrintWriter(new FileOutputStream("memo.xml"));
    out.println("<?xml version=\"1.0\" ?>");
    ...
    out.println("</memo>");
    out.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

Many people are against hard coding the document, instead the correct way to do this is to create the document object.

2.9.3.3. Creating The Document Object

The Xerces Parser

The Xerces parser is distributed in a compressed archive. Simply extract the archive to the location of your choice. Within the contents of the archive you will find two jar files that are required, namely: `xercesImpl.jar` and `xml-apis.jar`. You need to add both to your classpath.

Creating The Document

The first thing we need to do is to actually create the document object through the use of a document builder. We first get a `javax.xml.parsers.DocumentBuilderFactory` instance and use it to create a `javax.xml.parsers.DocumentBuilder` which in turn is used to create the `org.w3c.dom.Document` instance.

```
DocumentBuilderFactory builderFactory =
    DocumentBuilderFactory.newInstance();
DocumentBuilder builder = builderFactory.newDocumentBuilder();
Document document = builder.newDocument();
```

Creating The Root Element

Next we create the root element of the document. This is done through the `createElement` method of the `document` instance which returns a `org.w3c.dom.Element`.

```
Element root = document.createElement("memo");
```

Creating Additional Elements

We now create the `to` and `subject` elements. This is again done using the `createElement` method of the `document` instance. In each case we need to add a sub element which is a text node. Text elements contain the actual data in the tag.

```
Element item = document.createElement("to");
item.appendChild(document.createTextNode("John"));
root.appendChild(item);

item = document.createElement("subject");
item.appendChild(document.createTextNode("Re: Progress Meeting"));
root.appendChild(item);
```

Creating The Message Element

The `message` element is interesting in that it contains both text and an `emphasis` subelement. In this instance we have three child elements, the first text element, the `emphasis` element and the second text element.

```
item = document.createElement("message");
item.appendChild(document.createTextNode("Don\'t forget that we "
    + "must book the boardroom well in advance or we'll have "
    + "to have the meeting in one of our offices which will "
    + "turn out to be ") );
Element subitem = document.createElement("emphasis");
subitem.appendChild(document.createTextNode("very cramped"));
item.appendChild(subitem);
item.appendChild(document.createTextNode(". See you later!"));
root.appendChild(item);
```

Adding The Root Element

We add the `from` element and then add the root element to the document.

```
item = document.createElement("from");
item.appendChild(document.createTextNode("Trevor"));
root.appendChild(item);

document.appendChild(root);
```

Output The Document

In order to output the document we use an `org.apache.xml.serialize.XMLSerializer` instance which requires an `org.apache.xml.serialize.OutputFormat` instance and a writer object.

```
OutputFormat format = new OutputFormat(document);
PrintWriter  writer =
    new PrintWriter(new FileOutputStream("memo.xml"));
XMLSerializer serializer = new XMLSerializer(writer, format);
serializer.asDOMSerializer();
serializer.serialize(document.getDocumentElement());
```

You can get the complete source code for this chapter here.

2.9.4. Reading XML

In this section we will look at ways to read an xml document. This is typically known as parsing XML.

2.9.4.1. Two Ways To Read XML

When one wishes to parse an XML document one will typically use an XML Parser for this purpose. There are two ways of parsing an XML document, one is through the use of Simple API for XML (SAX) and the other is using the Document Object Model (DOM).

SAX is easy to implement and easy to understand. Its is also very quick to parse an XML Document. The way sax works is that it parses the document in a serial fashion, so you get each element as it is parsed. This means that once an element is parsed you cannot go back to it and you cannot read elements that occur later in the document in random access fashion.

DOM allows you to access any element in the document in random access fashion. You first parse the entire document and keep it in memory then you can read elements from it. DOM is a bit more difficult to implement and does not work well with extremely large documents. It is better suited to small documents like our memo. Keep the differences between SAX and DOM in mind as we explore them.

2.9.4.2. Simple API for XML (SAX)

We said that SAX parses the document in a serial fashion, one element at a time. How this works is as follows, as soon as an element is found an event occurs and any event handlers associated with that event are fired. Your program will typically inherit from one of the handler classes or implement the `Handler` interface which defines these event handlers.

We are going to use SAX to read in our memo document from a file named "`memo.xml`" and print out the memo. We first define our class SAXReader to implement ContentHandler and implement all the required methods. The method bodies are all empty.

```
public class SAXReader implements ContentHandler {
    public void setDocumentLocator(Locator arg0) {}
```

```
    public void startDocument() throws SAXException {}
    public void endDocument() throws SAXException {}
    public void startPrefixMapping(String arg0,
        String arg1) throws SAXException {}
    public void endPrefixMapping(String arg0)
        throws SAXException {}
    public void startElement(String uri, String local,
        String raw, Attributes attrs) throws SAXException {}
    public void endElement(String arg0, String arg1,
        String arg2) throws SAXException {}
    public void characters(char[] ch, int start,
        int length) throws SAXException {}
    public void ignorableWhitespace(char[] arg0,
        int arg1, int arg2) throws SAXException {}
    public void processingInstruction(String arg0,
        String arg1) throws SAXException {}
    public void skippedEntity(String arg0) throws SAXException {}
}
```

The class should also include a main method since this is our program. In the main method we create an `org.xml.sax.XMLReader` instance using the `org.xml.sax.helpers.XMLReaderFactory`. We set its content handler to our class and then parse the file.

```
public static void main(String[] args) {
    try {
        XMLReader parser =
            XMLReaderFactory.createXMLReader(
                                "org.apache.xerces.parsers.SAXParser");
        parser.setContentHandler(new SAXReader());
        parser.parse("memo.xml");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Finally we implement two of the methods in order to print out our memo. We print the name of each element in the `startElement` method and then we print out the character data for each text node.

```
public void startElement(String uri, String local,
        String raw, Attributes attrs) throws SAXException {
    System.out.print(local + ": ");
}

public void characters(char[] ch, int start,
        int length) throws SAXException {
    char[] array = new char[length];
    System.arraycopy(ch, start, array, 0, length);
    System.out.print(array);
```

```
}
```

When we run the program, we get the following output:

```
memo:
        to: John
        subject: Re: Progress Meeting
        message: Don't forget that we must book the boardroom
                 well in advance or we'll have to have the meeting in
                 one of our offices which will turn out to be
                 emphasis: very cramped.
                 See you later!
        from: Trevor
```

2.9.4.3. Document Object Model (DOM)

Using DOM, we first parse the entire document into memory before we can use it. Using the document means actually using the set of interfaces and classes in the DOM API. We will use these to print out the memo we parse in from file.

We can write a printNode method that is responsible for printing out the Node and then recursively calls the method to print each child node. This is fairly straight forward.

```
public static void printNode(Node node) {
    if (node == null) return;

    int type = node.getNodeType();
    if (type == Node.TEXT_NODE) {
        Text text = (Text)node;
        System.out.println(text.getWholeText());
    } else if (type == Node.ELEMENT_NODE) {
        Element element = (Element) node;
        System.out.print(element.getTagName() +  ": ");

        NodeList children = node.getChildNodes();
        for (int i = 0; i < children.getLength(); i++)
            printNode(children.item(i));
    }
}
```

The last thing we need to do is to define our program's main method. This follows a similar pattern to writing out the XML using DOM. We use a DOM builder to actually parse the XML. After the document is parsed, we call the `printNode` method using the document's root element.

```
public static void main(String[] args) {
    try {
        DocumentBuilderFactory builderFactory =
```

```
            DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = builderFactory.newDocumentBuilder();
        Document document = builder.parse("memo.xml");
        printNode(document.getDocumentElement());
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

2.9.5. Transforming XML

XML is designed to be a format for structuring data and not for presenting data. Countless other formats exist that focus on presenting data such as HTML. We may want to transform our XML into a format that is more capable of presenting the data.

In order to transform our XML we make use of another tool called eXtensible Stylesheet Laguage. Using XSL we are going to transform our XML memo into an HTML version. We will be using **Xalan** to do the transformation for us.

2.9.5.1. eXtensible Stylesheet Language (XSL)

An XSL stylesheet is a document that is defined in XML and describes the process of transforming an XMLdocument into another format. It does this by using templates, typically a template would match a specific element in the XML document and describe how to convert that XML element into the desired format.

Lets get started by defining our stylesheet document.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

</xsl:stylesheet>
```

As you can see we define an XML document using the XML declaration and then declare that this document is an xsl stylesheet using the `xsl:stylesheet` declaration. All XSL stylesheets are defined in this way.

The next thing we do is to define a template that matches the root element. The root element of the document is used to create the output document, in this case the HTML docuement, so in this template we define an HTML document. We also use the `xsl:apply-templates` instruction to continue processing.

```
<xsl:template match="/">
    <HTML>
```

```
        <HEAD>
            <TITLE>Memo</TITLE>
        </HEAD>
        <BODY>
            <H2>Memo</H2>
            <TABLE>
                <xsl:apply-templates/>
            </TABLE>
        </BODY>
    </HTML>
</xsl:template>
```

A number of people code their HTML in lowercase, I prefer to use uppercase in order to distinguish between the XML for the stylesheet and the HTML output.

The templates for the `to`, `from` and `subject` elements are given next.

```
<xsl:template match="to">
    <TR>
        <TD>To:</TD>
        <TD><xsl:value-of select="."/></TD>
    </TR>
</xsl:template>

<xsl:template match="from">
    <TR>
        <TD>From:</TD>
        <TD><xsl:value-of select="."/></TD>
    </TR>
</xsl:template>

<xsl:template match="subject">
    <TR>
        <TD>Subject:</TD>
        <TD><xsl:value-of select="."/></TD>
    </TR>
</xsl:template>
```

The `xsl:value-of select="."` instruction is used to print the actual text content of the XML element. Because the message element does not contain only text elements but other elements, namely the `emphasis` element, we cannot use this instruction. Instead we tell the transformer to continue processing which prints the text and processes other elements.

```
<xsl:template match="message">
    <TR>
        <TD>Message:</TD>
        <TD><xsl:apply-templates/></TD>
    </TR>
</xsl:template>
```

The last template is for the emphasis element:

```
<xsl:template match="emphasis">
    <EM><xsl:value-of select="."/></EM>
</xsl:template>
```

2.9.5.2. Performing the Transformation

Xalan is distributed as a compressed archive, once you have uncompressed the archive to the location of your choice you will see that it contains a number of jar files, You will need all these jar files on your classpath.

We use a `javax.xml.transform.Transformer` to do the transformation for us. We need to create an instance using a `javax.xml.transform.TransformerFactory`. We load the XML and XSL as `javax.xml.transform.stream.StreamSource` objects and write the output to a `javax.xml.transform.stream.StreamResult`.

```
TransformerFactory factory = TransformerFactory.newInstance();
StreamSource stylesheet = new StreamSource("xml2html.xsl");
Transformer transformer = factory.newTransformer(stylesheet);
StreamSource source = new StreamSource("memo.xml");
StreamResult result =
    new StreamResult(new FileOutputStream("memo.html"));
transformer.transform(source, result);
```

After running the program we end up with the file named "`memo.html`" which is an HTML version of the memo.

2.9.6. Summary

XML is widely used for a number of purposes since it is easily read and understood by both humans and computers. XML can be written, read and transformed into different formats. DOM can be used to write and read XML while SAX provides a method for reading XML. XSL is used to transform XML.

2.9.7. Exercise Set

2.9.7.1. Multiple Choice

1. XML stands for eXtensible Markup Language
    1. True
    2. False
2. Possible ways to write XML are:

    1. Manually with a text editor.
    2. Programmtically: Hard-Coded
    3. Programmtically: Using a Document Object
    4. All the above
    5. None of the above
3. When writing to a file you will use
    1. `java.io.OutputStream`
    2. `java.io.Writer`
    3. `java.io.PrintWriter`
    4. `java.io.FileOutputStream`
    5. 1 and 2
    6. 3 and 4
4. To get a document builder you need to
    1. Create a builder factory and get a document builder instance.
    2. Create a document builder instance.
    3. Get a new instance of a builder factory and get a new document builder from it.
5. Every document contains a single root element.
    1. True
    2. False
6. To create a new element:
    1. Create a new instance using a constructor.
    2. Use the `createElement` method on a document object.
7. To output a document you use an XMLSerializer.
    1. True
    2. False
8. Possible ways to read XML are:
    1. Programmatically
    2. Using SAX
    3. Using DOM
    4. None of the above
    5. 2 and 3
9. For a large document, you would parse it using
    1. SAX
    2. DOM
    3. Neither
10. What XSL instruction is used to retrieve the text of an xml element?
    1. <xsl:template match="to">
    2. <xsl:value-of select="."/>

3. <xsl:apply-templates/>

2.9.7.2. Exercises

1. Find out about the `DefaultHandler` and how it differs from `ContentHandler`.

2.9.7.3. Programming Exercises

1. Complete the following:
   1. Write a program to generate a memo in xml format and save it to a file.
   2. Now write a program to display the contents of this xml memo.
   3. Write a program to transform the memo into HTML and raw text.
2. Implement a SAX parser using `DefaultHandler` instead of `ContentHandler`.

3. Appendices

3.1. Appendix A: ASCII Table

Value	Character	Value	Character	Value	Character	Value	Character
0		64	@	128	#	192	À
1		65	A	129	#	193	Á
2		66	B	130	#	194	Â
3		67	C	131	#	195	Ã
4		68	D	132	#	196	Ä
5		69	E	133	#	197	Å
6		70	F	134	#	198	Æ
7		71	G	135	#	199	Ç
8		72	H	136	#	200	È
9		73	I	137	#	201	É
10		74	J	138	#	202	Ê
11		75	K	139	#	203	Ë
12		76	L	140	#	204	Ì
13		77	M	141	#	205	Í
14		78	N	142	#	206	Î

15		79	O	143	#	207	Ï
16		80	P	144	#	208	Ð
17		81	Q	145	#	209	Ñ
18		82	R	146	#	210	Ò
19		83	S	147	#	211	Ó
20		84	T	148	#	212	Ô
21		85	U	149	#	213	Õ
22		86	V	150	#	214	Ö
23		87	W	151	#	215	×
24		88	X	152	#	216	Ø
25		89	Y	153	#	217	Ù
26		90	Z	154	#	218	Ú
27		91	[	155	#	219	Û
28		92	\	156	#	220	Ü
29		93	]	157	#	221	Ý
30		94	^	158	#	222	Þ
31		95	_	159	#	223	ß
32		96	`	160		224	à
33	!	97	a	161	¡	225	á
34	"	98	b	162	¢	226	â
35	#	99	c	163	£	227	ã
36	$	100	d	164	¤	228	ä
37	%	101	e	165	¥	229	å
38	&	102	f	166	¦	230	æ
39	'	103	g	167	§	231	ç
40	(	104	h	168	¨	232	è
41	)	105	i	169	©	233	é

42	*	106	j	170	ª	234	ê
43	+	107	k	171	«	235	ë
44	,	108	l	172	¬	236	ì
45	-	109	m	173	-	237	í
46	.	110	n	174	®	238	î
47	/	111	o	175	¯	239	ï
48	0	112	p	176	°	240	ð
49	1	113	q	177	±	241	ñ
50	2	114	r	178	²	242	ò
51	3	115	s	179	³	243	ó
52	4	116	t	180	´	244	ô
53	5	117	u	181	µ	245	õ
54	6	118	v	182	¶	246	ö
55	7	119	w	183	·	247	÷
56	8	120	x	184	¸	248	ø
57	9	121	y	185	¹	249	ù
58	:	122	z	186	º	250	ú
59	;	123	{	187	»	251	û
60	<	124	|	188	¼	252	ü
61	=	125	}	189	½	253	ý
62	>	126	~	190	¾	254	þ
63	?	127	#	191	¿	255	ÿ

Table 1: ASCII Character Values

3.2. Appendix B: Answers To Exercises

3.2.1. Chapter 1

3.2.1.1. Multiple Choice

1. 5. All of the above
2. 5. All of the above
3. 3. Running the installation binary.
4. 5. All of the above.
5. 4. 1 and 2.

3.2.1.2. Programming Exercises

1.
```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Trevor Miller");
    }
}
```

2.
```
public static void main(String[] args) {
    for (int i = 0; i < 20; i++) {
            System.out.println(Math.pow(2, i+1));
    }
}
```

3.
```
public static void main(String[] args) {
    for (int i = 0; i < 100; i++) {
        System.out.println(i + "C is equal to " + ((1.8*i)+32) + "F");
    }
}
```

4.
```
public static void main(String[] args) {
    int x = 1;
    int y = 1;
    for (int i = 0; i < 20; i++) {
        System.out.print(x + " " + y + " ");
        x = x+y;
        y = x+y;
    }
}
```

5.
```
public static void main(String[] args) {
    try {
        BufferedReader reader
            = new BufferedReader(new InputStreamReader(System.in));
        String in = reader.readLine();
        double d = Double.parseDouble(in);
        System.out.println(Math.sqrt(d));
    } catch (Exception e) {
        e.printStackTrace();
```

```
    }
}
```

3.2.2. Chapter 2

3.2.2.1. Multiple Choice

1. 4. Java
2. 3. Open Source Software
3. 5. Eclipse has JUnit integrated into it.
4. 5. All of the above.
5. 4. 1 and 2.

3.2.2.2. Programming Exercises

1.
```
public static void main(String[] args) {
    try {
        // Initial loan amount
        double A = Double.parseDouble(args[0]);

        // interest rate per annum
        double r = Double.parseDouble(args[1]);

        // Payment per period
        double P = Double.parseDouble(args[2]);

        // Current period in months
        int n = Integer.parseInt(args[3]);

        double i = (r / 12)/100;
        System.out.println("i = " + i);

        double amount_left = A * Math.pow((1+i),n)
            - (P / i) * (Math.pow((1+i),n) - 1);
        System.out.println(amount_left);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

2.
```
public static void main(String[] args) {
    try {
        double total = 0.0;
        int num = 0;
        BufferedReader reader
            = new BufferedReader(new InputStreamReader(System.in));
        String in = reader.readLine();
        double d = Double.parseDouble(in);
        while (d != 0) {
```

```
            total += d;
            in = reader.readLine();
            d = Double.parseDouble(in);
            num++;
        }
        System.out.println(total/num);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

3. Similar to above

3.2.3. Chapter 3

3.2.3.1. Multiple Choice

1. 2. Resolve naming conflicts among programmers.
2. 1. Finding the classes you want is difficult.
3. 3. package za.co.nephila.maculata
4. 3. import za.co.nephila.maculata.ChatServer
5. 3. /Maculata/src/za/co/nephila/maculata

3.2.3.2. Programming Exercises

Download here.

3.2.4. Chapter 4

3.2.4.1. Multiple Choice

1. 2. False
2. 5. All the above
3. 1. True
4. 5. All the above
5. 2. False
6. 4. @returns
7. 2. Protected
8. 1. -d
9. 2. javadoc -sourcepath ./src *.java
10. 3. javadoc -d ./docs/api -sourcepath g:\packages\Maculata\src;g:\packages\Cruentata\src nephila.maculata nephila.cruentata

3.2.5. Chapter 5

3.2.5.1. Multiple Choice

1. 1. True
2. 2. False
3. 5. All the Above
4. 3. jar cf jar-filename input-files
5. 2. jar tf jar-filename
6. 5. jar cmf seal.txt myJar.jar MyCompany
7. 1. java -jar project.jar
8. 4. jar xf jar-filename archived-files
9. 3. Main-Class: net.sourceforge.jinit.Main
10. 1. JarEntry

3.2.5.2. Programming Exercises

1.
```
public static void main(String[] args) {
    try {
            File dir= new File("output");
        JarFile jarFile = new JarFile("test.jar");
        Enumeration entries = jarFile.entries();
        while (entries.hasMoreElements()) {
            JarEntry entry = (JarEntry) entries.nextElement();
            File outFile = new File(dir, entry.getName());
            if (entry.isDirectory()) {
                outFile.mkdir();
            } else {
                InputStream in = jarFile.getInputStream(entry);
                OutputStream out = new FileOutputStream(outFile);
                int c;
                while ((c = in.read()) != -1)
                    out.write(c);
                in.close();
                out.close();
            }
        }
        jarFile.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

2.
```
public static void main(String[] args) {
    try {
        File outFile = new File(args[0]);
        JarOutputStream zout = new JarOutputStream(new
FileOutputStream(outFile));
        byte[] buf = new byte[1024];
```

```
        for (int i=1; i < args.length; i++) {
            FileInputStream in = new FileInputStream(args[i]);
            zout.putNextEntry(new ZipEntry(args[i]));

            int len;
            while ((len = in.read(buf)) > 0) {
                zout.write(buf, 0, len);
            }

            zout.closeEntry();
            in.close();
        }
        zout.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

3.2.6. Chapter 6

3.2.6.1. Multiple Choice

1. 4. bin
2. 3. A java build tool.
3. 2. Is an extensible markup language.
4. 3. <?xml version="1.0" standalone="yes" encoding="UTF-8" ?>
5. 1. True
6. 1. True
7. 5. All the above
8. 1. mkdir
9. 5. All the above
10. 1. True

3.2.7. Chapter 7

3.2.7.1. Multiple Choice

1. 4. All the above
2. 1. True
3. 1. Simplicity
4. 4. All the above
5. 1. True
6. 3. Test - Code - Design
7. 1. True
8. 3. junit.framework.TestSuite

9. 2. setUp
10. 1. True

3.2.8. Chapter 8

3.2.8.1. Multiple Choice

1. 1. True
2. 1. True
3. 5. All the above
4. 2. Its destination
5. 4. EXCEPTION
6. 4. WriterAppender
7. 4. All the above
8. 2. False
9. 4. All the above
10. 2. log4j.debug=false

3.2.9. Chapter 9

3.2.9.1. Multiple Choice

1. 1. True
2. 4. All the above
3. 6. 3 and 4
4. 3. Get a new instance of a builder factory and get a new document builder from it.
5. 1. True
6. 2. Use the createElement method on a document object.
7. 1. True
8. 5. 2 and 3
9. 1. SAX
10. 2. <xsl:value-of select="."/>

3.2.9.2. Programming Exercises

1.
```
public static void main(String[] args) {
    try {
        PrintWriter out =
            new PrintWriter(new FileOutputStream("memo.xml"));
        out.println("<?xml version=\"1.0\" ?>");
        out.println("");
        out.println("<!DOCTYPE memo SYSTEM \"memo.dtd\">");
        out.println("");
        out.println("<memo>");
```

```
        out.println("    <to>John</to>");
        out.println("    <subject>Re: Progress Meeting</subject>");
        out.println("    <message>Don't forget that we must book the
boardroom");
        out.println("        well in advance or we'll have to have the
meeting in");
        out.println("        one of our offices which will turn out to
be");
        out.println("        <emphasis>very cramped</emphasis>.");
        out.println("        See you later!");
        out.println("    </message>");
        out.println("    <from>Trevor</from>");
        out.println("</memo>");
        out.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

2.

```
public class Main  {

    public static void printNode(Node node) {
        if (node == null) return;

        int type = node.getNodeType();
        if (type == Node.TEXT_NODE) {
            Text text = (Text)node;
            System.out.println(text.getWholeText());
        } else if (type == Node.ELEMENT_NODE) {
            Element element = (Element) node;
            System.out.print(element.getTagName() +  ": ");

            NodeList children = node.getChildNodes();
            for (int i = 0; i < children.getLength(); i++)
                printNode(children.item(i));
        }
    }

    public static void main(String[] args) {
        try {
            DocumentBuilderFactory builderFactory =
                DocumentBuilderFactory.newInstance();
            DocumentBuilder builder =
            builderFactory.newDocumentBuilder();
            Document document = builder.parse("memo.xml");
            printNode(document.getDocumentElement());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

4. End Matter

4.1. Bibliography

4.1.1. Books

Burke, Eric, M. Java and XSLT. *O'Reilly, 2001*

Epp, Susanna S. Discrete Mathematics With Applications. *PWS, 1995*

Gehtland, Justin & Tate, Bruce. Better, Faster, Lighter Java. *O'Reilly, 2004*

Harold, Elliotte Rusty XML Bible. *Hungry Minds, 2001*

Hightower, Richard & Lesiecki, Nicholas. Java Tools For eXtreme Programming. *John Wiley & Sons, 2002*

Holzner, Steve. Eclipse, A Java Developer's Guide. *O' Reilly, 2004*

Tilly, Jesse & Burke, Eric. Ant: The Definitive Guide. *O'Reilly, 2002*

Wake, William C. Extreme Programming Explored. *Addison-Wesley, 2000*

4.1.2. Online Resources

Beck, Kent & Gamma, Erich JUnit Cookbook *http://junit.sourceforge.net*

Gulcu, Ceki. Short introduction to log4j.
http://logging.apache.org/log4j/documentation.html

Various The Java Tutorial. *http://java.sun.com/tutorial*

4.2. Glossary

Abbreviation
A shortened form of a word or phrase used chiefly in writing to represent the complete form.

Abstract Class
An abstract class is a type of class that cannot be instantiated, it merely exists to be extended and contains methods and varaibles common to all sub-classes.

Abstraction
A representation that only captures essential aspects of something thereby reducing complexity.

Access Modifier
An access modifier is a modifier that changes the visibility of a class or its members.
Algorithm
A step-by-step problem-solving procedure, especially an established, recursive computational procedure for solving a problem in a finite number of steps.
API
Application Programming Interface, Set of related classes and methods that provide certain functionality. The API represents the parts of a class exposed through access modifiers to code written by other programmers.
Application Failure
A situation resulting from a fatal error in a computer program that renders the program useless.
Application
A computer program that helps the user accomplish a specific task; for example, a word processing program. Application programs should be distinguished from system programs, which control the computer.
Archive
A file that contains a group of files and possibly directories which must be extracted in order to use them. An archive may optionally be compressed which would require it to be decompressed before the files can be extracted.
Attribute
A single piece of information that represents a property present in all instances of a class. An attribute is often modelled as a variable in a class.
Automated
Acting or operating in a manner essentially independent of external influence or control.
Binaries
An application binary is the compiled form of a program, the actual machine code that is executed by a system in order to run the program.
Browser
A program such as Mosaic, Netscape, Internet Explorer, FireFox and others that are used to view hypertext documents on the World Wide Web.
Bug
A colloquial term for a defect.
Build
The process of compiling and integrating all components of a piece of software, incorporating all changes made since the last build.
Byte

The amount of memory space used to store one ASCII character, which is usually 8 bits. A bit is the smallest unit of information a computer can hold; short for binary digit, the value can be either one or zero.

Child Node

A child node or descendant node is a node in a tree data structure that is linked to by a parent node.

Class Member

Class members are items that belong to that class, usually methods and variables and also nested classes.

Class

A class is the definition of a programmer defined type, the blueprint used to construct objects of that type. It is a logical unit that provides procedural and data abstraction.

Clean

A part of the build process in which temporary files used during the build process are removed.

Click

To press a button on a mouse or other pointer.Clicking is used to place the cursor, when working in text, or to select an object on the screen or a menu option.

Command

An instruction given to the computer, by means of a keyboard, punch card, mouse, voice command, or other method.

Comment

A portion of source code used to provide information about the source code. This section is ignored by the compiler.

Compiler

A computer program that translates a high-level programming language into machine language. The program fed into the compiler is called the source program; the generated machine language program is called the object program.

Component

A piece of software with a clear function that can be isolated and replaced by another component with equivalent functionality.

Compression

The temporary coding of data in a way that saves storage space or transmission time.Most text files can be compressed to about half their normal size.Graphics can be compressed to 10 percent of their original size.

Computer

An electronic device that has the ability to store, retrieve, and process data, and

can be programmed with instructions that it remembers. The physical parts that make up a computer (the central processing unit, input, output, and memory) are called hardware. Programs that tell a computer what to do are called software.

Configuration
The way that a program or computer is set up, various settings are used in the configuration.

Console
The command prompt window that recieves output from a `System.out.print` call.

Constructor
An object instance is created by calling the constructor of a class. A constructor is method-like block of code that is called when the object is created. Typically constructors initialize data members and acquire resources the abject may require.

Debugging
The process of locating errors in source code whether logical or syntactic and fixing them often through the use of a debugger.

Declaration
A statement that declares a class, interface, method, package or variable in a source file. It can also explicitly initialize a variable by assigning a value to it.

Default
An instruction that a computer assumes, unless the user gives it other instructions. For example, if the default typeface on a word processing program is Times Roman, the user may instruct the machine to use a different default typeface.

Defect
A flaw in any aspect of the system. See Error.

Deflated
An extracted archive file. See Archive.

Deprecated
An API item that is considered obsolete and on its way out, usually in favor of something better. Usually, though the item may have been originally included as part of an API, the use of it is no longer advised, and slowly support for the item is phased out.

Design
Process to find and describe a way to implement the system's requirements.

Developer
A computer programmer whose responsibilty is the development of new software rather than the maintenance and updating of existing software.

Development
The process of developing software.
Dialog
A box on the computer screen that lets the user communicate with the computer. A dialog box can be used to enter information, set options, or give commands to the computer. The dialog box gives the user choices (such as open file, delete, save) which can be selected by clicking with the mouse.
Digital Signing
An attempt to mimic the offline act of a person applying their signature to a paper document.
Directory
On Macintosh and Windows 95 screens, files can be organized by placing them into folders that look like office file folders. These file folders are directories.
Distributed System
A system in which various compnents occur on different systems usually connected via a network.
Documentation
Instructions that come with a software program, which may include paper or electronic manuals, README files, and online help.
Download
To transfer files or data from one computer to another. To download means to receive; to upload means to transmit
Drive
A device that spins disks or tapes in order to read and write data; for example, a hard drive, floppy drive, CD-ROM drive, or tape drive.
Editor
A program that is used to make changes to a file; for example a text editor or an image editor.
Element
A block in a markup document like a tag in an HTML document.
Embedded
An object, such as a graphic, which has been placed in a document from another file.
Enumeration
An object that assists in iterating over a set of objects.
Environment Variable
A variable that specifies how an operating system or another program runs, or the devices that the operating system recognizes.
Error

A mistake made by a software developer that introduces a defect in a computer program.

Exception

Refers to an "exceptional condition" which is an abnormal situation that alters the flow of an application.

Executable

A program that a computer can directly execute. See Binaries.

Expression

An expression which uses symbols to represent numbers or abstract concepts for use in operations similar to arithmetic.

Extract

To deflate an archive. See Archive.

Fatal Error

An error that causes a program to stop executing. See Error and Application Failure.

File

A block of information in the form of bytes, stored together on a computer or external digital storage medium, and given a name. A file may be a program, a document, a database, or some other collection of bytes.

Fileset

An abstract representation of a set of files. For instance "`*.java`" refers to all files with a ".java" extension.

Font

A complete set of type characters in a particular style and size.

Framework

A Skeletal software component that performs functions required by a system and which is incorporated into the design of such systems.

Function

A method that performs some form of processing in order to return a result. For example, a function to calculate the sum of two integers.

Functionality

The capabilities or behaviors of a program, application, or system; the total set of its features.

Generate

The automated creationg of something.

Getter

A method that is responsible for relaying information about a class's state by returning the value of one of its variables.

Hard Coded

Hard-code refers to the software development practice of embedding data directly into the source code of a program or other executable object, instead of obtaining that data from external sources such as a configurations file or command-line parameters.

HTML

Hypertext Markup Language, The language used to create World Wide Web pages, with hyperlinks and markup for text formatting.

IDE

Integrated Development Environment, an IDE combines the editor, compiler and other useful tools in the same software package. Its advantage is that when a program with syntax errors is compiled, the programmer sees the error messages and the original program at the same time -- this makes debugging much easier.

Import

The act of including another source file's declarations for use in one's own source file.

Initialize

Set a variable to a starting value.

Input

Something put into a system or expended in its operation to achieve output or a result.

Install

To load and configure a piece of software on a computer.

Interface

Publically vsiible operations provided by an API.

Internet

A network of computer networks which operates world-wide using a common set of communications protocols.

Iterator

A data type used to mark a position in a collection of data (eg a linked list) and to move from item to item within the collection.

Jar

Java Archive file, An archive (like a ZIP file) contraining Java class files and images. JAR files are used to package Java applications for deployment.

Java

Java is an object-oriented programming language developed initially by James Gosling and colleagues at Sun Microsystems. The language, initially called Oak (named after the oak trees outside Gosling's office), was intended to replace C++, although the feature set better resembles that of Objective C. Java should

not be confused with JavaScript, which shares only the name and a similar C-like syntax. Sun Microsystems currently maintains and updates Java regularly.

Javadoc

Pertaining to the tool that parses the declarations and documentation comments in a set of source files and produces a set of HTML pages describing the classes, inner classes, interfaces, constructors, methods, and fields. The Javadoc tool from Sun Microsystems is used for generating API documentation in HTML format from doc comments in source code.

JDK

Java Development Kit, A software package that contains the minimal set of tools needed to write, compile, debug, and run Java applets and applications.

JRE

Java Runtime Environment, The JRE includes the Java Virtual Machine (JVM), core classes, and supporting files required to run Java programs.

JVM

Java Virtual Machine, An abstract computing machine, or virtual machine, JVM is a platform-independent programming language that converts Java bytecode into machine language and executes it.

Keyword

A keyword is an identifier which indicates a specific command. Many times the keyword is restricted to that core language usage, in which case it is also considered a reserved word.

Library

A library is a collection of subprograms used to develop software. Libraries are distinguished from executables in that they are not independent programs.

Logger

A component with sole responsibility of handling the logging operations in a logging framework.

Logging

The process of storing information about events that occurred in an application, used for debugging purposes.

Logical Unit

A group of code blocks or statements that together form a logical cohesive whole.

Maintain

The process of fixing bugs in, and adding new features to, existing software.

Manifest

File containing meta information incorporated into a Jar file.

Markup

Syntactically delimited characters added to the data of a document to represent its structure. There are four different kinds of markup: descriptive markup (tags), references, markup declarations, and processing instructions.

Member

Elements of a class, including methods, variables and nested classes.

Memory

The terms "storage" and "memory" refer to the parts of a digital computer that retain physical state (data) for some interval of time, possibly even after electrical power to the computer is turned off.

Message

Any information sent as a component interacts with another.

Method

Section of source code that performs a specific function, has a name, may be passed parameters and may return a result. Methods occur only within classes.

Multiline

Spanning across multiple lines.

Namespace

A scoping construct to subdivide the set of names and their visibility within a system. In many languages, this is tied to other computational constructs, eg, classes, procedures, modules, packages. A mechanism used to resolve naming conflicts.

Object

An instance, once a class is instantiated it becomes an onject.

Options

Alternatives or choices, often refers to settings or preferences in a program that may be set according to the users preference or taste.

Output

Information that has been manipulated by the central processing unit (CPU) of the computer, and displayed either on the video monitor or rendered on paper or film as hard copy, or saved on disk in a digital format.

Package

An entity that groups related classes together, the name must reflect the directory structure used to store the classes.

Parameter

Input passed to a method for processing. Parameters are a way of allowing the same method to operate on different data without re-specifying the instructions.

Parse

Parsing is the process of splitting up a continuous stream of characters (read from a file or keyboard input, for example) into meaningful tokens, and then

building a parse tree from those tokens.

Performance

A major factor in determining the overall productivity of a system, performance is primarily tied to availability, throughput and response time.

Platform

In computing, a platform describes some sort of framework, either in hardware or software, which allows software to run. Typical platforms include a computer's architecture, operating system, or programming languages and their runtime libraries.

Plug-In

A small piece of software that adds features to already existing, usually large, programs.

Popup

A Menu that provides no visual cue to its presence, but simply pops up when a user performs a particular action. Popup Menus are associated with a particular area of the workspace, such as the client area of an application, and a user must memorize where these areas are.

Portability

A measure of system independence; portable programs can be moved to a new system by recompiling without having to make any other changes.

Preferences

See Options.

Priority

Precedence: status established in order of importance or urgency.

Private

Access modifier that renders members of a class invisible to components outside of that class.

Process

A process is a running instance of a program, including all variables and other state. A multitasking operating system switches between processes to give the appearance of simultaneous execution, though in fact only one process can be executing at once per CPU core.

Program

See Application.

Programmer

Someone who writes source code that is compiled into a program. See Developer.

Project

A project is a temporary endeavor undertaken to create a unique product or

service.

Properties
A set of variables in the form `name=value` often saved in a file and used for configuration.

Property
A variable in the form `name=value`.

Protected
Access modifier that renders class members invisible to components outside of a class with the exception of sub-classes.

Random Access
The process of selecting information in a arbitrary order, not based on the physical order or sequence of its storage.

Reader
An object in Java that is used to read a character stream. See Stream.

Recursive
Referring back to itself. A method that calls intself until some base condition is true.

Refactor
Refactoring is the process of rewriting written material to improve its readability or structure, with the explicit purpose of keeping its meaning or behavior.

Return
The act of passing a result back to the caller of a method or function.

Run
To execute a program.

Runtime
During the time in which a program is executing.

Script
A file containing operating system commands that are processed in a batch method, one at a time, until complete.

Sealed
A Sealed Jar guarantees that all classes in a package come from the same code source.

Security
Techniques and practices that preserve the integrity of computer systems, and digital library services and collections.

Serialize
Encode a data structure as a sequence of bytes.

Setter
Method that transforms the state of an object by set the value of one of its

variables.

Software Developemnt
A set of activities that results in software products. Software development may include new development, modification, reuse, re-engineering, maintenance, or any other activities that result in software products.

Software
See Application.

Solution
The result of solving a problem.

Standalone
A program, function, files or system that operates on its own and has no dependencies on other programs, functions, files or systems.

Statement
An entity in a programming language which is typically the smallest indivisible unit of execution.

Stream
A continuous flow of data, usually digitally encoded, designed to be processed sequentially. Also called a bitstream.

String
A group or sequence of characters.

Stylesheet
An ASCII text document that is attached to an SGML or XML encoded document and that contains instructions to specify the formatting and display of the encoded document, or to transform it into another format.

Sub-class
A class that is an extension of another class and inherits public and protected variables and methods from the other class. Aslo known as a derived class.

Super Constructing
A derived or sub class calls the base or super class's constructor in order for the super class to initialize itself properly. This is achieved through the `super` mechanism in Java.

Super-class
A base class from which another class derives.

Swing
Swing is a GUI toolkit for Java. Swing is one part of the Java Foundation Classes (JFC) and includes graphical user interface (GUI) widgets such as text boxes, buttons, split-panes, and tables.

Syntax Highlight
Syntax highlighting is a feature of some text editors that displays text, especially

source code, in different colors and fonts according to the category of terms. This feature eases writing in a structured language such as a programming language or a markup language as both structures and syntax errors are visually distinct.

Syntax

The rules by which the words in a program are combined to form commands to a computer.

Tag

A tag is a marker embedded in a document that indicates the purpose or function of the element. Each element has a beginning tag and an end tag.

Target

The step in a build process that can be executed, usually by Apache Ant in a build file.

Task

A sequence of instructions treated as a basic unit of work.

Template

A set of pre-designed formats for text and graphics on which new pages and webs can be based.

Test Case

A set of conditions or variables under which a tester will determine if a requirement upon an application is partially or fully satisfied. It may take many test cases to determine that a requirement is fully satisfied.

Test Driven Development

Test-driven development (TDD) is a programming technique heavily emphasized in Extreme Programming. Essentially the technique involves writing your tests first then implementing the code to make them pass

Test Suite

A test suite is a set of related tests, usually pertaining to a group of features or software component.

Text

Data consisting of a sequence of characters, as opposed to binary numbers, images, graphics commands, executable programs, and the like.

Throw

In Java terms, an exception is said to be thrown if the exception is raised. A method may throw an exception if an error occurs in its processing.

Tool

A device that provides a mechanical or mental advantage in accomplishing a task.

Transformation

Transformation is a form of conversion in which a file is converted into a file

format with a comparable structure (eg from XML to XML, or from SGML to HTML). Usually, this form of conversion can be carried out very well.

Tutorial

A mode of instruction that presents content, checks understanding or performance, and continues on to the next relevant selection of content. Tutorials may be linear or branched.

Unit Test

A unit test is a method of testing the correctness of a particular module of source code.

User

A person who requires a computer for the performance of a task or recreational activity. Also called an end-user.

Variable

A quantity that may assume any one of a set of values.

Versioning

Versioning is a mechanism that keeps track of all changes in content and code and allows any change to be 'rolled back' to any previous version. This also means that a deleted file can be recovered to its last saved state.

Visibility

The accessibility of methods and instance variables to other classes and packages, through the use of access modifiers: public, protected, package or private.

Wildcard

A special character such as an asterisk (*) or a question mark (?) that you can use to represent one or more characters. Any character or set of characters can replace a pattern matching character.

Window

A window is a visual area, usually rectangular in shape, containing some kind of user interface, displaying the the output of and allowing input for one of a number of simultaneously running computer processes. Windows are primarily associated with graphical displays, where they can be manipulated with a pointer.

Windows

Microsoft Windows is a range of closed source proprietary commercial operating systems for personal computers and servers.

Zip

To zip a file is to compress it into an archive so that it occupies less disk space.