

# **Add a note hereApplied Software Engineering Using Apache *Jakarta Commons***

Add a note hereAdd a note here**Christian Gross**

# Chapter 1: **Introduction**

## **Add a note hereAdd a note here****Overview**

Add a note hereAdd a note hereIn this chapter, we will do the following:

* Add a note hereAdd a note hereIntroduce the book
* Add a note hereAdd a note hereIntroduce XML
* Add a note hereAdd a note hereIntroduce Open Source

## **What This Book Is About**

Add a note hereAdd a note hereThis book is a task-oriented solutions book that illustrates solutions to common programming problems using the Apache Jakarta Commons. Say the name Apache and many people will associate the name with a Web Server. Maybe some will associate Apache with other projects. What few do is associate Apache as an organization that helps developers create Open Source products. At the highest level is the Apache Software Foundation (ASF); below that are individual projects like the Apache Web Server, Extensible Markup Language (XML), and Jakarta.

Add a note hereAdd a note hereThe Jakarta and XML projects contain a number of Java-based projects, which include Web Servers, XML processors, and other useful solutions. Many of these solutions share components that allow you to do common things like logging, class instantiation, and serialization. Those common components are called the Commons, which is the focus of this book.

Add a note hereAdd a note hereThe Commons are reusable components created through using experience, not abstract thinking. When a developer wants to create a component, he typically evaluates the need, comes up with a design, and then implements it. The problem with this approach is that it may or may not solve the needs of the developer who is using the components. Another problem is that there are many different frameworks that do this. The difference between other frameworks and Commons components is that components from the Commons are based on experience and need.

Add a note hereAdd a note hereFor example, let's say somebody needs some logging routines. He will write the first version and think, "Hey, somebody else could use this." The code is then released to the Commons, but not the main Commons. The code is put into something called the Commons Sandbox, which is a pre-staging area that checks the need of the reusable code. In the Sandbox, the code will be inspected, tweaked, updated, and debated. Especially important in the Sandbox is that the code is truly reusable and used in many different projects. If a component in the Sandbox is popular and has gone through the teething stages, then it is promoted to a mainstream Commons component.

Add a note hereAdd a note hereThis teething and experimentation make a Commons component different from all other reusable component frameworks. A user of a Commons component knows that the component has been used elsewhere and solves a purpose in the best way that it can be solved.

Add a note hereAdd a note hereYou may be wondering how software engineering ties into the book. Engineering is the application of practical generic solutions to solve problems. In essence, this is what the Commons is all about. It is about being practical and pragmatic, and using proven solutions when solving common problems (such as creating objects, reading configuration files, and generating log messages).

Add a note hereAdd a note hereTying this back to the first sentence of the chapter, the focus of this book will be to present the components from the Commons that solve common problems. For example, when you're creating an object, what is the best way to create an object? The answer lies in [Chapter 3](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=164#164), which shows several different strategies offered by the Commons. Another problem is how to effectively write components; this is covered in [Chapter 4](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=250#250).

Add a note hereAdd a note hereThroughout this book, we pick a problem theme and then show the solutions using components and strategies from the Commons. What is important to remember when you're reading the material is that all of the solutions presented are tried and tested, and work in the most efficient manner possible.

Add a note hereAdd a note hereIt is important that you read the book starting from the front to the back because the questions at the end of each chapter build on top of each other. The idea is to get you using the ideas presented in the chapter to solve a problem that is representative of a real-life programming scenario.

## **If Only Coding Were Simple**

Add a note hereAdd a note hereThe code in [Listing 1.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=544675466) is for a Java program that outputs the text "hello world" to the command line. The listing is essentially a complete program, and it's simple. A bigger program, which is not rocket science to create, would involve code like that in [Listing 1.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=544675466) but repeated with

## **An Introduction to XML**

Add a note hereAdd a note hereXML is a way of defining structured documents or, in programmer terms, data. Typically, programmers store data in some type of format. Java programmers would probably store their data using Java persistence, which stores the data in a binary file specific to Java. A Java-persisted piece of data is useful to only Java, not to C++ or other languages. Using Structured Query Language (SQL) databases allows a developer to exchange data with programs written in different languages on different platforms. However, SQL databases require you to use specific Application Programming Interfaces (APIs), which adds extra programming steps. Although SQL adds complexity to the program, it is still used because a SQL database can efficiently manage large chunks of data.

### Add a note hereAdd a note hereWhy Use XML and How It Relates to a Stereo System

Add a note hereAdd a note hereIf SQL works, why use XML? The answer lies in the current architecture of XML itself. Consider a stereo system. When a guy buys a stereo system, does he ever worry about various pieces not working together? The answer is no. A stereo system is comprised of a receiver and a set of speakers. To connect the receiver to the speakers, the owner uses a cable that has a red connector and a black connector on each end. The owner of the stereo system then connects one end to the receiver and another end to the speakers. And at that point, the owner can listen to music.

Add a note hereAdd a note hereThis sounds very obvious, but there is a bigger picture. Consider the case where the owner of the stereo system decides that the quality of the sound is not good enough. In that case, he decides to buy an equalizer. To make the equalizer work, he gets another cable with the red and black connectors. This time, however, the owner plugs the stereo-out jacks to the equalizer-in jacks, and the equalizer-out jacks to the speaker-out jacks. And again the owner can listen to music, but with better sound quality. The owner can improve this sound quality even more. By simply adding more cables he can use multiple sound channels and so on. This process allows the user to add or remove cables without buying different kinds of cables.

Add a note hereAdd a note hereThe sound industry has managed to build an infrastructure that revolves around plugging standard audio cables in and out. In addition, users of the stereo system can improve or alter the sound by directing the audio content into the appropriate devices.

Add a note hereAdd a note hereIn this analogy, XML is the audio cable in the stereo system and is moved from one device to another. An XML database is the XML storage, which is like a Digital Audio Tape (DAT) deck that is responsible for reading and writing the audio. The programs that manipulate the XML are like the equalizer that reads audio from one channel, modifies the audio, and writes the audio to another channel. XML is considered to be the stream of information that connects devices. And like a stereo system, an XML processor can be added or removed from the chain without affecting other devices. This process starkly contrasts to the way technologies today work.

Add a note hereAdd a note hereOur current software industry suffers from the problem of different pieces of software being incompatible with each other because we have created a dependency on the programs that manipulate the data. For example, using today's word processors, can a person edit content from very early versions of a word processor? No. The translators have not been written since the data was stored in a proprietary format. When a piece of software writes data today, the written data becomes a legacy because there is no assurance that the program that created the data will be available tomorrow.

Add a note hereAdd a note hereAn argument could be made that cassettes and CDs store their data in very different formats, and hence cannot be compared to XML but are more like the early versions of the word processor. While this is true, it is not entirely correct. An XML database can store the data in a binary XML format or standard XML format. The way it is stored depends on the XML database. What is important is that the XML database can be addressed using Internet technologies and XML queries. While an XML front end could be written for the early word processors, such a front end does not exist and would require a de-emphasis on the proprietary nature of the word processing document.

Add a note hereAdd a note hereThe difference in an XML approach is that the focus lies in taking an XML data structure, transforming it, and sending it to the next device. The XML approach assumes that the data is more important than the application and allows a developer to plug and play with various XML processing devices.

### Add a note hereAdd a note hereHow XML Is Structured

Add a note hereAdd a note hereAt the simplest level, XML can be learned in a minute or two. Consider the XML in [Listing 1.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=738330865).

Add a note hereAdd a note here**Listing 1.4**

Add a note hereAdd a note here<doc attribute="value">

<key>value</key>

</doc>

Add a note hereAdd a note hereIn XML, the special characters < and > define blocks of XML. The blocks of XML are encapsulated one in another, as if you took smaller boxes and packed them in a larger one. In [Listing 1.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=738330865), the outermost XML block is doc; nested within is the XML block key. An XML block by definition has an opening XML tag (a sequence of < and > characters) and a closing tag. In an opening XML tag, the < character is followed by a complete word, which in [Listing 1.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=738330865) is either the doc or key word. A closing XML tag is the < character followed by the / character and a complete word. The complete word must match in nesting terms the opening XML tag. It is not possible to have XML blocks overlap each other as shown in [Listing 1.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=738330865). However, within an XML block you can embed text like the text value in [Listing 1.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=738330865).

Add a note hereAdd a note here**Listing 1.5**

Add a note hereAdd a note here<doc attribute="value">

<key>value

</doc></key>

Add a note hereAdd a note hereYou use XML attributes to describe the contained XML, just like you would write something on a box to describe the contents. In [Listings 1.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=738330865) and 1.5 is the XML attribute attribute, which has the value value. Specific XML attributes can occur only once within an opening XML tag. It should also be noted that XML is case sensitive, which means the XML identifiers open and Open are not identical.

Add a note hereAdd a note hereThis book assumes you have certain knowledge about XML. If you don't know the following XML technologies and want to learn about them, please see another text. Examples include: *XML: Introduction, Second Edition* (Book & CD), by Michael Hazard (ISBN: 0758048386), or *Learning XML*, by Erik T. Ray and Christopher R. Maden (ISBN: 0596000464).

* Add a note hereAdd a note hereeXtensible Stylesheet Language Template(XSLT): This XML programming language is used to transform an XML document into another type of content, which may or may not be XML.
* Add a note hereAdd a note hereXML Schema: This XML syntax is used to validate an XML document for correct structure.
* Add a note hereAdd a note hereSimple Object Access Protocol (SOAP): This XML description is used to define protocol communications between a client and server.
* Add a note hereAdd a note hereXML Document Object Model (DOM): This XML programming model is used to manipulate XML documents.
* Add a note hereAdd a note hereXML HyperText Markup Language (XHTML): This is XML data that is HTML based but conforms to the XML specification. For example, in previous HTML, the tag <br> was legal; however, XML requires a closing <br> tag.
* Add a note hereAdd a note hereXLink/XPointer: These XML extensions are used to reference other XML documents or resources from within an XML document.
* Add a note hereAdd a note hereXPath: This language is used in XSLT and other XML extensions to query and reference specific XML nodes in a document.

## **An Introduction to Open Source**

Add a note hereAdd a note hereThis book is about software engineering using the Commons, but we need to introduce how Open Source works. Some readers may have heard about Open Source but may have no idea what Open Source is about beyond the availability of the source code.

Add a note hereAdd a note hereOpen Source is about considering software as a product of one's effort, much like an author who writes a book, or a musician who composes music. The creator of the software believes in giving away the sources for everybody to look at, manipulate, and experiment with.

### Add a note hereAdd a note hereThe Three Cs

Add a note hereAdd a note hereAt an O'Reilly keynote address in 2003, Tim O'Reilly, the CEO of O'Reilly, talked about Open Source and its impact on the industry (see [*www.oreillynet.com/pub/wlg/3468*](http://www.oreillynet.com/pub/wlg/3468)). In that talk he mentioned the three Cs of Open Source:

1. Add a note hereAdd a note here**Commodization** of software
2. Add a note hereAdd a note hereUser-**Customizable** systems and architectures
3. Add a note hereAdd a note hereNetwork-enabled **Collaboration**

#### **Commodization of Software**

Add a note hereAdd a note hereSoftware is quickly becoming a commodity, with Open Source leading the way. Software is plentiful and easily available, which means that the price of software has dropped massively. This does not mean that nobody cares about software, just that people know it is available and plentiful. As a result of commodization, software applications have become plug-compatible and can interact with other pieces of software. People do not question whether or not two pieces of software can interact; it is assumed that they can.

#### **User-Customizable Systems and Architectures**

Add a note hereAdd a note hereIn a traditional software application paradigm, software is programmed against specific interfaces that may not change for years at a time. In an Open Source paradigm, change is constant. The objective is not to sell the software, but to sell the services associated with the software. So, if a customer needs an extra database field for a specific custom solution, that field can be added without much fanfare from the developers of the original software. In contrast, with today's closed source software often that is not the case. In the end what drives the application development cycle in the Open Source module is not the APIs but the data that needs to be manipulated.

#### **Network-Enabled Collaboration**

Add a note hereAdd a note hereOpen Source could have happened only with the advent of Usenet and the Internet. Developers on the Internet collaborate virtually and make things happen. Traditionally, making things happen required setting up meetings, making phone calls, and discussing the situation. The effort required to start an Open Source project when you are using Usenet and e-mail is virtually nil, so the effort required to shift resources to another project is just as low. The result is that Open Source applications constantly shift as determined by the needs of the individual projects and their programmers.

### Add a note hereAdd a note hereSo What Is Apache Server?

Add a note hereAdd a note hereApache Web Server is primarily a HyperText Transfer Protocol (HTTP) server used to serve HTML content. Apache Server is a mutation of the original National Center for Supercomputing Applications (NCSA) Web Server. In the early years of HTTP programming, an HTTP server called the NCSA HTTP server was developed at the NCSA. With time, however, another team created a set of patches to modify the original NCSA sources. The patches improved the NCSA HTTP server and provided extra capabilities. Soon the patches became so numerous that people started referring to the patched server as a "patchy server." At this point the "patchy server" became a new HTTP server called Apache Server. Apache's quality, flexibility, and availability of sources separated it from the rest of the field (it was a novelty that the sources were available). Around the time of Apache's inception, in 1995, Open Source was beginning to be coined as an expression.

Add a note hereAdd a note hereAs of this writing, Apache has a 62 percent market share of currently running Web servers, but the statistics can be debated. The debate says that in terms of the Secure HTTP (HTTPS) (typically e-commerce) protocol, the statistics are not so lopsided in favor of Apache Web Server. Although this is correct, it is misleading because these types of sites use commercial products, which are based on the Apache source code base.

Add a note hereAdd a note hereApache works for four major reasons:

* Add a note hereAdd a note here**It is Open Source**: Apache is not owned by anyone; rather, it is the intellectual property of the Apache Software Foundation (ASF). It's available in source code format, so anyone can download the sources, modify them, and distribute the changes in binary or source code format. Apache works because its license scheme is very liberal and allows users to do whatever they please with the sources. The only real restrictions are that you cannot call Apache your own development, and if you release a product using Apache sources you must reference Apache.
* Add a note hereAdd a note here**It works very well**: Apache works anywhere and at any time. The Internet has introduced the concept of working constantly because the Internet does not rest and information can be requested all the time. Hence, software must work continually. Consider the PC. For all of what it has brought to the industry, it continues to be unstable. Unlike PCs, HTTP applications must always work. Apache fulfilled this requirement and hence has been used extensively at large Internet Service Providers (ISPs).
* Add a note hereAdd a note here**It offers third-party support**: There is a huge amount of third-party support for Open Source. Literally thousands of third-party modules can do whatever is required. It is possible to write any type of server-side application using third-party support.
* Add a note hereAdd a note here**It works**: Was this mentioned? The fact that Apache works so well should not be underestimated.

### Add a note hereAdd a note hereBut Is There a "Real" Company?

Add a note hereAdd a note hereMany people who want to use a piece of software want to have a company they can complain to when there are problems. No such official support is offered for the Apache Server software. However, there is plenty of free support, and many third-party companies—such as IBM or HP—offer paid support. The ASF is a non-profit organization and is responsible only for the organizational, legal, and technical support of various projects that improve Apache software. Some of these are listed here:

* Add a note hereAdd a note here**Apache HTTPD Server**: The Apache HTTP Server is an Open Source-based HTTP server usable on multiple operating systems and environments. From humble beginnings, the Apache HTTP Server has become an Internet Server platform in its own right.
* Add a note hereAdd a note here**The Apache XML Project**: This project is intended to provide base XML applications based on standards created by known standard bodies. Examples include XML (Xerces) and XSLT (Xalan) processors.
* Add a note hereAdd a note here**Jakarta**: Jakarta is the home for the ASF server-side Java projects.
* Add a note hereAdd a note here**Commons**: This group is dedicated to managing a number of components shared by various teams. Many of the projects outlined in this book are part of this group.
* Add a note hereAdd a note here**Web Services**: Within the Web Services group are a number of Web service-related projects: SOAP, the Axis toolkit (discussed later in this book), and XML Remote Procedure Call (XMLRPC).
* Add a note hereAdd a note here**Ant**: This project is dedicated to maintaining and developing the Ant build utility.
* Add a note hereAdd a note here**Perl**: The Apache/Perl integration project brings together the full power of the Perl programming language and the Apache HTTP Server. Using mod\_perl, which is an Apache HTTPD Server extension, you can write Apache modules entirely in Perl. In addition, the persistent interpreter embedded in the server avoids the overhead of starting an external interpreter and the penalty of Perl startup time.
* Add a note hereAdd a note here**PHP**: PHP is a server-side, cross-platform, HTML embedded scripting language that uses the Apache HTTP Server.

Add a note hereAdd a note hereThe ASF and the projects that it supports or collaborates with are all OSS projects. The ASF supports other groups not discussed here. If you want to investigate the various groups, visit [*www.apache.org*](http://www.apache.org).

### Add a note hereAdd a note hereHow You Can Interact with Open Source

Add a note hereAdd a note hereIn classical terms, when somebody mentions Open Source and using it, the thinking is that any development using Open Source must be Open Source as well. That is a fallacy. There are two ways to interact with Open Source: as a provider and as a consumer. As a provider of Open Source, the developer creates software that will be used by other people. That aspect of Open Source is beyond the scope of this book. This book focuses on the consumer aspect of Open Source, just like anybody would use an API from another software vendor.

#### **Using OSS Programs As Components**

Add a note hereAdd a note hereThe Open Source components focused on in this book are from the Commons. However, you should not stop there; you should be actively looking and trying out new components. Only by being aware of what is available can you best solve your software problems. Following are a number of steps to take when you're experimenting with different Open Source components:

Add a note hereAdd a note here**Visit source code sites every day**: Regularly visit sites that feature OSS so that you can be aware of what packages are available. Two good source code sites are [*www.freshmeat.net*](http://www.freshmeat.net) and [*www.sourceforge.net*](http://www.sourceforge.net). If you have a problem, you can often find its solution in a different package. For example, if you need HTTP client code, then that code may be found in an HTTP proxy server. An HTTP proxy server might be acceptable because a proxy requires an HTTP client; hence, the code might exist and be well written.

Add a note hereAdd a note here**Learn to use search engines**: Searching the Internet is a learned skill. Only by practicing can you find what you need. For example, use Google ([*www.google.com*](http://www.google.com)) to find help with generic topics. When you get your search results, look for words that are used repeatedly and then execute a new search based on those found words.

Add a note hereAdd a note here**Download software that looks interesting**: You can get new ideas by downloading and looking at interesting software. Doing so allows you to think laterally and look at problems in a different context.

Add a note hereAdd a note here**For a specific problem, download multiple packages**: When you're investigating a solution to a particular problem, always download multiple packages. This is important because often the first or most common OSS program may not solve your problem.

Add a note hereAdd a note here**Unzip and compile the downloaded packages**: When you find a package that you consider a solution to a problem you're having, the first step when you're evaluating the package is to compile it. If it does not compile and you cannot fix the problem in under five minutes, downgrade the usability of the package. A good OSS package always compiles within five minutes out of the box. Once the package compiles, run the demos and downgrade the package if the demos do not automatically execute successfully.

Add a note hereAdd a note here**Compile the package as a static library within your application**: The purpose of this step is to integrate the OSS program with your sources. At this point you can see if there are any compile problems. Typical examples of problems are redefinition of standard data types and conflicting error-handling schemes. The biggest problem with OSS is that every OSS package has its own error-handling scheme. If compilation fails, downgrade the usability of the OSS program.

Add a note hereAdd a note here**Compile the package as a shared library without your application**: As with the previous step, the purpose of this step is to see if there are any compile-time problems. In addition, note that in each of these steps, the OSS program was not called or utilized. If compilation fails, downgrade the usability of the OSS program.

Add a note hereAdd a note here**Make a single isolated use of the package**: In a part of your program, create a function or class and make a simple call. The purpose of this step is to see if there are any runtime errors such as data misalignment or error-handling problems. If there are any problems, downgrade the OSS program.

Add a note hereAdd a note here**Attempt to read the source code**: In this step you're looking to see if the code is understandable. Experience will show you that legible source code is easier to debug, maintain, and extend than illegible code. Hard-to-read or hard-to-manage code makes your programming life much more difficult. Hard-to-read code downgrades the usability of the OSS program.

Add a note hereAdd a note here**See what support is given**: Consider the support options of the OSS programs. Good OSS programs like Apache have mailing lists, books, and corporate support. The number of extensions is also important. Extensions of the program provide sample programs, test the program, and show that there is actual interest. The more support there is, the more usable the OSS program is.

Add a note hereAdd a note here**Back up your project**: This is the last step of no return with respect to your program. Beyond this step you will actually be integrating the OSS program into your sources. By performing regular backups of the project, you will be able to roll back to the original versions of your sources.

Add a note hereAdd a note here**Integrate the OSS program into your sources**. This is the last major step of integrating the OSS program into your sources. At this point, you can expect a high level of success as far as making the OSS program work for you.

Add a note hereAdd a note here**Follow the package**: At this point, the OSS program is part of your sources. Hence it is very important that you follow the mailing lists, report bugs, and contribute changes. Only in this way can you ensure that you know how to fix problems when they arise and use the OSS program efficiently.

**Summary**

Add a note hereAdd a note hereThis chapter explained some of the basics that will be presented in future chapters. It essentially explained that software engineering using the Commons is a task-driven approach, not a pure object-oriented approach. Using tasks is similar to using patterns, except that the tasks are defined in more generic terms.

Add a note hereAdd a note hereThe Commons includes a number of components that have proven their worth in other projects. That is the point of the Commons. In the charter of the Commons, it is a requirement that the component already be popular before it is accepted. In fact, before being even accepted into the Sandbox, the component must have some type of following.

**Open Source Golden Rule**

Add a note hereAdd a note hereMany people say that the advantage of Open Source is that you have the sources and can modify them to suit your needs. Some people say that this is a disadvantage and makes for second-rate or unprofessional software. The truth of the matter is that Open Source has nothing to do with quality. There is bad Open Source software and good Open Source software, just as there is good and bad Closed Source software. You can distinguish Open Source and Closed Source by thinking of it like this: imagine you are checking the oil in your car. With Open Source, you can check and add your own oil. You could even change your own oil. It is your choice. With Closed Source, you don’t have these options; a specialized technician needs to handle every change and update.

Add a note hereAdd a note hereWhen you are in the process of selecting which Open Source to use, of course you need to look at its quality. As a consumer of components, you will probably not select a package that does not compile out of the box. If a couple of tweaks related to jar locations are required, then that is acceptable, but it’s unacceptable if the Open Source requires manual source code or build file updates.

Add a note hereAdd a note hereIf you are willing to invest the time to update the component, that is acceptable. However, if you can’t update the Open Source component within five minutes, it most likely will not compile out of the box. If you must update it, that may take between a day and a week, assuming that you are very knowledgeable in the field.

Add a note hereAdd a note hereOne problem that the Commons could run into is that it could be overrun by thousands of potential frameworks that solve all of the problems of the software world. Therefore, the requirement that the component has to already be popular and used ensures that only the most useful and versatile of components will enter the Commons. Maybe this policy discriminates against new components, but the Commons wants to ensure that only the best of breed survive. This only enhances the value of the components in the Commons for end users.

Add a note hereAdd a note here**On the CD** The CD-ROM contains all of the sources accompanying this book. If a complete Java 1.4.x SDK and Java J2EE SDK environment is installed, the remaining needed tools are provided by the accompanying CD-ROM. At the end of each chapter the location of the appropriate sources that demonstrate the concepts shown in the chapter are given. At the URL [*http://www.devspace.com/content/jseng.htm*](http://www.devspace.com/content/jseng.htm) the latest changes and updates to the sources can be found.

## **Questions**

Add a note hereAdd a note here**1.1:** Complexity arises naturally and is decomposable into simpler instructions. For example, the number series 1,3,4,7,11,18 is based on the formula n1 + n2 = n3. The result looks complex, but the formula is simple. List some examples in nature that could be written using a simple program and would yield something that looked complex.

Add a note hereAdd a note here**1.2:** List some sample Open Source programs that could be described as being commodity-type applications.

Add a note hereAdd a note here**1.3:** Some business people would argue that Open Source in today's form represents a monopoly and is illegal. Explain why this is both correct and incorrect.

Add a note hereAdd a note here**1.4:** Name some ASF groups not listed in this book.

**Chapter 2: Four-and-a-Half Levels of Granularization**

**Add a note hereAdd a note here****Overview**

Add a note hereAdd a note hereIn this chapter, we will do the following:

* Add a note hereAdd a note hereDefine the different granularization techniques
* Add a note hereAdd a note hereOutline the two best practices that serve as fundamental concepts for all future concepts taught in the book
* Add a note hereAdd a note hereDefine common-sense class types
* Add a note hereAdd a note hereIllustrate various programming techniques based on the Jakarta sources
* Add a note hereAdd a note hereIntroduce Java Generics

## **The Purpose of This Chapter**

Add a note hereAdd a note hereEvery piece of Java code starts with a class. Maybe that class will use another class or subclass. Maybe that class is part of a package that is part of a component. Maybe the class is not even a class, but an interface. Regardless, once multiple classes, components, and interfaces are mixed together, the end result can be great or awful, much like bread that has either too much yeast or not enough. The trick is to learn how to know how much yeast to add so that you make the perfect bread or, in the case of this book, the perfect class, component, or interface. The task that is described in this chapter is how to organize your classes so that the resulting architecture is versatile.

## **Introduction to the Commons**

Add a note hereAdd a note hereWriting good code is always a challenge. The key to writing good code is understanding a few techniques and then using them often in different combinations. Knowing a few good techniques and their variations makes it possible to understand a complex system. In the Apache Jakarta project, there are literally millions of lines of Java source code. The Apache Jakarta source code would be a good place to start if you want to see what programming techniques do and don't work. In this chapter, we show you some of the more important and common Jakarta programming techniques. Most reusable techniques tend to appear in the projects Commons, Commons Sandbox, and XML Commons. These projects are components that have shown themselves to be reusable and commonly used.

## **The Four-and-a-Half Levels of Granularization**

Add a note hereAdd a note hereWhen writing object-oriented programs in the Java language, you need to know when to hide or expose details of a class, package, or component. You use Java interfaces, scope identifiers, inheritance, or package names to hide or expose a subset of functionality that another class can utilize (you do this to simplify development).

Add a note hereAdd a note hereImagine for a minute that it's the Middle Ages. The king—who is a good one—has to deal with some good people and some bad ones. The good people want to trade with him. The bad people want to take over his kingdom. The king could be one of two types, and each type is similar to how software is developed. All will be revealed shortly.

Add a note hereAdd a note hereIn response to the bad people, a paranoid king shields himself from society, effectively making it impossible for the bad people to remove him. The side effect, however, is that the king removes himself from contact with his people. As a result, the king always relies on other people for accurate information. The people who provide the information can usurp power from the king since they control the flow of information. This results in a delicate situation where the king could find himself the victim of false circumstances.

Add a note hereAdd a note hereOn the other hand, another type of king would be in constant contact with the people. The people would love the king, but because of the constant contact, the king would be vulnerable to assassination. Therefore, the king needs to somehow protect himself, while still being in contact with the people. The solution is that the king needs to construct a castle, a sort of house that encloses a community. In times of distress, the castle changes into a fortress to repel any potential invader. But the castle also serves a more important purpose: it provides an access point where guards can watch over everyone who enters the castle. Suspicious people are questioned.

Add a note hereAdd a note hereYou may have guessed by now that developing software is like being a king who builds various castles. While you may not have to worry about bad guys like the first type of king does, you will encounter some programmers who will do things they should not—either on purpose or by mistake.

Add a note hereAdd a note hereSoftware written too abstractly is like the king who does not want to be a target of assassination. This kind of software typically promises to be future-proof and can supposedly deal with any future changes in software technology. However, the future-proofing is only as good as the programmers who write the software. If the programmers are badly advised or miss certain details, then the future-proofing will become useless. Software that is written too loosely and that trusts the other programmers will become problematic because everybody will abuse the software and use it how they see fit. This is like the king who is worried about assassination because he trusts his people too much.

Add a note hereAdd a note hereHiding or exposing software details is called *writing components*. Components are like individual castles in that what is exposed from the component is like the guard at the castle who controls the flow of people. The different ways of exposing a component is called *granularization*. There are four-and-a-half levels of granularization. The first four are class level, package level, interface level, and Web Service level. The half level is generic-level granularization.

### Add a note hereAdd a note hereClass-Level Granularization

Add a note hereAdd a note hereIn Java, class-level granularization is limited to scope identifiers, like public, private, etc. [Listing 2.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707) shows the Java class definition.

Add a note hereAdd a note here**Listing 2.1**

Add a note hereAdd a note herepublic class ClassScope {

void methodPackage() {

System.out.println( "Scope package");

}

public void methodPublic() {

System.out.println( "Scope public");

}

protected void methodProtected() {

System.out.println( "Scope protected");

}

private void methodPrivate() {

System.out.println( "Scope private");

}

Add a note hereAdd a note hereIn [Listing 2.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707), the class ClassScope has been declared with four individual methods. Each method has a different scope. For this level of granularization, what is important is the difference between the public method modifier and the other modifiers. When a class, method, or field is declared public, then that class, field, or modifier is available for usage by all other classes. This means that a public modifier gives the highest amount of access to the individual class and its contents. Likewise, this means you need to consider the public modifier very carefully before using it.

Add a note hereAdd a note hereWhen you declare a class or interface public, it is important to consider the ramifications of this action. The problem is that once another class uses a public class or interface, that public class or interface is locked into a specific implementation signature. As in the castle example, we are providing a place for people to enter the castle and then one day changing the location. People will figure out where the new entrance is, but before everybody adjusts, there will be confusion, even if people are forewarned. Therefore, when you make classes or interfaces public, consider the publication final.

### Add a note hereAdd a note herePackage-Level Granularization

Add a note hereAdd a note hereIn the class example shown in [Listing 2.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707), some methods were declared with scope identifiers other than public. Other than the private scope modifier, the other methods are accessible from only other packages. [Listing 2.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707) shows a class that is visible only to the local package.

Add a note hereAdd a note here**Listing 2.2**

Add a note hereAdd a note hereclass PackageScope {

void method( ClassScope cls) {

cls.methodPackage();

cls.methodProtected();

}

}

Add a note hereAdd a note hereThe class PackageScope declared in [Listing 2.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707) has no scope modifiers, meaning that the default scope is used, which is package scope. Package scope means that only classes within the same package can interact with the class in [Listing 2.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707). The method method has no scope modifiers either and is therefore package scope as well. [Listing 2.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707) shows how the method method can call the package methods of the class ClassScope that have a package scope.

Add a note hereAdd a note herePackages in Java are a nice feature because they allow you to group together a number of classes and interfaces as one unit. Typically, packages are created to reflect the functionality contained within the package.

### Add a note hereAdd a note hereInterface-Level Granularization

Add a note hereAdd a note hereInterfaces are a higher level of granularization than packages because of the binding types and scope of the bindings. When referencing a class or a package, you make a direct reference to the identifier of the class or package. An interface is a bit different in that the interface that is referenced uses an unknown implementation. So, while the interface is referenced directly, the actual implementation is referenced indirectly. This makes the interface a more sophisticated kind of reference than a class or package. [Listing 2.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707) shows a sample interface and implementation.

Add a note hereAdd a note here**Listing 2.3**

Add a note hereAdd a note hereinterface IFlyingDevice {

void whatAmI();

}

class Plane implements IFlyingDevice {

void whatAmI() {

System.out.println( " am a plane");

}

}

class HotAirBalloon implements IFlyingDevice {

void whatAmI() {

System.out.println( "I am a hot air balloon");

}

}

class UserClass {

void runProgram( IFlyingDevice device) {

device.whatAmI();

}

}

Add a note hereAdd a note hereIn [Listing 2.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707), the interface IFlyingDevice represents an interface that will be used elsewhere. Two classes, Plane and HotAirBalloon, implement the interface IFlyingDevice. The implementation of the method whatAmI is different in each case. The class UserClass has a single method, runProgram, which is passed in an instance of the interface IFlyingDevice. Within the implementation of the method runProgram, the interface method whatAmI is called. The question is which version of the method whatAmI will be called? The answer is that it is entirely unknown. The answer depends on whether the class Plane or HotAirBalloon has been instantiated. You can't figure that out from [Listing 2.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707).

Add a note hereAdd a note hereWhat should be evident from the example in [Listing 2.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707) is that you can generically define an interface and then have multiple implementations of the interface. Generally, the purpose of the interface is to define an intention of action. For example, an intention could be to save data somewhere. Then, an implementation of the intention would be to save the data on a hard disk; another implementation could be to save data on a network.

### Add a note hereAdd a note hereWeb Service-Level Granularization

Add a note hereAdd a note hereThis level of granularization did not exist until very recently. Using a Web Service is similar to using a Java Remote Method Invocation (RMI) call because both involve communication over a network between a client application and a server service. The difference with a Web Service is that it is based on XML, which is not object oriented; rather, it is a mixture of object-oriented and structure-oriented data. The advantage of a Web Service is that it is generic and can be used by any type of client, regardless of platform and technology. However, exposing a Web Service is not like exposing an interface or a class, even though the Java mappings make it possible.

### Add a note hereAdd a note hereGeneric-Level Granularization, the Half Level

Add a note hereAdd a note hereThis level of granularization does not exist, except in the alpha versions (at the time of this writing) of the Java Software Development Kit (SDK). It is a half-level granularization because generics are useful helper functions that simplify specific aspects of Java programming. Generics are very similar to templates from the C++ world. The motivation for generics can easily be described by [Listing 2.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707).

Add a note hereAdd a note here**Listing 2.4**

Add a note hereAdd a note hereclass GenericType {

public synchronized Object get(int index) {

return null;

}

}

Add a note hereAdd a note hereIn [Listing 2.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707), the method returns an Object type. The data type Object is not a specific object but rather a generic object and the base of all other objects. Object is returned because in Java, if generics don't return a specific data type, more code needs to be written. For example, the core Java class Vector would have to be rewritten for each data type. If we rewrote [Listing 2.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707) to use generics, we would get [Listing 2.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707).

Add a note hereAdd a note here**Listing 2.5**

Add a note hereAdd a note hereclass GenericType< A> {

public synchronized A get(int index) {

return null;

}

}

Add a note hereAdd a note hereIn [Listing 2.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707), the class is defined using a generic data type A, which is referenced within the context of the class. The definition of A is like the declaration of a variable, except the declaration is not a variable but a data type. In addition, in places where data types are defined, you can use the declaration A. It is a very effective way of writing type-safe code, as shown by [Listing 2.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707).

Add a note hereAdd a note here**Listing 2.6**

Add a note hereAdd a note hereGenericType< IFlyingDevice> variable;

Add a note hereAdd a note hereIn [Listing 2.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707), the interface IFlyingDevice replaces wherever the data type A is defined in [Listing 2.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=906730707). Therefore, it would seem that generics solve the type problem very well. Generics are covered in more detail at the end of this chapter in the section entitled [*Generics*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=138#138).

## **Best Practice: Commons Bridge**

Add a note hereAdd a note hereThe Commons Bridge best practice is very similar to the bridge pattern as described in the *Design Patterns, Elements of Reusable Object Oriented Software* book. Both are used to separate intention from implementation. The difference between the Commons Bridge and the bridge pattern is implementation and variation. The purpose of the original bridge pattern was to separate an intention from implementation. The intention is the signature of the interface, and the implementation is a class that inherits the interface and implements the interface signature. The consumer of the Commons Bridge interacts with the interface. This frees the class that inherits the interface to change implementation details or even change the name of the class, without interfering with the client.

Add a note hereAdd a note hereThe Commons Bridge was created (instead of directly referencing the bridge pattern) because the bridge pattern is focused on one context. In reality, the concept of the bridge is very powerful because of its separation of intention and implementation. The problem is that the different solutions can be applied and must be applied depending on the context. In this book, all the examples are from the Apache projects, so it is only appropriate that the name used is Commons Bridge. In the simplest case, the Commons Bridge is implemented as shown in [Listing 2.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=718101136).

Add a note hereAdd a note here**Listing 2.7**

Add a note hereAdd a note herepackage example;

package org.apache.commons.cache;

public interface Cache extends Serializable {

// ...

}

package org.apache.commons.cache;

public abstract class BaseCache implements Cache {

// ...

}

Add a note hereAdd a note hereIn [Listing 2.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=718101136), the base interface Cache is defined. The abstract class BaseCache subclasses the class Cache. However, neither Cache nor BaseCache can be instantiated, because Cache is an interface and BaseCache is an abstract class. Notice that both the class and interface reside in the same Java package. This is on purpose because it indicates a package—this is intended for external use or is a core package for the component or application.

Add a note hereAdd a note hereA common implementation technique in the Jakarta projects is to combine the use of an interface and an abstract class. The interface defines an intention, but the abstract class defines a default implementation. Consider [Listing 2.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=718101136), which shows a more complete implementation of the Commons Bridge.

Larger View

Add a note hereAdd a note here**Listing 2.8**

Add a note hereAdd a note herepackage com.devspace.jseng.granularization;

interface BaseInterface {

void method();

}

abstract class DefaultFunctionality implements BaseInterface {

public void method() {

System.out.println( "Default functionality");

}

}

class UserImplementation extends DefaultFunctionality {

public void method() {

System.out.println( "Overridden functionality");

}

}

Add a note hereAdd a note hereThe interface BaseInterface defines an interface, which is then subclassed by the abstract class DefaultFunctionality. The abstract class DefaultFunctionality defines an implementation of the method method, which is a default functionality. When the UserImplementation class extends DefaultFunctionality, UserImplementation does not need to implement the method method. It is the choice of the class UserImplementation. This allows the UserImplementation class to implement the methods it needs to, when it needs to.

Add a note hereAdd a note hereOften, an implementation may need to override only specific pieces of functionality, not the entire interface. The technique of using both an interface and an abstract is not used everywhere in the Jakarta projects. The decision depends on whether or not each implementation should implement all methods of the interfaces. A case where not all methods should be implemented is when the interface exposes structural methods, as shown in [Listing 2.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=718101136).

Add a note hereAdd a note here**Listing 2.9**

Add a note hereAdd a note hereinterface Callback {

void addListener( BaseInterface interf);

void deleteListener( BaseInterface interf);

void method();

}

Add a note hereAdd a note hereIn [Listing 2.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=718101136), the interface Callback has three methods, but two are considered structural. The methods addListener and deleteListener are structural because they define methods used by a specific implementation to execute other code. A simpler indicator is if the methods, when implemented, appear similar in all of the implementations. In [Listing 2.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=718101136), the method addListener would add an interface instance to some collection. Regardless of the implementation, most likely all implementations would use a collection. You could consider the structural methods as those that perform generic operations on other interfaces or classes.

Add a note hereAdd a note hereA *structural class* or *interface* is a programming construct where the specified construct is not used to execute any programming logic related to the problem to be solved. The purpose of the structural class or interface is to organize another class or interface for later use in some kind of programming logic. An example of a structural class or interface is the management of an array of interface implementations. Typically, most structural classes are implemented as abstract classes that need to be subclassed.

Add a note hereAdd a note hereIn [Listing 2.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=718101136), the structural methods would operate on the interface BaseInterface, which was shown in [Listing 2.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=718101136). Therefore, any class that implements the Callback interface will need to implement the structural functionality. If you want to simplify the structural functionality, use an abstract class to implement the basics; then, the client class that subclasses the abstract class finishes the implementation. Therefore, the rule of thumb should be that an abstract class provides an infrastructure defined by the interface the client-derived class utilizes. Look back to [Listing 2.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=718101136); the Cache code in the Jakarta Commons code is doing just that. If you inspect the Jakarta sources, you'll see that this practice of creating a structural abstract class is commonly used and is labeled with the word Base or Skeleton in the class identifier.

### Add a note hereAdd a note hereVariation: Local Utilization

Add a note hereAdd a note hereThere is a variation of the Commons Bridge where the implementation of Commons Bridge is used locally. A local utilization happens when a defined interface and defined class are within the same package or same component. In this variation, the Commons Bridge does not have an interface definition, only an abstract class definition. The abstract class still serves the same purpose of defining structural code. However, because the client is local, the interface is not used. This is so because using an interface in the specific situation would be overkill. An interface defines a general intention and is good for general high-level contracts between components and applications. The problem with using interfaces everywhere is that it becomes tedious. When an abstract class is used locally, it defines both the interface and default structural code.

### Add a note hereAdd a note hereVariation: Private Implementation

Add a note hereAdd a note hereAnother variation of the Commons Bridge is to make the various user implementations of the abstract class or interface a private class. This variation is represented in the Commons Project *vfs*, which is a virtual file system class library. From the status document, the purpose of the vfs project is: "Commons VFS provides a single API for accessing various different file systems. It presents a uniform view of the files from various different sources, such as the files on local disk, on an HTTP server, or inside a Zip archive."

Add a note hereAdd a note hereA standard API will be defined from the status document, but most likely the user of the API will never want to implement the API. Instead, the user of the API will want to use a specific implementation, which could be a Zip file or Tar file (both are archives that reference a collection of files as a single file). As a result of the *vfs* API's, the user should not, under any circumstances, have access to the individual implementations. The implementation of this variation is shown in [Listing 2.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=718101136).

Add a note hereAdd a note here**Listing 2.10**

Add a note hereAdd a note herepublic interface FileObject {

}

public abstract class AbstractFileObject

implements FileObject {

}

final class FtpFileObject extends AbstractFileObject {

}

Add a note hereAdd a note hereIn [Listing 2.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=718101136), the interface FileObject has been declared as the API that the external application or component is using. The class AbstractFileObject implements FileObject, and the class FtpFileObject extends AbstractFileObject. Both the interface FileObject and class AbstractFileObject are defined as public, which means that other classes can publicly share both the class and interface. However, the class FtpFileObject is private to the package and final. This implies that the class FtpFileObject cannot be externally instantiated or extended.

Add a note hereAdd a note hereThe reason for keeping the class FtpFileObject private and final relates to the castle metaphor we discussed earlier in the chapter. The key is to control which people can get into the castle and become part of the community. In the case of the Commons Bridge, the instantiation of the structure is the responsibility of the Commons Bridge, not the client. This ensures that the Commons Bridge has a certain amount of control that only specific implementations can be used in the context of the package. The exact technique of instantiating the implementation is explained in [Chapter 3](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=164#164).

### Add a note hereAdd a note hereVariation: Nested Classes

Add a note hereAdd a note hereWhen you look through the Commons sources you'll see another variation of the Commons Bridge that is used quite often. In this variation, the protected class FtpFileObject is replaced with a nested class; this is shown in the Commons Project *lang*, which is a project for Java language extensions that did not make the runtime as distributed by Sun Microsystems. [Listing 2.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=718101136) shows an example of the variation.

Add a note hereAdd a note here**Listing 2.11**

Add a note hereAdd a note herepublic class FactoryUtils {

private static class PrototypeCloneFactory

implements Factory, Serializable {

}

}

Add a note hereAdd a note hereThe class PrototypeCloneFactory has the same role as the class FtpFileObject, and the interface Factory has the same role as the interface FileObject. In this example of the Commons Bridge variation, however, there is no counterpart for the class AbstractFileObject. This is unnecessary because it lacks structural code or methods. The difference in this class declaration is that the user implementation class PrototypeCloneFactory is private and cannot be extended or instantiated by any class other than FactoryUtils.

Add a note hereAdd a note hereThis strategy using nested classes is very useful when you're implementing small blocks of varying functionality. However, implementing interfaces with many methods will cause the editor to be stressed since an implementation class could easily extend into thousands of lines of code. As in [Listing 2.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=718101136), there most likely will be no abstract class of default functionality. If you do define an abstract class, ensure that there is a very good reason; otherwise, the line count will increase exponentially. If an abstract class is defined as being structural code, then declare the abstract class as a nested class in the same parent class as the implementations.

## **Best Practice: Commons Composite**

Add a note hereAdd a note hereThe Commons Composite is a best practice that encompasses the adapter, proxy, composite, and state patterns. In essence, all of the encompassed patterns are based on having a conductor manage a number of classes as action items. The action items are typically logic classes or interfaces. The conductor is responsible for organizing and knowing when and how to call the action items. In a typical implementation a client using the Commons Composite will instantiate both the conductor and action items, but will interact with only the conductor.

Add a note hereAdd a note hereIn the simplest example of the Composite, there is only one structural class or interface, and one logic class or interface. A *logic class* or *interface* is a programming construct where the class or interface is primarily used to solve a programming logic that is related to the program problem. An example would be deciding whether or not to call a specific method on an interface, which is similar to the adapter pattern. Typically, the adapter pattern is used to fit an existing implementation with a different implementation that a consumer of the best practice expects.

Add a note hereAdd a note hereThe problem with the Composite variation of the best practice is that it implies fitting two classes together using a middle class. Doing things this way can be inefficient. Open Source projects like the Jakarta projects do not like to make an interface or implementation intended for one purpose fit into another purpose. Instead, the Jakarta group defines a new architecture that reflects the new approach. You can find the simplest example of this approach in the Jakarta Tomcat sources. In the sources there are three classes, Tomcat3Request, org.apache.coyote.tomcat4.CoyoteRequest, and org.apache.coyote.tomcat5.CoyoteRequest. Each class represents a request in that version of the application.

Add a note hereAdd a note hereWhen new classes are used to define new implementations, it is a clean-slate approach where the old may or may not be kept and the new will be a better and potentially incompatible version. The result is that the new version of the software is the best it can be. The Jakarta Tomcat project can do this because it is an Open Source project not constrained by budgets, time, and current customers. In the for-profit world, constantly changing the interfaces can cause problems and tends to be avoided. However, the result is that the stagnant interfaces have to carry around a large amount of baggage. The Composite best practice attempts to be a compromise between the changing and stagnant interfaces. However, there is a warning: using too many composites makes for bad programming style and potentially buggy code.

### Add a note hereAdd a note hereThe Basic Implementation of Composite

Add a note hereAdd a note hereThe essence of Composite is that there is some already existing functionality, which could be a class or an interface associated with an implementation. To keep things simple, [Listing 2.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166) uses a class.

Add a note hereAdd a note here**Listing 2.12**

Add a note hereAdd a note hereclass ClassToHide {

public void hiddenStuff() {

}

}

Add a note hereAdd a note hereIn [Listing 2.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166), the class ClassToHide has a method hiddenStuff, which should be called, but not exposed, to the public world. Note that the class and method may already be public; therefore, hiding the class and method from a Java implementation point of view is not possible. The conductor considers this class the action item. The task of the conductor is to implement an interface and expose the action item using the implemented interface's methods. Most likely, the conductor is implemented as a Commons Bridge. [Listing 2.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166) shows an example of this implementation.

Add a note hereAdd a note here**Listing 2.13**

Add a note hereAdd a note hereinterface ExposeToAll {

void method();

}

final class ClassToExpose extends ClassToHide

implements ExposeToAll {

public void method() {

hiddenStuff();

}

}

Add a note hereAdd a note hereIn [Listing 2.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166), the interface ExposeToAll is what a client expects. The conductor is the class ClassToExpose, which subclasses the interface ExposeToAll and extends the action item class ClassToHide. Then, in the method method, a method call is made to the method hiddenStuff, which represents the already existing functionality. The implementation of the method method need not be only a single method call; it could contain more complicated logic. The idea is to delegate the new interface functionality to the original implementation.

### Add a note hereAdd a note hereVariation: Encapsulation

Add a note hereAdd a note hereIn [Listing 2.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166), the class ClassToExpose subclasses the class ClassToHide. While this works very well, ClassToHide may expose undesired functionality from the inheritance. An example of an undesirable side effect would be if the developer bought the action item as a component and then one day replaced the action with a new component. If a developer used the class ClassToExpose directly and not the interface ExposeToAll, then some code might break. While the developer may have done this intentionally or unintentionally, the damage is done. The solution is not to buy a specific component but to encapsulate the component instead. Using encapsulation, a developer cannot get direct access to the action item, as shown in [Listing 2.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166).

Add a note hereAdd a note here**Listing 2.14**

Add a note hereAdd a note hereclass VariationClassToExpose implements ExposeToAll {

private ClassToHide \_cls = new ClassToHide();

public void method() {

\_cls.hiddenStuff();

}

}

Add a note hereAdd a note hereIn [Listing 2.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166), the class ClassToHide is a private data member of the class VariationClassToExpose. When the method method from the interface ExposeToAll is called, the private data member \_cls is referenced to call the action item implementation.

### Add a note hereAdd a note hereVariation: Delegation

Add a note hereAdd a note hereIn the encapsulation variation of the Commons Composite, the client has no way to call methods exposed by the class ClassToHide. The Composite shown in the basic variation has the problem of exposing methods that should not be exposed, solved by the encapsulated variation. The third variation (called delegation of the Composite) is a hybrid between the basic and encapsulated variation. In the delegation variation, an interface exposes all of the methods like the basic variation, but the original implementation is referenced using the encapsulation variation.

Add a note hereAdd a note hereThe encapsulation variation of the Commons Composite, for reference purposes, is very similar to the proxy pattern. The only difference between them is intent. A sample delegation variation is shown in [Listing 2.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166).

Add a note hereAdd a note here**Listing 2.15**

Add a note hereAdd a note herepackage com.devspace.jseng.granularization;

interface SomeInterface {

void method();

}

class OriginalImplementation implements SomeInterface {

public void method() {

}

}

class Proxy implements SomeInterface {

OriginalImplementation \_cls = new

OriginalImplementation();

public void method() {

\_cls.method();

}

}

Add a note hereAdd a note hereIn [Listing 2.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166), the class Proxy implements the interface SomeInterface, making the delegation variation similar to the basic variation. In the implementation of the Proxy class is the reference to the class OriginalImplementation, which makes the delegation variation similar to the encapsulation variation. The two variations combined make up the delegation variation. The advantage of this approach is that a client can take advantage of a new functionality without having to completely rewrite the old functionality. This approach is very good when you need to create implementations that include service pack fix-ups or specific implementation details specific to a client.

### Add a note hereAdd a note hereVariation: Many Action Items

Add a note hereAdd a note hereThis variation is used extensively throughout the Jakarta sources. The main difference between this variation and the other Composite variations is that the conductor manages multiple action items instead of only a single action item. The conductor in this variation takes on the role of managing the collection of action items.

Add a note hereAdd a note hereIn this variation of the Commons Composite, the objective is to be able to group together various components that support a common class or interface into a collection and then perform an operation on the collection. A simple example of this variation of Commons Composite is a file system management tool. In a typical file system, there is a file and a directory entry. Both could be represented as an interface that has an identifier and some type of content. Some type of collection system would manage the files and directories. If a search were be performed, it would be necessary only to query the collections, iterate through the various components, and query the identifier of each component. The conductor would manage and query the individual components on behalf of the client. The advantage of this approach is that the user does not need to understand the individual functionality of the action items. The conductor will manage all of the details.

Add a note hereAdd a note hereIn the Jakarta sources, the word "composite" is used, but so are other words such as "pipeline" and "chain." All of these words are examples of the many action item variation. The simplest and best example of the many action item variation can be found in the Jakarta Tomcat project, mentioned earlier. In the Tomcat project, requests are processed using a Pipeline and Valve, as illustrated in [Listing 2.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166).

|  |  |  |
| --- | --- | --- |
|  | **Note** | Add a note hereAdd a note herePlease note that some readers may comment that the Jakarta Cocoon or Jakarta Ant projects are very good examples of a composite. That is absolutely correct, but both of those projects include other aspects in their composite that are beyond the scope of this book. Discussing these would confuse you more than help you. |

Add a note hereAdd a note here**Listing 2.16**

Add a note hereAdd a note herepackage org.apache.catalina;

public interface Valve {

public String getInfo();

public void invoke(Request request, Response response,

ValveContext context)

throws IOException, ServletException;

}

package org.apache.catalina;

public interface Pipeline {

public Valve getBasic();

public void setBasic(Valve valve);

public void addValve(Valve valve);

public Valve[] getValves();

public void invoke(Request request, Response response)

throws IOException, ServletException;

public void removeValve(Valve valve);

}

Add a note hereAdd a note hereIn [Listing 2.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166), the interface Pipeline includes a series of methods that manipulate the interface Valve. The objective of the interface Pipeline is to create a collection of valves that are then processed when the interface pipeline method invoke is called. The interface Pipeline is the conductor. In [Listing 2.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166), for the interface Pipeline there would most likely be a structural class that manages the details of the action items. For the interface Pipeline, the structural methods are setBasic, getBasic, addValve, removeValve, and getValves. On the same interface, the method invoke is a logic method that manipulates the managed valves. What will typically happen is the user class that subclasses the abstract structural class (which subclasses the interface Pipeline) will execute some method on the interface Valve. What has been ignored is how the classes that implement the interfaces Pipeline and Valve are instantiated. Each interface would be implemented using a Commons Bridge, so the instantiation would be managed using those best practices. A typical implementation would be [Listing 2.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166).

Add a note hereAdd a note here**Listing 2.17**

Add a note hereAdd a note herepackage com.devspace.jseng.granularization;

import org.apache.catalina.Pipeline;

import org.apache.catalina.Valve;

import org.apache.catalina.Request;

import org.apache.catalina.Response;

import org.apache.catalina.ValveContext;

import java.io.IOException;

import javax.servlet.ServletException;

class MyLogic implements Valve {

public String getInfo() {

return "";

}

public void invoke(Request request, Response response,

ValveContext context) throws

IOException,ServletException {

}

}

abstract class DefaultStructural implements Pipeline {

public void setBasic(Valve valve) {

}

public Valve getBasic() {

return null;

}

public void removeValve(Valve valve) {

}

public void addValve(Valve valve) {

}

public Valve[] getValves() {

return null;

}

public abstract void invoke(Request request,

Response response) throws IOException,ServletException;

}

class User extends DefaultStructural {

public void invoke(Request request, Response response)

throws IOException,ServletException {

}

}

Add a note hereAdd a note hereIn [Listing 2.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166), the class MyLogic subclasses the interface Valve and represents a logic implementation of the interface. In a typical implementation, there would be multiple different implementations of the interface Valve. The class DefaultStructural is an abstract class implementation of the interface Pipeline and represents the structural class. The methods that are implemented in the abstract class all relate to structural methods that manage the collection of valves. There is also another method, invoke, which is abstract. It is added only for notation purposes. It explicitly defines that any class that extends the class DefaultStructural must also implement the method invoke.

### Add a note hereAdd a note hereVariation: Related Action Items

Add a note hereAdd a note hereIn the design pattern community, there is debate on whether the conductor should be a separate interface or a class. The answer is not either one; rather, it hinges on whether the action item and conductor should be related.

Add a note hereAdd a note hereIn a composite, a related action item and conductor is when the consumer cannot easily distinguish between the two. In the delegation variation, the consumer did not know if he was using the original class or a proxy class delegating to the original class. In the basic variation or many action item variation, there was a distinction between the two classes. The related action item variation is a hybrid of the many action items and the delegation variations. An example of the related action item variation is shown in the XML axis project. In the XML Axis project, which is a SOAP toolkit, a related action item variation is created by defining the interface and then having both the conductor and action item derive from the same interface, as shown in [Listing 2.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166).

Add a note hereAdd a note here**Listing 2.18**

Add a note hereAdd a note herepackage org.apache.axis ;

public interface Handler extends Serializable {

public void init();

public void cleanup();

public void invoke(MessageContext msgContext)

throws AxisFault ;

public void onFault(MessageContext msgContext);

public boolean canHandleBlock(QName qname);

public List getUnderstoodHeaders();

public void setOption(String name, Object value);

public Object getOption(String name);

public void setName(String name);

public String getName();

public Hashtable getOptions();

public void setOptions(Hashtable opts);

public Element getDeploymentData(Document doc);

public void generateWSDL(MessageContext msgContext)

throws AxisFault;

};

package org.apache.axis ;

public interface Chain extends Handler {

public void addHandler(Handler handler);

public boolean contains(Handler handler);

public Handler[] getHandlers();

};

Add a note hereAdd a note hereThe conductor is the interface Chain, but notice how the interface Chain subclasses the interface Handler, which is the action item interface. The reason for doing this is to make logic interfaces self-reliant. Consider it from the scope of the client. Instead of having to deal with two interfaces, where one is the conductor and the other the action item, you have only one interface. Therefore, when calling the action method, the consumer would call either an action item or a conductor that calls the action items managed by the conductor. This approach allows a system to dynamically configure itself without having to indicate to the consumer that the configuration has changed.

### Add a note hereAdd a note hereSome Smaller Details

Add a note hereAdd a note hereThe Commons Composite best practice is used extensively throughout the Jakarta sources. The variations discussed earlier in this chapter generally use only interfaces. However, in the Jakarta sources the conductor may very often be a class, not an interface. Granted, the action item is always an interface. Whether or not that the conductor is an implementation or an interface depends entirely on the context. For example, if the conductor is defined locally and used within a package or subsystem, then using an interface is tedious. The rule on when to use a class and when to use an interface depends on the scope of the conductor. If the composite is used globally, as is shown in [Listing 2.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166), then an interface is the only possible solution. However, if the composite is used only within the local package or subsystem, a class definition of the composite is correct. An example of a class-based conductor approach is [Listing 2.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166).

Add a note hereAdd a note here**Listing 2.19**

Add a note hereAdd a note herepackage org.apache.commons.configuration;

public interface Configuration

{

Configuration subset(String prefix);

boolean isEmpty();

boolean containsKey(String key);

// Other methods...

}

public class BaseConfiguration implements Configuration

{

// methods...

}

public class CompositeConfiguration implements Configuration

{

// methods...

}

Add a note hereAdd a note hereIn [Listing 2.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=455421166), the action item is the interface Configuration. The conductor is the class CompositeConfiguration. The class CompositeConfiguration contains methods to add and remove implementations based on the interface Configuration. The class Base- Configuration is a structural class for the interface Configuration because the interface Configuration is very extensive and because it's tedious if you have to constantly implement all of the methods. Therefore, for the logic interface, it's a good idea to use a structural class. Remember that the structural class BaseConfiguration should be an abstract class.

## **Writing Code Based on the Two Best Practices**

Add a note hereAdd a note hereIf you inspect the Jakarta sources, basically all of the code falls into these two best practices. This is a very sweeping statement, because the watchful reader will wonder about serialization, persistence, and instance creation. Well, those algorithm constructs do exist as well, and they will be covered in later chapters. However, the two best practices are used even in those algorithm constructs. It is amazing that the Jakarta team writes code so consistently. It proves the point that to create high-quality, stable code, you do not need to know a large array of programming techniques, just a few techniques and their associated variations.

Add a note hereAdd a note hereOf course, there are other types of coding techniques beyond the ones you use with the best practices. Such coding is called logic coding, which is when the program is coded to do something like add a user to the database. The best practices defined are skeleton programming constructs. Consider using the best practices to be like building a house. The skeleton of the house is the individual walls, which are made of wood or concrete. The logic of the house is the color of the walls, type of flooring, and type of finishing. A skeleton of a house is essentially identical in every house on this planet. The difference lies in the logic as far as how they are placed in relation to each other and how the interior appears. Of course, the skeleton might be optimized to use more modern building constructs, just like you might use more modern best-practice constructs.

Add a note hereAdd a note hereThe rest of this chapter outlines the individual common sense-based programming logic constructs. We call these constructs *Commons Sense* because they are part of the common sense found in the Commons projects and other Jakarta sources.

### Add a note hereAdd a note hereCommons Sense: Package Names

Add a note hereAdd a note hereWith literally millions of lines of source code in the Jakarta project, it would seem futile to attempt to figure out what the sources do. In fact, though, it is relatively simple to do because the coding standard is more or less consistent. There are small exceptions to this rule, but it is very easy to figure out what those classes or interfaces mean. When you write your own code, it is absolutely imperative that you use a good naming standard. In fact, ideally you would adopt the Jakarta naming technique. Doing so would make it easier to integrate your sources and the Jakarta sources. Package names have three major sections. Let's take apart the package name in [Listing 2.7](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=66#66). Here are the three sections of the package name:

* Add a note hereAdd a note here**org.apache**: This gives the identity of the entity that distributes the application. This could be a corporation or organization. Typically, the identity of the entity is a two-token identifier, which is typically the domain name of the entity. In [Listing 2.7](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=66#66), it is the non-profit organization Apache.
* Add a note hereAdd a note here**commons**: This is the main identifier of the project. Typically, the identifier of the project is the overall project name or an important part of a project. In Listing of 2.7, the name of the project is Jakarta Commons.
* Add a note hereAdd a note here**cache**: This is the subsystem project identifier. Any tokens used beyond the main identifier of the project represent a subsystem.

Add a note hereAdd a note hereTypically in Apache Jakarta projects, the naming of the package is consistent and each section has a significant purpose. For example, in some projects there is no sub-system identifier. In those cases, it means that the root-level classes or identifiers are fairly important and used throughout the rest of the project. The concept of a root and dependencies is used when a project has subsystems. For example, the package name com.companyname. project.sub-system is used to define classes and interfaces for the implementation package com.companyname[.project](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053).sub-system.implementation.

Add a note hereAdd a note hereThe implementation package is a child of the definition package name. The parent-child relationship modularizes the application or component in an organized manner.

### Add a note hereAdd a note hereCommons Sense: Identifying Class Types

Add a note hereAdd a note hereUsing consistent identifiers to identify various pieces of your architecture makes it simpler to figure out what the individual pieces do. Following is a list of class types in use in the Jakarta sources. Each class identifier will be presented with the token *[identifier]* prefixed or followed by a specific identifier. The specific identifier identifies the role of the class. The following list contains examples of how the Jakarta group identifies its classes, packages, or interfaces (note that the identifiers presented may only be part of an identifier; e.g., AttributeException, and the important part is Exception):

#### **Type: Exception - [identifier]Exception**

Add a note hereAdd a note hereThis type of class is an exception class used to throw an exception. Exceptions are thrown to signal that something went wrong. The typical structure of an exception class from the Jakarta Commons sources is shown in [Listing 2.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204).

Larger View

Add a note hereAdd a note here**Listing 2.20**

Add a note hereAdd a note herepublic class BufferOverflowException extends RuntimeException {

private final Throwable m\_throwable;

public BufferOverflowException() {

super();

m\_throwable = null;

}

public BufferOverflowException(String message) {

this(message, null);

}

public BufferOverflowException(String message,

Throwable exception) {

super(message);

m\_throwable = exception;

}

public final Throwable getCause() {

return m\_throwable;

}

}

Add a note hereAdd a note hereShown in [Listing 2.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204) is an exception that subclasses the Java standard exception RuntimeException. Also used as exception classes are the classes Exception and IOException. Using a quick inspection, the different examples of exceptions within the Jakarta Source do not seem to follow any pattern on when to use a specific exception type.

Add a note hereAdd a note hereA closer inspection reveals the rule that an exception is subclassed from the class RuntimeException if the exception is related to a component error. It is supposed to indicate a fundamental error in program state, such as reading corrupted data. The corrupted data cannot be read no matter how intelligent the routines are.

Add a note hereAdd a note hereWhen an exception is subclassed from the class Exception, then the exception is related to a problem in the program. For example, a component exposes five elements to be read, and the consumer wants to read the sixth element. The component did everything correct and the consumer insisted on reading an element that does not exist, which means that the consumer is doing something wrong.

Add a note hereAdd a note hereA general note is that when you subclass an exception, it is advisable to see if there already exists a similar exception in the Java 2 Standard Edition (J2SE) runtime.

Add a note hereAdd a note hereYou saw the essentials of an exception class in [Listing 2.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204). The class BufferOverflowException has three constructors and the method getCause. The different constructors are used to specify the type of error:

* Add a note hereAdd a note hereThe constructor without any parameters is the default error generator, which generally should not be used since that exception will not generate any meaningful information.
* Add a note hereAdd a note hereThe constructor with only one parameter is an exception generator intended for one system. More about this in a moment.
* Add a note hereAdd a note hereThe last constructor is intended for an exception handler that redirects another exception.

Add a note hereAdd a note hereRedirecting exceptions is a useful mechanism if the subsystem has other independent subsystems. Consider [Listing 2.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204).

Add a note hereAdd a note here**Listing 2.21**

Add a note hereAdd a note here public void anotherSubSystem() {

throw new RuntimeException( "error in subsystem");

}

public void mySubSystem() throws RuntimeException {

try {

anotherSubSystem();

throw new MyException( "oops");

}

catch( MyException ex) {

throw ex;

}

}

Add a note hereAdd a note hereIn [Listing 2.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204), the method anotherSubSystem represents some other independent subsystem used for illustration purposes. The method mySubSystem represents the local subsystem. Both subsystems will generate exceptions. The consumer of the local subsystem will implicitly use the subsystem created for illustration purposes. The problem arises when there is an exception. When the other subsystem generates an error, the consumer of the local subsystem will put the fault at the local subsystem. The truth is that the local subsystem did everything right, but it is being blamed because the consumer cannot distinguish between the two exceptions and who is responsible.

Add a note hereAdd a note hereOne solution to distinguish the exceptions is shown in [Listing 2.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204), where the local subsystem generates the exception MyException to indicate that the error occurred locally. If the error occurred in the other subsystem, then that error will be thrown to the consumer directly. The problem with the solution is that the consumer knows the error occurred in the other subsystem but does not know why. There is no context on which step in the local subsystem caused the error in the other subsystem. [Listing 2.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204) is an attempt to distinguish the steps.

Add a note hereAdd a note here**Listing 2.22**

Add a note hereAdd a note here public void mySubSystemComplex() {

try {

anotherSubSystem();

}

catch( RuntimeException ex) {

throw ex;

}

try {

anotherSubSystem();

}

catch( RuntimeException ex) {

throw ex;

}

}

Add a note hereAdd a note hereIn [Listing 2.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204), the method mySubSystemComplex represents a complex subsystem. In addition, the method anotherSubSystem is called in two places. The two different places are different line numbers, but one could imagine that in a larger system, different classes or methods would be used. In each case of the method call of another subsystem, an exception is caught and the context can be determined. The conditions for the first method call to the method anotherSubSystem may be very different than those for the second method call to the method anotherSubSystem.

Add a note hereAdd a note hereThe problem with the solution provided in [Listing 2.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204) is that the code in the local subsystem has become hard to read because the different places where an exception block is typed out and lost are the context of the error in the local subsystem. A compromise solution that improves readability and attempts to pinpoint the error is shown in [Listing 2.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204).

Add a note hereAdd a note here**Listing 2.23**

Add a note hereAdd a note here public void mySubSystemComplex() {

try {

anotherSubSystem();

}

catch( MyException ex) {

throw ex;

}

catch( RuntimeException ex) {

throw new MyException( "Place1", ex);

}

}

Add a note hereAdd a note hereThis time, in [Listing 2.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204), the two-parameter constructor is used to indicate where the error occurred, and the original exception is stored as a reference within the newly generated exception. Then, at the higher level, the main exception handling mechanism can decipher what errors occurred. The details of how to log these errors are explained [Chapter 8](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=632#632).

#### **Type: Test - [identifier]Test | [identifier]TestSuite | [identifier]TestClass | [identifier]TestCase**

Add a note hereAdd a note hereThe classes or interfaces with the identifier Test indicate that these classes are responsible for testing specific classes or interfaces in the package, subsystem, component, or application. The Test class is part of the JUnit test suites, discussed in [Chapter 11](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=859#859). For now, we'll explain that the identifiers Test and TestClass indicate a unique test. In addition, the identifiers TestSuit and TestCase indicate a full test scenario that will call multiple unique tests.

#### **Type: Default - [identifier]Default**

Add a note hereAdd a note hereA class will typically use the Default identifier when the class is defined to be a default implementation of an interface. For example, when you're implementing an interface, there are defaults that could be used in several places. Instead of repeating the same code, a default class is used. If you relate this back to the Commons Composite best practice, discussed earlier, you will see that you might create a default class for an action item. Consider, for example, [Listing 2.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204), which is a default implementation for the Commons Project *attributes*, which is a project used to read metadata from Java classes.

Add a note hereAdd a note here**Listing 2.24**

Add a note hereAdd a note herepackage org.apache.commons.attributes.impl;

import org.apache.commons.attributes.Attribute;

public class DefaultAttribute implements Attribute {

private String name;

private String value;

public DefaultAttribute(String name, String value) {

this.name = name;

this.value = value;

}

public String getName() {

return name;

}

public String getValue() {

return value;

}

}

Add a note hereAdd a note hereIn [Listing 2.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204), the class DefaultAttribute is a simple implementation of the interface Attribute. The constructor DefaultAttribute assigns the internal data members, and the getters (methods that support the Java Bean property notation) getName and getValue retrieve the set values. The class DefaultAttribute is not fancy. It carries out only the minimum of operations, which is the intent of a class' default implementation. A default class is not a structural class because a structural class is not a complete class. The default class is a complete class with a minimum of implementation details and can be extended or directly instantiated.

#### **Type: Algorithm: [identifier]Builder | [identifier]Compiler | [identifier]Parser, etc.**

Add a note hereAdd a note hereA class or interface with the keyword Builder, Compiler, Parser, or any word that is a verb and has an "er" ending, processes data in a more sophisticated computer algorithm than the default Java SDK implementations. Let's explain the concept of a more sophisticated computer algorithm a bit. A simple algorithm is the assignment of data from one object to another, or the testing of an object to see if the object validates according to some conditions. A more complicated algorithm is when the data is not simply assigned, but collected and then analyzed. The analysis could involve tokenization or ordering of the data in a tree. These types of algorithms tend to be computer science-oriented and easy to isolate in their own package.

Add a note hereAdd a note hereIn the Apache Jakarta sources, the computer science routines are distinctly marked by the verbs with an "er" ending. Notice that these routines are encapsulated within a class or a package. This is a good coding practice because it means that when bugs or major changes have to be carried out, the changes are localized to a single class or package. Discussing the various computer science routines is beyond the scope of this chapter. However, in later chapters of this book, we will discuss how to use these classes in your own development.

#### **Type: Handler - [identifier]Handler**

Add a note hereAdd a note hereThe class type with a Handler identifier would seem to fall into an *Algorithm*-type class type due to the purpose and the identifier used. That is correct; the Handler class type is a specific type of class that deserves its own class-type identification. A Handler class type is used as a callback mechanism. More about this class type will be discussed in [Chapter 6](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=392#392).

#### **Type: Source -[identifier]Provider**

Add a note hereAdd a note hereIt would seem that the Provider class type, like the Handler class type, should be an Algorithm class type due to the naming convention. However, a Provider class type is very special because it may or may not be related to computer science. A Provider class type is a class or interface that provides a bridge to another resource. It's a source of data, which is consumed by the user of the provider. Database access classes are a common example of a Provider class type. More about this class type will be discussed in [Chapter 7](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=468#468).

#### **Type: Navigation - [identifier]Child | [identifier]Parent | [identifier]Manager, etc.**

Add a note hereAdd a note hereThe notion of a navigation identifier is very common in various component structures. The navigation identifier is not limited to Child, Parent, or Manager, but could include Sibling, Descendent, and so on. In addition, the navigation identifiers do not need to be restricted to class or interface identifiers. They present an unusual situation where methods can be used as identifiers. Navigation identifiers allow you to construct a complex data structure that encompasses many different objects. Typically, when a class or interface is a navigation type, it indicates that the class or interface is a bridge that combines several references into a useful structure. Navigation identifiers allow you to move from one part of the data structure to another. This idea is used extensively in the XML Document Object Model.

Add a note hereAdd a note here[Listing 2.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204) shows how to use a class or interface to manage a complex structure. The listing is from the Commons Project *Jelly*, which is a scripting infrastructure. (Note that the class has been abbreviated for clarity.)

Larger View

Add a note hereAdd a note here**Listing 2.25**

Add a note hereAdd a note herepackage org.apache.commons.jelly.tags.jface;

public class MenuManagerTag extends UseBeanTag {

private String text;

private MenuManager mm;

public Window getParentWindow() {

}

public void doTag(XMLOutput output)

throws MissingAttributeException, JellyTagException {

}

public MenuManager getMenuManager() {

}

}

Add a note hereAdd a note hereIn [Listing 2.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204), the class MenuManagerTag manages the class MenuManager as well as manages a place in the window hierarchy. The class MenuManagerTag, which is navigational, has navigational methods, as demonstrated by the method getParentWindow. This sort of class arrangement is very common.

Add a note hereAdd a note here[Listing 2.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204) shows an example of exposing an interface with many navigational methods. The code comes from the Jakarta Commons Project *vfs*.

Add a note hereAdd a note here**Listing 2.26**

Add a note hereAdd a note herepackage org.apache.commons.vfs;

public interface FileName

{

char SEPARATOR\_CHAR = '/';

String SEPARATOR = "/";

String ROOT\_PATH = "/";

String getBaseName();

String getPath();

String getExtension();

int getDepth();

String getScheme();

String getURI();

String getRootURI();

FileName getParent();

FileName resolveName( String name )

throws FileSystemException;

FileName resolveName( String name, NameScope scope )

throws FileSystemException;

String getRelativeName( FileName name )

throws FileSystemException;

boolean isAncestor( FileName ancestor );

boolean isDescendent( FileName descendent );

boolean isDescendent( FileName descendent,

NameScope nameScope );

}

Add a note hereAdd a note hereIn [Listing 2.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204), the interface FileName has multiple methods that allow access to various parts to a file type object. The classical accessor to retrieve a higher-level interface FileName instance is the method getParent. However, what is new, and also very common, are the test functions isAncestor and isDescendent. These functions test if the input parameter is either an ancestor or descendent of the current interface FileName instance, respectively.

Add a note hereAdd a note hereThe interface FileName, which is common, is primarily composed of accessors, meaning that the methods are used to retrieve specific attributes or to test specific attributes. Using the interface FileName, you can't alter the data structure. Consider the situation where a file structure is read from the Internet base on a Web site. In most cases, it would not be possible to modify this structure and, therefore, exposing writable methods would be silly. If you want to be able to read/write, the most common solution is to subclass the read-only interface and then add the necessary write methods, as shown in [Listing 2.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204).

Add a note hereAdd a note here**Listing 2.27**

Add a note hereAdd a note hereinterface FileNameWritable extends FileName{

void setParent( FileName parent);

void addChild( FileName child);

}

#### **Type: Bean - [identifier]Bean**

Add a note hereAdd a note hereThe *Bean* class type may seem to be a silly identifier, because if a Java class supports a certain method, it is considered a Java Bean by default. However, in the Jakarta sources, when a class is identified as a *Bean*, then that class is really a full-fledged Java Bean or works with Java Beans. To understand what is meant, consider the context. Imagine building a subsystem that dynamically loads and saves Java classes. To be able to do this, you need to use reflection and metadata. These tasks do not relate to application programming. A developer would not want to add the Java Bean support classes, and he relies on the infrastructure to provide those extra routines, which are marked as *Bean* in the Common sources.

Add a note hereAdd a note hereLetting the Commons provide the support routines is a big advantage of the Commons approach. Sure, it is possible to build all of the support routines, but having the Commons create the infrastructure makes it simpler to software engineer better application. The responsibility of the developer is to know which routines are available. This is why it is necessary to know the class naming convention. Knowing the convention means being able to use the naming convention and make any consumer code consistent with the Commons naming standard.

#### **Type: Utility - [identifier]Utils**

Add a note hereAdd a note hereThe *Utility* class type is a common type used to provide basic runtime support. For example, in the Commons Project *lang* you'll see the Factory interface, as shown in [Listing 2.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204).

Add a note hereAdd a note here**Listing 2.28**

Add a note hereAdd a note herepackage org.apache.commons.lang.functor;

public interface Factory {

public Object create();

}

Add a note hereAdd a note hereThe interface Factory in [Listing 2.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204) is used to instantiate specific objects. Using this interface, you can delegate the object instantiation process to something generic and reusable. The problem, though, is that an object may be instantiated in multiple ways, such as fresh creation or the cloning of an already existing object. In any case, those types of factory instantiations have to be managed by some implementation.

Add a note hereAdd a note hereOne solution would be to create individual implementations for each instantiation type. Then, the developer would have to memorize the individual instantiation classes and use them appropriately. The problem with this approach is that the developer needs to remember many classes.

Add a note hereAdd a note hereA simpler approach to solve the memorization problem is to introduce a single class that encapsulates all of the instantiation types. Then, each instantiation type is exposed as a single method call. This is the approach taken in the *lang* project and is shown in [Listing 2.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204). (Note the class has been abbreviated for clarity.)

Larger View

Add a note hereAdd a note here**Listing 2.29**

Add a note hereAdd a note herepackage org.apache.commons.lang.functor;

public class FactoryUtils {

private static final Factory EXCEPTION\_FACTORY =

new ExceptionFactory();

private static final Factory NULL\_FACTORY =

new ConstantFactory(null);

protected FactoryUtils() {

}

public static Factory exceptionFactory() {

}

public static Factory nullFactory() {

}

public static Factory constantFactory(

Object constantToReturn) {

}

public static Factory prototypeFactory(Object prototype) {

}

public static Factory reflectionFactory(

Class classToInstantiate) {

}

public static Factory reflectionFactory(

Class classToInstantiate, Class[] paramTypes,

Object[] args) {

}

}

Add a note hereAdd a note hereIn [Listing 2.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204), the class FactoryUtils provides five functions to instantiate the interface Factory based on the choice of instantiation process. Within the implementation of the class FactoryUtils there is quite a bit of source code to perform the operation. However, the class FactoryUtils hides the complexity, and is the preferred approach because the client should not be aware of how to instantiate the object. The client only wants to be able to determine the algorithm to use and wants a neat wrapper to the functionality. (For those readers versed in patterns, this is often called a *façade pattern*.)

#### **Type: Structural - [identifier]Support | [identifier]Base**

Add a note hereAdd a note hereWe have already discussed the *Structural* class type in this chapter. However, we do need to note that the structural class often is labeled with the identifiers Support or Base.

#### **Type: Iteration - [identifier]Iterator**

Add a note hereAdd a note hereThe *Iteration* class type specifies a class that explicitly loops through a collection of items and retrieves each individual item. While the user of the Iterator class type could do this manually, sometimes it would be tedious to constantly re-implement certain special algorithms. More about this class type will be discussed in [Chapter 7](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=468#468).

#### **Type: Collection - [identifier]List | [identifier]Map | [identifier]Bag | [identifier]Set**

Add a note hereAdd a note hereThe *Collection* class type is a class that is used to manage a collection of objects. The objects can be associated or arranged using different techniques. As a result, many identifiers are used to identify specific types of collections. More about this class type will be discussed in [Chapter 7](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=468#468).

#### **Type: Constants - [identifier]Constants**

Add a note hereAdd a note hereIn the C and C++ languages, you can define constants using global variables. In Java, however, you can't because global variables do not exist in the language. To create constants, the Jakarta sources use the concept of *Constant* class types. *Constant* class types serve a single purpose: to define variables that will be used throughout the subsystem, application, or component. [Listing 2.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204) shows an example of a *Constant* class type.

Add a note hereAdd a note here**Listing 2.30**

Add a note hereAdd a note herepackage org.apache.commons.jxpath.servlet;

public final class Constants {

public static final String APPLICATION\_SCOPE =

"application";

public static final String SESSION\_SCOPE = "session";

public static final String REQUEST\_SCOPE = "request";

public static final String PAGE\_SCOPE = "page";

public static final String JXPATH\_CONTEXT =

"org.apache.commons.jxpath.JXPATH\_CONTEXT";

}

Add a note hereAdd a note hereIn [Listing 2.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204), the constant class is called Constants. This is typical, but you don't have to name it this. You could call it JXPathConstants. In either case, the class Constants is a good example of a Constants-type class because it is a public class with a final declaration. This means that no one can subclass the class. You would not want anyone to subclass the class Constants because that would confuse the developer and make a simple concept complicated. Declared within the class Constants are a number of string variables declared to be public, final, and static. The technique used to declare a constant string is consistently used in multiple places and is a good idea because the string cannot change and is stored only once in the running application.

Add a note hereAdd a note hereThe assignment of the variable in [Listing 2.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=636662204) is acceptable, but you could have also used a static constructor if the declaration had been more complicated. The Jakarta sources contain some examples of more complicated operations in the class Constants. As outlined when we discussed the class Constants, the way to instantiate a constant value is a preferred practice.

## **Generics**

Add a note hereAdd a note hereJava Generics is a new topic in the Java environment and at the time of this writing is considered alpha status. Hence, it is important to realize that changes from the time of this writing may or will still happen. The Generics that will be explained are based on the compiler version 2.2 that is being presented as a proposal for Java 1.5 release. More information can be found at [*http://developer.java.sun.com/developer/technicalArticles/releases/generics/,*](http://developer.java.sun.com/developer/technicalArticles/releases/generics/)including where to download the Java Generic tools.

### Add a note hereAdd a note hereA Simple Example

Add a note hereAdd a note hereThe concept of the Generic construct is defined by a series of angle brackets that contain some identifiers. [Listing 2.5](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=62#62) was a simple example of a Generic definition. [Listing 2.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122) is another example of a simple Generic class definition.

Larger View

Add a note hereAdd a note here**Listing 2.31**

Add a note hereAdd a note hereclass Generic1< classtype> {

private classtype \_something;

public classtype getSomething() { return \_something };

}

Add a note hereAdd a note hereIn [Listing 2.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122), the class identifier Generic1 is appended with a set of angle brackets. Contained within the angle brackets is the identifier classtype. Put all together, the first line of [Listing 2.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122) is saying, "The class Generic1 is a generic class that is associated with the type identifier classtype." Within the class declaration Generic1, the type identifier classtype will be recognized as another type, like the class types Integer or String. It is important to realize that the type identifier classtype can reference only other class types, not primitive types like int, long, or double.

Add a note hereAdd a note hereIn [Listing 2.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122), the type identifier classtype is used to declare the private data member \_something, and the property method getSomething(). Whenever any consumer uses the class Generic1, the property method will always have a type associated with it. The class Generic1 can be used as shown in [Listing 2.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122).

Add a note hereAdd a note here**Listing 2.32**

Add a note hereAdd a note hereGeneric1< Integer> val = new Generic1< Integer>();

Integer getval = val.getSomething();

Generic1< Integer> val2 = val;

Add a note hereAdd a note hereIn [Listing 2.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122), the generic class Generic1 is instantiated just like a regular Java class would be, with the only additional identifier Integer being the generic declaration contained by the angle brackets. The easiest way to understand the instantiation is that declarations are abstract, but instantiations must be specific. Hence, to make the class Generic1 specific, you have to associate a type with the type identifier, which is the class type Integer. The type identifier has to be the same in the declaration of the variable val, and in the instantiation of the variable val.

Add a note hereAdd a note hereThe instantiated generic class is used like any other instantiated Java class, with the exception being that an instantiated generic class is specific and has an associated type with it. Therefore, in the second line of [Listing 2.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122) the variable getval has to be declared as the type Integer, because when the method getSomething is called, the result of the method is expected to assign the type Integer. If the declaration were any other data type, then a compile error would be generated.

Add a note hereAdd a note hereThe last line of [Listing 2.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122) is used to illustrate that when a generic class is being declared, the declaration must be specific and the right type to match what it is being assigned to. There is an exception to the last line of [Listing 2.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122); this is shown in [Listing 2.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122). (Note that [Listing 2.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122) assumes that the variables used are declared in [Listing 2.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122).)

Add a note hereAdd a note here**Listing 2.33**

Add a note hereAdd a note hereGeneric1< ?> val2 = val;

val2.getSomething();

Integer intVal = ((Generic1<Integer>)val2).getSomething();

String strval = ((Generic1<String>)val2).getSomething();

Add a note hereAdd a note hereIn [Listing 2.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122), the variable val2 is declared with a question mark where the type identifier should be. The question mark indicates to the Java compiler that the type of the generic class is as yet unknown, and hence should be kept that way. The assignment of the variable val2 by the type of val will work and not cause a problem. However, what is a problem is that the variable val2 is some type of reference, where the type is unknown to the program at compile time and will be determined at runtime. This means that using the variable val2 at compile time has some restrictions.

Add a note hereAdd a note hereThe second to fourth lines in [Listing 2.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122) show the restrictions. The second line calls the method getSomething but does not assign the return value. This is allowed because there is no need to make a decision on what type is being referenced because the return value is not being assigned. At runtime, the generic variable val2 will execute and know what to do because the object instance knows how it is specified.

Add a note hereAdd a note hereProblems of knowing the specific details of the generic object arise in the third and fourth lines of [Listing 2.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122). The third and fourth lines cast the variable val2 to a specific type and then call the method getSomething. Both of the casts are legal at compile time, but one of the casts will be wrong at runtime. What will end up happening with one of the casts is that a runtime cast exception error will be generated.

Add a note hereAdd a note hereIn the declarations shown thus far, it is important to realize that multiple type identifiers can be used between the angle brackets. Each type identifier would have to be separated by a comma.

### Add a note hereAdd a note hereGeneric Methods and Wildcards

Add a note hereAdd a note hereThe question mark introduced in [Listing 2.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122) is very powerful in Java Generics. The question mark is a wildcard character that indicates to the compiler that the generic type is being used with a type identifier that is as yet unknown. Using that sort of identifier allows a developer to define what kind of class to use in which context, as shown in [Listing 2.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122).

Larger View

Add a note hereAdd a note here**Listing 2.34**

Add a note hereAdd a note hereclass Generic3 {

public void method1( Generic1<?> obj) {

return;

}

public <classtype> void method2( Generic1< classtype> obj) {

return;

}

}

Add a note hereAdd a note hereIn [Listing 2.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122), the class Generic3 has been declared with two methods: method1 and method2. They are identical methods but are declared in two different variations. To prove to yourself that both methods are identical, remove the number one and two from each identifier, and the compiler will come back with an identical method error.

Add a note hereAdd a note hereThe method method1 is declared with the first parameter of the generic type Generic1, and the question mark used in the type identifier indicates that the specific type will be determined at runtime. The method method2 is declared with the angle brackets after the public scope identifier and before the return type void identifier. Within the angle brackets is the generic type associated with the method. When the method is declared this way, the method is considered to be a generic method. Then, the first parameter is the generic class Generic1, but instead of a question mark the type identifier classtype is used.

Add a note hereAdd a note hereIn either case of the declaration, the specific types are known only when the method is used in some other place in the source code. [Listing 2.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122) shows how method1 and method2 could be consumed.

Add a note hereAdd a note here**Listing 2.35**

Add a note hereAdd a note hereGeneric3 cls = new Generic3();

cls.method1( new Generic1< Integer>());

cls.method2( new Generic1< Integer>());

Add a note hereAdd a note hereIn [Listing 2.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122), the class Generic3 is instantiated like any Java class, because the class Generic3 does not have the angle brackets to define it as a generic class. Then, to call the methods method1 and method2, the generic class type Generic1<Integer> is used. Both methods are called the same, proving that both method signatures are identical.

Add a note hereAdd a note hereThe reason for using the different method signatures lies in the implementation of each of methods which are shown in [Listing 2.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122).

Larger View

Add a note hereAdd a note here**Listing 2.36**

Add a note hereAdd a note hereclass Generic3 {

public void method1( Generic1<?> obj) {

obj.getSomething();

return;

}

public <classtype> void method2( Generic1< classtype> obj) {

classtype val = obj.getSomething();

return;

}

}

Add a note hereAdd a note hereIn [Listing 2.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122), we again encounter the problem of not knowing what the type of the object was, as first shown in [Listing 2.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122). In the implementation of the method method1, the method getSomething returns a specific type. The question mark was used in the declaration of method1, so the specific type of the generic class Generic1 is not known. The programmer would have to do much thinking, searching, and juggling to figure out the type identifier.

Add a note hereAdd a note hereIn the case of the method method2, the generic type is known because the generic type identifier classtype is declared. Therefore, in the implementation of method2 when the variable val is being assigned, both the variable val and the method getSomething have the same type, which causes no problems.

### Add a note hereAdd a note hereUsing Inheritance with Generics

Add a note hereAdd a note hereJava generics can inherit and determine what kind of classes can be used. In the simplest example, classes can inherit from other generic classes. In [Listing 2.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122), a generic interface is defined, which is then implemented by two classes.

Add a note hereAdd a note here**Listing 2.37**

Add a note hereAdd a note hereinterface Interface1< classtype> {

void func( classtype param);

}

class Generic7 implements Interface1< Integer> {

public void func( Integer param) {

}

}

class Generic11< clstyp> implements Interface1< clstyp> {

public void func( clstype param) {

}

}

Add a note hereAdd a note hereIn [Listing 2.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122), the interface Interface1 is a generic interface that uses the type identifier classtype in one of the methods func defined by the interface. The interface is declared using a simple generic declaration.

Add a note hereAdd a note hereThe class Generic7 in [Listing 2.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122) is a regular Java class that implements the generic interface Interface1. As the class Generic7 is a regular Java class, the generic interface, which is the class type Integer, must be specialized. When the class Generic7 implements the methods of the Interface1, the type identifier is Integer, and therefore the method func must be declared with a parameter of the type Integer.

Add a note hereAdd a note hereThe class Generic11 is different because it is a generic Java class, with the type identifier clstyp. When the generic class implements the methods of Interface1, you can use a type identifier like in the case of the class Generic7, or use a class specialization, which is shown by the method implementation of the generic class Generic11. In either case, the identifier must already exist. For example, in [Listing 2.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122), the type identifier used for generic interface Interface1 in the context of the generic class Generic11 is clstyp. The type identifier clstyp is declared as a type identifier by the parent class Generic11. At no time could the type identifier for the generic interface Interface1 be some other randomly defined identifier. The Java compiler in that case would spit out an error saying that the identifier had not been defined.

Add a note hereAdd a note hereThere is a catch to inheritance in that inheritance cannot be used to bypass the Java multiple inheritance restriction. For example, the declaration in [Listing 2.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122) would be illegal.

Add a note hereAdd a note here**Listing 2.38**

Add a note hereAdd a note hereinterface Interface1< classtype> {

}

class Cls1 implements Interface1< Integer> {

}

class Cls2 extends Cls1 implements Interface1< String> {

}

Add a note hereAdd a note hereIn [Listing 2.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122), the interface Interface1 is declared as a generic interface. The class Cls1 implements Interface1 and uses the class type Integer as the type identifier. The class Cls2 extends the class Cls1 as well as implements Interface1, but it uses the class type String as the type identifier. The class Cls2 uses a different type identifier as the class Cls1, so there are two different implementations being inherited, which is not legal with Java Generics.

### Add a note hereAdd a note hereUsing Bounded Classes in Generic Declarations

Add a note hereAdd a note hereWhen you use inheritance in the declaration of the generics, the purpose is to create a specialized type in the context of a generic class or method. An example of using inheritance in a declaration is shown in [Listing 2.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122).

Add a note hereAdd a note here**Listing 2.39**

Add a note hereAdd a note hereclass Generic1< obj> {

obj \_something;

public obj getSomething() { return \_something; }}

class Generic2< A extends Generic1< C >, C> {

A \_something;

Generic2() {

something.\_something = null;

}

}

class Generic5< A extends Generic1< ? > {

A \_something;

Generic5() {

something.\_something = null;

}

}

Add a note hereAdd a note hereIn [Listing 2.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122), the class Generic1 is defined as a generic class, and it defines a private data member something and property getSomething. The class Generic2 is a generic class declaration as well, except that the first type identifier A extends the generic class Generic1. This way of defining a type identifier says that when the generic class Generic2 is used, it must subclass the generic class Generic1 somewhere in the inheritance hierarchy. The declaration of the class Generic1 is a tight bind because the type must be the same as the second type identifier C. The declaration of the generic class Generic5 is also saying that the type identifier must inherit from Generic1, except the type identifier has been replaced with a question mark, meaning that any type identifier is acceptable.

Add a note hereAdd a note here[Listing 2.40](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122) is an example of how to use the generic classes Generic2 and Generic5.

Larger View

Add a note hereAdd a note here**Listing 2.40**

Add a note hereAdd a note hereclass AnotherBase extends Generic1< String> {

}

class YetAnotherBase extends AnotherBase {

}

Generic2< YetAnotherBase, String> val1 = new Generic2< YetAnotherBase, String>();

Generic5< AnotherBase > val2 = new Generic2< YetAnotherBase>();

Add a note hereAdd a note hereIn [Listing 2.40](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122), the class AnotherBase subclasses the generic class Generic1, which is subclassed by the class YetAnotherBase. The classes AnotherBase and YetAnotherBase could be used in conjunction with the generic classes Generic2 and Generic5. The last two lines of [Listing 2.40](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122) show how to instantiate the generic classes Generic2 and Generic5.

Add a note hereAdd a note hereBesides using the keyword extends in [Listing 2.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122), you can use the keyword super, as shown in [Listing 2.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122). (Note that in [Listing 2.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=769110122) the type identifier B is the bounding type.)

Add a note hereAdd a note here**Listing 2.41**

Add a note hereAdd a note here? super B

### Add a note hereAdd a note hereUsing Generics

Add a note hereAdd a note hereWhen you're developing applications, Generics will help you solve certain problems like creating lists in a more consistent and type-safe manner. Especially interesting is the ability of Java Generics to enforce boundaries. This is useful because it means that a passed-in-type identifier will, at the minimum, implement a specific interface or extend a specific class. The ability to define a boundary when tied into the Commons Bridge makes it simpler to build generic interface-driven code than by using the Commons Bridge alone. Note, however, that all of what we have explained here could change, since all of the materials presented are alpha status.

## **Summary**

Add a note hereAdd a note hereIn this chapter, we showed that development is not that difficult and that the Jakarta sources prove it. We started the chapter started out by explaining the four-and-a-half levels of granularization. Granularization is the core concept of developing good code. As you saw, there are different levels of granularization, which is a core concept of this book. The chapter then moved on to discuss the two best practices that you should use to develop a large part of your code. The Jakarta programmers use these techniques—and the Commons Sense class types—extensively. The Commons Sense class types are attempts to identify specific class types that are programmed using a specific logic. The techniques shown in this chapter are used extensively in the Commons and have been proven to work.

## **Questions**

Add a note hereAdd a note here**2.1:** Define the scenarios where a nested class and package scope class are used.

Add a note hereAdd a note here**2.2:** Some people say that components need to be defined using interfaces. Explain why this statement may be true or false.

Add a note hereAdd a note here**2.3:** Design a calculator component using the Commons Bridge. Add some methods that would seem appropriate.

Add a note hereAdd a note here**2.4:** Extend the calculator component to include generic operations that are implemented outside of the calculator component. The calculator code knows only about the intentions of the extensions, not the implementation of the extensions. The idea is to be able to load an extension, use it, and then replace it when necessary with another extension. You will need to recode the calculator code when extensions are subbed in. A sample extension could be memory operations like M+ (Memory Add), M- (Memory Subtract), or MRC (Memory recall). The generic operations operate on the same data that the core calculator component operates on. This means if the core calculator component uses numbers, the extensions use the same numbers. Tie the extensions to the core calculator component using hard-coded references.

Add a note hereAdd a note here**2.5:** A new requirement is made to the calculator component. There are multiple variations of the calculator. Some people want a business calculator, while others want a scientific one. Rewrite the calculator component that implements all of the features defined in questions 2.3 and 2.4. Don't forget to name the component(s).

**Add a note hereChapter 3: Instantiating Objects**

**Add a note hereAdd a note here****Overview**

Add a note hereAdd a note hereIn this chapter, we will do the following:

* Add a note hereAdd a note hereDescribe the problem of creating instances of objects
* Add a note hereAdd a note hereDefine a basic factory
* Add a note hereAdd a note hereUse the *Collections* package to create an instance of an object using a formal factory
* Add a note hereAdd a note hereUse the *Discovery* package to discover resources and service implementations
* Add a note hereAdd a note hereUse the *Launcher* package to create child processes using a configurable launcher file

## **The Purpose of This Chapter**

Add a note hereAdd a note hereCreating an object is not difficult; you do so using the keyword new. Is that the extent of creating an object? What about creating an object based on a dynamic program condition? How about creating an object based on some administrative configuration files? In each of these situations, you cannot use the keyword new because you don't know which class to instantiate. The task of this chapter is to outline the different ways to instantiate a class depending on common scenarios that a programmer will encounter.

## **Creating an Object**

Add a note hereAdd a note hereThe simplest way to instantiate a Java object is shown in [Listing 3.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=295261080).

Add a note hereAdd a note here**Listing 3.1**

Add a note hereAdd a note here public static void main(String args[]) {

Main cls = new Main();

}

Add a note hereAdd a note hereIn [Listing 3.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=295261080), the class Main is instantiated using the new keyword. Once the object has been instantiated, the developer can manipulate the object however he pleases. Might this pose a problem? Well, yes; objects are allocated using a hard reference, in that the consumer references directly the class to manipulate. In the [previous chapter](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=42#42), we learned that the Commons Bridge is a good practice to follow when you're writing code, but we didn't show how to instantiate the Commons Bridge classes. [Listing 3.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=295261080) illustrates a very simple example of the Commons Bridge.

Add a note hereAdd a note here**Listing 3.2**

Add a note hereAdd a note hereinterface InterfaceToBeShared {

void availability();

}

class SomeFunctionality implements InterfaceToBeShared {

public void availability() {

System.out.println( "Yes we have bananas today");

}

}

Add a note hereAdd a note hereIn [Listing 3.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=295261080), the interface InterfaceToBeShared is an interface that will be used globally. The class SomeFunctionality implements the interface InterfaceToBeShared. A consumer of the classes can therefore use the interface InterfaceToBeShared if he requires the functionality of the class SomeFunctionality. [Listing 3.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=295261080) shows the simplest way to instantiate the classes defined in [Listing 3.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=295261080).

Add a note hereAdd a note here**Listing 3.3**

Add a note hereAdd a note hereInterfaceToBeShared intrf = new SomeFunctionality();

intrf.availability();

Add a note hereAdd a note hereThe instantiation in [Listing 3.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=295261080) is simple and logical, but it shows a huge problem. To be able to use the interface InterfaceToBeShared, the class SomeFunctionality has to be instantiated. The advantage of abstracting the interface from the implementation is lost in that single instantiation. The instantiation of the interface requires a reference to the class SomeFunctionality. The rest of this chapter shows how to get around the problem of a direct class reference.

## **Creating Objects Using a Factory**

Add a note hereAdd a note hereThe *Factory* pattern is a solution to the problem of not having to use the new keyword. You use the *Factory* pattern when you want to create an instance of an interface or abstract class using patterns. A helper class will instantiate an object for you, saving you the trouble of instantiating it yourself. The helper class creates an implicit instantiation where the consumer simply uses the helper class and expects everything to work properly. In the simplest case, the indirection would appear similar to [Listing 3.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=391873952).

Add a note hereAdd a note here**Listing 3.4**

Add a note hereAdd a note hereclass SharedFactory {

public static InterfaceToBeShared createObject() {

return new SomeFunctionality();

}

}

Add a note hereAdd a note hereIn [Listing 3.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=391873952), the class SharedFactory has a static method createObject that instantiates the specific object and then returns the supported interface. Rewriting [Listing 3.3](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=170#170) to use the factory would result in [Listing 3.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=391873952).

Add a note hereAdd a note here**Listing 3.5**

Add a note hereAdd a note hereInterfaceToBeShared interf =

SharedFactory.createObject();

interf.availability();

Add a note hereAdd a note hereIn [Listing 3.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=391873952), to get an interface instance the consumer of the interface InterfaceToBeShared would only need to use the shared class SharedFactory and then call the method createObject. We now ask ourselves if we have achieved anything. In Java programming terms, nothing much has changed. The class SomeFunctionality has been replaced with the class SharedFactory, which means a hard reference still exists. However, complexity has been added because there is now an intermediate class. In a way, we might think that it's no longer possible to maintain the code.

Add a note hereAdd a note hereThe code is not more complex, but simpler. The reason is related to the fact that, in terms of linkage change, the class SharedFactory will be written once and will never change again. In the implementation of SharedFactory, a different implementation may be instantiated, but the client code does not care about this. The client that uses the classes cares only that the linkage changes as little as possible. [Listing 3.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=391873952) is an illustration of when the linkage changes very little, even though the implementation does change. As you will remember from [Chapter 2](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=42#42), the key to good code is to be able to separate the linkage from the implementation, as demonstrated by the Commons Bridge best practice.

## **Creating Multiple Objects Using a Factory**

Add a note hereAdd a note hereIn some frameworks, it is acceptable to create one factory class for every implementation that is to be exposed. However, doing so can very quickly become tedious. In addition, a hard reference to the implementation is created indirectly through the use of the helper class. For example, imagine an implementation class called MyActionImplementation; the factory class helper might be called MyActionFactory. If a new class such as MyActionNewImplementation is defined, will the factory class be called MyActionNewFactory? If the naming of the factory follows the naming of the implementation, the client locks into specific implementations, which we hoped to avoid in the first place. In code terms, the problem of new implementations is shown in [Listing 3.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=288455935).

Larger View

Add a note hereAdd a note here**Listing 3.6**

Add a note hereAdd a note hereinterface InterfaceToBeShared {

void availability();

}

class SomeFunctionality implements InterfaceToBeShared {

public void availability() {

System.out.println( "Yes we have bananas today");

}

}

class SomeFunctionality2 implements IntefaceToBeShared {

public void availability() {

System.out.println( "Yes we still have bananas today");

}

}

class SharedFactory {

public static InterfaceToBeShared createObject() {

//return new SomeFunctionality(); correct ???

//return new SomeFunctionality2(); correct ???

}

}

Add a note hereAdd a note hereIn [Listing 3.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=288455935), the interface InterfaceToBeShared is still the public interface. However, this time, we have the original implementation, SomeFunctionality, and the new one, SomeFunctionality2. The latter class is a new implementation that is supposed to be used instead of the original implementation. The problem is in the class SharedFactory because the createObject method needs to be updated. Specifically, which object the method createObject should return is a mystery.

Add a note hereAdd a note hereThe quick and simple answer on which implementation to instantiate is the newest version, but that is incorrect and can cause problems. The problem is that there are bug fixes and system updates. A bug fix typically will not require you to define a new class. A system update will require more complicated changes, like the ones proposed in [Listing 3.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=288455935). However, the complicated update is not supposed to impact the client. Remember, this goes back to the Commons Bridge best practice of separating interface and implementation. This means that returning an instance of the class SomeFunctionality2 could cause some consumers to react incorrectly because some bugs are fixed. Of course, one can argue that the consumer took into account how the interface reacted to certain conditions, which broke the Commons Bridge best practice. This is absolutely correct, but the problem is that we can argue like this for days and still not be closer to solving the problem of which class instance to return.

Add a note hereAdd a note hereThe correct approach generally is not to create a specific class factory for an instance. Instead, consider the factory as a task-specific undertaking. If you take this approach, it becomes simpler to insert new versions and at times use specific versions, as shown in [Listing 3.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=288455935).

Larger View

Add a note hereAdd a note here**Listing 3.7**

Add a note hereAdd a note herefinal class MultiObjectSharedFactory {

static final int INITIAL\_VERSION = 1;

static final int UPDATED\_VERSION = 2;

public static InterfaceToBeShared anImplementation(

int version) {

switch( version) {

case MultiObjectSharedFactory.INITIAL\_VERSION:

return new SomeFunctionality();

case MultiObjectSharedFactory.UPDATED\_VERSION:

return new SomeFunctionality2();

}

return null;

}

public static InterfaceToBeShared anotherImplementation() {

return new AnotherFunctionality();

}

}

Add a note hereAdd a note hereIn [Listing 3.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=288455935), the class SharedFactory has been changed and it called the class MultiObjectSharedFactory to indicate the varying capabilities of the factory. The class MultiObjectSharedFactory has its declaration changed to final, indicating that nobody can subclass the factory. Within the class are two methods to instantiate different implementations that support the interface InterfaceToBeShared. The method anotherImplementation is similar in purpose to that of the createObject method of the class SharedFactory. The method anImplementation is entirely different than previous object creation methods because it requires a parameter, which is an integer value. The parameter represents a version number of which object to instantiate. For the old implementations, the initial version will be used. However, the newer version will be used for the new implementations.

Add a note hereAdd a note hereWhen you use this approach to solve the factory problem, the objective is not to look at the details of the implementation. Instead, the objective is to notice that the factory is a task-based class used to instantiate specific objects. This means a factory for one subsystem will not be identical to a factory for another subsystem. This problem of figuring out the correct implementation of a factory is an issue regardless of the way that the object is instantiated. In later sections of this chapter, we will show you different factory approaches, but they do not solve the problem of cross-referencing the instance with the client. The different factory approaches make it simpler to develop a generic framework.

## **Creating Objects Using the *Collections* Factory**

Add a note hereAdd a note hereThe Commons project *Collections* contains a number of extensions that help Java programmers carry out everyday programming tasks. One part of the library is the factory helper classes. Consider [Listing 3.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590), which implements a modified version of the class SharedFactory.

Add a note hereAdd a note here**Listing 3.8**

Add a note hereAdd a note herefinal class SharedLangFactory {

static final SomeFunctionality \_orig =

new SomeFunctionality();

public static Factory getFactory() {

return FactoryUtils.prototypeFactory( \_orig);

}

}

Add a note hereAdd a note hereIn [Listing 3.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590), the class SharedLangFactory has a single method called getFactory. The purpose of the method getFactory is to retrieve a standard interface instance of the *langs* package factory instance. Relative to the patterns-based factory, the method getFactory is a reference to the factory that will instantiate the object. This means there are two levels of indirection. Now consider [Listing 3.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590), which finishes off [Listing 3.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590).

Add a note hereAdd a note here**Listing 3.9**

Add a note hereAdd a note hereFactory fac = SharedLangFactory.getFactory();

InterfaceToBeShared intrf =

(InterfaceToBeShared)fac.create();

intrf.availability();

Add a note hereAdd a note hereIn [Listing 3.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590), the method getFactory retrieves an interface-based instance of the interface Factory. The instance fac is then used to instantiate an interface-based implementation of the interface InterfaceToBeShared using the method call create. Once the object has been instantiated, the method availability can be called.

Add a note hereAdd a note hereThis method of allocation may seem an exercise in overkill because you have to create a factory before you can use it. If you used the instantiated factory in this context, then it absolutely would be overkill. However, that is not the purpose of this library. Rather, its purpose is to be able to generically define a factory that will be used throughout the subsystem or application. Then, that factory can be defined to execute in different contexts without having the client update its code. For example, it is possible to define a factory that will either create a new instance every time or use a singleton of the implementation to instantiate.

### Add a note hereAdd a note hereTechnical Details for the *lang* Factory

Add a note hereAdd a note here[Tables 3.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590) and [3.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590) contain abbreviated details necessary to use the *lang* factory package.

**Add a note hereAdd a note hereTable 3.1:** Repository details for the *lang* factory

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note herecollections |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.collections |

**Add a note hereAdd a note hereTable 3.2:** Package and class details (legend: [collections] = org.apache.commons.collections).

|  |
| --- |
| Add a note hereAdd a note hereClass/Interface Details |
| Add a note hereAdd a note here[collections].functor.Factory | Add a note hereAdd a note hereA neutral interface used to instantiate an object |
| Add a note hereAdd a note here[collections].functor.FactoryUtils | Add a note hereAdd a note hereThe main class that contains all of the functions used to instantiate the different implementations of the Factory interface |
| Add a note hereAdd a note here[collections].functor.FactoryException | Add a note hereAdd a note hereAn exception that is thrown if the Factoryclasses have encountered a problem |

### Add a note hereAdd a note hereInstantiating a Cloned Object

Add a note hereAdd a note hereThe [Listings 3.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590) and 3.9 are functionally correct. However, they would not work because the object to be instantiated is the wrong type of object. The idea behind the *Collections* factory is to instantiate objects based on already existing instances. The *Collections* factory uses this general approach so that it's possible to define a generic default state and have that state propagated whenever the object is instantiated. As a result, a component, subsystem, or application requires an initialization stage, which initializes the objects that will be created in the execution of the program.

Add a note hereAdd a note hereA class that could be used in the *Collections* factory is shown in [Listing 3.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590).

Larger View

Add a note hereAdd a note here**Listing 3.10**

Add a note hereAdd a note herepackage com.devspace.jseng.create;

public class ObjectToBeCreated

implements InterfaceToBeShared, Cloneable {

public Object clone() throws CloneNotSupportedException {

ObjectToBeCreated cls =

(ObjectToBeCreated)super.clone();

return cls;

}

public void availability() {

System.out.println( "Yes we have bananas today");

}

}

Add a note hereAdd a note hereThere are several things to note in [Listing 3.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590). The first is that the object must be declared as public, which means nested classes and inner classes will not work. You must do this because the implementation of the clone factory requires public classes. In order for the *lang* package clone factory to work, the object to be created must have a public scoped method clone. The method clone is responsible for instantiating a new instance of the object.

#### **Implementing the Method clone**

Add a note hereAdd a note hereImplementing the method clone is simple, but the details of the implementation require great care. The purpose of cloning is to be able to create a brand-new instance of the current object. However, by itself the method clone does help the developer implement the method. For example, consider the class declaration in [Listing 3.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590).

Add a note hereAdd a note here**Listing 3.11**

Add a note hereAdd a note hereclass NonDeepCloneExample implements Cloneable {

private ArrayList \_array;

public Object clone() throws CloneNotSupportException {

NonDeepCloneExample ex = new NonDeepCloneExample();

ex.\_array = \_array;

return ex;

}

}

Add a note hereAdd a note hereIn [Listing 3.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590), the method clone allocates a new instance of the object NonDeepCloneExample. Then, the data member \_array is assigned to the data member \_array of the newly allocated object ex. The problem with this way of copying is that it is not incorrect in a technical sense. Doing an assignment is OK because, once the local instance object goes out of scope, the cloned instance will still be able to reference the array list. The problem is at a higher level. An assignment of the array list means that two object instances will point to the same array list. This means any changes made by one instance will automatically be reflected in the other instance, which we don't want. A quick and simple solution would be to change the implementation of the method clone to what is shown in [Listing 3.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590).

Add a note hereAdd a note here**Listing 3.12**

Add a note hereAdd a note herepublic Object clone() throws CloneNotSupportException {

NonDeepCloneExample ex = new NonDeepCloneExample();

ex.\_array = (ArrayList)\_array.clone();

return ex;

}

Add a note hereAdd a note hereThe modified version of the method clone in [Listing 3.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590) would seem to be the ideal solution because the method clone of the array list is used. However, this may not be true in reality. The reality of what happens lies in the implementation of the method ArrayList.clone, as shown in [Listing 3.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590).

Add a note hereAdd a note here**Listing 3.13**

Add a note hereAdd a note herepublic Object clone() {

try {

ArrayList v = (ArrayList)super.clone();

v.elementData = new Object[size];

System.arraycopy(elementData,

0, v.elementData, 0, size);

v.modCount = 0;

return v;

} catch (CloneNotSupportedException e) {

// This shouldn't happen

throw new InternalError();

}

}

Add a note hereAdd a note hereIn Listing of 3.13, which was taken from the class ArrayList, the method clone creates a new array to hold the object references. However, instead of copying the individual array elements, only the object references are copied. This means that while the class method ArrayList.clone creates a new instance of the class ArrayList, the elements are still referenced by two instances of ArrayList. Now, instead of referencing the same ArrayList instance, both instances of the class NonDeepCloneExample are referencing the same set of objects referenced by two different instances of the ArrayList. The lesson is that the overall idea of the method clone works, except that most clone methods are shallow clone implementations.

#### **Implementing a Deep Clone**

Add a note hereAdd a note hereA *shallow clone* is when an object clones itself as well as the data members that the object references. However, with a shallow clone, the references of the individual object references are not cloned. To convert a shallow clone into a deep clone, each individual object has to be cloned, as shown in [Listing 3.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590).

Larger View

Add a note hereAdd a note here**Listing 3.14**

Add a note hereAdd a note hereclass NonDeepCloneExample implements Cloneable {

private ArrayList \_array;

public Object clone() throws CloneNotSupportedException {

NonDeepCloneExample ex = new NonDeepCloneExample();

ex.\_array = (ArrayList)\_array.clone();

for( int c1 = 0; c1 < \_array.size(); c1 ++) {

OtherClass obj = (OtherClass)\_array.get( c1);

ex.\_array.add( obj.clone());

}

return ex;

}

}

Add a note hereAdd a note hereIn the modified version of the class method NonDeepCloneExample.clone shown in [Listing 3.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590), the individual array list elements are iterated and cloned. [Listing 3.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590) makes extensive use of the method clone. Often, in situations similar to that shown in [Listing 3.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590), the individual classes are allocated and then initialized. That methodology is fine and works. However, in the context of the *Collections* factory, where it is required to support the method> clone, it is better to be consistent and use the method clone everywhere. Not being consistent could cause specific bugs to be introduced with version changes.

Add a note hereAdd a note hereOne modification in [Listing 3.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590) could have been to use the class Object instead of the class OtherClass when extracting the elements from the array list. The problem with this approach is that the method clone is protected and hence not publicly available. To make [Listing 3.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590) work, you must cast the class to a specific type, which means defining a generic interface. The generic interface would support the method clone and be public scoped. The generic interface would be used to clone an object, so the generic interface would have to support the interface Cloneable. The interface Cloneable itself does not have any methods, but it's required when you use the method call super.clone. [Listing 3.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590) shows how to define the generic interface.

Add a note hereAdd a note here**Listing 3.15**

Add a note hereAdd a note hereinterface BaseOfAllInterfaces extends Cloneable

{

public Object clone() throws CloneNotSupportedException;

}

Add a note hereAdd a note hereWhen using this interface, you can cast the instance to the interface BaseOfAllInterfaces and use that instead of the class OtherClass in [Listing 3.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590).

Add a note hereAdd a note hereOne last note before we discuss the other parts of the *Collections* factory. Traditionally, you must implement the interface Cloneable to indicate that an object instance can be instantiated. The *Collections* factory is flexible and does not require explicit support of the interface Cloneable. Not using the Cloneable interface would require using the new keyword in [Listing 3.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590) instead of super.clone.

### Add a note hereAdd a note hereInstantiating a Singleton Object

Add a note hereAdd a note hereIn [Listing 3.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590), a class was instantiated using the class method FactoryUtils.prototypeFactory to instantiate an object instance. With each usage of the Factory.create method, another object was instantiated. However, it is sometimes desirable to return the same object, which is called a *singleton*. This allows you to define an instance of a class that is referenced whenever the interface method Factory.create is called. If we modified [Listing 3.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590) to add singleton functionality, you would get [Listing 3.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590).

Add a note hereAdd a note here**Listing 3.16**

Add a note hereAdd a note herefinal class SharedLangFactory {

static final ObjectToBeCreated \_cloned =

new ObjectToBeCreated();

static final ObjectToBeCreated \_singleton =

new ObjectToBeCreated();

public static Factory getClonedFactory() {

return FactoryUtils.prototypeFactory( \_cloned);

}

public static Factory getSingletonFactory() {

return FactoryUtils.constantFactory( \_singleton);

}

}

Add a note hereAdd a note hereThe class SharedLangFactory in [Listing 3.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590) has changed fundamentally from what it was in [Listing 3.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590). The class has become more like a multiple objects factory, described in [Chapter 2](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=42#42). The method getClonedFactory is using the cloned instance method. What concerns us right now is the method getSingletonFactory. In the implementation of the method, the class method FactoryUtils.constantFactory is called. The parameter passed to the class method FactoryUtils.constantFactory is another static object. However, the data member \_singleton is an important object. Whenever the interface method Factory.create is called, the reference to the data member \_singleton is returned.

### Add a note hereAdd a note hereKnowing What Is a Singleton and What Is Not

Add a note hereAdd a note hereKnowing that you can create a singleton from the *lang* factory is good but then again you need to know what is and is not a singleton. In [Listing 3.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590), two data members—\_cloned and \_singleton—are defined. Each data member is a singleton in the class SharedLangFactory. When the methods getClonedFactory or getSingletonFactory are called, the interface instances of Factory are separate instances. This means that five calls to the method getClonedFactory yield five different instances of the interface Factory.

Add a note hereAdd a note hereHowever, these different instances are misleading. Consider the implementation of ConstantFactory shown in [Listing 3.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590).

Add a note hereAdd a note here**Listing 3.17**

Add a note hereAdd a note here private static class ConstantFactory

implements Factory, Serializable {

private final Object iConstant;

private ConstantFactory(Object constant) {

super();

iConstant = constant;

}

public Object create() {

return iConstant;

}

}

Add a note hereAdd a note hereThe constructor of the class ConstantFactory in [Listing 3.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590) requires a parameter, which is the object to be used as a basis for the factory. The data member iConstant references the passed-in object. Relating this back to [Listing 3.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590), the object constant iConstant is the data member \_singleton. Therefore, if five constant factories are instantiated, each iConstant will reference the same reference that the data member \_singleton references.

Add a note hereAdd a note hereThis coding pattern does impact how the *Collections* factory operates. For example, if the ConstantFactory is passed a different object instance, then a different singleton is created. In addition, in the case of the clone factory, the method clone may need synchronization, since several objects may be cloning the same instance. The only place where this is critical is if the clone operation requires access to a state that may not be available when the object is to be cloned. Of course, though, programmers would never do that since it's a bad idea to clone a state that morphs. Nonetheless, it is important when you use the *lang* factory that the object used as a basis for the individual factories is a global item, as we saw in [Listing 3.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590).

### Add a note hereAdd a note hereInstantiating a Reflected Object

Add a note hereAdd a note hereThe last type of instantiation that you can perform using the *Collections* factory is the reflection instantiation. The reflection approach is the simplest and least complicated. If we modify the class SharedLangFactory that we saw in [Listing 3.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590) and add the reflection code, we'll get what's shown in [Listing 3.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590).

Add a note hereAdd a note here**Listing 3.18**

Add a note hereAdd a note herefinal class SharedLangFactory {

static final ObjectToBeCreated \_cloned =

new ObjectToBeCreated();

static final ObjectToBeCreated \_singleton =

new ObjectToBeCreated();

static final ObjectToBeCreated \_reflected =

new ObjectToBeCreated();

public static Factory getClonedFactory() {

return FactoryUtils.prototypeFactory( \_cloned);

}

public static Factory getSingletonFactory() {

return FactoryUtils.constantFactory( \_singleton);

}

public static Factory getReflectionFactory() {

return FactoryUtils.reflectionFactory(

\_reflected.getClass());

}

}

Add a note hereAdd a note hereIn [Listing 3.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=539023590), the data member \_reflected and method getReflectionFactory are the new factory classes that use reflection. To use the reflection factory, an object is still instantiated. However, instead of passing in a reference to the instantiated object, a reference to the class object is passed. Another way of passing the class would be to use the class definition (as defined by the class type) using the data member ObjectToBe-Created.class. The reflection factory uses the built-in reflection techniques of the Java Virtual Machine (VM), which is why this method is the simplest of all *lang* factory methods.

## **The Service *Discovery* Package**

Add a note hereAdd a note hereWithin the Commons library is a library called *Service Discovery*, or *Discovery* for short. The purpose of the *Service Discovery* package is defined as follows (source: Discovery Proposal): The *Discovery* package is a very formalized approach to instantiating an object. In Java, you can use the Service Discovery technique built into the JDK. The *Discovery* package enhances the basic technique into a full-blown configurable service provider interface, and users of this package should keep this in mind. It is not easier to instantiate objects using the *Discovery* package than using the factory. However, the *Discovery* package makes it easier for a developer to develop an application that is configured by an administrator.

### Add a note hereAdd a note hereLoading a Plug-in

Add a note hereAdd a note hereImagine building a dynamic Web Server, meaning that it will support the dynamic loading of components to serve requests. To serve requests, you would use a configuration file that would load the mappings. [Listing 3.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) shows a sample mapping.

Add a note hereAdd a note here**Listing 3.19**

Add a note hereAdd a note here<mappings>

<request uri="/home"

class="com.devspace.jseng.create.ObjectToBeShared" />

<request uri="/user"

class="com.devspace.jseng.create.AnotherObjectToBeCreated"

/>

</mappings>

**Rationale**

Add a note hereAdd a note hereThe *Discovery* Component is about discovering, or finding, implementations for pluggable interfaces. It provides facilities instantiating classes in general, and for lifecycle management of singleton (factory) classes.

Add a note hereAdd a note hereFundamentally, *Discovery* locates classes that implement a given Java interface. The discovery pattern, though not necessarily this package, is used in many projects including JAXP (SaxParserFactory and others) and commons-logging (LogFactory). By extracting this pattern, other projects can (re)use it and take advantage of improvements to the pattern as Discovery evolves.

Add a note hereAdd a note here*Discovery* improves over previous implementations by establishing facilities for working within managed environments. These allow configuration and property overrides without appealing to the global System properties (which are scoped across an entire JVM)."

Add a note hereAdd a note hereIn [Listing 3.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935), there are two XML tags that contain a mapping of a request to a Java class name. When the request /home is received, the class com.devspace.jseng.create. ObjectToBeShared will be instantiated and called. The process of instantiating the class is what the *Discovery* package does. The details of loading and parsing this configuration will be discussed in [Chapter 8](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=632#632). In the actual implementation, the Web Server will not use the class directly, but rather an interface that both classes in [Listing 3.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) implement.

Add a note hereAdd a note hereIn simplest terms, you would think that the *Discovery* package is absolute overkill because doing a dynamic class load in Java is a relatively simple operation. However, there is another situation to consider. Imagine that the Web server loads both classes defined in the mapping of [Listing 3.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935). The classes loaded by the request /home load version 1.0 of the class libraries package xyz. However, the classes loaded by the request /request load version 1.2 of the class libraries package xyz. We have a dilemma because there are two versions of the same library. The class loader at this point becomes very complicated. This is where the *Discovery* package helps you. The *Discovery* package does not create the different class loaders per se, but it helps you set up the class loaders and then will automatically manage and load the right class.

Add a note hereAdd a note hereLet's finish off [Listing 3.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935); the way to load and instantiate the class is shown in [Listing 3.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935).

Add a note hereAdd a note here**Listing 3.20**

Add a note hereAdd a note hereimport

org.apache.commons.discovery.tools.DiscoverClass;

DiscoverClass discoverClass = new DiscoverClass();

InterfaceToBeShared interf =

(InterfaceToBeShared)discoverClass.newInstance(

InterfaceToBeShared.class,

"com.devspace.jseng.create.ObjectToBeCreated");

interf.availability();

Add a note hereAdd a note hereIn [Listing 3.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935), the utility class DiscoverClass makes it simpler to use the *Discovery* package. The method newInstance is used to instantiate a class. The type is determined by the first parameter InterfaceToBeShared.class. However, since the type is an inter- face, it cannot be instantiated directly. The second parameter, "com.devspace.jseng. create.ObjectToBeCreated", is the default class that will be instantiated if another implementation cannot be found. Since [Listing 3.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) is simple, the default class will be instantiated. Once the class has been instantiated, the returned object is type cast to the interface InterfaceToBeShared. Finally, the interface method interf.availability can be called just like another interface instance.

### Add a note hereAdd a note hereTechnical Details for the *Discovery* Package

Add a note hereAdd a note here[Tables 3.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) and [3.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) contain an abbreviated details necessary to use the *Discovery* package.

**Add a note hereAdd a note hereTable 3.3:** Repository details for the *Discovery* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note herediscovery |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.discovery, org.apache.commons.discovery.resource, org.apache.commons.discovery.tools; org.apache.commons.discovery.jdk (optional) |

**Add a note hereAdd a note hereTable 3.4:** Package and class details (legend: [discovery] = org.apache.commons.discovery).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[discovery].tools.DiscoverClass | Add a note hereAdd a note hereA utility class used to find a specific service that implements a specific interface. |
| Add a note hereAdd a note here[discovery].tools.Singleton | Add a note hereAdd a note hereA utility class used to find a specific service that implements a specific interface, except that the returned instance will be a singleton service. |
| Add a note hereAdd a note here[discovery].tools.ClassUtils | Add a note hereAdd a note hereA utility class used to help find out specific attributes regarding a class, such as package name. |
| Add a note hereAdd a note here[discovery].tools.Properties | Add a note hereAdd a note hereA class used by the *Discovery* library to store certain service definitions and their respective implementation. |
| Add a note hereAdd a note here[discovery].tools.ManagedProperties | Add a note hereAdd a note hereA class used by the *Discovery* library to store certain service definitions and their respective implementation except that the class is a static global class. |
| Add a note hereAdd a note here[discovery].resource.ClassLoaders | Add a note hereAdd a note hereA class used to store and reference a specific set of class loaders that can be created at runtime. The combination of class loaders managed by the ClassLoaders class can be reused for later class instantiations. |
| Add a note hereAdd a note here[discovery].ResourceClass | Add a note hereAdd a note hereA class that contains a reference to the service class found by the *Discovery* framework. |
| Add a note hereAdd a note here[discovery].ResourceClassIterator | Add a note hereAdd a note hereA class used to iterate the ResourceClasses that are found. |
| Add a note hereAdd a note here[discovery].resource.DiscoverResources | Add a note hereAdd a note hereA class used to find a specific set of resources that is helpful for finding service descriptors. |
| Add a note hereAdd a note here[discovery].Resource | Add a note hereAdd a note hereA class used to encapsulate a found resource. |
| Add a note hereAdd a note here[discovery].ResourceIterator | Add a note hereAdd a note hereA class used to iterate the Resources found. |
| Add a note hereAdd a note here[discovery].jdk.JDKHooks | Add a note hereAdd a note hereA very useful utility class that encapsulates the class loading functionality of the JDK. |

### Add a note hereAdd a note hereUsing Service Descriptors

Add a note hereAdd a note here[Listing 3.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) showed how to instantiate and find a service based on a default object. [Listing 3.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) shows you how to use service descriptors to find and load a class.

Add a note hereAdd a note here**Listing 3.21**

Add a note hereAdd a note hereorg.apache.commons.discovery.log.SimpleLog.setLevel(2);

DiscoverClass discover = new DiscoverClass();

Class implClass = discover.find(InterfaceToBeShared.class);

Add a note hereAdd a note hereAs in [Listing 3.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935), the DiscoverClass in [Listing 3.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) is instantiated. This time, though, instead of instantiating a new instance of the interface InterfaceToBeShared, the class descriptor is located. The method find returns an object of type Class, which can be used to instantiate a class instance. In this case, the class Class serves the same purpose as the interface Factory we saw in the Lang Factory section of this chapter. [Listing 3.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) assumes that certain configuration items are defined, which will be explained in later sections.

Add a note hereAdd a note hereIf you run [Listing 3.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) as is, an error will occur. You can find the cause of the error in the output that is generated by the method call SimpleLog.setLevel. The debug code that is generated is shown in [Listing 3.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935).

Larger View

Add a note hereAdd a note here**Listing 3.22**

Add a note hereAdd a note here[DEBUG] DiscoveryLogFactory - -Class meets requirements: org.apache.commons.discovery.tools.ClassUtils

[DEBUG] DiscoveryLogFactory - -Class meets requirements:

org.apache.commons.discovery.resource.names.DiscoverNamesInFile

[DEBUG] DiscoveryLogFactory - -Class meets requirements:

org.apache.commons.discovery.resource.DiscoverResources

[DEBUG] DiscoverNamesInFile - -find: fileName='META-

INF/services/com.devspace.jseng.create.InterfaceToBeShared'

[DEBUG] DiscoverResources - -find:

resourceName='com.devspace.jseng.create.InterfaceToBeShared'

[DEBUG] DiscoveryLogFactory - -Class meets requirements: org.apache.commons.discovery.resource.classes.DiscoverClasses

[DEBUG] DiscoverResources - -getNextResources: search using ClassLoader 'sun.misc.Launcher$AppClassLoader@4b222f'

[DEBUG] DiscoverResources - -getNextResources: search using ClassLoader 'sun.misc.Launcher$AppClassLoader@4b222f'

[DEBUG] DiscoverResources - -getNextResources: search using ClassLoader 'sun.misc.Launcher$AppClassLoader@4b222f'

[DEBUG] DiscoverResources - -getNextResources: search using ClassLoader 'sun.misc.Launcher$AppClassLoader@4b222f'

Add a note hereAdd a note hereIn [Listing 3.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935), find the section DiscoverNamesInFile and notice the filename, which is META-INF/services/com.devspace.jseng.create.InterfaceToBeShared. This is a rather odd name for a class that needs to be loaded. At first, it would seem that the compiled class file for the interface InterfaceToBeShared needs to be stored underneath the META-INF/services directory. However, that impression is incorrect because the class name is in fact a real filename.

Add a note hereAdd a note hereEarlier in this chapter we mentioned the idea of a configuration stored in XML. At that time, we didn't worry about that file. However, in this specific case, we need to concern ourselves with it because the file that is to be searched for is a file that contains a reference to a class that implements the interface InterfaceToBeShared. In this case, the name of the file is com.devspace.jseng.create.InterfaceToBeShared, the dots should not be replaced with subdirectories. [Listing 3.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) shows a sample implementation of the file.

Add a note hereAdd a note here**Listing 3.23**

Add a note hereAdd a note here# comment

com.devspace.jseng.create.ObjectToBeCreated

Add a note hereAdd a note hereIn [Listing 3.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935), a single line identifies the class that implements the interface InterfaceToBeShared. This time when [Listing 3.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) is executed, the class ObjectToBeCreated is instantiated and loaded into the class loader. It is possible to put multiple service implementations into the file in [Listing 3.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935). In a multiple service implementation scenario, only the first line is used to load a class.

### Add a note hereAdd a note hereUsing Properties

Add a note hereAdd a note hereThe service can be stored as settings in a configuration file. However, when the data is in memory, knowing where to store the data can be more complicated. In the *Discovery* package, the individual interfaces and their service providers could be stored as a property. There are two property storage techniques. The local storage mechanism is to the use the Java common library class Properties. The global storage mechanism is to use the *Discovery* package class ManagedProperties. An example of using the class Properties is shown in [Listing 3.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935).

Add a note hereAdd a note here**Listing 3.24**

Add a note hereAdd a note hereDiscoverClass discover = new DiscoverClass();

Properties props = new Properties();

props.setProperty( InterfaceToBeShared.class.getName(),

ObjectToBeCreated.class.getName());

String found[] = discover.discoverClassNames(

new SPInterface(InterfaceToBeShared.class), props);

if( found != null) {

for( int c1 = 0; c1 < found.length; c1 ++) {

System.out.println( "Found " + found[ c1]);

}

}

Add a note hereAdd a note hereIn [Listing 3.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935), the class Properties is used to store the name of the interface and the name of the class that provides the service for the interface. Both the interface and class identifier are stored as strings. Therefore, if you introduce a typo in either the interface or class identifier, the *Discovery* package will not work as intended.

Add a note hereAdd a note hereOnce the class Properties has been populated using the method setProperty, a search is performed. The search is the method discoverClassNames, which searches for the name of the services that implement a specific interface. For the method discoverClassNames, the first parameter is the class SPInterface, which references the class identifier of the interface to be found. The class SPInterface is a placeholder class for the service implementation class descriptor. In the documentation, the class SPInterface is short for "Service Provider Interface." Part of the purpose of this class is to store the descriptors about the constructor and its parameters. The second parameter of the method discoverClassNames is the instance of the class Properties. The second parameter does not need to be an actual value and could be substituted with a null value. The method discoverClassNames returns a list of registered service providers in string form.

Add a note hereAdd a note hereThis then raises another question, because the class Properties is based on the class Hashtable, meaning that there can be only one key value combination. In the case of [Listing 3.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935), the key value combination is the interface and the service that implements the interface. This means that if a second service implements the same interface, the second service identifier will overwrite the first service identifier. While this behavior is not unintended, it's questionable that it's possible to return an array of service identifiers for a specific interface, as in the case of discoverClassNames. The truth lies in the implementation of discoverClassNames, which searches multiple places for service providers. An example of another place to search is the global properties represented by the class ManagedProperties. [Listing 3.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) is an example of setting a key value combination in the class ManagedProperties.

Add a note hereAdd a note here**Listing 3.25**

Add a note hereAdd a note hereDiscoverClass discover = new DiscoverClass();

Properties props = new Properties();

props.setProperty( InterfaceToBeShared.class.getName(),

ObjectToBeCreated.class.getName());

ManagedProperties.setProperty(

InterfaceToBeShared.class.getName(),

AnotherObjectToBeCreated.class.getName());

String found[] = discover.discoverClassNames(

new SPInterface(InterfaceToBeShared.class), props);

if( found != null) {

for( int c1 = 0; c1 < found.length; c1 ++) {

System.out.println( "Found " + found[ c1]);

}

}

Add a note hereAdd a note hereThe code in [Listing 3.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) is very similar to that in [Listing 3.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935), except the additional class method call ManagedProperties.setProperty. The class ManagedProperties is essentially a class containing only static methods and data members. In [Listing 3.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935), you set the key value combination using the same technique as in the class Properties example (see [Listing 3.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935)). However, if the code in [Listing 3.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) were now executed, the method discoverClassNames would return two services that implement the same interface. It would then be up to the programmer to decide which service to use. It is important to realize that the class ManagedProperties manages global properties that are based on the thread context class loader. This means that a class loader for one thread may or may not be the same class loader for another thread. It depends on the application's runtime configuration.

### Add a note hereAdd a note hereLow-Level Resource Discovery

Add a note hereAdd a note hereThus far, all of the examples have been used to discover specific services that implement specific interfaces. However, within the *Discovery* package are several low-level classes that allow you to discover the specific files manually. Imagine that a Web Server is serving requests to multiple clients on multiple domains. Typically, different Web site administrators will manage the multiple domains. There will also be an administrator managing both domains. The problem will be when the Web site administrators want to update their respective Web sites. Do the administrators need to be informed of every change and add the items in a configuration file? This approach would require a constant update of the Java execution environment and Java classpath. The administrators would want a facility to find a specific resource within the Java classpath that can be used to carry out some other action. The resource to find could be a service descriptor file or Java class file. Once the service descriptor file has been found, further actions, such as expanding the jar file, may be carried out. [Listing 3.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) shows an example of this type of resource location.

Add a note hereAdd a note here**Listing 3.26**

Add a note hereAdd a note hereClassLoaders loaders = new ClassLoaders();

ClassLoader cl = getClass().getClassLoader();

if( cl != null)

loaders.put(getClass().getClassLoader(), true);

else

loaders.put(

JDKHooks.getJDKHooks().getSystemClassLoader(), true);

String name = "build.xml";

DiscoverResources discovery = new

DiscoverResources(loaders);

ResourceIterator iter = discovery.findResources(name);

while( iter.hasNext()) {

Resource resource = iter.nextResource();

URL url = resource.getResource();

if ( url != null ) {

System.out.println("URL = " + url.toString());

}

}

Add a note hereAdd a note hereIn [Listing 3.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935), there are two very important parts to identify. The first part is the how the various class loaders are referenced and added to the class loader collection. The second part is the iteration of the discovered resources. In the first part, the class loaders are referenced using two different techniques. The first technique is to use the Java-defined method getClass().getClassLoader(). The second technique is to use the class JDKHooks to retrieve the system class loader using the method getSystemClassLoader. The class JDKHooks is a global JDK helper class that you can use to retrieve the different class loaders. It is a useful helper class because the messiness of using the JDK class loader methods is avoided.

Add a note hereAdd a note hereOnce the individual class loaders have been referenced and added to the list managed by the class ClassLoaders, you can query for a resource. You can use the class DiscoverResources to query for the resources. The constructor of the class DiscoverResources accepts as a parameter a collection of class loaders, as we saw in [Listing 3.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935). When the method findResources is called, the list of class loaders is searched. The method findResources has a single parameter that represents the resource to be found. The resources that are found are iterators that use the class ResourceIterator. We'll discuss iterators in more detail in [Chapter 7](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=468#468).

Add a note hereAdd a note hereOnce a resource has been found, the description of the resource is stored in the class Resource. Various pieces of information can be retrieved from the class Resource. [Listing 3.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) shows the Uniform Resource Locator (URL) or location of the resource. In addition, you can discover the name of the resource or load the resource as an InputStream by using the method getResourceAsStream. It's useful to convert the resource to a stream if the resource is a configuration file that needs additional parsing.

Add a note hereAdd a note hereIf the resource to be found is a class file, then a variation of the query is to query for an individual class. The entire process of setting up the query by defining the class loaders and then querying is virtually identical in both cases. The difference is that you use the class DiscoverClasses instead of DiscoverResources. [Listing 3.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) shows an example of this.

Larger View

Add a note hereAdd a note here**Listing 3.27**

Add a note hereAdd a note hereClassLoaders loaders = ClassLoaders.getAppLoaders(

InterfaceToBeShared.class, getClass(), false);

String name = "com.devspace.jseng.create.InterfaceToBeShared";

DiscoverClasses discovery = new

DiscoverClasses(loaders);

ResourceClassIterator iter=

discovery.findResourceClasses(name);

while (iter.hasNext()) {

ResourceClass resource = iter.nextResourceClass();

try {

Class implClass = resource.loadClass();

if ( implClass != null ) {

System.out.println( "Resource is " +

implClass.getName());

return;

}

}

catch (Exception e) {

throw new RuntimeException(

"Could not load service: " + resource );

}

}

Add a note hereAdd a note hereThe difference between [Listings 3.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) and 3.27 is that [Listing 3.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) uses the class DiscoverClasses instead of DiscoverResources. Notice, though, in [Listing 3.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) how the class name is specified as a Java complete name. In contrast, the class DiscoverResources does not use a class name, but a filename. [Listing 3.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) when executed will generate output similar to [Listing 3.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935), which contains the names of the classes that were searched for.

Larger View

Add a note hereAdd a note here**Listing 3.28**

Add a note hereAdd a note here[java] [DEBUG] DiscoverClasses - -find:

className='com.devspace.jseng.create.InterfaceToBeShared'

[java] [DEBUG] DiscoverClasses - -getNextClass: next

URL='jar:file:/D:/mydocs/book/bookappliedpatterns/src/jseng.jar!/com/devspace/jseng/create/InterfaceToBeShared.class'

Add a note hereAdd a note hereNotice that the found file has the class extension to indicate that the found item is a class, which was not specified when we assigned the variable name in [Listing 3.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935). Of course, this may seem logical since in [Listing 3.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) the resource to be found is a file and in [Listing 3.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=515105935) the resource to be found is a class. However, if a developer chose to find the class file using the class DiscoverResources, then he would have to use the query string "InterfaceToBeShared.class".

Add a note hereAdd a note hereWhen you are querying for a class name, you need to remember several specifics. Using the class DiscoverResources to query for a class name requires that the developer add the package details. For example, the package com.devspace.jseng.create would have to be converted to com/devspace/jseng/create. Trying to search for a class name without a package identifier will not work, although you can find a regular file without a directory specifier. The directories for a class file have a special purpose. When you're attempting to find a class using the DiscoverClasses class, it is important to use the package name; otherwise, you will not be able to find the class.

### Add a note hereAdd a note hereUsing the ClassJDKHooks

Add a note hereAdd a note hereThe class JDKHooks is part of the *Discovery* package because it is needed to make some things in the Java environment simpler. It is not the most interesting class, but knowing what it can do is useful in the odd situation. The essence of the class JDKHooks is to make it simpler to get the thread context and system context class loaders. This is very useful when you are manually loading classes based on specific class paths. The class JDKHooks is a static class within the JDKHooks class definition. To retrieve the instance, you use the static method JDKHooks.getJDKHooks. You need an accessor because the class JDKHooks is an abstract class that has two private implementations. This type of architecture is identical to that of the Commons Bridge, defined in [Chapter 2](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=42#42). The two private implementations are for JDK 1.1 and JDK 1.2.

## **The *Launcher* Project**

Add a note hereAdd a note hereThe *Launcher* project in the Jakarta Commons is not a strategy used to instantiate another class within a component or subsystem. Rather, it allows you to launch another Java component or subsystem as an application. As we've progressed in this chapter, each project description's factory has become a bit more abstract. In the beginning of the chapter, we illustrated the factory that created a class within the same application. The *Discovery* package was capable of loading a class into an application. The *Launcher* package is not a factory in the strictest sense of the word. However, it's a way to launch another Java class using a specific set of environment constraints. The *Launcher* package is great when you're creating server applications that want to launch Java applications executing in different contexts. An example of such an application type would be a Web Server with multiple domains. With the *Launcher* package, you can create the main Web Server processor, which manages a number of child processes responsible for each virtual domain.

Add a note hereAdd a note hereThe simplest case of using the *Launcher* classes is not that simple, even though using the Launcher class is very simple. An example of its simplicity from the user point of view is shown in [Listing 3.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156).

Add a note hereAdd a note here**Listing 3.29**

Add a note hereAdd a note hereLauncher launch = new Launcher();

String args[] = new String[ 3];

args[ 0] = "-launchfile";

args[ 1] =

"/home/cgross/mydocs/book/src/common/launcher.xml";

args[ 2] = "echo";

launch.start( args);

Add a note hereAdd a note hereIn [Listing 3.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), the main class being used is the class Launcher, which has one main method, start, which is used to instantiate the other class. What is puzzling is that instead of starting a class directly, the arguments defined in [Listing 3.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) reference some type of launch file and another identifier, echo. At first impression, the identifier echo would appear to be a class identifier. The reality, though, is that the identifier references something else entirely.

### Add a note hereAdd a note hereTechnical Details for the *Launcher* Package

Add a note hereAdd a note hereThe *Launcher* package, with its various classes, presents a problem: it's very hard to figure out what it does. Consider the description of the package described in the proposal document: "*The* Launcher *component is designed to be a cross-platform Java application launcher.*"

Add a note hereAdd a note hereThis description tells you nothing of the classes that are defined or implemented. As a developer, you are left entirely in the dark. Add to that the fact that you have a *Image from book*[*bin*](http://cdcontent.books24x7.com/id_7265/bin.zip) directory with a bunch of files that somebody could click on; however, the clicks don't tell you much. Therefore, it's very hard to conceptualize the *Launcher* package unless you identify the various classes. [Tables 3.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) and [3.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) do just that.

**Add a note hereAdd a note hereTable 3.5:** Repository details for the *Launcher* factory.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note herelauncher |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.lang.launcher |

**Add a note hereAdd a note hereTable 3.6:** Package and class details (legend: [launcher] = org.apache.commons.launcher).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[launcher].Launcher | Add a note hereAdd a note hereA main class used by the caller to instantiate another class |
| Add a note hereAdd a note here[lang].\* | Add a note hereAdd a note hereClasses used by the infrastructure to instantiate the Java class (needed by Ant) |
| Add a note hereAdd a note here[lang].types.\* | Add a note hereAdd a note hereClasses used by the infrastructure to instantiate the Java class (needed by Ant) |

#### **Details of Building the *Launcher* Package**

Add a note hereAdd a note hereBuilding the *Launcher* package from the CVS repository is not complicated. You need to use Ant and ensure that the Ant jar files are in the class path; otherwise, you'll get some compilation errors. When you compile, use the command ant dist because that will generate a directory *dist* within the *Launcher* directory. The *dist* directory contains a number of subdirectories, including the *Image from book*[*bin*](http://cdcontent.books24x7.com/id_7265/bin.zip) and *Image from book*[*lib*](http://cdcontent.books24x7.com/id_7265/lib.zip) directories. The bin directory is the most important because it contains the *Image from book*[*commons-launcher.jar*](http://cdcontent.books24x7.com/id_7265/commons-launcher.jar) file, which is the compiled *Launcher* package. The other files are important and will be explained later in the chapter.

### Add a note hereAdd a note hereArchitectural Details

Add a note hereAdd a note hereThe *Launcher* package depends on the Ant jar files. The *Launcher* package is a very clever package because it provides the programmer with a way of instantiating a class using a specific set of environment variables. This is achieved through the infrastructure of the Ant toolkit, which is a way of compiling a set of Java files. Lately, however, Ant has become an execution infrastructure for Java and other tools. When a developer clicks on a typical operating system executable, the executable will run, but the environment will be based on whatever the defaults are. Using Ant, you can define the classpaths, environment variables, and how many applications to start. Ant has become the ideal bootstrap environment. The default Ant execution environment is very extensive; for application bootstrapping, it is a bit of overkill. The *Launcher* is a stripped-down version of Ant that allows only a number of Ant constructs, such as task and variable definitions. If you summarize *Launcher*, you get aversion of Ant that makes it simpler to launch Java tasks. [Listing 3.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) shows the Ant tasks and variables that are defined. (Note that the class definition has been shortened so that it's easier to understand.) We will discuss Ant in more detail in [Chapter 11](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=859#859).

Add a note hereAdd a note here**Listing 3.30**

Add a note hereAdd a note herepublic class Launcher implements Runnable {

/\*\*

\* List of supported Ant tasks.

\*/

public final static Object[] SUPPORTED\_ANT\_TASKS =

new Object[] {

LaunchTask.TASK\_NAME, LaunchTask.class,

"ant", Ant.class,

"antcall", CallTarget.class,

"available", Available.class,

"condition", ConditionTask.class,

"fail", Exit.class,

"property", Property.class

};

/\*\*

\* List of supported Ant types.

\*/

public final static Object[] SUPPORTED\_ANT\_TYPES =

new Object[] {

ArgumentSet.TYPE\_NAME, ArgumentSet.class,

JVMArgumentSet.TYPE\_NAME, JVMArgumentSet.class,

SysPropertySet.TYPE\_NAME, SysPropertySet.class,

"description", Description.class,

"fileset", FileSet.class,

"filelist", FileList.class,

"path", Path.class,

"patternset", PatternSet.class

};

}

Add a note hereAdd a note hereIn [Listing 3.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), the supported Ant types and tasks are a combination of already existing tasks and types, and newly defined tasks and types. The already defined tasks and types are the string-defined tasks and types, which are defined in the Ant jar file. The other definitions are specified in the *Launcher* package. The class Launcher is a very simple class. You could copy it and make it part of another package if you need to expand the range of supported tasks and types.

### Add a note hereAdd a note hereDefining a Launch File

Add a note hereAdd a note hereThe defined tasks and types are used in a launch file, which was specified in [Listing 3.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) as Image from book[launcher.xml](http://cdcontent.books24x7.com/id_7265/launcher.xml). In [Listing 3.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), the Image from book[launcher.xml](http://cdcontent.books24x7.com/id_7265/launcher.xml) file is basically identical to an Ant Image from book[build.xml](http://cdcontent.books24x7.com/id_7265/build.xml) file. This means that the launch file has a project descriptor, some definitions, and some targets. [Listing 3.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) contained the identifier echo, which is a launch file target. This means that when the *Launcher* parses the launch file, the XML tag target will be searched. One of the found target XML tags will contain an attribute name with the value echo. The Image from book[launcher.xml](http://cdcontent.books24x7.com/id_7265/launcher.xml) file used in [Listing 3.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) is shown in [Listing 3.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156).

Add a note hereAdd a note here**Listing 3.31**

Add a note hereAdd a note here<project name="Ant Launcher" default="ant" basedir=".">

<property environment="env"/>

<property file="build.properties"/>

<target name="echo">

<launch

classname="com.devspace.jseng.create.ClassToLaunch"

classpath="${standard-jars}:${project-

src}/classes">

</launch>

</target>

</project>

Add a note hereAdd a note hereIn [Listing 3.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), the target echo contains a single task, launch. This task has two XML attributes, which specify the classpath and classname. The attribute classpath identifies the Java classpath of the application that is about to be executed. The attribute classname identifies the Java class that will be launched. In this case, the Java class that will be launched as shown in [Listing 3.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156).

Add a note hereAdd a note here**Listing 3.32**

Add a note hereAdd a note herepackage com.devspace.jseng.create;

public class ClassToLaunch {

public static void main( String [] args) {

System.out.println( "oooweee");

}

}

Add a note hereAdd a note hereThe class ClassToLaunch in [Listing 3.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) is a very simple class and does not contain any interesting functionality other than supporting the method main.

#### **Launching a Java Class**

Add a note hereAdd a note hereWhen you launch a Java class file, the biggest problem is defining the class path. The basic solution has been to define the CLASSPATH environment variable with all of the appropriate jar files. However, that approach has proven to be extremely tedious since literally millions of jar files are available on the Internet these days. The launcher file massively improves this situation because you can define variables as properties, which reference individual jar files. Let's go back and consider [Listing 3.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) again, and focus in on the XML tag property as illustrated in [Listing 3.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156).

Add a note hereAdd a note here**Listing 3.33**

Add a note hereAdd a note here<property environment="env"/>

<property file="build.properties"/>

Add a note hereAdd a note hereThe XML tag property in [Listing 3.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) defines a set of static variables that can be later referenced. The first property definition of [Listing 3.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) is used to define the environment variables. This is the case because the XML tag property has an attribute environment. Ant will translate this definition as saying, "Please expose the environment variables predicated by the value env." The second property definition has a file attribute that indicates you should import the file Image from book[build.properties](http://cdcontent.books24x7.com/id_7265/build.properties). The variables defined within the imported file should be considered static variables. A sample Image from book[build.properties](http://cdcontent.books24x7.com/id_7265/build.properties) is shown in [Listing 3.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156).

Larger View

Add a note hereAdd a note here**Listing 3.34**

Add a note hereAdd a note hereclib=${env.CATALINA\_HOME}/common/lib

project-root=/home/cgross/mydocs/book

project-src=${project-root}/src

commons-jars=:${clib}/commons-logging.jar:${clib}/commons-logging-api.jar

src=.

build=${project-root}/classes

debug=on

Add a note hereAdd a note hereEach line in a properties definition file like [Listing 3.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) has the [identifier]=[value] pattern. The identifier defines a static variable like the tag property in the launcher file. The identifiers can then be referenced using the notation ${variable}, as we saw in [Listings 3.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) and 3.34. The notation is a sequence of characters that tells the Ant parser to consider the identifier within the curly brackets as a variable to be substituted by its associated value. For example, in [Listing 3.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), the variable project-root is defined. One line later, that definition is used to define the variable project-src. The value of the variable project-src would be the concatenation of variable project-root and /src, which would result in the value of /home/cgross/mydocs/book/src.

Add a note hereAdd a note hereOne twist in [Listing 3.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) is the variable reference env.CATALINA\_HOME. In the launcher file, there was a property reference to the environment variables being the value env. Therefore, a variable that starts with env has a period as a separator to the identifier, which is an environment variable. In [Listing 3.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), that environment variable is CATALINA\_HOME. Using a dot to separate two identifiers is very similar notation to that used in an object-oriented system. Don't let this confuse you, however, because referencing the environment variables like this is a special case. Otherwise, a dot is part of the identifier, much like an underscore can be used to separate two identifiers.

Add a note hereAdd a note hereThe strategy to take when developing your own launcher file is to abstract definitions such as the class path to the Image from book[build.properties](http://cdcontent.books24x7.com/id_7265/build.properties) file. Then, within the launcher file, reference the class path using either one or a few class path variables as you saw in [Listing 3.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156). This approach allows an administrator to change jar files without having to update the launcher file or the Java executable.

#### **Launching a Class File Depending on the Context**

Add a note hereAdd a note hereWhen you're building an application launcher, you also need to make specific decisions based on the conditions. For example, one very problematic issue is the ability to execute specific files dependent on the operating system, or dependent on whether the component should be debugged. In either of these situations, the decision on what to do and how to do it depends on when the launcher file is executed. In [Listing 3.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), the simplest condition, which is whether the operating system is Microsoft Windows or not, is tested.

Larger View

Add a note hereAdd a note here**Listing 3.35**

Add a note hereAdd a note here<target name="testcondition">

<condition property="classtoexecute"

value="com.devspace.jseng.create.window.ClassToLaunch">

<os family="windows" />

</condition>

<condition property="classtoexecute"

value="com.devspace.jseng.create.unix.ClassToLaunch">

<not>

<os family="windows" />

</not>

</condition>

</target>

<target name="echo" depends="testcondition">

<launch classname="${classtoexecute}"

classpath="${standard-jars}:${project-src}/classes">

</launch>

</target>

Add a note hereAdd a note hereIn [Listing 3.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), there are two targets. The target echo still exists, but it has an additional attribute depends. The purpose of the depends attribute is to execute another target before the target in question is executed. In [Listing 3.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), the depends attribute identifies the testcondition target, which is a target with two contained condition XML tags. The condition XML tags are tasks that execute based on the contained child tags. This way of processing a condition is counter to how a programming language processes one. In a programming language like Java, the condition result executes the instructions within the condition block. In Ant, the condition result is the XML tag, and the condition tests are the instructions within the condition block. Let's look back to [Listing 3.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156). Here, the condition test is the XML tag os and the result is the attributes property and value. The condition is saying that if the child tags allow, please define the property classtoexecute with a value of com.devspace. In the first condition, the child tag was an operating system descriptor os, which returns a true or false value. In the second condition tag, the child tag os is embedded within a not child tag, meaning that the os result value should be inverted. These conditions then define a class to be executed dependent on the operating system.

#### **Launch Arguments**

Add a note hereAdd a note hereThe *Launcher* is based on the concept of calling Ant, which calls the Java class file. You can define the class to be launched as well as the Java classpath. The launch task has many more options other than classname and classpath. The following list defines them all and what they do:

* Add a note hereAdd a note here**debug (true | false)** This is a flag to start the child JVM in the Java Debugger (JDB) debugger. If you want to use Java Platform Debugger Architecture (JPDA) debugging, please refer to the *Image from book*[*bin*](http://cdcontent.books24x7.com/id_7265/bin.zip) directory of the *Launcher* package. In the *Image from book*[*bin*](http://cdcontent.books24x7.com/id_7265/bin.zip) directory is a launcher file example on how to start the launched process to be debugged using JPDA. The default is false.
* Add a note hereAdd a note here**displayMinimizedWindow (true | false)**: If set, this displays the window as minimized.
* Add a note hereAdd a note here**disposeMinimizedWindow (true | false)**: This is a flag specific to the Windows platform that causes the window to be discarded once the child process is completed.
* Add a note hereAdd a note here**failonerror (true | false)**: If this is set and the child process fails because of an exit, the calling process will fail and exit as well.
* Add a note hereAdd a note here**filterclassname**: This specifies the file that implements a filter. A filter is discussed later in this chapter.
* Add a note hereAdd a note here**filterclasspath**: This specifies the Java classpath used to load the filter class.
* Add a note hereAdd a note here**minimizedWindowIcon**: This is an attribute specific to the Windows platform. It represents a filename that contains an icon that is displayed when the child process window is minimized.
* Add a note hereAdd a note here**minimizedWindowTitle**: This is an attribute specific to the Windows platform that sets the name of the process in the taskbar when the window is minimized.
* Add a note hereAdd a note here**output**: This is the name of a file that will contain the output of the child process.
* Add a note hereAdd a note here**print (true | false)**: This outputs the command used to start the child JVM to the standard output stream. The default is false.
* Add a note hereAdd a note here**redirectoutput (true | false)**: When this flag is set to true, the output of the child process is redirected to the file specified by the output attribute. The default is false.
* Add a note hereAdd a note here**requiretools (true | false)**: If you set this flag when executing the process, it will cause the *tools.jar* to be appended to the classpath. The major side effect of this is that the child JVM process must be executed using a JDK and not JRE (Java runtime). The default is false.
* Add a note hereAdd a note here**useargs (true | false)**: In [Listing 3.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), there were a number of arguments created manually. These arguments are passed as arguments to the child process. We will discuss this further later in this chapter. The default is false.
* Add a note hereAdd a note here**usesystem (true | false)**: This is a flag specific for some UNIX platforms to not read the standard input stream. The default is true, which means read the standard input stream.
* Add a note hereAdd a note here**waitforchild (true | false)**: This causes the calling process to wait for the child process to finish before continuing any further processing. The default is false.

#### **Passing in Command Line Arguments**

Add a note hereAdd a note hereWhen the child process is executed, it is possible to specify some command line arguments. The command line arguments can be specified in two different locations. The nature of the command line argument determines where the command line argument can be optimally stored. If the command line argument is static and does not change, then the best place to define it would be in the launcher file, as shown in [Listing 3.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156).

Add a note hereAdd a note here**Listing 3.36**

Add a note hereAdd a note here<launch

classname="${classtoexecute}"

classpath="${standard-jars}:${project-src}/classes">

<argset>

<arg value="hello" />

</argset>

</launch>

Add a note hereAdd a note hereIn [Listing 3.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), the XML tag argset defines a placeholder for what could be a single or multiple command line arguments. In [Listing 3.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), there is a single argument as specified by the XML tag arg. The value of the argument is specified by the attribute value, which in this case is "hello." Modifying [Listing 3.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) to output the command line arguments results in [Listing 3.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156).

Add a note hereAdd a note here**Listing 3.37**

Add a note hereAdd a note herepackage com.devspace.jseng.create;

public class ClassToLaunch {

public static void main( String [] args) {

if( args.length > 0) {

for( int c1 = 0; c1 < args.length; c1 ++) {

System.out.println( "Arg: " + args[ c1]);

}

}

System.out.println( "oooweee");

}

}

Add a note hereAdd a note hereRunning the launcher file would generate the one argument as output. The problem with using the launcher file is that the command line argument is static and cannot be changed dynamically. To generate a dynamic command line argument, you need to generate the command line argument in the client code, as shown in [Listing 3.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156).

Add a note hereAdd a note here**Listing 3.38**

Add a note hereAdd a note hereLauncher launch = new Launcher();

String args[] = new String[ 4];

args[ 0] = "-launchfile";

args[ 1] =

"/home/cgross/mydocs/book/src/common/launcher.xml";

args[ 2] = "echo";

args[ 3] = "command line argument";

launch.start( args);

Add a note hereAdd a note hereWhen using the method launch.start as shown in [Listing 3.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), it is very important to use the correct command line arguments. The args[ 0] element references the string -launchfile, which is a command line argument consumed by the class Launcher. This command line argument specifies that the launcher file will be located at a specific location. Otherwise, the Launcher class will assume that the file Image from book[launcher.xml](http://cdcontent.books24x7.com/id_7265/launcher.xml) exists in the same directory where the Image from book[commons-launcher.jar](http://cdcontent.books24x7.com/id_7265/commons-launcher.jar) file is located. The args[ 1] element references the location of the Image from book[launcher.xml](http://cdcontent.books24x7.com/id_7265/launcher.xml) file, which does not need to be called Image from book[launcher.xml](http://cdcontent.books24x7.com/id_7265/launcher.xml) since a specific launch file is defined. The args[ 2] element defines the target that will be executed in the launch file. Any argument after the target, such as args[ 3], can be a command line argument that is passed to the client process.

Add a note hereAdd a note hereThe command line argument, by default, is not passed to the client process. The launch task in the launcher file has to have the useargs attribute set to true, as shown in [Listing 3.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156).

Add a note hereAdd a note here**Listing 3.39**

Add a note hereAdd a note here<launch

classname="${classtoexecute}"

classpath="${standard-jars}:${project-src}/classes"

useargs="true">

<argset>

<arg value="hello" />

</argset>

</launch>

Add a note hereAdd a note hereWhen the client process executes, a command line argument will be displayed. However, [Listing 3.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) will create an array of two arguments. The question is which arguments are placed at the beginning of the array and which arguments at the end. The static arguments are at the front of the argument array, and the dynamic arguments are placed afterwards. However, this is a practical observation and not a common rule. Therefore, when defining your own arguments, use key value combinations such as -argument value.

Add a note hereAdd a note hereIn [Listing 3.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), the method launch.start is used to start a child process. You can also use the method launch.stop to stop the child process. If the caller wants to kill the child process, the method launch.killChildProcesses is used. The methods launch. isStarted and launch.isStopped make it possible for the caller to query for the status of the child process. Additionally, the class Launcher has methods to query the classpaths and other attributes of the client task.

#### **Specifying Java VM Arguments**

Add a note hereAdd a note hereThe arguments defined thus far are command line arguments. It is possible to specify Java VM arguments that control how the client process JVM is started. The way to specify those arguments is to use the XML tags jvmargset and jvmarg, as shown in [Listing 3.40](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156).

Add a note hereAdd a note here**Listing 3.40**

Add a note hereAdd a note here<launch

classname="${classtoexecute}"

classpath="${standard-jars}:${project-src}/classes">

<jvmargset>

<jvmarg value="-verbose" />

</jvmargset>

</launch>

Add a note hereAdd a note hereIn [Listing 3.40](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), the XML tag jvmarg has a value of -verbose, which means that when the client process JVM starts, the JVM should be verbose and display plenty of messages. The purpose of being able to specify JVM arguments is to be able to debug the client process, as shown in [Listing 3.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156).

Add a note hereAdd a note here**Listing 3.41**

Add a note hereAdd a note here <jvmargset id="base.jvm.args">

<jvmarg value="-sourcepath" if="jdb"/>

<jvmarg path="${app.home}/../src/java" if="jdb"/>

<jvmarg value="-Xdebug" if="jpda.settings"/>

<jvmarg value="-Xrunjdwp:${jpda.settings}"

if="jpda.settings"/>

</jvmargset>

Add a note hereAdd a note hereIn [Listing 3.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), the jvmarg tags define a series of JVM instructions that will start the JPDA debugger. Contrasting [Listing 3.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156) with [Listing 3.40](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), the jvmarg tags have an additional if attribute. The if attribute is a conditional for the jvmarg. The attribute specifies that if jdb or jpda.settings are defined, then the jvmarg is defined.

#### **Using Args**

Add a note hereAdd a note hereIn addition to command line arguments, you can define system properties. System properties are different from command line arguments in that system properties are key value combinations. The key value combinations are specified in the launcher file, as shown by [Listing 3.42](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156).

Add a note hereAdd a note here**Listing 3.42**

Add a note hereAdd a note here<launch

classname="${classtoexecute}"

classpath="${standard-jars}:${project-src}/classes">

<syspropertyset>

<sysproperty key="val" value="another"/>

</syspropertyset>

</launch>

Add a note hereAdd a note hereThe key value combination is specified by the XML tags syspropertyset and sysproperty, as illustrated in [Listing 3.42](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156). These XML tags are not unique and could have been specified as a jvmarg using the notation >"-Dval=another". However, the XML tag sysproperty is a cleaner approach. To be able to read the system property, the method call System.getProperty is made. The method getProperty retrieves the key based on the input parameter, which is the key to search for. If we relate the method call to [Listing 3.42](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156), the key would be val, and the method getProperty would return the value another. Using the syspropertyset approach is tidier than having to specify the properties using a jvmarg tag.

### Add a note hereAdd a note hereFiltering the Command

Add a note hereAdd a note hereJust as a child process is about to be started, you can change all of the things that you defined using the client code or launcher file. The way to do this is to specify a filter class. The purpose of the filter class is to give the client application one last chance to modify the startup characteristics before the child process starts. A sample implementation is shown in [Listing 3.43](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156).

Add a note hereAdd a note here**Listing 3.43**

Add a note hereAdd a note herepackage com.devspace.jseng.create;

import org.apache.commons.launcher.LaunchFilter;

import org.apache.commons.launcher.LaunchCommand;

public class LocalLauncherFilter implements LaunchFilter {

public void filter(LaunchCommand launchCommand) {

launchCommand.setPrint( true);

}

}

Add a note hereAdd a note hereTo create a filter, the class has to implement the interface LaunchFilter, as shown in [Listing 3.43](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156). The interface LaunchFilter has only the method filter to be implemented. All of the child process startup characteristics are stored in the class LaunchCommand. Going through this class definition would be redundant because the individual operations mirror the individual attributes and arguments that have been specified thus far. For example, it is possible to manually set the launch attribute print using the method setPrint and getPrint. You should use a filter only when you need to do very specific manual tweaking that cannot be coded in another technique.

Add a note hereAdd a note hereTo activate the filter, the launch task has an additional attribute, filterclassname, as shown in [Listing 3.44](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=622714156).

Add a note hereAdd a note here**Listing 3.44**

Add a note hereAdd a note here<launch

classname="${classtoexecute}"

classpath="${standard-jars}:${project-src}/classes"

filterclassname=

"com.devspace.jseng.create.LocalLauncherFilter">

</launch>

Add a note hereAdd a note hereIf the filterclass specified is not on the classpath, then you can use the filterclasspath attribute to specify an additional classpath that contains the filter class.

## **Summary**

Add a note hereAdd a note hereThis chapter outlined how a Java class can be instantiated. The chapter started out by describing the factory problem and why it was necessary. Although writing your own factory is not that difficult, it is better to use a formalized factory approach, as defined in the *Lang* package. The purpose of the factory is to reduce the number of hard dependencies that can occur in a project. Hard dependencies are bad because they cause source code to become brittle and difficult to modify. As source code becomes brittle, more and more problems surface, which increases maintenance costs. The factory always returns an interface that represents some instance of an implementation that the user of the interface is not aware of. The factory is the missing piece of the Commons Bridge best practice.

Add a note hereAdd a note hereHowever, using a factory like the one presented in the *lang* package is not the only way to instantiate an object. There are other approaches, because developers often write frameworks that have plug-ins. For those situations, the factory is totally inappropriate to use since it is assumed that the factory knows which implementation to instantiate. Using the *Discovery* package, a developer can let the system administrator define the implementation. Of course, even with the *Discovery* package the client sees only an interface instance.

Add a note hereAdd a note hereFinally, for those situations where a separate task needs to be instantiated, the *Launcher* package comes to assistance. Using the *Launcher* package, you can start another JVM process and assign it a number of characteristics. However, the *Launcher* package's downside is that the client has no interface to interact with. The client interacts only with the class Launcher, or the client has to establish some type of inter-process communication.

Add a note hereAdd a note hereThe objective of this chapter was to illustrate that creating an instance of a class is not always a simple and straightforward task. There are different strategies and objectives to solve, and you should consider each strategy carefully.

Add a note hereAdd a note here**On the CD** The sources to the concepts presented in this chapter are located under the directory *[CDROM]/jseng/src/com.devspace.jseng.create*.

## **Questions**

Add a note hereAdd a note here**3.1:** Explain the problems associated with instantiating classes directly.

Add a note hereAdd a note here**3.2:** Rewrite the calculator component from [Chapter 2](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=42#42) to use factories instead of direct class instantiations.

Add a note hereAdd a note here**3.3:** The calculator component has extensions, as created in the [previous chapter](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=42#42). The extensions were hard-coded using strings in a class file. Change the extension to be a service that is discovered using the *Discovery* package. Ideally, the administrator should be able to define which extension the calculator loads.

Add a note hereAdd a note here**3.4:** Create a batch file extension to the calculator. For the time being, the file contains an integer number on a line, where each file can have multiple lines. The extension would be able to open the file and add each number, and the result would be a running total. The extension is used by the calculator component by having a user type in the filename, which is passed to the extension for further processing. The challenge in this part of the exercise is how to extend the calculator to accept letters and not just numbers. The batch file extension starts another Java process using the *Launcher* package, which adds the numbers and returns a result. How the result is communicated back to the original calculator process is left up to the programmer.

**Chapter 4: Scalability**

**Add a note hereAdd a note here****Overview**

Add a note hereAdd a note hereIn this chapter, we will do the following:

* Add a note hereAdd a note hereLearn what a pooled object is, based on the *Pool* package
* Add a note hereAdd a note hereLearn the individual pool types
* Add a note hereAdd a note hereFigure out how to decide which objects need to be pooled
* Add a note hereAdd a note hereUse a pool of threads using the *Threadpool* package
* Add a note hereAdd a note hereExecute tasks based on immutable objects

## **The Purpose of This Chapter**

Add a note hereAdd a note hereOne of the more difficult tasks when you're writing applications is partitioning an application to increase performance or scalability. The partitioning of the program is simple, but it's not simple to make the partitioning work effectively. It is difficult to make the program work because it is hard to write the infrastructure classes necessary to effectively partition an application. Often, the programmer does not have enough time to consider all of the ramifications. Hence, a partitioning of the application could result in slower execution times.

Add a note hereAdd a note hereEffective partitioning requires two major algorithm techniques: multi-threading and pooling of objects. A third algorithm technique—creating immutable objects—is more of a best practice. The task in this chapter is to show how to use the infrastructure code that will make an application easier to scale.

## **Pooling Objects**

Add a note hereAdd a note hereWhen you're developing complex applications, you need to be able to manage your resources. The resources could be a database connection, file, or some other kind of resource where the cost of initialization and destruction is high. In the Jakarta Commons project, several projects deal with managing resources using pools and caches. However, the *Pool* package is the simplest and most useful. For example, the *dbcp* package allows you to pool database connections, which is generally a good idea. You could very easily do that using the *Pool* package; just define a business object that contains a database connection. You don't use pooling for every object due to the overhead. With pooling, you have to write some small pieces of infrastructure code that manage objects that can be pooled.

### Add a note hereAdd a note hereTechnical Details for the *Pool* Package

Add a note hereAdd a note here[Tables 4.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939) and [4.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939) contain the abbreviated details necessary to use the *Pool* package.

**Add a note hereAdd a note hereTable 4.1:** Repository details for the *Pool* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note herepool |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.pool, org.apache.commons.pool.impl |

**Add a note hereAdd a note hereTable 4.2:** Package and class details (legend: [pool] = org.apache.commons.pool).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[pool].ObjectPool | Add a note hereAdd a note hereA base interface that represents an object pool. The client may use this interface but must instantiate a type of object pool found in the [pool].impl package. |
| Add a note hereAdd a note hereGenericObjectPool: | Add a note hereAdd a note hereA simple object pool implementation. |
| Add a note hereAdd a note hereGenericKeyedObjectPool | Add a note hereAdd a note hereAn object pool that marks individual objects by a specific key. |
| Add a note hereAdd a note hereSoftReferenceObjectPool | Add a note hereAdd a note hereAn object pool that is managed by the JVM. The JVM will decide when to compact or expand the pool. |
| Add a note hereAdd a note hereStackObjectPool | Add a note hereAdd a note hereA specific type of pool that uses a specific algorithm for managing object instances. Typically, the advantage of this class is that there is no limit to active objects, but the maximum of idle number of objects is strictly enforced. |
| Add a note hereAdd a note hereStackKeyedObjectPool | Add a note hereAdd a note hereAn object pool that manages individual objects using a specific key. The object management algorithm is the same as for the class StackObjectPool. |
| Add a note hereAdd a note here[pool].PoolableObjectFactory | Add a note hereAdd a note hereA structural interface used by the Object pool to instantiate and manage the various object instances. |
| Add a note hereAdd a note here[pool].BaseObjectPool | Add a note hereAdd a note hereAn abstract class used by pool implementations; it contains the basic functionality of a pool. |
| Add a note hereAdd a note here[pool].KeyedObjectPool | Add a note hereAdd a note hereAn object pool interface that uses keys to uniquely identify specific instance types of pools. Such pools can hold multiple different types of pools. |
| Add a note hereAdd a note here[pool].KeyedPoolableObjectFactory | Add a note hereAdd a note hereA structural interface used by key-based object pools to manage and instantiate objects. |

### Add a note hereAdd a note hereA Simple Pool

Add a note hereAdd a note hereWhen building a pooled object component or subsystem, you need to implement one structural interface. The architecture of the pool is that the client uses a pool and the pool manages all of the details related to when to instantiate and destroy objects. However, the pool does not instantiate and destroy the individual object. A helper class will do that on behalf of the pool. In pool speak, this is known as an *object pool factory*. The object pool factory is a bit more complicated than the standard factories described in [Chapter 3](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=164#164).

Add a note hereAdd a note here[Listing 4.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939) shows an implementation of the object pool factory.

Larger View

Add a note hereAdd a note here**Listing 4.1**

Add a note hereAdd a note hereclass SimpleFactory implements PoolableObjectFactory {

public Object makeObject() { return new PooledObject(); }

public void destroyObject(Object obj) { }

public boolean validateObject(Object obj) { return true; }

public void activateObject(Object obj) { }

public void passivateObject(Object obj) { }

}

Add a note hereAdd a note hereIn [Listing 4.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939), the class SimpleFactory implements the interface PoolableObject- Factory. In this sample implementation, the class SimpleFactory does nothing much other than instantiating the class PooledObject and returning a value of true for the method validateObject. In such simple implementations, it is better to extend the class BasePoolableObjectFactory since that class definition implements the essentials. If the class SimpleFactory subclasses the class BasePoolableObjectFactory, then SimpleFactory only needs to implement the method makeObject. To make use of a pooled object, a client needs to instantiate a pool and then pass the class SimpleFactory to it, as shown in [Listing 4.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939).

Add a note hereAdd a note here**Listing 4.2**

Add a note hereAdd a note hereGenericObjectPool pool = new GenericObjectPool(

new SimpleFactory());

PooledObject obj = (PooledObject)pool.borrowObject();

obj.method();

pool.returnObject(obj);

Add a note hereAdd a note hereIn [Listing 4.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939), the pool used is the class GenericObjectPool, which provides a basic pool implementation. For the class' GenericObjectPool constructor, an instance of the class SimpleFactory is passed in. Once the pool has been instantiated, it is possible to start using the pool's object instances. To get an object instance, the method pool.borrowObject is called. The method is called "borrow" because the pool decides if a new object instance needs to be allocated or if an existing object instance can be returned. Once the program is finished with the object instance, it needs to be returned to the pool using the method pool.returnObject. If the object instance is not returned to the pool, then the pool will constantly grow and objects will not be reused.

### Add a note hereAdd a note hereDetails of the *Pool* Package

Add a note hereAdd a note hereA pool in the default case is relatively simple object to use. However, you can use many options to manage the pool. The type of pool used and how it is implemented are other considerations. Regardless of the pool, however, you need a helper factory to manage individual object instances, as you saw in [Listing 4.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939).

Add a note hereAdd a note hereIn a pool, some objects are idle, whereas some are active. *Idle objects* are objects available in the pool and available for usage. *Active objects* are the objects that have been made available by the pool and are in usage. If you add the number of idle and active objects, you generate the total object instance count. The hard limit on the number of object instances is based on the total active count. Once the hard limit has been reached, a pool can take various actions, such as expanding the pool size or forcing the caller to wait for idle object instances.

Add a note hereAdd a note hereWhen an object is borrowed from the pool, the state of the object instance changes from idle to active. If there are no idle objects, then the pool will create an object instance and mark it as active. A pool will always recycle objects from the object pool and activate the object before it is borrowed from the pool. When an object is being activated, the situation is like when the constructor of the object is called, except that the helper object activation method is called.

Add a note hereAdd a note hereEvery pool object factory implements either the interface PoolableObjectFactory or KeyedPoolableObjectFactory. The interface PoolableObjectFactory has the following methods:

* Add a note hereAdd a note here**Object makeObject()**: Creates an object instance.
* Add a note hereAdd a note here**void destroyObject(Object obj)**: Destroys the object. This is a bit of a strange concept because in Java there is no such thing as object destruction in memory terms. The purpose of this method is to release the resources managed by the object. Remember that the pooled object is an object that is expensive in resource terms to instantiate and release.
* Add a note hereAdd a note here**boolean validateObject(Object obj)**: Checks the object for state integrality. This method is not called generally, only when the pool caller requests it.
* Add a note hereAdd a note here**void activateObject(Object obj)**: Resets the object to an initialized state. It's usually called before the object's state changes from idle to active.
* Add a note hereAdd a note here**void passivateObject(Object obj)**: Resets the object to an uninitialized state. It's usually called before the object's state changes from active to idle.

Add a note hereAdd a note hereThe methods for the interface KeyedPoolableObjectFactory are identical to those for PoolableObjectFactory, except that there is an additional parameter. For example, the interface PoolableObjectFactory has the method makeObject(), which on the interface KeyedPoolableObjectFactory would be makeObject( Object key). The key could be any object instance; it is a reference marker that is not translated by the pools. However, each key must be unique. This means that each object instance has to be unique and not just contain unique data, since the pools will not process for uniqueness the data that the object holds.

Add a note hereAdd a note hereThe developer will interact with the interfaces ObjectPool and KeyedObjectPool when managing a pool of objects. As with the interfaces PoolableObjectFactory and KeyedPoolableObjectFactory, the difference between the interface ObjectPool and the interface KeyedObjectPool is that the interface KeyedObjectPool has an additional parameter key. The key is provided by the caller and is then passed onto the object factory helpers. The interfaces ObjectPool and KeyedObjectPool have the following methods: (note that the methods in square brackets are variations of the methods for the interface KeyedObjectPool):

* Add a note hereAdd a note here**Object borrowObject() [Object borrowObject(Object key)]**: This method retrieves an object instance from the pool. The object instance's state changes from idle to active. If the pool has reached a hard limit on the number of active object instances, there is no predefined result. The result depends entirely on the implementation of the pool.
* Add a note hereAdd a note here**void returnObject(Object obj) [void returnObject(Object key, Object obj)]**: This returns the object instance back into the pool. The object instance's state changes from active to idle. If there are an excessive number of idle objects, then the extra ones are destroyed. The exact nature of how this method acts depends on the implementation of the pool.
* Add a note hereAdd a note here**void invalidateObject(Object obj) [void invalidateObject(Object key, Object obj)]**: An object is returned to the pool and should be destroyed because the object has become invalid. This could be due to any reason, but the thinking is as follows: When an object is activated or set passivated, the object's state is reset. These operations tend not to be too expensive. However, when the object becomes invalid, the cost of resetting the object becomes too great, and it would be simpler to delete the object and instantiate a new one if necessary.
* Add a note hereAdd a note here**int getNumIdle() [int getNumIdle(Object key)]**: This retrieves the number of idle objects.
* Add a note hereAdd a note here**int getNumActive() [int getNumActive(Object key)]**: This retrieves the number of active objects.
* Add a note hereAdd a note here**void clear()**: This releases all of the idle objects from the pool.
* Add a note hereAdd a note here**void close()**: This closes and finalizes the pool and releases all resources associated with the pool.
* Add a note hereAdd a note here**void setFactory(PoolableObjectFactory factory)**: This sets the factory that is used to manage individual object instances.

Add a note hereAdd a note hereThe details of the managing a pool may seem overwhelming; however, because of the nature of the library, it is essential that there be plenty of parameters you can tweak. Remember that we are talking about scalability, and often that is discovered only when we're testing the application. Being able to tweak the settings optimizes the performance of the application. When we use the pool, it is important that the object does need a pool. The key thing to remember is that the cost of object instantiation should be high.

### Add a note hereAdd a note hereIntegrating the *Pool* Package

Add a note hereAdd a note hereAlthough we're glad that the *Pool* package exists, the problem is that it introduces yet another factory. We introduced several factories in [Chapter 3](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=164#164). The *Pool* package-based factory was not one of those factories. Ideally, the best solution would be to use the *lang* factory or the *Discovery* factory. [Listing 4.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939) shows how to wrap the *lang* factory in the *Pool* object factory.

Add a note hereAdd a note here**Listing 4.3**

Add a note hereAdd a note herepublic class BasePooledFactory

extends BasePoolableObjectFactory {

protected Factory \_factory;

public BasePooledFactory( Factory factory) {

\_factory = factory;

}

public Object makeObject() throws Exception {

return \_factory.create();

}

}

Add a note hereAdd a note hereIn [Listing 4.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939), the class BasePooledFactory delegates the makeObject method to the \_factory.create method. This way, the strategy used in [Chapter 3](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=164#164) has not been negated. However, the object created will not be subjected to any activation and deactivation method calls, which are part of the interface PoolableObjectFactory. This is OK if the object to be used is stateless.

Add a note hereAdd a note hereIf the object is not stateless, then there is a problem. Right now, the class BasePooledFactory knows nothing about the implementation of the class created by the generic Factory interface. However, the *lang* factory does not have any methods that relate to activate or deactivation. Therefore, the only solution to this problem is to add object-specific instructions. Doing so will preclude the need for using the interface Factory, as shown in [Listing 4.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939).

Larger View

Add a note hereAdd a note here**Listing 4.4**

Add a note hereAdd a note herepublic class BasePooledFactory

extends BasePoolableObjectFactory {

protected Factory \_factory;

public BasePooledFactory( Factory factory) {

\_factory = factory;

}

public Object makeObject() throws Exception {

return \_factory.create();

}

public void activateObject( Object obj) throws Exception {

MyClass cls = (MyClass)obj;

// ... do something with the instance

}

}

Add a note hereAdd a note hereIn [Listing 4.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939), the interface instance Factory is used to instantiate the object. However, the method activateObject has the parameter variable obj typecast to the specific class MyClass. Contrast that to the method makeObject, which returns a generic object type. The result is that class BasePooledFactory, which was supposed to be generic, has a reference to a specific class, and that is not desirable.

Add a note hereAdd a note hereThe solution to the specific class reference problem is to look back at Listing 3.15 in [Chapter 3](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=164#164). To recap, Listing 3.15 showed how to create some type of base interface to expose the method clone and extend the interface Cloneable. Listing 3.15 could be used as a basis for a more complex interface, as shown in [Listing 4.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939).

Add a note hereAdd a note here**Listing 4.5**

Add a note hereAdd a note herepublic interface BaseObject extends Cloneable {

public Object clone() throws CloneNotSupportedException;

public void activateObject() throws Exception;

public void destroyObject() throws Exception;

public boolean validateObject();

public void passivateObject() throws Exception;

}

Add a note hereAdd a note hereThe newly defined interface BaseObject adds the methods used to activate, deactivate, and destroy the object. Of course, the original method clone is still available. Knowing that this interface is the basis of all objects, you could substitute the reference to MyClass in [Listing 4.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939) with a reference to BaseObject. The actual code would appear similar to [Listing 4.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939).

Add a note hereAdd a note here**Listing 4.6**

Add a note hereAdd a note here public void activateObject(Object obj) throws

Exception {

try {

BaseInterface interf;

interf = (BaseInterface)obj;

interf.activateObject();

}

catch( java.lang.ClassCastException ec) {

;

}

}

Add a note hereAdd a note hereIn [Listing 4.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939), the input variable obj is cast to the interface BaseInterface. Once the object has been cast, the method activateObject can be called. What is important to realize is that the entire code is wrapped in an exception block that catches the exception ClassCastException. The class ClassCatchException represents cast failure, which could occur when the variable obj is cast to the interface BaseInterface. However, no actions are taken if the cast fails. This is a good, logical solution because it allows the developer to have the option to implement the interface BaseInterface. BaseInterface must be implemented if the object is stateful, even if the object used in the pooled context is a legacy object.

### Add a note hereAdd a note hereEncapsulating the Pool

Add a note hereAdd a note hereIn [Listing 4.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939), the client used the pool directly and associated the object factory manually. This is another example of how the object factory is used as a hard reference, even though the rest of the code would use the interface. The object factory as shown in previous chapters could be converted into a soft reference, but there is a logic problem. The designer of the class used by the object factory knows what is expensive in resources terms of the class. Therefore, it's probably a bad idea to have the client programmer decide which pool to use and which parameters to set. It is not that the class designer should take away rights from the client programmer, but that the class designer should set meaningful defaults. This way, the client programmer does not have to look through the class sources to figure out what the class is trying to do. The simplest example of predefining what to do for the client programmer is shown in [Listing 4.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939).

Add a note hereAdd a note here**Listing 4.7**

Add a note hereAdd a note herepublic class IntegrationExamplePool extends

GenericObjectPool {

public IntegrationExamplePool() {

setFactory( new GenericPooledFactory(

FactoryUtils.reflectionFactory(

IntegrationExample.class)));

}

}

Add a note hereAdd a note hereIn [Listing 4.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939), the class IntegrationExamplePool subclasses the pool class Generic-ObjectPool. In the constructor of IntegrationExamplePool, the default object factory is set using the method call setFactory. The other class, GenericPooledFactory, is a class developed by the author of this book, which can be retrieved from the Web site *www. devspace.com*. Finally, the method call FactoryUtils.reflectionFactory is a method from the *lang* factory package.

Add a note hereAdd a note hereEven though [Listing 4.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939) is compact, it accomplishes everything that is necessary in one step. The *lang* factory is used, which means the material discussed in previous chapters is not for naught, or at least it is not an either-solution situation. If, however, the *lang* factory is not to your liking, then you can use the *pool* factory as the base factory interface. The interface BaseInterface is used because the class IntegrationExample will implement the interface. The class GenericPooledFactory implements the interface PoolableObjectFactory and uses the *lang* factory internally to instantiate objects. In addition, the client programmer has only one hard reference to the class IntegrationExamplePool, which is responsible for maintaining all of the hard references of the specific implementations internally. [Listing 4.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939) shows an example of a client using the class.

Add a note hereAdd a note here**Listing 4.8**

Add a note hereAdd a note hereIntegrationExamplePool pool = new

IntegrationExamplePool();

Object temp = pool.borrowObject();

InterfaceToBeShared obj = (InterfaceToBeShared)temp;

obj.availability();

pool.returnObject(obj);

Add a note hereAdd a note here[Listing 4.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939) looks very similar to [Listing 4.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939). The difference is that the hard references to specific implementations and object factories have been removed. The client only needs to know about the interface InterfaceToBeShared and the common pool class IntegrationExamplePool. This is a clean approach that does not violate any of the best practices we have learned thus far.

### Add a note hereAdd a note hereEncapsulating a Multiple Class Type Pool

Add a note hereAdd a note hereNow that we have defined a strategy for defining a custom pool, we can use it to manage multiple class type pools, which are pools that can contain different class instance types. In [Chapter 3](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=164#164), we introduced the idea of creating multiple objects using one factory class. The same sort of idea can be used to create multiple objects with a pool. Doing so is relatively simple; instead of subclassing the class GenericObjectPool, we subclass the class GenericKeyed- ObjectPool. The caller class would then define a number of static variables that can be used to instantiate the different types. [Listing 4.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939) shows a sample implementation of this class.

Larger View

Add a note hereAdd a note here**Listing 4.9**

Add a note hereAdd a note herepublic class MultiIntegrationExamplePool extends

GenericKeyedObjectPool {

public final static String OBJ\_OBJECTTOBECREATED =

ObjectToBeCreated.class.getName();

public final static String OBJ\_ANOTHEROBJECTTOBECREATED =

AnotherObjectToBeCreated.class.getName();

private KeyedGenericPooledFactory \_helperFactory;

public MultiIntegrationExamplePool() {

\_helperFactory = new KeyedGenericPooledFactory();

\_helperFactory.setFactory( OBJ\_OBJECTTOBECREATED,

FactoryUtils.reflectionFactory(

ObjectToBeCreated.class));

\_helperFactory.setFactory( OBJ\_ANOTHEROBJECTTOBECREATED,

FactoryUtils.reflectionFactory(

AnotherObjectToBeCreated.class));

setFactory( \_helperFactory);

}

}

Add a note hereAdd a note hereIn [Listing 4.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939), the class MultiIntegrationExamplePool contains two static final strings that are used to describe two unique keys. The unique keys are used to identify the object type when you are borrowing or returning objects to the pool. In [Listing 4.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939), the class KeyedGenericPooledFactory is a custom-defined class part of the book's source code. The purpose of the class KeyedGenericPooledFactory is to associate a key to a specific factory interface instance. In this case, the class KeyedGenericPooledFactory is a data member, but it does not need to be. [Listing 4.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939) shows how to instantiate a specific object type.

Add a note hereAdd a note here**Listing 4.10**

Add a note hereAdd a note hereObject temp = pool.borrowObject(

MultiIntegrationExamplePool.OBJ\_OBJECTTOBECREATED);

Add a note hereAdd a note hereIn [Listing 4.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939), the method borrowObject requires a key, which is the final string defined in [Listing 4.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939). The result from the method call is an object instance that is the class instance ObjectToBeCreated. The unique token does not need to be named something similar to the class name. The unique token could be MY\_SUPER\_DUPER\_OBJECT. However, it is crucial that the object represent something that all developers understand.

Add a note hereAdd a note hereThe examples shown thus far use the *lang* factory package, but we could just as easily have used the *Discovery* factory package. We didn't because the details of the implementation of the *Discovery* package implementation are beyond the scope of this book.

### Add a note hereAdd a note hereControlling the Characteristics of the Pool

Add a note hereAdd a note hereIn [Listings 4.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939) and 4.9, we used the generic implementations of the pool when subclassing to custom pools. We can control the generic pool by using the following attributes, which can be set using standard Java bean setters:

* Add a note hereAdd a note here**maxActive**: This is an integer value that specifies the maximum number of active objects.
* Add a note hereAdd a note here**maxIdle**: This is an integer value that specifics the maximum number of idle objects. If the idle count is exceeded, then those objects are destroyed.
* Add a note hereAdd a note here**maxWait**: This value is used when the whenExhaustedAction flag is set to WHEN\_ EXHAUSTED\_BLOCK. If a client is blocked, then the client will wait the number of milliseconds as specified by the value. If a timeout occurs, the exception NoSuchElementException is thrown.
* Add a note hereAdd a note here**minEvictionIdleTimeMillis**: This determines the amount of time an object can be idle before being destroyed. If an object has been idle for four hours, it has been sitting around too long without being used and hence is wasting resources.
* Add a note hereAdd a note here**testOnBorrow**: This causes the interface method PoolableObjectFactory.validate to be called whenever an object is borrowed from the pool.
* Add a note hereAdd a note here**testOnReturn**: This causes the interface method PoolableObjectFactory.validate to be called whenever an object is returned to the pool.
* Add a note hereAdd a note here**testWhileIdle**: When the idle object inspector thread checks how idle an idle object has been, this flag will force the object inspector thread to validate the idle object. If the validation fails, then the object is automatically discarded.
* Add a note hereAdd a note here**timeBetweenEvictionRunsMillis**: This determines the time that the idle object inspector thread sleeps before making another inspection run of all possible idle objects that can be removed. The time unit is milliseconds.
* Add a note hereAdd a note here**whenExhaustedAction**: This is an enumeration-type value that specifies what occurs when the number of active instances exceeds the maxActive count. Setting this flag with the value WHEN\_EXHAUSTED\_FAIL indicates that an exception of type NoSuchElementException will be thrown. Setting the flag with the value WHEN\_EXHAUSTED\_GROW indicates that the method borrowObject can still be called. Using this value makes the value of the maxActive value irrelevant, since an object will be allocated anyway. Setting the flag with the value WHEN\_EXHAUSTED\_BLOCK indicates that the client will wait until an idle object becomes available.

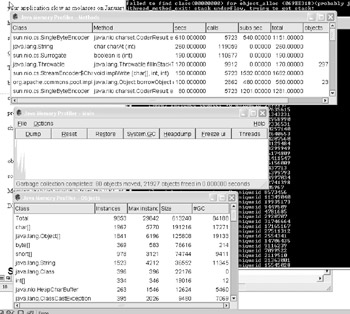
### Add a note hereAdd a note herePool Monitoring

Add a note hereAdd a note hereUsing pools can make your application more responsive. However, setting some of the pool characteristics incorrectly can severely slow down your application because pools are system level-type classes. When you experiment with the various settings, funny things can happen. For example, in the default case of the GenericObjectPool, a too small active object pool size can cause the system to deadlock. The deadlock occurs because all of the active objects have been given away, no extra objects can be instantiated, and no thread is returning an active object back to the pool. Another situation could be that exceptions are thrown and the application reacts incorrectly and causes further problems.

Add a note hereAdd a note hereThe question then becomes how to know which objects should be part of a pool and which objects should not. The answer to this problem is based on monitoring the application using some kind of profiling tool. In this book, we've used Java Memory Profiler (JMP), a monitoring tool available by going to [*www.khelekore.org/jmp/*](http://www.khelekore.org/jmp/). The JMP tool is a shared library or Dynamic Link Library (DLL) that is placed in the operating system path. When the JVM calls the JMP profiler, direct access to the shared library or DLL is required. To activate the JMP profiler, the Java command line needs to include the command line option -Xrunjmp. The JVM uses the -Xrun command to execute a shared library or DLL named jmp (with a .dll or .so extension). When a Java application is executed and the processing time of the application exceeds five seconds, the windows in [Figure 4.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939) appear.

Add a note hereAdd a note hereIn [Figure 4.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939), the three windows that appear monitor all aspects of the application that is being executed. These windows will tell you which objects to manage. The problem in many Java applications is that the Memory Profiler graphic has a saw tooth pattern, as shown in [Figure 4.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939).

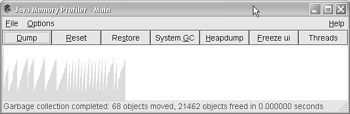
Larger View



Add a note hereAdd a note hereFigure 4.1: An example of JMP windows showing method calls, memory usage, and object usage.

Add a note hereAdd a note hereThe saw tooth pattern in [Figure 4.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939) indicates that the program allocates a number of objects and then the garbage collector comes along and frees a large group of objects at once. Using pools does not allow you to remove the entire saw tooth effect because Java is a garbage-collected environment, which means garbage will be generated. However, you do want to generate less garbage.

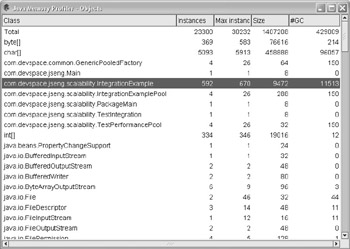
Larger View



Add a note hereAdd a note hereFigure 4.2: A saw tooth pattern that shows memory usage.

Add a note hereAdd a note hereIf a saw tooth pattern is generated, then the next step is to inspect which objects are being garbage-collected. [Figure 4.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939) shows a snapshot of the object usage profile for a particular application.

Larger View



Add a note hereAdd a note hereFigure 4.3: Object profiling.

Add a note hereAdd a note hereIn [Figure 4.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939), the highlighted object is the class IntegrationExample. Beside the class name are four columns:

* Add a note hereAdd a note here**Instances**: This indicates the current number of active instances of the object.
* Add a note hereAdd a note here**Max Instances**: This tells you the maximum number of active instances at any point in time.
* Add a note hereAdd a note here**Size**: This is the size of the object.
* Add a note hereAdd a note here**#GC**: This shows you the number of objects of this type that have been garbage- collected thus far.

Add a note hereAdd a note hereThe class IntegrationExample was highlighted because it shows an excellent example of an object that could be pooled. The first indicator of an object that should be pooled is the total number of times that the object has been garbage-collected. If the number is high, this indicates that the object has been used very often and then discarded. A pooled object would therefore not be garbage-collected as often and hence could improve performance. Also important to consider is the size of the object. Of course, the example class IntegrationExample is not that large; however, when the object is large, it might be useful to put the object into the pool. Another indicator of whether or not the object should pooled is if the number of instances that have been garbage-collected is much greater than the active count. This is called the *factor of difference* and is calculated by dividing the *#GC* by the *Max Instances*. For example, if the factor of difference is five or more, the object should be in the pool. In [Figure 4.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939), the factor is 17, indicating that objects are very commonly created and then deleted.

Add a note hereAdd a note hereOnce an object has been declared for pooling, you should set the parameters. The maximum count should be set to 1.5 times the *Max Instances* count for initial testing purposes. The maximum idle object count should be no larger than half the *Max Instances* count. However, this number depends on the cost of creating the objects and the fluctuation of the object count. The fluctuation of the object is not that simple to measure; you find it out by having experience testing the application.

Add a note hereAdd a note hereYou measure the fluctuation by noting the number of objects that change state as compared to those that remain in a specific state. A high fluctuation means that objects are retrieved very often and then quickly discarded. When the factor of difference is very high (typically greater than 20 or 30), the fluctuation is high and the minimum number of idle objects should be about the same as how many *Max Instances* you have. A lower fluctuation does not require as many idle objects. Of course, if the fluctuation is low, you don't need the pool since the low fluctuation could indicate that objects are referenced for a long time. The key to properly defining the idle count is to be able to keep a constant supply of objects available. If the supply of objects in a high fluctuating system is not available, then the system will be constantly garbage-collecting and creating new instances of objects. As a result, you will be managing the object without a pool. After all is done and said, however, there is no fixed rule about knowing how many idle objects to keep. It is something that is defined for each scenario.

Add a note hereAdd a note hereLook back at [Figure 4.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=118522939), which shows a different class and the statistics highlighted for the class IntegrationExamplePool. The highlighted class was generated artificially to show the situation where an object needs to be managed using pools. The sample numbers should be used to find potentially poolable objects in your sources

## **Threading Using the *Threads* Package**

Add a note hereAdd a note hereOne way to increase scalability is to use multiple threads to work on multiple tasks concurrently. Of course, some may argue that dedicating a process to processing a series of tasks is just as fast as doing all of the tasks in parallel. Part of that argument is correct, but then one could say that DOS would have been good enough since it was not a multi-tasking operating system. And, these days, DOS applications would probably run like a cheetah chasing its kill. However, the reality is that operating systems and programs should do things in parallel because people like to do things in parallel, like start a print job, modify a document, send some e-mail, etc. In all of those conditions, the computer can run all of those tasks concurrently without slowing down the computer. The traditional UNIX user would also make the comment that processes can act like threads and perform operations concurrently. Again, this is true, but running concurrent multiple processes has other problems, like process communications, sharing of data, and speed of thread startup and shutdown. In the end, threads are a good thing, and they increase scalability.

Add a note hereAdd a note hereIn the Jakarta Commons package, there are two threading packages, *threading* and *Threadpool*. The *threading* package has many different capabilities that assist a developer in writing asynchronous-type programs. Asynchronous programs do increase scalability because of their ability to distribute a task among computers and physical locations. However, we will discuss the *threading* package in more detail in [Chapter 6](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=392#392).

Add a note hereAdd a note hereThe *Threadpool* package is a package that deals specifically with making it simpler to write multithreaded applications. In this section, we will discussing lower-level issues that improve scalability.

### Add a note hereAdd a note hereTechnical Details for the *Threadpool* Package

Add a note hereAdd a note here[Tables 4.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679) and [4.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679) contain the abbreviated details necessary to use the *Threadpool* package.

**Add a note hereAdd a note hereTable 4.3:** Repository details for the *Threadpool* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons-sandbox](http://cdcontent.books24x7.com/id_7265/jakarta-commons-sandbox.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note herethreadpool |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.threadpool |

**Add a note hereAdd a note hereTable 4.4:** Package and class details (legend: [threadp] = org.apache.commons.threadpool).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[threadp].ThreadPool | Add a note hereAdd a note hereThe same as for the class StackObjectPool (see [Table 4.2](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=256#256)) |
| Add a note hereAdd a note here[threadp].DefaultThreadPool | Add a note hereAdd a note hereA structural interface used by the Object pool to instantiate and manage the various object instances. |

### Add a note hereAdd a note hereRunning a Thread

Add a note hereAdd a note hereWriting a multithreaded application with the thread pool is extremely simple. The thread pool expects to execute a class that subclasses the class Runnable. [Listing 4.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679) shows an example.

Add a note hereAdd a note here**Listing 4.11**

Add a note hereAdd a note hereclass ExampleThread implements Runnable {

public void run() {

System.out.println( "a task from a thread");

}

}

Add a note hereAdd a note hereIn [Listing 4.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679), the method run is executed by the thread pool and a simple output is generated. To execute this class, use [Listing 4.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679).

Add a note hereAdd a note here**Listing 4.12**

Add a note hereAdd a note hereDefaultThreadPool pool = new DefaultThreadPool();

pool.invokeLater( new ExampleThread());

Add a note hereAdd a note hereIn [Listing 4.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679), the class DefaultThreadPool is the default implementation of the thread pool. To execute a task, the method invokeLater is called. The method will add the instance of the class ExampleThread into a queue. The thread pool then retrieves the task from the queue and executes the task in the context of thread.

Add a note hereAdd a note hereThe thread pool in [Listing 4.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679) is a pool of a single thread because, when the class DefaultThreadPool is instantiated without any parameters, only one thread is created. What happens in the thread pool is that upon creation of the thread pool, a number of threads are started. The exact number depends on the constructor parameter. Once the individual threads have started, they begin pooling the task queue. If the task queue is empty, then the thread will wait for a specific time period before pooling the task queue again. If the task queue has an object instance, then the class method Runnable.run is called. The advantage of using a thread pool is that the number of threads is limited and the threads will execute tasks as they arrive. Of course, in the default case of a thread pool of one thread and two tasks, one task will be waiting, while one is being executed. The exact number of threads you should have in the thread pool depends on the situation—feel free to experiment.

### Add a note hereAdd a note hereImmutable Classes Are Scalable Classes

Add a note hereAdd a note hereThe quality of scalable applications depends on how well the application is written. The sentence sounds too dumb to be logical, but there is quite a bit of truth to the comment. The classical example of this comment is a pregnant woman. Typically, it takes one woman, one man, and nine months to produce a baby. It is not possible under any circumstance to take nine women and one month to produce a baby. This is a fact of nature and indicates that some tasks cannot be split into multiple tasks. The problem, though, is that many people program applications as if the applications were a pregnant woman. Often, business object design is incorrect and objects are too complicated.

Add a note hereAdd a note hereA scalable class is a class that uses the keyword synchronized as little as possible. The keyword synchronized blocks multiple threads from accessing the same piece of code concurrently. [Listing 4.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679) uses a synchronized method.

Add a note hereAdd a note here**Listing 4.13**

Add a note hereAdd a note hereclass ExampleSynchronized {

public synchronized void onlyone() {

System.out.println( "Hello world");

}

}

Add a note hereAdd a note hereIn [Listing 4.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679), the method onlyone allows only one thread access at a time. In this example, the method did not do much other than output some text. Some methods are more complex and will do more work. In those cases, the single method may take longer to return; in addition, if many threads are waiting, a bottleneck is created. Singletons can be bottlenecks because only a single object will typically restrict access. A factory, discussed in [Chapter 3](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=164#164), could be a bottleneck because a factory may allow only one thread to create objects. To be sure that a bottleneck is not created, you should inspect the various sources.

Add a note hereAdd a note hereThe best type of class to write in a multithread and multiprocess scenario is an immutable class. An *immutable class* is a class that does not allow its data to be modified. The best example of an immutable class is String. The class String is an immutable class because the contents can never be changed once they are set. Consider, for example [Listing 4.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679), which is the concatenation of two strings.

Add a note hereAdd a note here**Listing 4.14**

Add a note hereAdd a note hereString a += c;

Add a note hereAdd a note hereIn [Listing 4.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679), the string buffer c is concatenated to the string buffer a. In programming terms, what should happen is that the contents of string buffer c are appended to the already existing string buffer a. The string buffer a would either be expanded or extended to include the contents of c. However, that is not what happens, as illustrated in [Listing 4.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679).

Add a note hereAdd a note here**Listing 4.15**

Add a note hereAdd a note hereString a = "Starting point ";

String temp = a;

String c = "ending point";

a += c;

System.out.println(

"before (" + temp + ") after (" + a + ")");

Add a note hereAdd a note hereThe string buffer a contains a specific buffer. The string buffer Image from book[temp](http://cdcontent.books24x7.com/id_7265/temp.zip) is assigned a reference to string buffer a. The purpose of string buffer Image from book[temp](http://cdcontent.books24x7.com/id_7265/temp.zip) is to be another reference to the same buffer that string buffer a points to. String buffer c is assigned some content, which is then appended to string buffer a. The class method System.out.println then outputs the content of the original buffer pointed to by string buffer a and the buffer after the concatenation of c. This is shown in [Listing 4.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679).

Add a note hereAdd a note here**Listing 4.16**

Add a note hereAdd a note herebefore (Starting point ) after (Starting point ending point)

Add a note hereAdd a note hereThe output of [Listing 4.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679) is strange, because it indicates that string buffer a references a specific buffer and then another buffer after the concatenation. What has occurred is that the appended buffer is a new buffer that contains the contents of string buffers a and c. The old string buffer referenced by a is kept unmodified. This is immutability in action and it ends up being faster than expanding and appending the buffer.

Add a note hereAdd a note hereImmutability is faster because it is an optimization learned by the passage of computing time. In the days of the original C and C++, a buffer was allocated from the heap. A memory manager kept track of the various pieces of memory. In the C and C++ memory model, whenever a piece of data is allocated, the memory manager had to search the heap for an appropriate piece of memory that would fit the need. This searching and slicing of the memory, however, cost many CPU cycles. When an object is immutable, the memory buffer has a binary state: it is either used or not used. The condition that the memory buffer will be expanded or modified never exists. Therefore, the memory manager can optimize operations—although this may waste resources at times, it is faster overall.

Add a note hereAdd a note hereThe other reason why immutable objects are faster is that there is no need for synchronization. When multiple threads access a piece of data, synchronization is needed so that each thread manipulates stable data. If the data is always stable, then synchronization is not necessary. Therefore, immutable objects have a definite speed advantage over objects that have synchronization requirements.

Add a note hereAdd a note hereImmutable objects are not that difficult to write. [Listing 4.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679) shows a simple example.

Add a note hereAdd a note here**Listing 4.17**

Add a note hereAdd a note herefinal class SomeData {

private final int \_value;

public SomeData( int value) {

\_value = value;

}

public SomeData add( int value) {

return new SomeData( \_value + value);

}

public int getValue() {

return \_value;

}

}

Add a note hereAdd a note hereIn [Listing 4.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679), there are several indicators that the class SomeData is immutable:

1. Add a note hereAdd a note hereThe class is declared with the keyword final. This means that another class cannot subclass the class SomeData.
2. Add a note hereAdd a note hereThe data members are declared private and final. This means that data members can be assigned only once within the constructor, ensuring that no other data members can change the values. The data members are private. Therefore, the class may not need to be declared final since the final data members ensure read-only functionality.
3. Add a note hereAdd a note hereThe method add, which adds two numbers together, will return a new instance of the class SomeData. Returning a new instance ensures that the original data members are not updated.
4. Add a note hereAdd a note hereThe getter getValue returns the value of the data member \_value, but there is no associated setter. The lack of a setter ensures that no user of the class SomeData will unintentionally modify the value of the data members.

Add a note hereAdd a note hereIdeally, an immutable class should be a data member class. For example, very few people would consider subclassing the class String. Granted, it is not possible, but even if it were, not many people would consider doing so. When the class String is used, other classes typically encapsulate the class String.

Add a note hereAdd a note hereIn addition, an immutable class would ideally be a data class. A *data class* is a class that stores and manipulates data but does typically not operate on it. Going back to the class String example in [Listing 4.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679), there is an associated class StringBuffer. The class StringBuffer is used to modify the string data in place and is not an immutable class. The difference between using the String class and the StringBuffer class is that scalability may be an issue when you use the class StringBuffer. The separation of a scalable class from a less scalable one makes it easier for programmers to write efficient code.

Add a note hereAdd a note hereThere is a downside to immutable classes in that they can be resource intensive. For example, writing objects that use the operators of the immutable class may require constant object instantiation. As noted earlier, the memory manager will be able to optimize, but there are limitations. If a class allocates four megabytes with each instantiation, then that instantiation will become costly. In that situation, the solution would be to use pooled objects. However, using pooled objects requires that you rewrite [Listing 4.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679) so that it appears like [Listing 4.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679).

Add a note hereAdd a note here**Listing 4.18**

Add a note hereAdd a note hereprivate class PooledSomeData {

private int \_value;

private ObjectPool \_pool;

public PooledSomeData() {

}

public void assign( ObjectPool pool, int value) {

\_pool = pool;

\_value = value;

}

public void reset() {

\_pool = null;

\_value = 0;

}

public PooledSomeData add( int value)

throws Exception {

PooledSomeData cls =

(PooledSomeData)\_pool.borrowObject();

cls.assign( \_pool, \_value + value);

return cls;

}

public int getValue() {

return \_value;

}

}

Add a note hereAdd a note hereIn [Listing 4.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679), the class PooledSomeData has some fundamental changes. The keyword final has been removed because the object instance needs to be reused, which is not possible when you use the keyword final. The exception, of course, is the final used in the class declaration. We've also changed the constructor from one with parameters to one with none. We need to do this because the pool will allocate the class instance and not the method add, like in [Listing 4.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679). We have replaced the constructor with the method assign so that we can initialize the class instance. The method assign has an extra parameter, pool, which is used to borrow objects that will be assigned. We have also added the method reset to initialize the state of the object, which is required when the object pool activates an object.

Add a note hereAdd a note hereThe method add in [Listing 4.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679) has changed dramatically. The method add is still immutable, but instead an object being allocated directly, the class method \_pool.borrow- Object is called to get a pool instance. Then, the class method cls.assign is called to assign a state to the object instance. Finally, the class instance is returned.

Add a note hereAdd a note hereIn [Listing 4.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679), the rewritten class is still immutable even though pooled objects are used. Pooled objects are the ideal solution for data objects that have multiple data members and more complex structures. With pooled objects, the advantages of having larger immutable complex objects are possible. Immutable objects have an additional advantage in that they represent a consistent state. The method add in [Listings 4.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=768413679) and 4.18 is an operation that assigns a state that can be tested and validated. If the validation is not successful, it is very simple to pinpoint where the error occurred. This reduces debugging time and simplifies class maintenance.

## **Summary**

Add a note hereAdd a note hereIn many books, when performance and scalability are talked about, the focus is on raw performance issues, like being able to tweak the bits and bytes. While this type of performance management is important, this book attempts to show how to manage performance using good infrastructure components. Therefore, this book focuses on using Commons libraries that help the programmer write scalable code without having to resort to bit tweaking. There are two very common issues when you're writing scalable code: object instantiation costs and running tasks concurrently.

Add a note hereAdd a note hereAll programs have classes that are more comprehensive and more expensive in resource terms than simple "hello world" examples. It is simply not possible to expect developers to always write lightweight objects; after all, many business processes require you to manage multiple pieces of data. In those cases, the pooled object comes to the developer's rescue. A pool allows a developer to maintain a number of instances on hand for later usage.

Add a note hereAdd a note hereThis chapter also showed you how to use some simple profiling techniques, which help you decide which objects should be garbage-collected and which objects should not be.

Add a note hereAdd a note hereThe other scalability factor is the ability to split the task into execution tasks using a pool of threads. The pool of threads is not a complicated concept or technique; rather, it's a helper class that makes threading simpler. Using threading means that multiple threads can access the same object. Typically, this means that you need to use the keyword synchronize to block specific threads from accessing the same code block at the same time. However, with an immutable object, synchronization is not required. An immutable object is a read-only object that can be viewed, but not changed. If you want to change the state of the object, you use specific operators to create new instances of the object that contain a new state. Immutable objects could become computationally expensive because of the large number of class instantiations. To combat this problem, you use pooled objects and slightly change the design of the classic immutable class.

Add a note hereAdd a note here**On the CD** The sources to the concepts presented in this chapter are located under the directory *[CDROM]/jseng/src/com.devspace.jseng.scalability*

## **Questions**

Add a note hereAdd a note here**4.1:** For the batch file extension component, replace the *Launcher* package part with a thread pool.

Add a note hereAdd a note here**4.2:** Rewrite the calculator extension framework to use object pools. This means the *Discovery* package functionality and *Launcher* package functionality can be discarded and replaced with the new solution.

Add a note hereAdd a note here**4.3:** Rewrite the batch file extension to become a proxy to other extensions. The objective is to be able to specify a list of files, where the numbers are read in by the batch file extension and then stored in an array, which is then passed to the extension. Because the user is able to specify multiple batch files at a time, the batch file extension should use threads and object pools to manage the calculations.

**Chapter 5: Serialization of Java Objects**

**Add a note hereAdd a note here****Overview**

Add a note hereAdd a note hereIn this chapter, we will do the following:

* Add a note hereAdd a note hereConnect a Java application to a database
* Add a note hereAdd a note hereUse an independent storage mechanism that can write to any type of source
* Add a note hereAdd a note herePersist a Java object to an XML source
* Add a note hereAdd a note hereUse the *Lucene* text search engine for XML documents

## **The Purpose of This Chapter**

Add a note hereAdd a note hereA running application is probably a good application, especially if it does not crash. However, a running application without data is a useless application because the application will not be doing anything meaningful. A typical program will interact with the computer using files and databases. This means taking the state of an executing Java object and saving the state to a file or databases, or vice versa. This changing of state is called serialization or persistence.

Add a note hereAdd a note hereSerialization, in the context of this book, takes two forms: database or XML file. The database has proven itself to be a useful and versatile way of storing and manipulating data. With the advent of XML, though, you can cleanly handle unstructured data, in contrast to data in a database. The task in this chapter is to explain how to manage serialization to and from a database and to and from an XML file.

## **Using SQL Helpers**

Add a note hereAdd a note hereA relational database is these days more commonly known as a Structured Query Language (SQL) database. SQL is a programming language like Java, except that SQL deals with the many. For example, when an object is instantiated in Java, a single object is instantiated. Even when you use something like a pool, it always revolves around manipulating a single object. SQL is different in that you don't manipulate the single object or record instance; rather, the single SQL object is a set of data. To generate a set of data, you issue a single command. SQL itself is not properly equipped to deal with single items in a data set. Typically, the data is handed off to a programming language (like Java) that manipulates individual entries of the data set.

Add a note hereAdd a note hereIt's a challenge when you have to map the data from SQL to Java. Mapping from a result set language to a single object language is complex, even in the simplest case. The algorithms are not complex; however, the number of steps required to perform even the simplest of tasks is tedious. This is why component architects and developers invest so much time and effort in making SQL easier.

Add a note hereAdd a note hereThis chapter does not provide any magic bullets, just some solutions that you could apply. The Commons projects contain several SQL and database helper packages. However, the only packages we will discuss here will be *SQL* and *betwixt*. The others are not discussed because the problem of getting the packages to compile properly is too complex.

### Add a note hereAdd a note hereTechnical Details for the *SQL* Package

Add a note hereAdd a note here[Tables 5.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) and [5.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) contain the abbreviated details necessary to use the *SQL* package.

**Add a note hereAdd a note hereTable 5.1:** Repository details for the *SQL* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note heresql |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.sql.\* (there are many subpackages of this package) |

**Add a note hereAdd a note hereTable 5.2:** Package and class details (legend: [lang] = org.apache.commons.lang).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[lang].functor.Factory | Add a note hereAdd a note hereA neutral interface used to instantiate object |
| Add a note hereAdd a note here[lang].functor.FactoryUtils | Add a note hereAdd a note hereA main class that contains all of the functions used to instantiate the different implementations of the Factory interface |
| Add a note hereAdd a note here[lang].functor.FactoryException | Add a note hereAdd a note hereAn exception that is thrown if the Factory classes have encountered a problem |

### Add a note hereAdd a note hereConnecting to the Database

Add a note hereAdd a note hereDoing a simple query with the *SQL* package can be more complicated than just opening a file and reading or writing data. The reason is that the *SQL* package relies on Java Database Connection (JDBC). To get any JDBC connection running, you need a database server, which means more installation and configuration issues. The *SQL* package downloads and uses the Axion and HSQL databases. Both of these databases are Java-based relational databases that use the SQL language, and they are both lighter weight in distribution terms than a database like Oracle. However, the *SQL* package is not limited to these databases and uses the following list of supported databases: Axion, DB2, HSQL, Microsoft SQL Server ™, MySQL ™, Oracle ™, SAP SQL ™, and Sybase ™. Once you have established the JDBC connection parameters, you can create a simple demo, which connects to a database. An example is shown in [Listing 5.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054).

Add a note hereAdd a note here**Listing 5.1**

Add a note hereAdd a note hereDataSourceWrapper datasource = new DataSourceWrapper();

datasource.setDriverClassName("com.mysql.jdbc.Driver");

datasource.setJdbcURL("jdbc:MySQL://localhost/test ");

datasource.setUserName(null);

datasource.setPassword(null);

Connection connection = datasource.getConnection();

JdbcModelReader modelReader = new

JdbcModelReader(connection);

Database database = modelReader.getDatabase();

DynaSql dynaSql = new DynaSql(datasource, database);

Add a note hereAdd a note hereWhen you connect to a database using the *SQL* package, you must take two major steps. The first step is to connect the *SQL* package to the database at the JDBC level. In [Listing 5.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054), we took care of this by setting the correct parameters for the class DataSourceWrapper. We showed you four methods of the class DataSourceWrapper in [Listing 5.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) so that you could see the basics. These four methods basically show all of the functionality that is needed for most situations. The method setDriverClassName is the name of Java driver we'll use to connect to the database. The method setJdbcURL assigns the database connection string. Lastly, the methods setUserName and setPassword define the username and password. The class DataSourceWrapper can then be used to retrieve a connection to the database using the method getConnection. The connection retrieved is based on the JDBC class Connection. However, it is not necessary to call the method getConnection because other parts of the *SQL* package will do that automatically.

Add a note hereAdd a note hereOnce a connection has been established, the second step is to read the metadata from the database. The metadata can be read from an XML configuration file or from the JDBC connection directly. In [Listing 5.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054), the metadata is read from the JDBC connection using the class JdbcModelReader. Once the connection and metadata have been read, you can instantiate the class DynaSql. The class DynaSql is the central class to use when you are performing SQL-type operations such as querying for records or inserting new data.

### Add a note hereAdd a note hereSimplifying Connection Management

Add a note hereAdd a note hereOnce the developer has made the connection to the database, he will want to query, add, modify, or delete data from the database. The *SQL* package is a library that simplifies all of the SQL tasks by adding helper routines to manage the mundane tasks. For example, the manipulation of SQL tables is abstracted into an XML file, and the generation of result sets is managed using Java Beans.

Add a note hereAdd a note hereWhen you use the *SQL* package, you use several dependencies and only a few classes. The typical dependencies are JDBC URL, where the database is located, JDBC class name, and database type. The *SQL* package uses the database type to determine a SQL builder implementation. These dependencies are defined in a user-implemented class that subclasses a yet-to-be-defined *SQL* package helper class, as shown in [Listing 5.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054).

Add a note hereAdd a note here**Listing 5.2**

Add a note hereAdd a note hereclass MySQLImpl extends SQLHelper {

public String getURL() {

return "jdbc:MySQL://localhost/test";

}

public String getDBClass() {

return "com.mysql.jdbc.Driver";

}

public String getDatabaseType() {

return "mysql";

}

}

Add a note hereAdd a note hereIn [Listing 5.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054), the class MySQLImpl implements three methods that are defined as abstract in the subclassed class SQLHelper. We will explain the details of the class SQLHelper in various places throughout the source code, whenever it is appropriate. For now, let's just say that the code presented in [Listing 5.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) has been abstracted into the methods defined in [Listing 5.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054).

Larger View

Add a note hereAdd a note here**Listing 5.3**

Add a note hereAdd a note here protected DataSource createDataSource() throws Exception {

DataSourceWrapper wrapper = new DataSourceWrapper();

wrapper.setDriverClassName( getDBClass());

wrapper.setJdbcURL( getURL());

wrapper.setUserName( getUsername());

wrapper.setPassword( getPassword());

return wrapper;

}

public Database createJDBCDatabase() throws Exception {

DataSource datasource = createDataSource();

Connection connection = datasource.getConnection();

JdbcModelReader modelReader =

new JdbcModelReader(connection);

return modelReader.getDatabase();

}

public Database createXMLDatabase() throws Exception {

DatabaseReader reader = new DatabaseReader ();

return (Database)reader.parse(

new FileInputStream(getXMLDBMetaData()));

}

Add a note hereAdd a note hereThe method createDataSource defined in [Listing 5.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) prepares a connection to the database. The method createJDBCDatabase and createXMLDatabase are used to provide metadata about the database. The difference between the two methods is that createJDBCDatabase retrieves the metadata information from the JDBC driver, whereas createXMLDatabase retrieves it from an XML configuration file.

Add a note hereAdd a note hereThe *SQL* package requires meta information to delete, update, and manipulate records from tables. The package needs metadata because it is fairly difficult to reference a column by name if you do not have the metadata that contains the name of the column.

Add a note hereAdd a note hereWhen metadata is retrieved from the JDBC driver, it is a simple process because the class JdbcModelReader does all of the hard work. The class JdbcModelReader reads the data from the database and then generates a model using the classes defined in the package org.apache.commons.sql.model. However, the JDBC approach requires that the information exist in the first place. It would not be possible to create a table using the JDBC metadata since the table would not exist.

Add a note hereAdd a note hereThe solution the *SQL* package uses is to define an XML file that contains a number of XML elements that represent a database. It is then possible to read in the XML file and use that configuration to create a database. The XML configuration file is written using neutral data types. Various XML-based mapping files are available to the *SQL* package. Contained within the mapping files are data-type mappings from the neutral XML configuration file to the database-expected data type. The end programmer doesn't generally modify the XML mapping files.

### Add a note hereAdd a note hereConfiguring the XML Database

Add a note hereAdd a note hereIn [Listing 5.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054), the method createXMLDatabase loads and processes an XML configuration file. The XML configuration file contains the database configuration that defines the various tables in the database. [Listing 5.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) shows a simple XML configuration file.

Add a note hereAdd a note here**Listing 5.4**

Add a note hereAdd a note here<database name="bookstore">

<table name="author">

<column name="author\_id" type="INTEGER"

primaryKey="true" required="true"/>

<column name="name" type="VARCHAR"

size="50" required="true"/>

<column name="organization" type="VARCHAR"

size="50" required="false"/>

</table>

</database>

Add a note hereAdd a note hereIn [Listing 5.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054), the configuration file is based on the idea of defining a database with a number of tables that have various columns. The configuration file items are all based on XML tags like database, table, and column. The configuration file is read in using some configuration libraries that will be explained in later chapters. The XML structure is converted into a number of Java objects, all of which are located in the package org.apache. commons.sql.model. Here are all of the possible XML tags (and they are all from the package org.apache.commons.sql.model); the child bullets represent XML attributes associated with the XML tag:

* Add a note hereAdd a note here**database**: This represents a database and is the root element of a configuration file. The only acceptable child XML element is table. This tag cross-references with the class Database.
  + Add a note hereAdd a note here**name**: This is used to define the database's name. The name is not used to create a database but is purely for identification purposes.
* Add a note hereAdd a note here**table**: This represents a table in the database. The only acceptable child XML elements are column, foreign-key, index, and unique. This tag cross-references with the class Table. Note that the class Table has more properties, but the package doesn't use them. For example, you can set the attribute catalog and have it read by the configuration libraries, but that data will not be used.
  + Add a note hereAdd a note here**name**: This is used to define the name of the table. You use this identifier when creating the tables.
  + Add a note hereAdd a note here**description**: This is used to define the purpose or description of the table.
* Add a note hereAdd a note here**column**: This represents a table column. This tag cross-references with the class Column.
  + Add a note hereAdd a note here**name**: This is used to define the identifier of the column.
  + Add a note hereAdd a note here**primaryKey**: This is a Boolean value used to indicate if the column is a primary key for indexing purposes.
  + Add a note hereAdd a note here**required**: This is a Boolean value used to indicate if the column must have a value associated with it when a new row is added.
  + Add a note hereAdd a note here**autoIncrement**: This is a Boolean value used to indicate if the column is an auto-increment type. If this attribute is set, then the type of the column must be integer- based.
  + Add a note hereAdd a note here**type**: This is used to define the column's data type. Be sure that the database recognizes the data type, and that the data type has a mapping, so that an appropriate JDBC data type can be used. The class TypeMap contains references to the most commonly used data types.
  + Add a note hereAdd a note here**size**: This is an integer-based value that defines the size of the data type.
  + Add a note hereAdd a note here**defaultValue**: This is the initial value assigned to a record. When a new record is created and no value is assigned to the record, the value of defaultValue is used.
  + Add a note hereAdd a note here**scale**: When you use the DECIMAL and NUMERIC SQL data types, you can specify a number format. The attribute scale is one part of the number format. It indicates that a number is allowed to have as many decimal places as indicated by the value of scale. For example, if the value of scale is 2, then the number 89.99 is acceptable, whereas 89.999 is not.
  + Add a note hereAdd a note here**precisionRadix**: When you use the DECIMAL and NUMERIC SQL data types, you can specify a number format. The attribute precisionRadix is one part of the number format. It indicates the total number of spaces that a value takes. For example, if the value of precisionRadix is 5, then the number 99.99 is valid, whereas 999.99 is not. This particular number is invalid because the positive indicator (which is not shown) is part of the precisionRadix, as is the decimal point. When the sign indicator and numbers before and after the decimal point are added in terms of space, they must be equal to or less than the length specified by precisionRadix.
* Add a note hereAdd a note here**index**: This is used to specify a specific index used to organize a specific table. The only acceptable child XML element is index-column. This tag cross-references with the class Index.
  + Add a note hereAdd a note here**name**: This specifies the name of the index, which is used to reference the index in the database.
* Add a note hereAdd a note here**index-column**: This is used to specify a column used to create an index. The name of the column must equal the name of a column described in the parent XML element table. This tag cross-references with the class IndexColumn.
  + Add a note hereAdd a note here**name**: This is the name of the column used in the index.
* Add a note hereAdd a note here**foreign-key**: This is used to specify a specific foreign key that associates two column values between the parent XML element table and another table defined in the parent XML element database. The only acceptable child XML element is reference. This tag cross-references with the class ForeignKey.
  + Add a note hereAdd a note here**foreignTable**: This is the name of the foreign table being referenced.
* Add a note hereAdd a note here**reference**: This is used to cross-reference two columns in a foreign key relationship.
  + Add a note hereAdd a note here**local**: This is the name of a table column in the parent XML element table.
  + Add a note hereAdd a note here**foreign**: This is the name of a table column in the parent XML element foreign-key referenced table.

Add a note hereAdd a note hereGoing back to [Listing 5.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054), we can use the XML configuration file to generate a database and its tables using [Listing 5.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054).

Add a note hereAdd a note here**Listing 5.5**

Add a note hereAdd a note hereDatabase database = createXMLDatabase()

DataSource datasource = createDataSource();

SqlBuilder sqlBuilder =

SqlBuilderFactory.newSqlBuilder( getDatabaseType());

DDLExecutor executor = new DDLExecutor( datasource,

sqlBuilder);

executor.createDatabase( database, true);

Add a note hereAdd a note hereWhen you're creating a database and its associated tables, the database mappings discussed earlier are provided by the interface SqlBuilder. The interface SqlBuilder is a Commons Bridge implementation, where the implementations reside in the package org.apache.commons.sql.builder. The class method SqlBuilderFactory.newSqlBuilder is a factory used to instantiate the correct implementation. The class DDLExecutor ties together the JDBC connection to the database, the database metadata, and the database implementation details. Using the method createDatabase, all of the classes discussed in [Listing 5.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) are pulled together and used to create a database. The second parameter to the method is a Boolean used to indicate whether or not the tables should be dropped if they already exist. In [Listing 5.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054), the value is true and therefore the tables will be dropped.

Add a note hereAdd a note hereUsing a configuration file is a nice idea because it allows you to create a database on short notice. However, using a configuration file to create a database can also take a long time; [Listing 5.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) has to gather and process data from various sources. Also problematic is that [Listing 5.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) can easily drop tables and add new ones, which could happen if the developer is not careful. Another potential problem occurs when a Database Administrator (DBA) updates the metadata in the database, but does not update the XML configuration file. At that point, the program using the XML configuration file and the database will not be in synch, which could cause data-corruption issues. Using an XML configuration file is very appropriate where systems create their own local databases. In a corporate environment, it's probably not an ideal solution to use an XML configuration file, except if you created an administrative tool based on the XML configuration file. At that point, the administrator would always be in charge and could keep the database metadata in synch with the XML configuration metadata.

### Add a note hereAdd a note hereRunning a Batch Command

Add a note hereAdd a note hereThe SQL language contains commands that return data as well as those that process, but do not return, data (such as the SQL DELETE or CREATE TABLE commands). In either case, the only output that is returned from the command is a success or a failure. If a failure is returned, then the cause and details of the error are returned. The technique used to run a batch command is to use the class defined in [Listing 5.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) to retrieve the class DDLExecutor and then execute the command, as shown in [Listing 5.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054).

Add a note hereAdd a note here**Listing 5.6**

Add a note hereAdd a note hereMySQLImpl impl = new MySQLImpl();

DDLExecutor executor = impl.createDDLExecutor();

executor.evaluateBatch(

"CREATE TABLE 'example\_table'(" +

"'integer\_value' int(11) default '0'," +

"'string\_value' varchar(100) default ''," +

"'double\_value' double default '0')");

Add a note hereAdd a note hereIn [Listing 5.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054), we have used the method evaluateBatch to create a table based on a SQL statement. The method works, but as of this writing the method evaluateBatch was a protected method and not public. A patch has been put in to make the method public, but we don't know whether or not the patch has been accepted. If the patch is rejected, the programmer will have to use [Listing 5.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054), which has exposed the method evaluateBatch.

Add a note hereAdd a note here**Listing 5.7**

Add a note hereAdd a note hereMySQLImpl impl = new MySQLImpl();

impl.evaluateBatch(

"CREATE TABLE 'example\_table'(" +

"'integer\_value' int(11) default '0'," +

"'string\_value' varchar(100) default ''," +

"'double\_value' double default '0')");

Add a note hereAdd a note hereWe use this type of batch processing to allow specific commands to be executed. The SQL command demonstrated in [Listings 5.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) and 5.7 shows how to create a table in a database. This type of SQL command is a good example of what not to do. The problem with the CREATE TABLE command is that the command could be very specific to the database used. If we use the CREATE TABLE command, we'll be defeating the purpose of the *SQL* package. Other SQL commands—such as SELECT and INSERT—are not as database specific and hence could make effective use of the method evaluateBatch.

### Add a note hereAdd a note hereManipulating Database Data

Add a note hereAdd a note hereCertain SQL commands allow you to add, update, or remove data. In SQL terms, we're talking about the commands INSERT, UPDATE, and DELETE. However, in terms of the *SQL* package, we mean using dynamic beans that allow you to add, modify, or delete data. In the *SQL* package, there is a notion of property bag, which stores data using key value pairs. The property bag is a Java class that can be used in conjunction with SQL commands executed using the SQL builder class. In [Listing 5.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054), a record is inserted into the SQL table example\_table.

Add a note hereAdd a note here**Listing 5.8**

Add a note hereAdd a note hereMySQLImpl impl = new MySQLImpl();

DynaSql dynaSql = impl.createJDBCDynaSql();

DynaBean bean = dynaSql.newInstance( "example\_table");

bean.set( "integer\_value", new Integer( 1234));

bean.set( "string\_value", "hello how are you doing");

bean.set( "double\_value", new Double( 1234.567890));

dynaSql.insert( bean);

Add a note hereAdd a note hereRegardless of which SQL data manipulation command is called, you need an instance of the class DynaSql (as is done in [Listing 5.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054)). If you want to insert data into a specific table, you need to retrieve a reference to the table using the method newInstance. The method newInstance has one parameter, which is the name of the table, and it returns an instance to the class DynaBean. The class DynaBean is a sort of property bag where specific key value pairs are stored. In [Listing 5.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054), the key value pairs are column names and their associated values. Notice that when a numeric value is assigned, the classes Integer and Double are used. This is because the class DynaBean manipulates objects and not primitives like int and long. To add the contents of the class DynaBean, you call the method insert, where the parameter of the method is the instance of class DynaBean.

Add a note hereAdd a note hereThe technique that the method insert in [Listing 5.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) uses is introspection of the Dyna-Bean to generate a SQL command. Once the SQL command has been generated, it is executed. In the same fashion, the method delete will generate a SQL DELETE command. The SQL DELETE command has an additional requirement in that the record has to be specified. The *SQL* package searches for the foreign key collection and uses the foreign keys to specify which records to delete. If the table does not have a foreign key collection, you can't use the *SQL* package to delete or update records.

Add a note hereAdd a note hereUsing a query is very similar to inserting data, except that the query requires a string that contains a SQL SELECT command. When the query returns a result set, the result set is converted by the *SQL* package to an iterator. [Listing 5.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) shows a sample query.

Add a note hereAdd a note here**Listing 5.9**

Add a note hereAdd a note hereIterator iter = dynaSql.query(

"select integer\_value, linked\_value from " +

"example\_table, linked\_table " +

"where example\_table.integer\_value = " +

"linked\_table.linked\_value");

while( iter.hasNext() == true) {

DynaBean foundBean = (DynaBean) iter.next();

System.out.println( "Integer value is " +

foundBean.get( "integer\_value"));

}

Add a note hereAdd a note hereIn [Listing 5.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) the method query accepts any SQL command that returns a dataset. Two tables are linked together to generate the output. The method query returns an instance of the class Iterator. The Iterator class instance contains the entire dataset of the SQL query in terms of individual DynaBean class instances. To iterate the individual classes, you use the methods hasNext and next. To retrieve an individual value from the DynaBean class instance, you use the method get, where the first parameter is the name of the column to retrieve.

Add a note hereAdd a note hereFor the method query, the SQL command can be the SQL SELECT command, or it can be a stored procedure. In any case, *SQL* package does not translate or fix up the SQL command. When you use a stored procedure, you will have to be database-specific. A stored procedure is called like a function call, where there is function name and some parameters. [Listing 5.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) is an example where the query string is a complete string. When you call a stored procedure, the parameters need to be converted into a complete string. This process is time-consuming and potentially error prone. Another solution is to use parameterized queries, as shown in [Listing 5.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054).

Add a note hereAdd a note here**Listing 5.10**

Add a note hereAdd a note hereList params = new ArrayList();

params.add( new Integer( 1234));

Iterator iterpara = dynaSql.query(

"Select \* from example\_table where integer\_value = ?",

params);

while( iterpara.hasNext() == true) {

DynaBean foundBean = (DynaBean)iterpara.next();

System.out.println( "Integer value is " +

foundBean.get( "integer\_value"));

}

Add a note hereAdd a note hereTo make a parameterized query, you use the same method query that we used in [Listing 5.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054). However, instead of passing in a complete string, we use two parameters. The first parameter is still a complete string, and the second is an array of parameters that are substituted in the parameterized query. A parameterized query is a buffer that contains SQL commands, but in specific places a question mark represents a variable. The question marks are placeholders that are then replaced by values, which in [Listing 5.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=143254054) are array elements from the ArrayList class instance. The elements are replaced in sequential order, meaning that the first question mark in the SQL command is replaced with the first array element. In this instance, the Iterator class instance that is returned still contains the dataset from the query, which is a list of DynaBean class instances.

## **Serializing Beans to XML and Back Again**

Add a note hereAdd a note hereWriting applications using SQL commands is a very good strategy if you are using current technologies. For example, many companies will have an Oracle Database or Microsoft SQL Server Database. Using SQL has proven to be an effective way of storing data. The problem with SQL is in how you serialize to a data structure. This process can often be very complicated and requires some processing power. When you build applications using these technologies, it is typically called "building an *n*-tier application."

Add a note hereAdd a note hereIn this section of the chapter, XML is used to store and retrieve data. The power of SQL is its ability to sort and find data when the data is organized in a concise manner. The problem with using SQL is that in a longer-term application, earlier created database tables may not be compatible with ones designed more recently. Incompatibility occurs because you cannot upgrade SQL data partially. Either a column is added to the table or a column is removed. The only other option, if you want to partially upgrade a table, would be to create a new table. If you used this solution, you would have to copy and maintain the data in two locations. Regardless of how you define SQL tables, they are very hard to upgrade. Consider the XML in [Listing 5.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.11**

Add a note hereAdd a note here<data>

<today>I defined something</today>

</data>

Add a note hereAdd a note hereLet's say that the XML in [Listing 5.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) was created at a point in time called *X*. We would then create the XML in [Listing 5.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) at a point in time called *X* + *N*, where *N* is some later time.

Add a note hereAdd a note here**Listing 5.12**

Add a note hereAdd a note here<data>

<today>I defined something</today>

<tomorrow>Some more data</tomorrow>

</data>

Add a note hereAdd a note hereThe interesting part of XML is that the data added in [Listing 5.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) does not affect the original data. Therefore, any application that reads the original data will still be able to function. In addition, any application that modifies the new data can cope with the old data. This can occur in XML because XML is a structured document that allows two entirely different pieces of data to be embedded in the same document.

Add a note hereAdd a note hereIn the Internet world, searching and finding data has become a well-defined science. Take, for example, a search engine, which crawls and attempts to index the Internet. While it crawls the Internet, documents are archived and stored for later reference purposes. Therefore, when you ask a search engine to find a document or some information, you can pinpoint what you need. Put a search engine together with an XML document and the result is a modern database system.

### Add a note hereAdd a note hereTechnical Details for the *betwixt* Package

Add a note hereAdd a note hereThis section of the chapter will discuss the *betwixt* package. Using this library, you can serialize Java objects to XML and back again. [Tables 5.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) and [5.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) provide an abbreviated description of the *betwixt* package.

**Add a note hereAdd a note hereTable 5.3:** Repository details for the *betwixt* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[Jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note hereBetwixt |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.betwixt.io.\*, org.apache.commons.betwixt.\*, org.apache.commons.betwixt.strategy.\*, java.io.\* (for the serialization and deserialization) |

**Add a note hereAdd a note hereTable 5.4:** Package and class details (legend: [lang] = org.apache.commons.betwixt).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[betwixt].io.BeanWriter | Add a note hereAdd a note hereA class used to read XML files that are serialized beans. |
| Add a note hereAdd a note here[betwixt].io.BeanReader | Add a note hereAdd a note hereA class used to write XML from a Java Bean class. |
| Add a note hereAdd a note here[betwixt].XMLIntrospector | Add a note hereAdd a note hereA class used to introspect specific Java Bean classes used to serialize and deserialize. |
| Add a note hereAdd a note here[betwixt].strategy.\* | Add a note hereAdd a note hereAll of the classes and interfaces in this package are relatively important because they allow a developer to customize the overall serialization and deserialization process. |

### Add a note hereAdd a note hereWriting a Java Bean

Add a note hereAdd a note hereThe main objective of the *betwixt* package is to serialize a Java class to and from an XML file. It is important to remember that serialization is not the same as saving data to a SQL database. Serialization is the process of saving the state of an object to another medium, whereas saving data to a SQL database is saving the state of a running business process. The Java serialization file format is specific to Java. In contrast, practically every programming language that exists can parse the XML format. What the *betwixt* package does is override the default serialization mechanism and add support for XML. The *betwixt* package requires that a Java Bean be able to serialize XML content. [Listing 5.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) shows a sample Java Bean.

Add a note hereAdd a note here**Listing 5.13**

Add a note hereAdd a note herepackage com.devspace.jseng.serialization;

public class BeanToWrite implements java.io.Serializable {

private int \_iValue;

private String \_strValue;

public BeanToWrite( int ival, String sval) {

\_iValue = ival;

\_strValue = sval;

}

public int getIntegerValue() {

return \_iValue;

}

public void setIntegerValue( int val) {

\_iValue = val;

}

public String getStringValue() {

return \_strValue;

}

public void setStringValue( String val) {

\_strValue = val;

}

}

Add a note hereAdd a note hereFor a Java Bean to be suitable for XML serialization, the Java Bean must be a public class, as shown in [Listing 5.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445). If the class is not a public class, then an access error will occur. The problem is not that the exception is thrown but that the exception is caught and an unpredictable result occurs. This results in an empty data set. If the Java class exposes public data members like in [Listing 5.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), those data members will not be serialized.

Add a note hereAdd a note here**Listing 5.14**

Add a note hereAdd a note herepublic class TestClass {

public int SomePublicDataMember;

}

Add a note hereAdd a note hereOnce the Java Bean has been defined, which in our examples is by the class BeanToWrite, it can be used by the *betwixt* package and serialized to XML. The simplest example of XML serialization is shown in [Listing 5.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.15**

Add a note hereAdd a note hereimport org.apache.commons.betwixt.io.BeanWriter;

BeanToWrite somebean = new BeanToWrite( 1234, "hello

world");

BeanWriter writer = new BeanWriter();

writer.enablePrettyPrint();

writer.write( somebean );

Add a note hereAdd a note hereThe main class in [Listing 5.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is the class BeanWriter. This class is responsible for serializing the Java Bean to XML. In [Listing 5.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the method write has one parameter, which is the Java Bean to serialize. The class BeanWriter exposes several helper methods (and in [Listing 5.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), enablePrettyPrint is one of the methods):

* Add a note hereAdd a note here**enablePrettyPrint:** When the XML output is generated, this enables the pretty printing of the XML content, which means humans can read it easily.
  + Add a note hereAdd a note here**writeXmlDeclaration:** Before serialization, this enables the writing of an XML declaration to the serialized XML. An example is <?xml version='1.0' ?>.
  + Add a note hereAdd a note here**getIndent**, **setIndent:** When the you pretty print the XML output, this retrieves or sets the string used to specify an indent in the output string.
  + Add a note hereAdd a note here**getLog**, **setLog:** This retrieves or sets the logging handler used to capture log events.
  + Add a note hereAdd a note here**getWriteIDs**, **setWriteIDs:** These retrieve or set a Boolean flag used to determine whether or not the Java Bean IDs are written.
  + Add a note hereAdd a note here**getWriteEmptyElements**, **setWriteEmptyElements:** These retrieve or set a Boolean flag used to determine whether or not to write an empty element. An empty element is one where the value of the element does not reference any meaningful data. For example, in [Listing 5.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the getter for the property StringValue could have returned a null value. In that case, if the WriteEmptyElements flag were set to false, then the getter StringValue would not be serialized in the XML data stream. If the flag value were set to true, then the property StringValue would be written; however, an empty XML data set will be generated.

Add a note hereAdd a note here[Listing 5.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) shows the XML data stream that's generated when [Listing 5.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is executed.

Add a note hereAdd a note here**Listing 5.16**

Add a note hereAdd a note here<BeanToWrite>

<integerValue>1234</integerValue>

<stringValue>hello world</stringValue>

</BeanToWrite>

Add a note hereAdd a note herePlease note that [Listing 5.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is the default example of a Java Bean being written into an XML stream. We mention this because the XML data stream XML tags (BeanToWrite, integerValue, and stringValue) are based on the default *betwixt* naming strategy. The root XML element is BeanToWrite, which maps one-to-one to the name of the Java Bean being serialized. Notice, though, that the name of the XML element does not include the namespace of the Java Bean. This is an important consideration because if two identically named Java Beans with different namespaces were serialized, they would use the same XML tag. In [Listing 5.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), notice the XML tag element identifiers integerValue and stringValue. Java Bean Patterns uses these names to identify various properties.

### Add a note hereAdd a note hereConsistent XML Naming Conventions and Name Mappings

Add a note hereAdd a note hereThe name of XML tag in [Listing 5.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is identical to the name of the Java Bean property in [Listing 5.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445). This is not a problem and follows the Java Bean encoding rules. What is a problem is that in [Listing 5.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), some XML tags start with a capital letter and some start with a lower case letter. This is not consistent and not a good naming practice. Although we said that [Listing 5.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is the default XML serialization, the reality is that the default naming standards will never be used. What will be used, however, is all lower case, all upper case, all hyphenated, or some combination. The power of the *betwixt* package is not its ability to serialize a Java Bean to XML but its ability to let a developer control the serialization. To show you what we mean, we'll convert the XML output in [Listing 5.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) to first letter lowercase entirely. [Listing 5.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) shows the results.

Add a note hereAdd a note here**Listing 5.17**

Add a note hereAdd a note hereBeanToWrite somebean = new BeanToWrite( 1234, "hello world");

BeanWriter writer = new BeanWriter();

XMLIntrospector introspector = new XMLIntrospector();

introspector.setElementNameMapper( new

org.apache.commons.betwixt.strategy.DecapitalizeNameMapper()

);

writer.setXMLIntrospector( introspector);

writer.enablePrettyPrint();

writer.write( somebean );

Add a note hereAdd a note hereTo be able to generate a different case using the *betwixt* package, the class XMLIntrospector is used in [Listing 5.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445). The purpose of the class XMLIntrospector is to allow a developer to override specific defaults so that he can custom-tailor the XML serialization. In [Listing 5.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the method setElementNameMapper is used to replace the default XML tag naming to the class instance of DecapitalizeNameMapper. In the *betwixt* package, XML tag naming is called "element name mapping." Running [Listing 5.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) yields the output generated in [Listing 5.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.18**

Add a note hereAdd a note here<beanToWrite>

<integerValue>1234</integerValue>

<stringValue>hello world</stringValue>

</beanToWrite>

Add a note hereAdd a note hereNow, [Listing 5.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) contains consistent XML tag names because all of the first letters of the individual XML tags are lowercase. The following name mappers are available in the org.apache.commons.betwixt.strategy package:

* Add a note hereAdd a note here**DefaultNameMapper:** This does nothing; it's just a placeholder for the XMLIntrospector class.
* Add a note hereAdd a note here**DecapitalizeNameMapper:** This changes the first letter of all XML elements to be lowercase.
* Add a note hereAdd a note here**CapitalizeNameMapper:** This changes the first letter of all XML elements to be uppercase.
* Add a note hereAdd a note here**HyphenatedNameMapper:** This is a flexible name mapper that hyphenates and changes the case of the first letter for multiword identifiers. For example, in [Listing 5.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), there was the XML tag beanToWrite, which would be changed to the XML tag identifier to use hyphens instead of capitalization like bean-to-write. The hyphen is used to separate word sections in the identifier. The use of hyphens is a popular form in the XML world. To be able to separate an identifier into individual word sections, you need to use an algorithm. The algorithm that is used to detect a word break is based on *Camel Humped Naming*, which is when the first letter of a multiword identifier is a capital, and the second or thereafter letter is lowercase. When a word break is found, a separator is inserted. The default is a hyphen character. You can define the separator by using the class method HyphenatedNameMapper.setSeparator. In the default case, the individual broken words are set to lowercase letters. Using the class method HyphenatedNameMapper.setUpperCase(true), you can set the first letter of the individual broken words to an uppercase letter.

### Add a note hereAdd a note hereWriting Your Own Name Mapper

Add a note hereAdd a note hereThe name mappers available in the *betwixt* package cover most of the bases. However, sometimes a developer may want to create his own name mapping. The developer may want his own name mapping for a business process reason or to use a specific XML Schema. The reason is not important. What is important is how a custom name mapper can be implemented. It is not a difficult task and is as simple as the code in [Listing 5.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.19**

Add a note hereAdd a note hereclass ItsANameMapper implements NameMapper {

public String mapTypeToElementName(String typeName) {

return " its\_a\_" + typeName;

}

}

Add a note hereAdd a note hereIn [Listing 5.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the class ItsANameMapper implements the interface NameMapper, which implements the single method mapTypeToElementName. The purpose of the method mapTypeToElementName is to convert the input parameter typeName to an acceptable identifier and then return the modified value. In the case of [Listing 5.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the text "its\_a" is prefixed to all elements. The generated output shown in [Listing 5.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is a bit silly, but it shows the result of the sample name mapper.

Add a note hereAdd a note here**Listing 5.20**

Add a note hereAdd a note here<its\_a\_BeanToWrite>

<its\_a\_integerValue>1234</its\_a\_integerValue>

<its\_a\_stringValue>hello world</its\_a\_stringValue>

</ its\_a\_BeanToWrite>

Add a note hereAdd a note hereIf you read the XML result in [Listing 5.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), there is a typo. The identifier integerValue starts with a vowel; hence, the text should have read its\_an\_integerValue. We could have added that logic to [Listing 5.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), but its implementation is beyond the scope of this book. The point is that you can define a custom name mapper without affecting other parts of the XML generation process.

Add a note hereAdd a note hereAs a last note, the class ItsANameMapper is used in the same manner as the class DecapitalizeNameMapper is in [Listing 5.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445). [Listing 5.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is a shortened listing that uses the class ItsANameMapper.

Add a note hereAdd a note here**Listing 5.21**

Add a note hereAdd a note hereXMLIntrospector introspector = new XMLIntrospector();

introspector.setElementNameMapper( new ItsANameMapper());

### Add a note hereAdd a note hereSerializing Java Beans That Reference Java Beans

Add a note hereAdd a note hereThe serialization process for the Java Bean in [Listing 5.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is a simple single Java Bean that does not reference other Java Beans. While the *betwixt* package does automatic serialization for most Java Bean features, special circumstances sometimes arise. For example, if a Java Bean references a collection of other Java Beans, you might wonder how those objects would be serialized. The various Java Bean structures have predictable XML serialization structures, but it is important to know the various situations.

Add a note hereAdd a note hereAfter a single Java Bean serialization, the next level of complexity is when a Java Bean references another Java Bean. In [Listing 5.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), a new bean will be created and it will reference the Java Bean in [Listing 5.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.22**

Add a note hereAdd a note herepublic class ParentBean {

private String \_dataMember;

private BeanToWrite \_bean;

public ParentBean( String val) {

\_bean = new BeanToWrite( 1234, val);

\_dataMember = val;

}

public String getDataMember() {

return \_dataMember;

}

public void setDataMember( String val) {

\_dataMember = val;

}

public BeanToWrite getMyReferenceToAnotherBean() {

return \_bean;

}

}

Add a note hereAdd a note hereIn [Listing 5.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the class ParentBean references the class BeanToWrite and exposes the property myReferenceToAnotherBean. The additional property dataMember is added in the XML serialization to illustrate how a child object would be serialized. The output of serializing the class ParentBean is [Listing 5.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.23**

Add a note hereAdd a note here<ParentBean>

<dataMember>hello world</dataMember>

<myReferenceToAnotherBean>

<integerValue>1234</integerValue>

<stringValue>hello world</stringValue>

</myReferenceToAnotherBean>

</ParentBean>

Add a note hereAdd a note here[Listing 5.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) used default XML serialization to generate the output. What is important to notice is how the bean reference \_bean in [Listing 5.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is serialized. When the class BeanToWrite is serialized, there is no BeanToWrite XML tag. Compare that to the generated output of [Listing 5.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), where there is a BeanToWrite XML tag. The name of the XML tag, which should have been the XML tag BeanToWrite, is myReferenceToAnotherBean, which is the same name as the Java Bean property of the parent Java Bean defined in [Listing 5.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445). The problem with this serialization is that you can serialize the same structure in multiple ways. Another approach to serializing the same Java Beans is shown in [Listing 5.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.24**

Add a note hereAdd a note here<ParentBean>

<dataMember>hello world</dataMember>

<BeanToWrite>

<integerValue>1234</integerValue>

<stringValue>hello world</stringValue>

</BeanToWrite>

</ParentBean>

Add a note hereAdd a note hereIn the [Listing 5.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the XML tag myReferenceToAnotherBean is replaced with BeanToWrite. This solution is what many XML structures expect, because the class ParentBean and BeanToWrite have nothing in common with each other (other than that one class references the other). XML's power is that it can embed another document and structure without ruining the higher-level structure. Therefore, it would appear that we should have implemented the first approach.

Add a note hereAdd a note hereUsing the approach from [Listing 5.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) does have its issues as well. Imagine building a mortgage application. In the first release of the mortgage application, there can be only one signer of the mortgage. Imagine that another person wants to sign for the mortgage. There would be two co-signers. The problem then is how to store the two co-signers as XML elements in the document. From an XML point of view, the solution is simple; just store both of them because the XML document does not have a problem with that. Where the problem becomes difficult is when the *betwixt* package needs to read the two co-signers of the mortgage. Which one will be read and how will both be stored? In the original approach, each co-signer could be uniquely identified as a unique XML tag. A solution would be to use an array, but even that has problems; a mailing address and domicile address, while being the same type, have very different purposes logically.

Add a note hereAdd a note hereAnother approach to the serialization of the Java Bean is shown in [Listing 5.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.25**

Add a note hereAdd a note here<ParentBean>

<dataMember>hello world</dataMember>

<myReferenceToAnotherBean>

<BeanToWrite>

<integerValue>1234</integerValue>

<stringValue>hello world</stringValue>

</BeanToWrite>

</myReferenceToAnotherBean>

</ParentBean>

Add a note hereAdd a note hereIn [Listing 5.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), both the XML elements myReferenceToAnotherBean and BeanToWrite are embedded. In this approach, you don't have any problem figuring out which part of the document belongs to which Java Bean. However, this approach is not an ideal solution, since it indicates that two entirely different documents are embedded. In the case of a mortgage application, this is not the case. This approach would be legitimate when you are writing SOAP packets. SOAP is a Web Services XML specification and explicitly defines that there is an outer package and inner package. Think of it like a letter that's ready to mail; it's composed of the paper the letter is written on and the envelope that encloses the letter.

Add a note hereAdd a note hereWe can see from these examples that there is no ideal way of serialization and that the default mechanism shown in [Listing 5.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is good enough. Later in this chapter, we will explain a more sophisticated technique on how to control the serialization process.

### Add a note hereAdd a note hereSerializing Collections

Add a note hereAdd a note hereWhen a Java Bean references another Java Bean, that is a one-to-one relationship. Using collections, you can have a one-to-many relationship. A collection contains and references many other objects, which could contain another one-to-many relationship. To keep things simple, let's consider a simple collection using the class Vector, shown in [Listing 5.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.26**

Add a note hereAdd a note herepublic class CollectionBeanToWrite {

private java.util.Vector \_items =

new java.util.Vector();

public void addItem( BeanToWrite bean) {

\_items.add( bean);

}

public java.util.Iterator getItems() {

return \_items.iterator();

}

}

Add a note hereAdd a note hereIn [Listing 5.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), a *betwixt* package-type "Java Bean" is defined. [Listing 5.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) could also be defined as a *betwixt* bean. The methods addItem and getItems use a naming convention that the *betwixt* package programmers have defined. These two methods define a getter and a setter for collections. The difference between an individual getter and setter is that the *betwixt* collection getters and setters retrieve a collection and set individual objects. From a programming point of view, this makes sense because when serializing to XML, the *betwixt* package wants an interface instance of type Iterator to iterate through the individual objects. However, when serializing from XML, the *betwixt* package wants to be able to instantiate an object, serialize it, and then add the object to the collection.

Add a note hereAdd a note hereThe *betwixt* getter and setter notation is identical to [Listing 5.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.27**

Add a note hereAdd a note here public void add[PROPERTY NAME]( [OBJECT TYPE] bean) {

}

public Iterator get[PROPERTY NAME]s() {

}

Add a note hereAdd a note hereIn contrast to Java Beans, which require a get and set, the *betwixt* collection bean notation, shown in [Listing 5.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), requires an add and get. However, the get requires that you append an s after the property name. The concept of adding an s to the method get is called *pluralization*. Other plurals that are supported are Array, List, and Iterator. The object type is specified only by the method add. In [Listing 5.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the method get returns an interface instance of type Iterator. Also supported are get methods that return an object array, collection, enumeration, or map.

Add a note hereAdd a note hereThe code defined in 5.26 is executed using the same consumer source code as shown in [Listing 5.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445). The generated result is shown in [Listing 5.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.28**

Add a note hereAdd a note here <CollectionBeanToWrite>

<items>

<item>

<integerValue>1</integerValue>

<stringValue>first item</stringValue>

</item>

<item>

<integerValue>2</integerValue>

<stringValue>second item</stringValue>

</item>

<item>

<integerValue>3</integerValue>

<stringValue>third item</stringValue>

</item>

</items>

</CollectionBeanToWrite>

Add a note hereAdd a note hereIn [Listing 5.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the naming convention is very similar to all of the previously defined identifiers. The collection is encoded within the XML tag items. The identifier items is based on the property items from the class CollectionBeanToWrite. Within the XML element items are a number of XML child elements item. The identifier item is not the singular form of the identifier items; rather, it's a default identifier defined by the *betwixt* package. Then, within the individual XML child elements, item is the serialized contents from the class BeanToWrite.

### Add a note hereAdd a note hereSerializing Maps

Add a note hereAdd a note hereWhen you serialize a Map-based class, the output is slightly different than a simple collection or array. A Map is a key value pair type of collection. Unlike with the XML output, the changes we need to make in the listings are relatively minor. [Listing 5.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) shows the methods that need to be added to allow *betwixt* to serialize a Map.

Larger View

Add a note hereAdd a note here**Listing 5.29**

Add a note hereAdd a note herepublic class MappedBeanToWrite {

private java.util.HashMap \_items = new java.util.HashMap();

public MappedBeanToWrite() {

}

public void addItem( String key, BeanToWrite bean) {

\_items.put(key, bean);

}

public java.util.Map getItems() {

return \_items;

}

}

Add a note hereAdd a note hereIn [Listing 5.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the class MappedBeanToWrite uses the HashMap to store the key value pair. Using serializing Map's, the *betwixt* serialization routines search for the proper getter and adder methods, like when serializing arrays or lists. However, since a Map is associated with a key value pair, the adder method has two parameters: string (which represents the key) and object (which represents the value). When the class MappedBeanToWrite is serialized, the output in [Listing 5.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is generated.

Add a note hereAdd a note here**Listing 5.30**

Add a note hereAdd a note here <MappedBeanToWrite>

<items>

<entry>

<key>third item</key>

<value>

<integerValue>3</integerValue>

<stringValue>third item</stringValue>

</value>

</entry>

<entry>

<key>first item</key>

<value>

<integerValue>1</integerValue>

<stringValue>first item</stringValue>

</value>

</entry>

<entry>

<key>second item</key>

<value>

<integerValue>2</integerValue>

<stringValue>second item</stringValue>

</value>

</entry>

</items>

</MappedBeanToWrite>

Add a note hereAdd a note hereThe output that is generated is very similar to the output generated in [Listing 5.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445). The difference is that instead of the XML child elements item, the XML child element entry is generated. Within the XML child element entry are two XML child elements, key and value. The XML child element key contains the key that was given in the method add. The XML child element value contains the serialization of the class BeanToWrite. The XML elements entry, key, and value are defaults provided by the *betwixt* package.

### Add a note hereAdd a note hereGenerating and Renaming XML Attributes

Add a note hereAdd a note hereWe introduced the class XMLInspector when we talked about generating custom name mappings for XML elements. However, this class can do other things that allow the developer to influence how the XML content is serialized. In [Listing 5.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the XMLIntrospector class was used to define a new naming strategy for individual XML elements. You can use the same code to rename individual attributes, as shown in [Listing 5.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.31**

Add a note hereAdd a note hereParentBean bean = new ParentBean( "hello world");

BeanWriter writer = new BeanWriter();

XMLIntrospector introspector = new XMLIntrospector();

introspector.setElementNameMapper( new

org.apache.commons.betwixt.strategy.HyphenatedNameMapper());

introspector.setAttributeNameMapper( new

org.apache.commons.betwixt.strategy.HyphenatedNameMapper());

introspector.setAttributesForPrimitives( true);

writer.setXMLIntrospector( introspector);

writer.enablePrettyPrint();

writer.write( bean );

Add a note hereAdd a note here[Listing 5.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) uses the same ParentBean class used in the previous serialization examples, except now more *betwixt* classes are used. The class ParentBean is the multiple-reference Java Bean that will be serialized. The XMLIntrospector is allocated pretty much the same way as it was previously. The method setElementNameMapper modifies the individual XML elements. The method setAttributeNameMapper is a new method used to modify the naming of the individual XML attributes. This method, while useful, does not make much sense since all of the XML generated thus far has not had any XML attributes. A way of generating attributes is to call the method setAttributesForPrimitives with a parameter of true. This converts the class properties into XML attributes instead of XML child elements. When the class instance of XMLIntrospector is assigned to the class instance of BeanWriter, there can be only one assignment. It is not possible to assign multiple class instances of XMLIntrospector. If [Listing 5.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is executed, the output in [Listing 5.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is generated.

Add a note hereAdd a note here**Listing 5.32**

Add a note hereAdd a note here<parent-bean data-member="hello world">

<my-reference-to-another-bean

integer-value="1234" string-value="hello world"/>

</parent-bean>

Add a note hereAdd a note hereIn [Listing 5.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), XML elements are only generated for each of the class instances. The class ParentBean represents itself and the class BeanToWrite, so there are only two XML elements. The properties on each of the class instances are serialized to XML attributes. This strategy of serializing XML is OK to use; however, this approach tends to resemble a record set approach and is not that XML friendly. A bit later in this section of the chapter, we will discuss an XML serialization strategy.

### Add a note hereAdd a note hereFine-Tuning Plural Descriptors

Add a note hereAdd a note hereWhen *betwixt* generates collections, the default mechanism of finding both the add and get is to associate the singular form with the plural form. For example, in [Listing 5.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the singular is the identifier item and the plural form is items. The singular and plural default forms work only for the English language. Therefore, it would be beneficial if the singular and plural forms of other languages could be recognized as well. For example, in German the singular form of "book" is "buch," but the plural is "bßcher." The German plural form of the word "book" is rather complicated, because a new letter, ß, is substituted and an "er" is added to the end of the word. We need a more complicated pluralization rule. The way to implement this sort of rule is similar to how you implement a custom name mapping. However, you support a different interface. In [Listing 5.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), a collection class that uses German singular and plural forms of the word "book" is defined.

Add a note hereAdd a note here**Listing 5.33**

Add a note hereAdd a note herepublic class GermanCollection {

private java.util.Vector \_items = new

java.util.Vector();

public void addBuch( BeanToWrite bean) {

\_items.add( bean);

}

public java.util.Iterator getBuecher() {

return \_items.iterator();

}

}

Add a note hereAdd a note hereMaking the *betwixt* package aware of the German collection requires an implementation class that implements the interface PluralStemmer. And, like in the name-mapping example in [Listing 5.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the implementation class needs to be associated with the class XMLIntrospector. The implementation class is defined in [Listing 5.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Larger View

Add a note hereAdd a note here**Listing 5.34**

Add a note hereAdd a note hereclass GermanPluralMapper implements

org.apache.commons.betwixt.strategy.PluralStemmer

{

public org.apache.commons.betwixt.ElementDescriptor

findPluralDescriptor(String propertyName, Map map)

{

if( propertyName.equals( "buch"))

{

org.apache.commons.betwixt.ElementDescriptor answer

=

(org.apache.commons.betwixt.ElementDescriptor)

map.get( "buecher");

if( answer != null)

{

return answer;

}

}

return null;

}

}

Add a note hereAdd a note hereThe class GermanPluralMapper in [Listing 5.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) implements the interface PluralStemmer. The interface PluralStemmer requires that only the method findPluralDescriptor be implemented. In the implementation of findPluralDescriptor, the objective is to cross-reference the property name with a list of properties available. To find paired collection properties, *betwixt* first retrieves all of the methods that start with add. These methods have the add removed from the front of the identifier. Then, from those found methods, the interface method findPluralDescriptor is called to find the pairs of methods. Going back to [Listing 5.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the parameter propertyName will contain the value buch. The parameter map will contain a list of possible get methods, which in the case of [Listing 5.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), will be getBuecher. However, the names of the identifiers contained within the parameter map have the get trimmed, so that only buecher is left over. Within the implementation of findPluralDescription, we need to cross-reference the identifier buch with some identifier in the map. And, if a match is found, the associated instance of the class ElementDescriptor is returned. If a match is not found, a null value is returned.

Add a note hereAdd a note hereIf in [Listing 5.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) there were no addBuch method and only a getBuecher method, then the PluralStemmer interface instance would not be called. As a result, you might think that the collection will not be serialized; however, the collection is serialized because the default rule of serializing an Object of type Iterator is applied.

### Add a note hereAdd a note hereUsing *betwixt* Configuration Files

Add a note hereAdd a note hereAll of the techniques discussed thus far do not allow you to fine-tune how the data is serialized. The techniques outlined show you how to tune the overall process, which is useful as well. The point of fine-tuning is that it allows a developer to read and write data exactly to how the XML should be structured. For example, when we renamed the XML tags, all XML tags were renamed. It would have been difficult to rename an XML type based on a specific context, since the contexts used were simple. For example, in the renaming example, the context was the name of the property, which could be identical for two entirely different Java Bean classes.

Add a note hereAdd a note hereThe fine-tuning process works at the class-type level and allows a developer to specify how everything will be serialized. In the *betwixt* package, that means using *betwixt* configuration files. In theory, to use betwixt, it is not necessary to write code like that shown in all of the previous examples. The *betwixt* configuration files could do everything that is necessary. The only code that you would need to write is the code to consume the *betwixt* package. Following are general XML rules on how to generate data; for example, when to use XML tags and when to use XML attributes.

**General XML Object Structure Rule**

Add a note hereAdd a note hereWhen an XML file is created, you place XML elements in a certain structure. Typically, an XML element and its child elements are primary entities, and the XML attributes describe properties. For example, a car is a primary entity and so are the wheels, engine, and seats. However, the serial number of the model could be a property that describes the car. A property would typically be a property that provides a mechanism to quickly distinguish one XML element from another XML element, as shown by the following example.

Add a note hereAdd a note here<car serial="1234d32"><wheels>2</wheels></car>

Add a note hereAdd a note hereSome people may argue that the serial number is a primary entity and should be an XML element. The response to this would be that if the serial number were attached to an invoice, then it would be a primary entity. An invoice does not care if the car has four, five, or ten wheels. The invoice cares only if the car can be uniquely identified from another car. The simplest way to do that is to use the serial number. In the case of an invoice, the number of wheels might be a descriptor, as shown by the following example.

Add a note hereAdd a note here<invoice><item>  
 <car wheels="2"><serial>1234d32</serial></car>  
</item></invoice>

Add a note hereAdd a note hereHowever, this rule is not a heavy-handed rule that you must follow every time. Instead, apply this rule as necessary. Be sure, though, to avoid XML structures that are either all XML elements or XML attributes.

Add a note hereAdd a note hereThe *betwixt* configuration file allows you to alter the metadata stored about the object. Remember back to [Listing 5.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), where the class method BeanWriter.write was called. What happened in that step was that the bean was being introspected. The different fields, properties, and methods were inspected for potential serialization. However, doing introspection every time a bean is serialized would result in a massive performance hit. And that would not be very useful.

Add a note hereAdd a note hereSo that there is no major performance hit, the *betwixt* package uses the class XMLIntrospector to cache *betwixt* configuration files. We have used this class in various examples to control how XML content is serialized. The *betwixt* configuration files are cached in the XML-Introspector class, so it is an extremely important reference class that should be stored someplace as a singleton. For example, we could use the *lang* factory package singleton.

Add a note hereAdd a note hereIn the serialization process when a Java class is introspected, the class XMLIntrospector manages a number of settings that control XML serialization. Important is something called the *XML bean registry*. Every class that is serialized has its serialization process stored in the XML bean registry. This is the cache used so that an individual class does not have to be introspected twice. The class XMLBeanInfoRegistry represents the XML bean registry. The class XMLBeanInfoRegistry is considered a cache, and you can influence that cache by setting values using method calls or by creating a betwixt file.

Add a note hereAdd a note here[Listing 5.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is the basic Java class definition used to define a *betwixt* configuration file.

Add a note hereAdd a note here**Listing 5.35**

Add a note hereAdd a note herepublic class BetwixtBean {

private int \_iValue;

private String \_strValue;

public BetwixtBean( int ival, String sval) {

\_iValue = ival;

\_strValue = sval;

}

public int getIntegerValue() {

return \_iValue;

}

public void setIntegerValue( int val) {

\_iValue = val;

}

public String getStringValue() {

return \_strValue;

}

public void setStringValue( String val) {

\_strValue = val;

}

}

Add a note hereAdd a note here[Listing 5.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) shows the associated *betwixt* configuration file, which is called *BetwixtBean.betwixt*.

Add a note hereAdd a note here**Listing 5.36**

Add a note hereAdd a note here<info primitiveTypes="element">

<element name='better-name'>

<addDefaults/>

</element>

</info>

Add a note hereAdd a note hereIf [Listing 5.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) were executed to serialize the class BetwixtBean defined in [Listing 5.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the result would appear identical to [Listing 5.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.37**

Add a note hereAdd a note here <better-name>

<integerValue>1234</integerValue>

<stringValue>hello world</stringValue>

</better-name>

Add a note hereAdd a note hereIt is worth taking a few moments to consider [Listings 5.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), 5.36, and 5.37 because there are several points you should notice:

* Add a note hereAdd a note hereThe name of the betwixt file is identical to the name of class and resides in the same location as the class file. Therefore, if the class filename is *Image from book*[*hello.java*](http://cdcontent.books24x7.com/id_7265/hello.java), then the betwixt configuration filename is *hello.betwixt*.
* Add a note hereAdd a note hereThe betwixt file is XML based and determines the serialization characteristics of a single class. Each class has its own betwixt file.
* Add a note hereAdd a note hereThe XML element info is the root element of all betwixt files
* Add a note hereAdd a note hereThe XML child element element represents the outermost serialization XML element. In other words, in this serialization, the XML element element will associate itself with the class identifier BetwixtBean. If there are multiple XML elements element, as there are in [Listing 5.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), then the last one will be used for serialization.
* Add a note hereAdd a note hereYou can control attributes in the XMLIntrospector (like the XML attribute primitiveTypes and XML element addDefaults) by adding them in the betwixt file. The purpose of the XML attribute primitiveTypes is to define how the properties of the class BetwixtBean will be generated. In [Listing 5.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the properties are generated as XML elements. To generate properties as attributes, the XML attribute value element is replaced with attribute. The XML element addDefaults indicates that the XMLIntrospector should be populated with default values.
* Add a note hereAdd a note hereAll applications should define betwixt files because they are the only safe way of writing or reading the XML content properly.

#### **Adding Static Content**

Add a note hereAdd a note hereIn the XML serialization process, it might be useful to add static data. The static data could be version information. Alternatively, it could be data related to something that another process wants, or the static data could serve administrative purposes. For example, an application could be written that generates old data. To modernize the application, we could generate some static data that acts as a placeholder. This solution would be cheaper than having to rewrite the application to add the extra data. [Listing 5.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) shows a sample *betwixt* configuration that adds static content.

Add a note hereAdd a note here**Listing 5.38**

Add a note hereAdd a note here<info primitiveTypes="element">

<element name='documentation'>

<attribute name='version' value='1.0'/>

<element name='author'>

<element name="location" value="Zurich">

<attribute name='version' value='1.0'/>

</element>

<attribute name='version' value='1.0'/>

</element>

<addDefaults/>

</element>

</info>

Add a note hereAdd a note hereIn [Listing 5.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the XML elements attribute and all child XML elements element from the first XML element that has the attribute value documentation are static elements. The XML content that is generated is shown in [Listing 5.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.39**

Add a note hereAdd a note here<documentation version="1.0">

<author version="1.0">

<location version="1.0">Zurich</location>

</author>

</documentation>

Add a note hereAdd a note hereIf you compare [Listings 5.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) and 5.39, it would seem that all XML elements other than the first XML element element are static content elements. This is true in the simplest of situations, but only because a specific attribute is missing. When you define various XML elements element and attributes with associated value attributes, you create static content. For example, for the first XML element attribute, there are two XML attributes: name and value. These two attributes define an XML attribute that is created in the generated content.

Add a note hereAdd a note hereThe generated XML attribute is associated with the parent XML element in the *betwixt* configuration file. Practically speaking, this means that when the *betwixt* configuration file in [Listing 5.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is processed, the generated XML element documentation has an associated XML attribute version with a value of 1.0. When an XML element element is a child element, then the XML attributes name and value in the *betwixt* configuration file define an XML tag and value in the generated XML file. For example, in the *betwixt* configuration file in [Listing 5.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the XML element element with the XML attributes name and value will generate the XML element location, with a contained value of Zurich. We saw the result of this in [Listing 5.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

#### **Mapping a Property to a Different Element**

Add a note hereAdd a note hereWhen an object is serialized, the individual properties are iterated and serialized. It is possible to generate additional tags or remap the identifiers to something else. Remember back to [Listings 5.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), 5.23, and 5.24, where the problem was how to generate Java Beans that reference other Java Beans. We can easily solve this problem by defining a mapping within a betwixt file, as shown in [Listing 5.40](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.40**

Add a note hereAdd a note here<info primitiveTypes="element">

<element name='documentation'>

<element name="embedded">

<element name="ex-bean-to-write"

property="exBeanToWrite"/>

</element>

<addDefaults/>

</element>

</info>

Add a note hereAdd a note hereIn [Listing 5.40](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the XML element element with the attribute embedded is a static declaration. However, the child XML element element with the attribute ex-bean-to-write is mapped to the property exBeanToWrite. What happens in the serialization process is that an XML element with an identifier embedded will be generated, and embedded within that tag will be the XML element ex-bean-to-write, which represents the class property exBeanToWrite. The output will resemble [Listing 5.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.41**

Add a note hereAdd a note here <documentation>

<embedded>

<ex-bean-to-write>

<integerValue>1234</integerValue>

<stringValue>another bean</stringValue>

</ex-bean-to-write>

</embedded>

</documentation>

Add a note hereAdd a note here[Listing 5.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) shows how easily and neatly you can solve the problem of having one Java Bean reference and Java Bean.

#### **Mapping a Property to an Attribute**

Add a note hereAdd a note hereWhen you serialized the individual properties, the serialization occurred with all Java properties either being XML child elements or XML attributes. There was no happy middle where some properties were serialized as XML elements and others as XML attributes. Using a betwixt file, you can serialize the Java property as either an XML element or XML attribute. For example, let's say you have a Java property that is an integer and needs to be serialized as an attribute. The sample betwixt file is shown in [Listing 5.42](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.42**

Add a note hereAdd a note here<info primitiveTypes="element">

<element name='documentation'>

<attribute name="version" property="version" />

<addDefaults/>

</element>

</info>

Add a note hereAdd a note hereWhen this is executed, you get the results shown in [Listing 5.43](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.43**

Add a note hereAdd a note here<documentation version="1">

<!— Some other XML elements —>

</documentation>

Add a note hereAdd a note hereYou define a mapping of a property by using the XML element attribute and then adding an attribute property, as shown in [Listing 5.42](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445). This notation is very similar to that used with a static content declaration, except that instead of having an XML attribute value, there is an XML attribute property. You place the attribute according to the same rule as with static content.

#### **Mapping a Property to Text**

Add a note hereAdd a note hereIn the final example, you can map a property to an XML text. [Listing 5.44](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) shows how to do it.

Add a note hereAdd a note here**Listing 5.44**

Add a note hereAdd a note here<info primitiveTypes="element">

<element name="documentation">

<text property="version" />

</element>

</info>

Add a note hereAdd a note hereThe XML element text in [Listing 5.44](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) associates the property version with an output, which would generate the output shown in [Listing 5.45](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.45**

Add a note hereAdd a note here<documentation>1</documentation>

#### **Hiding a Property**

Add a note hereAdd a note hereIn all of the examples illustrated thus far, the introspection process has added all properties to the list of being serialized. In the betwixt file, you can indicate that a property should not be serialized. The way to do this is to use the XML element hide, as shown in [Listing 5.46](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445).

Add a note hereAdd a note here**Listing 5.46**

Add a note hereAdd a note here<info primitiveTypes="element">

<element name="documentation">

<hide property="version" />

</element>

</info>

Add a note hereAdd a note hereIn [Listing 5.46](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the property to be hidden is version.

### Add a note hereAdd a note hereReading an XML File

Add a note hereAdd a note hereIt is desirable to be able to read every bean that is written. Reading the bean is as simple as writing it. Things can get complicated, however, if you don't serialize the bean using a betwixt file. The betwixt file is a central part of the *betwixt* package and hence has been supported and tested extensively. [Listing 5.47](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445) is an example of where the class BetwixtBean is written to a string buffer, which is then read again using the class BeanReader.

Add a note hereAdd a note here**Listing 5.47**

Add a note hereAdd a note hereStringWriter stringWriter = new StringWriter();

BetwixtBean somebean = new BetwixtBean( 1234, "hello world");

BeanWriter writer = new BeanWriter( stringWriter);

writer.enablePrettyPrint();

writer.write( somebean );

stringWriter.flush();

String xml = "<?xml version='1.0'?>\n" +

stringWriter.toString();

BeanReader reader = new BeanReader();

reader.registerBeanClass( BetwixtBean.class );

BetwixtBean readBean =

(BetwixtBean)reader.parse(new StringReader(xml));

Add a note hereAdd a note hereIn [Listing 5.47](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445), the class StringWriter is a string buffer that will be written to. We could have used the class FileWriter or any other type of writer in its place. The class instance of StringWriter is passed to the class BeanWriter as a constructor parameter. The association created by the parameter passed in the constructor means that any serialization will automatically be saved to the writer. Combining an XML prolog with the contents of the string buffer referenced by variable stringWriter creates a fully complete XML buffer.

Add a note hereAdd a note hereOnce a complete XML buffer is available, the buffer can be parsed using the class BeanReader. The class BeanReader is the opposite of the class BeanWriter, but there are a couple of differences. First, for the class BeanReader to function correctly, the class BeanReader needs to know which classes can be instantiated. This is the purpose of the method registerBeanClass, which accepts as a parameter the class information of the classes that can be instantiated.

Add a note hereAdd a note hereIf the class information references other classes, then those other classes will be added to the available pools of classes to be instantiated. It is absolutely important, though, that all classes that can be instantiated from the pool can be instantiated without a constructor parameter(s). The classes that were serialized thus far did not have an empty constructor and therefore would generate an instantiation exception.

Add a note hereAdd a note hereTo reconstruct the class instance BetwixtBean, the method parse is called in [Listing 5.47](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=745604445). The method parse accepts as a parameter a string buffer, or in this case the class StringReader.

## **Organizing XML Data Using *Lucene***

Add a note hereAdd a note hereThe Jakarta *Lucene* project is text search engine that can do a full-text search of various types of documents. Even though Jakarta *Lucene* is not part of the Commons project, this project deserves notice since it is very unique and interesting. Within the Apache XML group is another project called *Xindice*, which is an XML database. That project will not be covered in this book. Had this book had been a pure XML development book, we would be discussing XML *Xindice* instead ofJakarta *Lucene*. The purpose of this book, however, is to show how you can easily create a document-processing framework using Java programming techniques.

Add a note hereAdd a note hereThe Jakarta *Lucene* text-search engine is unique because it combines text search with SQL abilities, which can uniquely define search attributes. This is an interesting combination because part of the strength of SQL is its ability to find specific pieces of data. Search engines are good at indexing any type of information but have traditionally had problems finding the "needle in the haystack." In the *betwixt* section, we converted Java classes into XML files. The XML files are used as the basis of the index in the Jakarta *Lucene* program.

### Add a note hereAdd a note hereTechnical Details for the Jakarta *Lucene* Program

Add a note hereAdd a note here[Tables 5.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275) and [5.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275) contain the abbreviated details necessary to use the *Lucene* program.

**Add a note hereAdd a note hereTable 5.5:** Repository details for the Jakarta *Lucene* program.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note herejakarta-lucene |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note hereNot applicable |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.lucene.index.\*, org.apache.lucene.analysis.\*, org.apache.lucene.document.\*, org.apache.lucene.search.\*, org.apache.lucene.queryParser.\* |

**Add a note hereAdd a note hereTable 5.6:** Package and class details (legend: [lucene] = org.apache.lucene).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[lucene].index.IndexWriter | Add a note hereAdd a note hereA class used to manage, write, and update an index. This class is not thread safe and therefore must have synchronized access to it. |
| Add a note hereAdd a note here[lucene].analysis.Analyzer | Add a note hereAdd a note hereA base interface used to define a word analyzer and tokenizer, which reads text and figures out the pieces of the text. |
| Add a note hereAdd a note here[lucene].analysis.standard.StandardAnalyzer | Add a note hereAdd a note hereA standard implementation of the interface Analyzer. It allows the case to be insensitive and it removes generic word searches. |
| Add a note hereAdd a note here[lucene].document.Document | Add a note hereAdd a note hereA class used to store the various fields used for indexing data that is referenced as a document. |
| Add a note hereAdd a note here[lucene].document.Field | Add a note hereAdd a note hereA class used to store an individual key value pair that represents a field that is part of a document. |
| Add a note hereAdd a note here[lucene].search.Query | Add a note hereAdd a note hereA class used to store a query that will be applied against an index. The query is independent from an index and hence could be stored for future reference. |
| Add a note hereAdd a note here[lucene].search.Searcher | Add a note hereAdd a note hereAn abstract class used to describe a generic searchable class instance. |
| Add a note hereAdd a note here[lucene].search.Hits | Add a note hereAdd a note hereA class that contains the results from running a query on a Searcher class instance. |
| Add a note hereAdd a note here[lucene].queryParser.QueryParser | Add a note hereAdd a note hereA factory class used to instantiate a Query class instance. |

#### **Details on Building the Jakarta *Lucene* Program**

Add a note hereAdd a note hereBuilding the Jakarta *Lucene* program from the sources is not that complicated. The only complication is that you need to download the *Image from book*[*JavaCC.zip*](http://cdcontent.books24x7.com/id_7265/javacc.zip.zip) package, a Java compiler. Then, you need to define the correct configuration in the *Image from book*[*default.properties*](http://cdcontent.books24x7.com/id_7265/default.properties) file. Specifically, in the *Image from book*[*default.properties*](http://cdcontent.books24x7.com/id_7265/default.properties) file, you need to set the settings in [Listing 5.48](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275).

Add a note hereAdd a note here**Listing 5.48**

Add a note hereAdd a note here# Home directory of JavaCC

javacc.home = /home/cgross/jars

javacc.zip.dir = ${javacc.home}/lib

javacc.zip = ${javacc.zip.dir}/JavaCC.zip

Add a note hereAdd a note hereIn [Listing 5.48](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), the javacc.home directory is set to the */home/cgross/jars* directory. However, when the *JavaCC* package is executed, the *Image from book*[*JavaCC.zip*](http://cdcontent.books24x7.com/id_7265/javacc.zip.zip) file is searched for in the directory */home/cgross/jars/lib*. For those of you who are Windows programmers, the *Image from book*[*JavaCC.zip*](http://cdcontent.books24x7.com/id_7265/javacc.zip.zip) file is not the same as the downloaded *Image from book*[*JavaCC.zip*](http://cdcontent.books24x7.com/id_7265/javacc.zip.zip) file that contains the *Image from book*[*JavaCC.jar*](http://cdcontent.books24x7.com/id_7265/javacc.jar.zip) file and appropriate Java Doc Files (documentation files). You will need the *Image from book*[*JavaCC.zip*](http://cdcontent.books24x7.com/id_7265/javacc.zip.zip) file, which you can create by unpacking the *Image from book*[*JavaCC.jar*](http://cdcontent.books24x7.com/id_7265/javacc.jar.zip) file and creating a *Image from book*[*JavaCC.zip*](http://cdcontent.books24x7.com/id_7265/javacc.zip.zip) file.

Add a note hereAdd a note hereOnce a compile has been successfully executed using Ant, you will have two jar files: *lucene-\*.jar* and *lucene-demos-\*.jar*. You need to add these two files to the class path. However, add these files to the root class path and not to a configuration file. Doing so will make testing and development much simpler.

### Add a note hereAdd a note hereIndexing a Document

Add a note hereAdd a note hereWhen you use the *Lucene* database system, data is indexed and a user can query for it, much like with a search engine. In a traditional relational database, when a piece of data is added to a table, most likely indexing will occur automatically. The data and its associated index is stored transparently from the end user. In the *Lucene* database, it is not transparent and requires some careful consideration because of the nature of the data managed by *Lucene*. The nature of the data is diverse, and therefore each document has its own characteristics on how it should be indexed and queried for.

Add a note hereAdd a note hereWhen you index a document using *Lucene*, you need to define a document and consider what is important in the document. The important parts of the document will be the parts that are indexed. To build some continuity from the *betwixt* package section, we will use the sample serialized bean XML files as the basis for building the documents. We will start with the simplest bean. We will use the class BeanToWrite, shown in [Listing 5.13](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=322#322), as a basis for indexing. [Listing 5.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275) is an example of how the class BeanToWrite can be indexed.

Add a note hereAdd a note here**Listing 5.49**

Add a note hereAdd a note herepublic void indexBeanToWrite( boolean isNewIndex,

String location, BeanToWrite bean) {

try {

saveBean( location, bean);

Analyzer analyzer = new StandardAnalyzer();

IndexWriter writer =

new IndexWriter( \_indexDir, analyzer, isNewIndex);

Document document = new Document();

document.add( Field.UnIndexed( "path", location));

document.add( Field.UnIndexed( "class",

bean.getClass().getName()));

document.add( Field.Text( "integerValue",

String.valueOf( bean.getIntegerValue())));

writer.addDocument(document);

writer.close();

}

catch( Exception ex) {

System.out.println( "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println( ex.getMessage());

ex.printStackTrace();

}

}

Add a note hereAdd a note hereIn [Listing 5.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), the method indexBeanToWrite has three parameters. The first parameter isNewIndex is used to either create a new index or add to an already existing one. The second parameter, location, is used to define the location where the serialized bean will be saved. The third and last parameter, bean, is the bean that will serialized and indexed.

Add a note hereAdd a note hereAfter the method indexBeanToWrite has been called, the method saveBean is called, which serializes the Java Bean to a file using the *betwixt* serialization routines similar to the examples that we have seen in the previous sections of this chapter. As far as the *Lucene* database, however, you don't need to take this step of serializing to a file because the *Lucene* database does not tie in to the *betwixt* package. The data could have been generated using an XML editing tool, and the *Lucene* database would not have cared either way.

Add a note hereAdd a note hereWhen a piece of information is indexed in *Lucene*, there are three parts to make an index work. The first part is the definition of an analyzer, which in [Listing 5.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275) is the class StandardAnalyzer. The analyzer is used to decipher the text that is to be indexed. For example, the standard analyzer will tokenize words, remove generic words, and convert them to lowercase to enable case-insensitive searches. The purpose of the tokenizer is to split the words apart into single words that can be indexed. Removing the generic words (more commonly called *stop words*) involves getting rid of words like "a," "this," and "or" because these words tend to return a large number of unapplicable results when you do a search. Some analyzers are available in the *org.apache.lucene.analysis* package.

Add a note hereAdd a note hereThe second part of indexing a document is to create a writer, which in [Listing 5.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275) is the class IndexWriter. The class IndexWriter is responsible for writing and managing the index.

Add a note hereAdd a note hereThe last part of indexing a document is to create a document, which in [Listing 5.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275) is the class Document. The purpose of the class Document is to define a document that is to be indexed. You define a document by creating a number of key value pairs that define a field that is added to the document using the method add. In [Listing 5.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), there are three different fields for the index. We will provide the exact specifics of the fields in a later section (["Defining the Different Field Types"](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275)).

Add a note hereAdd a note hereOnce all three parts have been defined, the class method writer.addDocument is called to add the document to the index. To write the index, the class method writer.close is called, and the index is organized and written to the hard disk.

### Add a note hereAdd a note hereQuerying for a Document

Add a note hereAdd a note hereOnce an index has been created and written, it can be queried. Querying an index is shown in [Listing 5.50](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275).

Add a note hereAdd a note here**Listing 5.50**

Add a note hereAdd a note hereSearcher searcher = new IndexSearcher( \_indexDir);

Query query = QueryParser.parse( String.valueOf( value),

"integerValue", new StandardAnalyzer());

Hits hits = searcher.search(query);

Vector vector = new Vector();

for (int i=0; i<hits.length(); i++) {

System.out.println( "Class identifier " +

hits.doc(i).get("class"));

vector.add( readBean( hits.doc(i).get("path"),

BeanToWrite.class));

};

Add a note hereAdd a note hereIn [Listing 5.50](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), querying an index is simpler than creating or adding to an index and requires only two parts. The first part, performed by the class IndexSearcher, is used to query an index. In [Listing 5.50](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), the IndexSearcher instantiation has a constructor, which is the directory where the index is located. The *Lucene* database can work either with directories or in memory indices. However, to keep it simple, we will use the directory-based index.

Add a note hereAdd a note hereThe second part, which is defined by the class Query, is the creation of the query. To create a class instance of the class Query, the class method QueryParser.parse is called. The class method requires two pieces of information: the query and the analyzer used to create the query. It is very important that you use the same analyzer in the query as you did to create the index. Using different analyzers will result in interpretation errors. For example, consider if one analyzer is case sensitive and another is not.

Add a note hereAdd a note hereOnce the two parts have been created, they are combined using the method searcher.search, which returns a class instance of Hits. The class Hits represents the results of the query. The class Hits contains a result set of class type Documents. In a relational environment, this would be a standard result set; however, in the *Lucene* database, there is an associated score. The scoring of a document result indicates how closely a match has been made. In [Listing 5.50](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), the result set of documents is then used to read a number of beans written in the indexing step.

### Add a note hereAdd a note hereDefining a Database Strategy

Add a note hereAdd a note hereWe on purpose "forgot" to mention why things were done the way they were in [Listings 5.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275) and 5.50. The reason is that for you to be able to fully comprehend how to save and index XML documents, or to understand the *Lucene* database strategy, the mechanics of *Lucene* need explanations. All of the following sections attempt to explain how to interact with the *Lucene* database and what is possible and not possible.

Add a note hereAdd a note hereIn [Listing 5.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), three fields were saved to the *Lucene* database. The first field was defined by the code Field.Unindexed( "path", location). In this case, the purpose of the field is to define the location of the serialized XML file. The XML file could be stored within the field, or it could be stored within a file or even within a SQL table. The idea is to define a unique reference where the content of the serialized bean can be retrieved. In Internet terms, consider it as a Uniform Resource Identifier (URI) or URL.

Add a note hereAdd a note hereThe second field was defined by the code Field.UnIndexed( "class", bean.getClass().getName()). In this case, the purpose of the field is to define the class name. Knowing the class name, the calling platform could dynamically load the class and then register the class descriptor with the betwixt platform.

Add a note hereAdd a note hereThe third and last field was defined by the code Field.Text( "integerValue", String.valueOf( bean.getIntegerValue()). The purpose of this field is to provide a field that is indexed and can be searched on.

Add a note hereAdd a note hereThe three fields each define a unique purpose that should be reflected in any business application. The first and second fields will have values specific to the technology used and will typically not be indexed. The last field changes from application to application. That type of field might not just be a single field but multiple fields.

Add a note hereAdd a note hereWhen you create an index, like in [Listing 5.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), you use a specific directory. In a relational database, the management of the indices is transparent to the end programmer. The *Lucene* database is partially transparent with its indices. While there is no requirement to individually manage the index, not all indices should be stored in one directory. Each individual index should reflect a business task. However, knowing how to split the various indices is not an exact science and should be experimented with.

### Add a note hereAdd a note hereDefining the Different Field Types

Add a note hereAdd a note hereWhen you define a field, like in [Listing 5.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), there are different field references. In [Listing 5.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), two fields were added using the method Field.Unindexed, and the other was added using the method Field.Text. Each type of method (Unindexed and Text) creates a field with specific characteristics. In the case of the method Text, the field is added to the index and the original text is stored in the database. This means that when a document that references the field defined is retrieved, you can retrieve the original content.

Add a note hereAdd a note hereAs an example, consider the text "my document is here". *Lucene* will split up and index the text. The words my, document, and here will be added to the index. When the words are added to the index, the original block is not added as a field that can be retrieved. If you want to add the text as a block, you must explicitly store it as a block of text.

Add a note hereAdd a note hereThe following other field types can be manipulated by the *Lucene* database:

* Add a note hereAdd a note here**Field.Keyword:** This a special field because the text block is not tokenized. This field is very useful for dealing with complex strings like dates, serial numbers, or titles. The purpose of this field is to index content, where the content is unique as a whole and not tokenized. For example, if the field value were "hello dolly", then the field would not be split into two tokens, but would be kept as one token. If the token were split into two, it would be easier to search for individual tokens. Using this type of field, the entire content of the field can be retrieved and is identical in storage terms to a SQL database column.
* Add a note hereAdd a note here**Field.UnIndexed:** This type of field is used to store blocks as text, but these blocks are not indexed. The purpose of these fields is to provide a quick reference to the actual data, so that you don't have to reference it elsewhere. In the case of [Listing 5.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), this field type is used to store reference information.
* Add a note hereAdd a note here**Field.Unstored:** This type of field is used to add information to the index, but the information is not stored for extraction.
* Add a note hereAdd a note here**Field.Text:** This type of field, which is used in many cases, is stored and tokenized in the database. This method has an overloaded version, where the data is not stored in a string but in a class instance of Reader. Using the class Reader, you can add large text fields to an index. The only catch is that the contents held by the class Reader is not stored and acts more like a Field.Unstored type.

Add a note hereAdd a note hereEach of the methods defined returns a class instance of Field. In addition, you can tweak the class Field property boost. You can use this property to modify queries and search results. A boosted field increases its priority in the result score that is generated for every query.

### Add a note hereAdd a note hereQuerying Using a Query String

Add a note hereAdd a note hereIn a relational database, you create a query by using the SQL programming language. With *Lucene*, instead of SQL, you use a querying language that's not related to SQL. It is more related to an Internet search engine that can search in specific fields. [Listing 5.51](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275) shows how to query using a query string, which is very similar to querying using a specific field.

Add a note hereAdd a note here**Listing 5.51**

Add a note hereAdd a note herepublic Vector queryStringBeanToWrite( String strQuery) {

try {

Searcher searcher = new IndexSearcher( \_indexDir);

QueryParser queryParser = new QueryParser(

"integerValue", new StandardAnalyzer());

queryParser.setOperator(

QueryParser.DEFAULT\_OPERATOR\_OR);

Query query = queryParser.parse(strQuery);

Hits hits = searcher.search(query);

Vector vector = new Vector();

for(int i=0; i<hits.length(); i++) {

System.out.println( "Class identifier " +

hits.doc(i).get("class"));

vector.add( readBean( hits.doc(i).get("path"),

BeanToWrite.class));

};

return vector;

} catch( Exception ex) {

System.out.println( "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println( ex.getMessage());

ex.printStackTrace();

}

return null;

}

Add a note hereAdd a note hereIn [Listing 5.51](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), instead of using the static class method QueryParser.parse, we instantiated the class QueryParser and then called the method parse. We did this because the analyzer needs to be associated with an instance of QueryParser, and the association is made when the class QueryParser is instantiated. The class Query is instantiated using the method queryParser.query, where the variable strQuery contains a query.

Add a note hereAdd a note hereAn initial query that could be used to call the method queryStringBeanToWrite is shown in [Listing 5.52](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275).

Add a note hereAdd a note here**Listing 5.52**

Add a note hereAdd a note here integerValue:1234 OR integerValue:2345

Add a note hereAdd a note hereThe query in [Listing 5.52](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275) finds the documents where the field integerValue has the value of 1234 or where the field integerValue has the value of 2345. In *Lucene*, a colon separates the fieldname and associated value. Another way to write the query in [Listing 5.52](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275) would be [Listing 5.53](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275).

Add a note hereAdd a note here**Listing 5.53**

Add a note hereAdd a note here integerValue:1234 integerValue:2345

Add a note hereAdd a note hereIn [Listing 5.53](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), an OR keyword is missing. This is OK because in [Listing 5.51](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), the method call queryParser.setOperator sets the default operator to be an OR to join multiple statements.

Add a note hereAdd a note hereYet another way to write query in [Listing 5.52](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275) is [Listing 5.54](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275).

Add a note hereAdd a note here**Listing 5.54**

Add a note hereAdd a note here 1234 2345

Add a note hereAdd a note hereIn [Listing 5.54](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275), there is no OR keyword, nor is there any field descriptor. The query still executes and returns the same results as [Listing 5.52](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275) because we set the default field in [Listing 5.51](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275). When the class QueryParser was instantiated, the field integerValue was given as the default field identifier.

Add a note hereAdd a note hereThere is even one more way to write [Listing 5.52](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275). This is shown in [Listing 5.55](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275).

Add a note hereAdd a note here**Listing 5.55**

Add a note hereAdd a note here integerValue:(1234 2345)

Add a note hereAdd a note here[Listing 5.55](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=771904275) uses brackets to group the items into a list that asks for the field integerValue to find the values of 1234 or 2345. The default operator between the two numbers is the OR operator, which could have been explicitly written out.

Add a note hereAdd a note hereHaving seen all these variations of the same query, we now ask whether or not it is advisable to rely on default field identifiers and operators. The answer is mostly no; reliance on default identifiers or operators may not be a good idea. It is better to uniquely describe each field and identifier. This leads to less confusion when somebody is trying to maintain already existing code or fix bugs.

Add a note hereAdd a note hereUp to this point, the example of queries used the OR keyword. Here is a list of all operators you can use when writing queries:

* Add a note hereAdd a note here**OR**: In a result set, this combines the result sets from multiple queries. It is as if the queries separated by the OR are combined to build one result set.
* Add a note hereAdd a note here**AND**: In a result set, this ensures that every item of the result set contains the requested data. Writing integerValue:1234 AND integerValue:2345 would be futile because the field integerValue cannot be both numbers at the same time. The AND operator is typically used in the context of a multiple field query.
* Add a note hereAdd a note here**NOT**: In result set, this ensures that the result set does not contain a specific value. For example, the query integerValue:1234 NOT stringValue:"hello world" says to select all the documents where the field integerValue has a value of 1234 but where the field stringValue cannot have a value of "hello world".
* Add a note hereAdd a note here**+:** In a result set, this ensures that every field has s specific value. For example, the query **+**integerValue:1234 OR stringValue:"hello world" says to select all of the documents that whose field integerValue has a value of 1234 and whose field stringValue may have a value of >"hello world".
* Add a note hereAdd a note here**-**: This is like the + operator except that instead of requiring each document to contain the value, this operator says that the result set should not contain the value.
* Add a note hereAdd a note here**[ ]**: This defines a group of values that has a range. For example, the query integerValue:[1000 3000] says that all documents with an integerValue in the range of 1,000 to 3,000 should be included in the result set.
* Add a note hereAdd a note here**~**: This defines a proximity word search. For example, the query stringValue: Hello~10 says that all documents with a stringValue should be within 10 words of the word Hello.
* Add a note hereAdd a note here**^**: This defines a boost to a specific term, resulting in a higher ordering in the result set. An example could be integerValue:1234^4.

Add a note hereAdd a note hereThe following operators can be performed on the words themselves:

* Add a note hereAdd a note here**Hello~**: Performs a fuzzy search for a specific word
* Add a note hereAdd a note here**Hel?o**: Performs a search where the question mark represents a single wildcard character
* Add a note hereAdd a note here**He\*** or **He\*o**: Performs a search where the asterisk represents a multiple wildcard character

## **Summary**

Add a note hereAdd a note hereThis chapter introduced the concept of serialization. Serialization is one of the most problematic pieces of the application because it is the part responsible for remembering what has transpired. Ideally, you could remember everything that happened using a very big RAM disk with programs running constantly. However, that is simply not feasible from a financial standpoint.

Add a note hereAdd a note hereUntil someone invents RAM disks, serialization will be an issue. The chapter started with a discussion on how to represent a relational database as a Java Bean-type object. The *SQL* package is a good package that does most of that work for the programmer. The advantage of this package is that it tries only to be a thin layer on top of SQL. A developer could very easily extend SQL to include special tasks that would make SQL simpler and more manageable.

Add a note hereAdd a note hereThen, the discussion switched to the *betwixt* package. We didn't discuss all features of the *betwixt* package because some have a lesser role. However, we did discuss the package's major aspects. The explanations and examples we gave there highlighted the *betwixt* package and made it easier to figure out what it does. The most important fact you should remember from that section is that a betwixt file gives developers the most flexibility when they serialize their Java classes to XML and back again.

Add a note hereAdd a note hereThe [last section](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=372#372) of the chapter covered the basics of the *Lucene* database, which highlights how XML data can be stored and indexed. As a result of the Internet era, we can do business processes differently, and *Lucene* is an example of this. Internet search engines have proven themselves to be scalable and very robust. *Lucene* extends the Internet search engine platform by adding precision of queries. This results in a superior way of searching for any kind of information.

Add a note hereAdd a note hereYou can now pat yourself on the back because you've gotten halfway through this book. We have already discussed basics such as the Commons Bridge, factories, scalability, and serialization. Knowing these basics, any programmer can write any type of algorithm safely. The basics are where most of the problems lie when you write programs because often some programmers get off on the wrong foot, falter, and instead of rewriting, keep going. This causes problems and should be avoided. In this book, we recommend that if you use what the Jakarta programmers use, your programming life will become simpler.

Add a note hereAdd a note hereThe rest of the book involves a layer of specific problems that we solve. Some of the problems could be configuration or logging. These problems are basic routine problems that all developers need to solve, so we look at how the Jakarta programmers solve them. Good luck in the second half.

Add a note hereAdd a note here**On the CD** The sources to the concepts presented in this chapter are located under the directory *[CDROM]/jseng/src/com.devspace.jseng.serializtion*. For the section *Using betwixt Con- figuration Files* the sample configuration files are located at *[CDROM]/jseng/src/classes/ com/devspace/jseng/serialization/BetwixtBean.betwixt,* and *[CDROM]/jseng/src/classes/ com/ devspace/jseng/serialization/MotherOfAllBeans.betwixt*.

## **Questions**

Add a note hereAdd a note here**5.1:** Outline the advantages of a SQL database and what types of applications are best suited to it.

Add a note hereAdd a note here**5.2:** Outline the advantages of an XML database and what types of applications are best suited to it.

Add a note hereAdd a note here**5.3:** Add a history to the calculator component that saves its data to a SQL database. The type of database is up to you.

Add a note hereAdd a note here**5.4:** Rewrite the history component to be a generic serialization framework that will save its history to either an XML file or to a database. It is important that the history component and the batch file component be merged and that they be able to read instructions and then execute them.

Add a note hereAdd a note here**5.5:** Add to the history serialization search capabilities that allow the calculator component to search for a specific history.

**Chapter 6: Messages and Asynchronous Processing**

**Add a note hereAdd a note here****Overview**

Add a note hereAdd a note hereIn this chapter, we will do the following:

* Add a note hereAdd a note hereWrite event-driven applications
* Add a note hereAdd a note hereWrite message-based programs
* Add a note hereAdd a note hereLearn how to use callbacks and asynchronous calls
* Add a note hereAdd a note hereCreate a service to process events

## **The Purpose of this Chapter**

Add a note hereAdd a note hereThe first software programs were entirely driven by a task. Typically, a developer would create a routine and that routine would be called in the scope of solving a larger task. When all of the routines were called, the task would end and generate a result. This result would be displayed and the program might run another task. When the GUI arrived, however, the programming model changed dramatically. Instead of expecting an answer, the system would have to wait and perform actions when they were requested. This is called *event- driven programming*, which is similar to message-based programming. With *event-driven programming* the idea is to react to events. An event could be the arrival of new data, clicking of a mouse, adding of a new user, etc. The objective of this chapter is to introduce, explain and show some solutions using message driven software.

## **Callbacks and Events**

Add a note hereAdd a note hereCallbacks and events are necessary because of how programs can be structured. For example, you could program a subsystem that needs to be extended or is based on functionality not defined by the subsystem. In either case, a user would define an implementation and then pass a reference of that implementation to the subsystem. The implementation of this would use the Commons Bridge, where the subsystem defines the interface and the user defines the implementation.

### Add a note hereAdd a note hereA Simple Implementation of a Callback

Add a note hereAdd a note here[Listing 6.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252) shows an example of a kernel that exposes some interfaces.

Larger View

Add a note hereAdd a note here**Listing 6.1**

Add a note hereAdd a note hereinterface Operation {

public long doSomething( long num1, long num2);

}

interface Result {

public void setResult( long result);

}

class SubSystem {

private Operation \_operation = null;

private Result \_result = null;

public SubSystem( Operation op, Result result) {

\_operation = op;

\_result = result;

}

public void operation( long num1, long num2) {

\_result.setResult( \_operation.doSomething(num1,num2));

}

}

Add a note hereAdd a note hereIn [Listing 6.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252), two interfaces—Operation and Result—are defined. We've defined two interfaces to show how a subsystem will execute only a generic process; it will not attempt to translate what the generic process is trying to do. The class SubSystem in [Listing 6.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252) defines a subsystem used to process two numbers. This class is responsible for binding together two pieces of functionality into one process. In [Listing 6.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252), the interface Operation is responsible for performing some kind of operation on two numbers. The Commons Bridge has already demonstrated this strategy on code abstraction.

Add a note hereAdd a note hereWhat is new in [Listing 6.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252) is that the interface Result is used to display a result. Yes, the interface Result and associated implementations are implemented using the Commons Bridge. However, the semantics of the interface Result are very different from those of the interface Operation. The interface Result receives data asynchronously and processes it. The interface Operation is very obviously used to process two numbers and to return a result. However, the interface Result receives a result and has to do something with it. You may notice that the statement "has to do something with it" is very vague. Well, it's vague because, unlike with a task-driven interface, an event-driven or asynchronous interface indicates that something has occurred. Once something has occurred, it is the responsibility of the interface implementation to do something.

Add a note hereAdd a note hereNotice how simple and elegant the class method SubSystem.operation is in [Listing 6.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252). The method is not responsible for the input, output, or processing. It's responsible only for combining the various interfaces and operations into something that relates to a task. This is very significant because it indicates that a micro kernel-type architecture is created. [Listing 6.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252) is a sample implementation of the two interfaces.

Add a note hereAdd a note here**Listing 6.2**

Add a note hereAdd a note hereclass MyOperation implements Operation {

public long doSomething(long num1, long num2) {

return num1 + num2;

}

}

class MyResult implements Result {

public void setResult(long result) {

System.out.println( "The result is (" + result +

")");

}

}

Add a note hereAdd a note hereIn [Listing 6.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252), the class MyResult implements the interface Result. The class MyResult outputs the result to the system standard output. This is a simple way of displaying the output. The code shown in [Listings 6.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252) and 6.2 can be wired together and utilized as shown in [Listing 6.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252).

Add a note hereAdd a note here**Listing 6.3**

Add a note hereAdd a note hereSubSystem subsys = new SubSystem( new MyOperation(),

new MyResult());

subsys.operation( 1, 3);

Add a note hereAdd a note here[Listing 6.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252) shows how simple it is to instantiate the classes MyOperation and MyResult and let the SubSystem do the rest of the work. However, this simplicity contains a hidden trap, which relates to how the classes MyOperation and MyResult are structured.

### Add a note hereAdd a note hereWriting Maintainable Asynchronous Code

Add a note hereAdd a note hereThe problem of an asynchronous or event-driven application is that when the callback happens, the original context is lost. The class MyResult does not have a reference back to the class MyOperation. The problem of cross-referencing is best illustrated in typical GUI applications. A typical GUI application has handlers to react to specific events, like button clicks and list box selections. When reacting to events, the handlers will need to reference other Java classes that are not local to the handler class. Look back to [Listing 6.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252); the class MyResult may need to reference some data in the class MyOperation. The simplest solution to this problem is to change the implementations to [Listing 6.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252).

Add a note hereAdd a note here**Listing 6.4**

Add a note hereAdd a note hereclass EverythingInOne implements Operation, Result {

// other stuff left out for simplicity

}

Add a note hereAdd a note hereIn [Listing 6.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252), the class EverythingInOne implements both the interfaces Operation and Result. This approach of implementing multiple interfaces with one class works, but it is not an ideal object-oriented solution. Implementing two interfaces does not break object-oriented design; what does is the overall solution to implement the interfaces in one class. That could result in an object implementing ten or more interfaces, which is not good object-oriented design.

Add a note hereAdd a note hereA more typical solution is shown in [Listing 6.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252).

Add a note hereAdd a note here**Listing 6.5**

Add a note hereAdd a note hereclass MyOperation implements Operation {

private Result \_result;

}

class MyResult implements Result {

private Operation \_operation;

}

Add a note hereAdd a note hereIn [Listing 6.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252), the solution is to have each class reference an instance of the interface that each class implements. This solution is adequate, but it typically leads to very complex and brittle classes because each class references too many other interface instances. This is a very typical problem when you're building GUI-based applications. Java can solve this problem very elegantly, as shown in [Listing 6.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252).

Add a note hereAdd a note here**Listing 6.6**

Add a note hereAdd a note hereclass MyOverallClass implements Operation {

private class LocalMyResult implements Result {

public void setResult(long result) {

doOutput( "Result is (" + result + ")");

}

}

private void doOutput( String buffer) {

System.out.println( buffer);

}

public Result newResult() {

return new LocalMyResult();

}

public long doSomething(long num1, long num2) {

return num1 + num2;

}

}

Add a note hereAdd a note hereIn [Listing 6.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252), the nested class LocalMyResult shows the clever Java trick. Notice in the implementation of the method setResult that a method call to doOutput is made. The trick is that the method implementation for doOutput is in the class MyOverallClass. This trick shows that an inner class can reference data in the outer class. When you're writing GUI code or code that has callbacks, this would be a good implementation strategy.

Add a note hereAdd a note hereAnother way of expressing the same notation is to use anonymous classes, as shown in [Listing 6.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252).

Add a note hereAdd a note here**Listing 6.7**

Add a note hereAdd a note hereclass MyAnonymousClass implements Operation {

private void doOutput( String buffer) {

System.out.println( buffer);

}

public Result newResult() {

return new Result() {

public void setResult(long result) {

doOutput( "Result is (" + result + ")");

}

};

}

public long doSomething(long num1, long num2) {

return num1 + num2;

}

}

Add a note hereAdd a note hereIn [Listing 6.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252), an anonymous class is defined by the method newResult. The declaration of the class in the method would be identical to the inner class of [Listing 6.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252), except that the class has no identity. However, the class does have a type and is automatically cast to the interface Result. The anonymous class is the type Result because the interface Result is instantiated in the method of newResult. The syntax looks a bit odd because it is part function call, part class declaration, and part class instantiation. No matter. The declaration is a class that is instantiated and used like any other interface instance of type Result.

### Add a note hereAdd a note hereUsing the Java Classes Observer and Observable

Add a note hereAdd a note hereThe technique we just described in [Listings 6.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252) to 6.7 is part of a pattern called the Observer. It is another way of explaining an event-driven architecture where the implementing class does not know when or how it is called. Within the Java class libraries you can use the classes Observer and Observable to create an event mechanism. In [Listing 6.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252), the class SubSystem could inform only one Result interface instance that something has occurred. If you wanted to be able to call up multiple Result interfaces, you would have to manage and maintain a collection. This is the purpose of the class Observable.

Add a note hereAdd a note hereThe class Observer is a common interface. All classes that want notifications must implement it, as shown in [Listing 6.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252).

Larger View

Add a note hereAdd a note here**Listing 6.8**

Add a note hereAdd a note hereclass ResultObserver implements Observer {

public void update(Observable o, Object arg) {

Long result = (Long)arg;

SubSystemObservable observable = (SubSystemObservable)o;

}

}

Add a note hereAdd a note hereIn [Listing 6.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252), the method update is called when a notification is generated. Think back to [Listing 6.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252) and how the interface Result was updated. In that example, the notification occurred when the class method Subsystem.operation called interface method Result.setResult. To notify the class ResultObserver, the class Observable is used, as shown in [Listing 6.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252).

Add a note hereAdd a note here**Listing 6.9**

Add a note hereAdd a note hereclass SubSystemObservable extends Observable {

private Operation \_operation = null;

public SubSystemObservable( Operation op) {

\_operation = op;

}

public void operation( long num1, long num2) {

long result = \_operation.doSomething(num1,num2);

Long longResult = new Long( result);

setChanged();

notifyObservers( longResult);

}

}

Add a note hereAdd a note hereIn [Listing 6.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252), the method operation has changed quite a bit (as compared to [Listing 6.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252)). In the implementation of the method operation, the method call to doSomething is still made. Only, this time the result value is assigned to the variable result. Then, the variable result is converted into an object of type Long, which is the variable longResult. This is a necessary step because the class Observable sends out the notifications using objects and not primitives. The call to method setChanged is necessary because it defines that the state has changed and therefore that it is OK to send a notification. To send a notification, the method notifyObservers is called. This method accepts as a single parameter the object that is sent to the receivers of the notification.

Add a note hereAdd a note hereLook back to [Listing 6.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252)—the method update would be called by the notification. The method update has two parameters. The first parameter is a reference to the class that extends the Observable class; the second is the object sent when the method notifyObservers is called. In [Listing 6.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252), you'll also see the type casts to the original data types that the notification sends.

Add a note hereAdd a note hereThe class Observable exposes the following methods, which manage the collection of observers:

* Add a note hereAdd a note here**addObserver**: Adds an observer to the collection
* Add a note hereAdd a note here**notifyObservers**: Notifies the observers about an event
* Add a note hereAdd a note here**deleteObservers**: Deletes all the observers from the collection
* Add a note hereAdd a note here**deleteObserver**: Deletes a specific observer from the collection
* Add a note hereAdd a note here**setChanged**: Sets the dirty flag, which is the flag to indicate a changed state, to true, indicating that if the method notifyObservers is called, all of the observers in the collection will receive the event
* Add a note hereAdd a note here**clearChanged**: Changes the dirty flag to false and none of the observers will receive a notification
* Add a note hereAdd a note here**hasChanged**: Retrieves the status of the dirty flag
* Add a note hereAdd a note here**countObservers**: Returns how many observers are in the collection

Add a note hereAdd a note hereWith the new sub-system class, [Listing 6.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252) changes to [Listing 6.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=149273252).

Larger View

Add a note hereAdd a note here**Listing 6.10**

Add a note hereAdd a note hereSubSystemObservable subsys = new SubSystemObservable( new MyOperation());

subsys.addObserver( new ResultObserver());

subsys.operation( 1, 2);

Add a note hereAdd a note hereUsing the Java classes Observer and Observable, a developer can create a very effective notification system, or a publish and subscribe framework. You may have combed through the Internet and questioned the effectiveness of these classes because all notification classes need to implement the interface Observer. Even so, that is not a justified reason to not use the classes. They are part of the Java class libraries and do their job correctly. Overall, they are useful and should be used when you require notifications in the application.

## **Using the *threading* Package**

Add a note hereAdd a note hereThe *threading* package in the Jakarta Commons project is a package used to help the programmer write threading applications. We have not yet discussed it because within this package are many routines that are useful for when you are writing asynchronous applications. In the *Using the Java Classes Observer and Observable* section of this chapter, we outlined the observer and callback mechanism. Consider the problem of having to send a notification to 10 observers. The creator of the notification will have to wait until all 10 observers have processed the message. This is a bottleneck and therefore should be optimized. Within the *threading* package there are such algorithms.

### Add a note hereAdd a note hereTechnical Details for the *threading* Package

Add a note hereAdd a note here[Tables 6.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=975304477) and [6.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=975304477) contain the abbreviated details necessary to use the *threading* package.

**Add a note hereAdd a note hereTable 6.1:** Repository details for the *threading* factory.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons-sandbox](http://cdcontent.books24x7.com/id_7265/jakarta-commons-sandbox.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note herethreading |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.threading |

**Add a note hereAdd a note hereTable 6.2:** Package and class details (legend: [threading] = org.apache.commons.threading).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[threading].AsyncPublisher | Add a note hereAdd a note hereA publisher class used to manage the various subscriptions. |
| Add a note hereAdd a note here[threading].Notifier | Add a note hereAdd a note hereA class used by the publisher to send an event to an individual subscriber. |
| Add a note hereAdd a note here[threading].Subscriber | Add a note hereAdd a note hereA class that is implemented by the programmer to receive subscription events. |
| Add a note hereAdd a note here[threading].FIFONotifier | Add a note hereAdd a note hereA default notifier class used to send an event to an individual subscriber, which uses a background thread. |
| Add a note hereAdd a note here[threading].Alarm | Add a note hereAdd a note hereA specific implementation of a publisher that does not accept individual events to propagate but that generates events. The event is triggered based on a time period having passed. |

### Add a note hereAdd a note hereRewriting the Observer Using Asynchronous Threads

Add a note hereAdd a note hereRecall that we talked about the Observer example in the *Using the Java Classes Observer and Observable* section. Now, we can entirely rewrite it using the same logic but with interfaces and classes from the *threading* package. We would rewrite the class Result as shown in [Listing 6.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=975304477).

Add a note hereAdd a note here**Listing 6.11**

Add a note hereAdd a note hereclass ResultSubscriber implements Subscriber {

public void receive(Object published) {

Long result = (Long)published;

}

}

Add a note hereAdd a note hereIn [Listing 6.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=975304477), the class ResultSubscriber implements the Subscriber interface instead of the previous Observer interface. The *threading* package does not implement the Observer concept, but rather the Publish and Subscribe concepts. Both concepts are identical, except their names are different. For the interface Subscriber, the method receive has to be implemented. The method receive has one published parameter; it represents the object created by the publisher.

Add a note hereAdd a note hereThe subsystem does not derive from the class Observable, but rather from the class AsyncPublisher. Both Observable and AsyncPublisher do the exact same thing, except that AsyncPublisher uses a background thread to delegate the notification. [Listing 6.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=975304477) is an implementation of the subsystem.

Add a note hereAdd a note here**Listing 6.12**

Add a note hereAdd a note hereclass SubSystemThreading extends AsynchPublisher {

private Operation \_operation = null;

public SubSystemThreading( Operation op) {

\_operation = op;

}

public void operation( long num1, long num2) {

long result = \_operation.doSomething(num1,num2);

Long longResult = new Long( result);

send( longResult);

}

}

Add a note hereAdd a note hereIn [Listing 6.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=975304477), the implementation of the method operation has a small change from [Listing 6.1](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=397#397) in that the method send is called. The parameter passed to the method send is the object that represents the data to be sent to the individual subscribers. Everything is wired together as shown by [Listing 6.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=975304477).

Larger View

Add a note hereAdd a note here**Listing 6.13**

Add a note hereAdd a note here SubSystemThreading subsys = new SubSystemThreading( new MyOperation());

subsys.subscribe( new ResultSubscriber());

subsys.operation( 1, 2);

Add a note hereAdd a note hereIn [Listing 6.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=975304477), you add a subscriber to the list by using the method subscribe.

### Add a note hereAdd a note hereManaging the Notification Process

Add a note hereAdd a note hereIn the *threading* package, the interface Notifier magically makes everything work. The class AsyncPublisher assumes that the notifier used is the default implementation FIFONotifier. The class FIFONotifier is a first-in first-out notification that delegates the tasks to a thread running in the background. The purpose of the interface Notifier is to handle the delegation of the event. The class AsynchPublisher manages the collection of observers. When a notification is triggered, the class AsynchPublisher will delegate the physical notification to the interface Notifier. At that point, the actual implementation of the interface Notifier can decide to do whatever is correct. [Listing 6.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=975304477) shows a synchronous implementation of the interface Notifier.

Larger View

Add a note hereAdd a note here**Listing 6.14**

Add a note hereAdd a note hereclass MyNotifier implements Notifier {

private String \_name = "MyNotifier";

public int getPriority() {

return 0;

}

public int size() {

return 0;

}

public void setName(String newName) {

\_name = newName;

}

public String getName() {

return \_name;

}

public void start() {

}

public void stop() {

}

public void setPriority(int newPriority) {

}

public void send(Subscriber subscriber, Object published) {

subscriber.receive(published);

}

}

Add a note hereAdd a note hereThe implementation of the interface Notifier has been reduced to the implementation of the method send. In that implementation, the subscriber is sent the object to be used as a notification object. The rest of the methods relate to how to control the delegation subsystem. The interface Notifier assumes that an implementation will be using some type of subsystem to delegate notifications.

Add a note hereAdd a note hereThe subsystem could be thread-based or process-based; it is up to the subsystem. In any case, the subsystem is responsible for calling the subscriber with a notification object. To start and stop the subsystem, you use the methods start and stop, respectively. The creator of the notification must be the one to start and stop the subsystem. The methods setPriority and getPriority are used to set the priority of the subsystem. The numbers assigned to the property priority are entirely dependent on the subsystem's implementation. The method size is used to retrieve how many notifications still need to be sent out.

### Add a note hereAdd a note hereOther Kinds of Notifications

Add a note hereAdd a note hereThe *threading* package has two other types of publishers. The class SynchPublisher is a direct counterpart to the classes Observer and Observable because all notifications are sent out in a serial manner without any background thread. While it would seem odd to add the class SyncPublisher to the Java classes if the processing actions are the same, the advantage is that it simplifies subbing in different notification implementations.

Add a note hereAdd a note hereAnother useful notification is the class Alarm. The class Alarm is a notification that does not accept any notifications (as is shown in [Listing 6.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=975304477)). The class Alarm accepts only subscribers that will be notified once the time has expired. An example of using the class Alarm is shown in [Listing 6.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=975304477).

Add a note hereAdd a note here**Listing 6.15**

Add a note hereAdd a note here Alarm alarm = new Alarm(Alarm.ONE\_SHOT, 1000);

alarm.subscribe( new AlarmSubscriber());

alarm.start();

Add a note hereAdd a note hereIn [Listing 6.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=975304477), the class Alarm is instantiated with one parameter. The parameter is the number 1000, which defines the number of milliseconds that will pass before the alarm will trigger. The alarm can be put into three modes: continuous, one shot, or multiple one shot. When the alarm is set to *continuous mode*, a notification is sent every time period. When the alarm is set to either *one shot* or *multiple one shot* mode, a single notification is generated when an alarm is triggered.

Add a note hereAdd a note hereThe difference between a one shot and multiple one shot is how the background thread is managed. For all modes, a background thread that manages the notifications is executed. In the case of one shot mode, which is used in [Listing 6.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=975304477), the background thread exits when the notification has finished. In multiple one shot mode, it does not exit the background thread. Regardless of whether you use one shot or multiple one shot mode, you can restart the alarm using the class method Alarm.start. To stop the alarm, the class method Alarm.stop is called.

## **Using Messenger Services**

Add a note hereAdd a note hereThe asynchronous programming techniques showed thus far work when both the caller and receiver are executing in the same Java virtual machine instance. When the caller and receiver are executing on different machines or at different times, a different strategy is needed. In the Java world, this is handled using Java Messaging Service (JMS). With JMS, applications are written using queues, not method calls. When writing applications use queues, it's very different than using the *threading* packages because it involves creating messages and then reacting to messages. The *messenger* package is a toolkit that makes it simpler to write JMS-type applications. It is a façade that makes it more efficient to write JMS applications without hiding access to the JMS classes.

|  |  |  |
| --- | --- | --- |
|  | **Tip** | Add a note hereAdd a note hereUsing the messenger package requires that you install the Java 2 Enterprise Edition SDK  (J2EE SDK), or it requires you to install the JMS package and the appropriate JMS server. The messenger package provides only client-side access to a JMS server. If you want the messenger package to function properly, the JMS client must be working and you must properly configure it. This can be a challenge and does require some knowledge about configuring a JMS server and JMS client. An administrator can probably help you with this configuration. Also, the J2EE SDK does have a JMS server for testing and debugging purposes. |

### Add a note hereAdd a note hereTechnical Details for the *messenger* Package

Add a note hereAdd a note here[Tables 6.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) and [6.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) contain the abbreviated details necessary to use the *messenger* package.

**Add a note hereAdd a note hereTable 6.3:** Repository details for the *messenger* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons-sandbox](http://cdcontent.books24x7.com/id_7265/jakarta-commons-sandbox.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note heremessenger |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.messenger, org.apache.commons.messagelet.\*, and javax.jms.\* (the *messenger* package requires some familiarity with the JMS specification) |

**Add a note hereAdd a note hereTable 6.4:** Package and class details (legend: [messenger] = org.apache.commons.messenger, [messagelet] = org.apache.commons.messagelet, [jms] = javax.jms).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[messenger].MessengerManager | Add a note hereAdd a note hereA static class instance that holds references to all currently loaded *messenger* objects. This class is at the center of the entire *messenger* package. |
| Add a note hereAdd a note here[messenger].Messenger | Add a note hereAdd a note hereThe main class used to interact with the JMS server. This class is a façade over the JMS client library. |
| Add a note hereAdd a note here[messagelet].SubscriptionManager | Add a note hereAdd a note hereA class used to manage the Subscriptions, references to the Connections, and the Message Listeners. |
| Add a note hereAdd a note here[messagelet].model.Subscription | Add a note hereAdd a note hereAn individual subscription from the subscription.xml class. |
| Add a note hereAdd a note here[messagelet].model.SubscriptionList | Add a note hereAdd a note hereA list of subscriptions. |
| Add a note hereAdd a note here[jms].\*Message | Add a note hereAdd a note hereA number of classes that represent an individual message. Read about and inspect these classes before you use them. |

### Add a note hereAdd a note hereConfiguring a *messenger* Client

Add a note hereAdd a note hereWhen you use JMS, using the various connection settings can be tedious. The simplest way to manage these settings is to create some generic routines and abstract the settings to either a class or a configuration file. In the case of the *messenger* package, the JMS settings are abstracted to a file called *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml). Within the *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml) file are the settings used to connect to the JMS implementation that you may be using.

Add a note hereAdd a note hereThe *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml) file can be located anywhere on the classpath, or you can specify the system property org.apache.commons.messenger. Once the file has been located, it is loaded into a singleton that is referenced whenever the *messenger* package is referenced. [Listing 6.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) is a sample *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml) file.

Add a note hereAdd a note here**Listing 6.16**

Add a note hereAdd a note here<manager>

<messenger name="topic">

<jndi lookupName="TopicConnectionFactory"

topic="true">

<property>

<name>

com.sun.jms.internal.java.naming.factory.initial

</name>

<value>

com.sun.enterprise.naming.SerialInitContextFactory

</value>

</property>

</jndi>

</messenger>

<messenger name="queue">

<jndi lookupName="QueueConnectionFactory"

topic="false">

<property>

<name>

com.sun.jms.internal.java.naming.factory.initial

</name>

<value>

com.sun.enterprise.naming.SerialInitContextFactory

</value>

</property>

</jndi>

</messenger>

</manager>

Add a note hereAdd a note here[Listing 6.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) contains two separate blocks of functionality. The first block is the XML tag messenger with the attribute value of topic. The second block is the XML tag messenger with the attribute value of queue. You can see that the messenger topic and messenger queue relate to the JMS topic and JMS queue, respectively. However, that does not need to be the case. It is a convention that has been defined. The XML attribute values topic and queue in [Listing 6.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) could have been named myfabtopic and myfabqueue. The child XML elements jndi are important to notice here.

Add a note hereAdd a note hereThe XML element jndi in [Listing 6.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) is the actual configuration used by the JMS layer to connect to the JMS server. The XML elements messenger and jndi are very different in purpose and scope. The XML element messenger determines connection settings and how messages are handled. The XML element jndi defines the JMS factories used to connect to the JMS server. It is important to realize that different JMS servers have different JMS client implementations. Within the directory *[jakarta-commons-sandbox]/messenger/src/conf* are a number of *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml) file examples that provide default connections to different JMS servers.

Add a note hereAdd a note hereThe following properties can be set on the XML element messenger (properties available from the class org.apache.commons.messenger.MessengerSupport):

* Add a note hereAdd a note here**name** **(String):** This is the name of the messenger used to identify the messenger in the client code.
* Add a note hereAdd a note here**jndiDestination** **(Boolean):** This specifies whether or not a Java Naming and Directory Interface (JNDI) destination should be used. Do not confuse this with a JMS destination. In implementation terms, the jndiDestination property either connects to an existing destination or creates a new destination if one does not already exist.
* Add a note hereAdd a note here**durable** **(Boolean):** This specifies whether or not the subscriber maintains a durable subscription. A *durable subscription* is when a subscriber will receive all messages even though an object instance used to receive the messages does not exist. Durable subscriptions cost a lot of overhead.
* Add a note hereAdd a note here**durableName** **(String):** This is an identifier to a message box. When a durable subscription is requested, an identifier is required so that the JMS server knows to which message box to send messages that do not have an active client connection.
* Add a note hereAdd a note here**cacheRequestor** **(Boolean):** This specifies whether or not the TopicRequestor is cached in the context of a thread.
* Add a note hereAdd a note here**noLocal** **(Boolean):** This specifies whether or not messages can be published on the same JMS connection.
* Add a note hereAdd a note here**cacheProducers** **(Boolean):** This specifies whether or not JMS producers should be cached. Note that the *messenger* package defaults to true.
* Add a note hereAdd a note here**deliveryMode** **(Integer):** This specifies the delivery mode of the individual JMS messages, which can be DeliveryMode.NON\_PERSISTENT or DeliveryMode.PERSISTENT. A *persistent delivery mode*, which is the default, means that a message will travel from the client to the JMS server and then to the individual subscribers or receivers. A *non- persistent delivery mode* means that the message may or may not travel to the JMS server and then to the individual subscribers or receivers. The message might get lost and remain lost forever.
* Add a note hereAdd a note here**persistentDelivery** **(Boolean):** This is a property to indicate whether or not a delivery is persistent. Unlike the property deliveryMode, which can be multiple different values, the property persistentDelivery is a Boolean value. This means that the deliveryMode can either be persistent or non-persistent. There are only two different delivery modes, so the Boolean flag is adequate to represent all delivery modes. You should generally use this property instead of deliveryMode.

Add a note hereAdd a note hereThe following properties can be set on the XML element jndi (properties available from the class org.apache.commons.messenger.SessionFactory):

* Add a note hereAdd a note here**transacted** **(Boolean):** This specifies whether or not the JMS session will support transactions.
* Add a note hereAdd a note here**acknowledgeMode** **(Integer):** This specifies the mode of message acknowledgment when a JMS session does not support transactions. Supported modes are:
  + Add a note hereAdd a note here**Session.DUPS\_OK\_ACKNOWLEDGE**: This is a lazy acknowledgement, which means that an acknowledgement may or may not be sent to the sender.
  + Add a note hereAdd a note here**Session.AUTO\_ACKNOWLEDGE**: This is an automatic acknowledgement provided by the session when the client receives the message.
  + Add a note hereAdd a note here**Session.CLIENT\_ACKNOWLEDGE**: This is a flag to indicate whether or not a client should acknowledge a message. When a client receives the message, an acknowledgment is manually created by explicitly calling the message acknowledgment method. The problem with this strategy is that a client might be busy processing a message and might not be able to send the acknowledgment until the processing is complete. This means that although the client has received the message, an acknowledgment of the message is delayed and thus might cause resource problems.
* Add a note hereAdd a note here**acknowledge** **(String):** This is like the property acknowledgeMode, except that the acknowledgment mode is specified by the strings dups\_ok, auto, or client. In general, you should use this property.
* Add a note hereAdd a note here**username** **(String):** This specifies the username used to connect to the JMS server.
* Add a note hereAdd a note here**password** **(String):** This specifies the password used to connect to the JMS server.
* Add a note hereAdd a note here**topic** **(Boolean):** This specifies whether or not the destination is a topic or a queue. A value of true indicates a topic, whereas a value of false indicates a queue. A topic is a Publish and Subscribe mechanism, whereas a queue is a message-based mechanism. For more details on the difference between topics and queues, please refer to the JMS documentation.
* Add a note hereAdd a note here**clientID** **(String):** This specifies the client ID used on the specific connection.
* Add a note hereAdd a note here**lookupName** **(String):** This specifies the JNDI lookup name.

Add a note hereAdd a note hereWithin the XML element jndi are a number of properties identified by a key value pair of XML elements, name and value. The individual properties are specific to the JMS server and most likely are defined in the JMS server documentation. However, the directory *[Jakarta-commons-sandbox]/messenger/src/conf* contains examples of defining *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml) files for various JMS servers.

### Add a note hereAdd a note hereConfiguring Multiple Messenger Clients

Add a note hereAdd a note hereWhen the client loads the *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml) file, the configuration is loaded into a singleton. A singleton is used so that the configuration is not repeatedly parsed. This is good to do for performance reasons, but bad for multiple configuration reasons. In the default scenario, it means that only one connection description can be used to connect. In theory, this would mean you could connect to only one JMS server and not another one. However, you can get around this problem in two ways: using a more complex configuration and manually managing the singleton.

Add a note hereAdd a note hereThe simplest solution programmatically is to use a more complicated *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml) file. Look back to [Listing 6.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585). Here, there is a defined topic and queue; however, like in [Listing 6.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), we could define additional topics and queues (note that the XML child elements of the XML element jndi have been removed for clarity).

Larger View

Add a note hereAdd a note here**Listing 6.17**

Add a note hereAdd a note here<manager>

<messenger name="topic-j2ee" jndiDestinations="true"

persistentDelivery="true">

<jndi lookupName="TopicConnectionFactory"

topic="true"

acknowledge="auto">

</jndi>

</messenger>

<messenger name="queue-j2ee" jndiDestinations="false">

<jndi lookupName="QueueConnectionFactory" topic="false">

</jndi>

</messenger>

<messenger name="topic-jboss" jndiDestinations="true">

<jndi lookupName="ConnectionFactory" topic="true">

</jndi>

</messenger>

<messenger name="queue-jboss" jndiDestinations="true">

<jndi lookupName="ConnectionFactory" topic="false"

transacted="true">

</jndi>

</messenger>

</manager>

Add a note hereAdd a note hereIn [Listing 6.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), there are four messenger XML elements. Each individual element represents a specific configuration. For explanation purposes, the XML attribute name has either an appended j2ee or jboss value. The appended j2ee value means that the configuration defined is used to connect to the default JMS server provided by the J2EE SDK. The appended jboss value means that the configuration defined is used to connection to the JBOSS JMS server.

Add a note hereAdd a note hereIt's acceptable to use the configuration approach to defining different connections. However, the problem with this strategy is that the programming environment has to be aware of the various configurations. The client application needs to know that two connections are used for the same logic, but that different JMS servers are being used. A better approach is to consider the *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml) configuration as a number of task-based configurations. A better name than queue-j2ee would be stock-publisher. To manage the stock-publisher messages, the client would write one set of method calls using the *messenger* package. The details of whether a topic or queue is used and whether or not a transaction is used are configuration issues. The *messenger* package is very useful because it lets the developer concentrate on writing JMS applications. The details of the session, connection, and other parameters are then configuration issues. Offloading the configuration issues to the administrators makes it simpler to tweak and tune the application appropriate for a specific situation.

Add a note hereAdd a note hereAs already noted, the other solution to managing multiple configurations is to load the configuration manually. Using manual techniques, the programmer can determine the exact configuration during the execution of the program, as shown in [Listing 6.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585).

Add a note hereAdd a note here**Listing 6.18**

Add a note hereAdd a note hereMessengerManager firstSingleton, secondSingleton;

MessengerManager.configure( "Messenger.xml");

firstSingleton = MessengerManager.getInstance();

MessengerManager.configure( "JBossMessenger.xml");

secondSingleton = MessengerManager.getInstance();

MessengerManager.setInstance( firstSingleton);

Add a note hereAdd a note hereIn [Listing 6.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), the singleton is stored as an instance to the class MessengerManager. Calling the class method MessengerManager.configure creates the singleton instance. However, calling this method also replaces the currently existing singleton instance. To retrieve the current singleton instance, the class method MessengerManager.getInstance is called. To replace the current singleton instance, without loading a new configuration, the class method MessengerManager.setInstance is called.

Add a note hereAdd a note hereManaging configurations manually does have a potential side effect. Consider the case of an application that sends messages and very often changes the singleton instance. In a single threaded application, this is not a problem. However, in a multi-threading scenario, it can be a huge problem. For example, thread A assigns singleton instance AA. A moment later, thread B assigns singleton instance BB. If a moment later thread A decides to send or receive messages, the wrong singleton instance will be referenced, since singleton instance BB is currently the singleton. In this scenario, there are no concurrency issues, since the *messenger* package uses synchronization. What is problematic is that correct singleton instance may not be set. And there is no way of solving this because that is how the *messenger* package is defined. The best solution is to define one configuration file and, if necessary, have it contain all of the needed references to different JMS servers.

### Add a note hereAdd a note hereA Simple Sender

Add a note hereAdd a note hereOnce you have created the configuration and added it to the classpath, a message can be sent. [Listing 6.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) is a simple message-sending application. The application is based on the configuration shown in [Listing 6.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585).

Add a note hereAdd a note here**Listing 6.19**

Add a note hereAdd a note hereMessenger messenger = MessengerManager.get( "queue");

Destination destination = messenger.getDestination(

"test.queue");

TextMessage message = messenger.createTextMessage(

"this is some text");

messenger.send( destination, message);

Add a note hereAdd a note hereIn [Listing 6.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), the class MessengerManager is used to retrieve an active instance of the class Messenger. Unlike [Listing 6.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), where the configuration was defined manually, [Listing 6.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) loads the configuration implicitly when the class method MessengerManager.get is called. If the variable messenger is null, then the configuration cannot be loaded. When a valid instance of the variable messenger is loaded, you can connect to remote queues or topics and create a destination. The method getDestination connects to a remote queue or topic and is transparent to the end programmer. In the case of [Listing 6.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), the identifier test.queue is a queue because the configuration in [Listing 6.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) specifies that the identifier queue is a queue. However, the queue could just as easily have been a topic. As already mentioned, which one it becomes depends on the configuration file *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml). Once a destination has been created, a message has to be created. In [Listing 6.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), the message is created using the method createTextMessage. To send the message, the method send is called, and the two parameters specify the destination and message.

Add a note hereAdd a note hereIt is important to note that the class and interface definitions Destination and TextMessage are directly from the JMS library. This means that the *messenger* package provides only a thin organizational layer on top of the JMS library. This is good because you don't have to worry about any extra baggage, nor do you have to learn anything else about the *messenger* package other than configuration file.

### Add a note hereAdd a note hereA Simple Receiver

Add a note hereAdd a note hereTo receive the message created in [Listing 6.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), we create a receiver, as shown in [Listing 6.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585).

Add a note hereAdd a note here**Listing 6.20**

Add a note hereAdd a note hereMessenger messenger = MessengerManager.get( "queue");

Destination destination = messenger.getDestination(

"test.queue");

Message message;

while((message = messenger.receive( destination)) != null) {

TextMessage messageText = (TextMessage)message;

System.out.println( "Message is " +

messageText.getText());

}

Add a note hereAdd a note hereLike in [Listing 6.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), the class MessengerManager in [Listing 6.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) is used to retrieve an instance to an active class instance of Messenger. In addition, to create a destination, the same method getDestination is used. What is important is that both the sender and receiver use the same configuration file. For this book's examples, the sender and receiver were on the same machine. However, in reality, this would usually not be the case. Therefore, it is important that the administrator does not have too many different versions of the file *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml) in use.

Add a note hereAdd a note hereTo receive a message, the method receive is called. This method accepts only one parameter that represents the destination from where messages are received. In the case of [Listing 6.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), the method receive will block waiting until there is a message. Having executed [Listing 6.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), we can retrieve a message. A type cast is made to convert the message into the class type TextMessage. This is a legal cast because the original message was text based. Then, to output the message that was retrieved, we call the method getText.

Add a note hereAdd a note here[Listings 6.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) and 6.20 show how simple it is to send and receive messages using JMS. It is important to stress that this is why the *messenger* package is so effective. However, it is also important to stress that most of the magic occurs within the configuration file *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml). Therefore, the administrator must take great care to make sure everything is set up correctly for proper performance and efficiency.

### Add a note hereAdd a note hereCreating Messages

Add a note hereAdd a note hereIn [Listings 6.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) and 6.20, the messages sent to and from the JMS server were text messages because the method createTextMessage was used. Other types of messages are interface-based and extend the root interface Message. The following sections describe these various types of messages.

#### **The javax.jms.Message**

Add a note hereAdd a note hereYou create the base interface Message by using the method createMessage. Creating a base message is not that useful if messages with content are being sent. The interface Message has no method to manipulate a message body. Primarily, this interface is geared towards manipulating the message headers.

Add a note hereAdd a note hereA generic JMS message has three kinds of sections: standard headers, custom-defined headers, and the body. *Standard headers* are headers that every JMS message has and needs. Examples include the destination, delivery mode, expiration, and priority. Typically, these header properties are modified using explicit method calls like getJMSMessageID and setJMSDestination.

Add a note hereAdd a note here*Custom-defined headers* are properties manipulated by the JMS provider or application. These types of headers are manipulated using a number of methods that have a naming convention similar to getBooleanProperty and setBooleanProperty. The difference in the various method names is that the type, which in our example is Boolean, would be replaced with Short, Long, or other data types. Each property is stored as a key value pair. It is important to realize that objects other than the default built-in object types like String cannot be stored as a key value pair.

Add a note hereAdd a note hereThe *body* is an abstract part of the class that is not implemented, and needs a specific interface or class type (as defined by the sections following this section).

#### **The javax.jms.BytesMessage**

Add a note hereAdd a note hereThe interface BytesMessage extends the interface Message and is used to store messages represented by a number of bytes. You create an instance by calling the method createBytesMessage. An example could be sending a binary picture. The entire message is based on an array of bytes. The arrays are manipulated using methods similar to readBoolean or writeChar. You read or write the array by reading and writing sections of data. This approach is very similar to reading and writing a file in C. The problem with this approach is that there are no boundaries to define the individual data types. For example, if the methods writeInteger and writeInteger were called, then the data could be read using the method call readDouble. The returned answer would be a double value, but not necessarily the correct double value. The byte stream in a Java context is not that useful because it requires the developer to ensure that the data is read like it was written. The only situation where this stream is useful is when you use C and C++ clients.

#### **The javax.jms.MapMessage**

Add a note hereAdd a note hereThe interface MapMessage extends the interface Message and is used to store key value pairs of data elements. You create an instance by calling the method createMapMessage. A message based on the interface MapMessage is like storing pieces of header information in the body. Custom application header information is stored in key value pairs in the same manner as the interface MapMessage. Like the interface BytesMessage, the interface MapMessage can read and write data based on the data type (like the methods setString and getObject). When you retrieve or set the value, you need to use a key. The key has to be unique and there cannot be multiple keys with the same name. Otherwise, the values will replace each other.

#### **The javax.jms.ObjectMessage**

Add a note hereAdd a note hereThe interface ObjectMessage extends the interface Message and is used to store an object or retrieve one. You create an instance using the method createObjectMessage. If you compare the interface MapMessage to the interface ObjectMessage, it would seem that the message ObjectMessage is superfluous. The interface MapMessage contains a method setObject, which allows a developer to store a key value pair that represents an object. However, this method is misleading because the only object types that can be stored are the Java Object Primitives or byte arrays. Therefore, you need to use the interface ObjectMessage in order to be able to store objects. There are only two methods on the interface ObjectsMessage: setObject and getObject. The object that will be saved as a message has to support the interface Serializable.

#### **The javax.jms.StreamMessage**

Add a note hereAdd a note hereThe interface StreamMessage extends the interface Message and is used to store an array of bytes. You create an instance using the method createStreamMessage. The interface StreamMessage and BytesMessage have identical purposes: to store a stream of byte-based data that represents some type of message. The large difference is that the interface StreamMessage has a type-safe data stream. In the implementation of the interface StreamMessage, whenever a double, long, or String is written, a type identifier is written. This means that if the message sender calls the method writeInteger and writeInteger, a reader will generate an exception if the method readDouble is called. You should generally use the interface StreamMessage instead of BytesMessage.

#### **The javax.jms.TextMessage**

Add a note hereAdd a note hereThe interface TextMessage extends the interface Message and is used to read and write a buffer of text. You create an instance using the method createTextMessage. In the context of this book, this is the preferred way of sending and receiving messages because it allows XML to be stored in the buffer. However, for those that do not want to serialize to XML, the interface ObjectMessage is also acceptable.

### Add a note hereAdd a note hereSending and Receiving Commands

Add a note hereAdd a note hereSending and receiving messages is not something that most programmers, or at least pure object-oriented programmers, enjoy. This is to a large degree why the concept of Remote Procedure Call (RPC) was invented. It is much easier to expose an object that is used by a client than to send a message and receive an answer. This goes back to the initial section of this chapter, where we showed you how to write asynchronous code. When you use messaging, the programming style is not much different from that of asynchronous programs, but it's very different from RPC's. The big difference between the asynchronous *threading* package and JMS is that the JMS infrastructure is more complex. With JMS, you can combine both the messaging and the asynchronous frameworks into a grand event-driven mechanism. However, that would be beside the point.

Add a note hereAdd a note hereWhen you're writing JMS applications that are not Publish and Subscribe mechanisms, they are workload-balancing applications. This means that for every single message added, there is only one read of the message. Therefore, if there are 10 queue readers, only one reader will retrieve the message and process it. Writing message-based applications using this technique makes it possible to almost linearly scale the processing workload.

Add a note hereAdd a note hereMessage-based workload-balancing applications are written using the design pattern *Command*. The original intent of the *Command* design pattern was to facilitate the ability to process generic macros in an application. However, the *Command* design pattern is much more useful as a workload-balancing best practice.

Add a note hereAdd a note hereThe essence of the *Command* best practice is the definition of a generic interface that exposes a single method to perform an operation. The generic interface is defined in [Listing 6.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585).

Add a note hereAdd a note here**Listing 6.21**

Add a note hereAdd a note herepublic interface Command {

public void execute();

}

Add a note hereAdd a note hereIn [Listing 6.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), the method execute is used to perform some execution. The idea of the *Command* best practice is to rely on the ability of an object to know what operations to perform.

Add a note hereAdd a note hereIn X-Windows or the Windows API, an event identifier is given to the receiver of the message. The event identifier is then used in a switch statement, and the correct action is executed. You can't maintain event identifiers because every new event requires you to update the receiver. A better approach is to use the Commons Bridge in conjunction with a generic interface that has an execution method to let the object itself decide what to do. This is the purpose of the *Command* best practice. An implementation of the interface Command is shown in [Listing 6.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585).

Add a note hereAdd a note here**Listing 6.22**

Add a note hereAdd a note herepublic class MessageBean implements Serializable, Command {

private String \_message;

public MessageBean( String inpMessage) {

\_message = inpMessage;

}

public MessageBean() { }

public void execute() {

System.out.println( "Message is " + \_message);

}

public String getMessage() {

return \_message;

}

public void setMessage( String value) {

\_message = value;

}

}

Add a note hereAdd a note hereIn [Listing 6.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), the class MessageBean implements the interfaces Serializable and Command. The class MessageBean contains one private data member, \_message, which contains a buffer that is sent by the sender and then will be processed by the receiver. Think back a moment to [Chapter 5](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=293#293); this bean conforms to all requirements required by the *betwixt* serialization package.

Add a note hereAdd a note hereIn [Listing 6.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), the message is retrieved and the text string is displayed. The abstract question is how does a receiver know what operation to perform when it receives a message? Maybe one message needs to have its contents displayed and another message requires storage of the information in the database. [Listing 6.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) does not address that issue. If you rewrote [Listings 6.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) and 6.20, which are responsible for sending and receiving messages, to use the class MessageBean, you would get [Listing 6.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585).

Larger View

Add a note hereAdd a note here**Listing 6.23**

Add a note hereAdd a note here public void send() {

try {

Messenger messenger = MessengerManager.get("queue");

Destination destination =

messenger.getDestination( "test.queue" );

MessageBean bean = new MessageBean(

"message to send");

TextMessage message = messenger.createTextMessage(

convertBeanToXML(bean));

messenger.send( destination, message);

}

catch( Exception ex) {

System.out.println( ex.getMessage());

ex.printStackTrace();

}

}

public void receive() {

try {

Messenger messenger =

MessengerManager.get("queue");

Destination destination =

messenger.getDestination( "test.queue" );

Message message;

while((message = messenger.receiveNoWait(

destination)) != null) {

TextMessage messageText = (TextMessage)message;

Command command =

convertXMLToBean( messageText.getText());

command.execute();

}

}

catch( Exception ex) {

System.out.println( ex.getMessage());

ex.printStackTrace();

}

}

Add a note hereAdd a note here[Listing 6.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) looks remarkably similar to [Listings 6.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) and 6.20. There are a few small changes, however. In the method send, the class MessageBean is instantiated and then passed to a method convertBeanToXML. The method convertBeanToXML contains code that uses the *betwixt* package to serialize the MessageBean to an XML buffer. The XML buffer is then sent as a text message.

Add a note hereAdd a note hereIn the method receive, something more complex happens. The method convertXMLToBean converts the XML buffer to a class instance. The implementation of the method convertXMLToBean is also based on the *betwixt* package. What is generic is that the method convertXMLToBean returns an interface instance of Command. The receiver therefore does not need to know any specifics regarding the implementation. The receiver now just needs to call the method execute to process the message.

Add a note hereAdd a note hereThe advantage of this approach is that the sender or receiver does not need to know any specifics about the actual bean. This approach has been used successfully in messaging applications where processing tasks are distributed on various computers. Sometimes, though, the interface Command as defined in [Listing 6.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) has to be experimented with because other generic data might need to be passed with every message. The interface as defined by [Listing 6.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) is very useful when there is only a single sender and a single receiver operation. Sometimes it is necessary to carry through workflow-type operations where there are multiple processing steps. In those cases, the interface needs to be an abstract class similar [Listing 6.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585).

Larger View

Add a note hereAdd a note here**Listing 6.24**

Add a note hereAdd a note hereabstract class CommandBase implements Command, Serializable {

private int \_executionStep;

public void execute() {

if( initialize() == true) {

executeLocal( \_executionStep);

destroy();

\_executionStep ++;

}

}

abstract public boolean initialize();

abstract public void execute( int step);

abstract public void destroy();

}

Add a note hereAdd a note hereIn [Listing 6.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), the class CommandBase still implements the interface Command. The class CommandBase implements the method execute, which calls the lifecycle methods initialize, execute, and destroy. The idea of the abstract class CommandBase is to be able to serialize and execute the business logic multiple times. The methods initialize and destroy are used to establish and cleanup resources. The method execute with a parameter is like the Command.execute method, except that the parameter identifies how many times the implementation has been executed.

Add a note hereAdd a note hereOne other variation is to pass in a context class to the method execute. In [Listing 6.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), the class MessageBean output some data. The problem with the method execute in this format is that the class does not know anything about the execution environment. In more complicated examples where database connections and storage locations are needed, this poses a problem. The implementation could figure things out using discovery mechanisms, as described in [Chapter 3](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=164#164).

Add a note hereAdd a note hereA better solution would be to pass in a Context object and let the implementation use it for specific environment information. An example of a context object and the modified Command interface is [Listing 6.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585).

Add a note hereAdd a note here**Listing 6.25**

Add a note hereAdd a note hereinterface Context {

public String getRootDirectory();

}

interface ContextCommand {

public void execute( Context ctxt);

}

### Add a note hereAdd a note hereListening for Messages

Add a note hereAdd a note hereInstead of waiting for messages, it is also possible to react to messages using a message listener. A *message listener* is a class that implements the interface MessageListener. Using the Messenger class, you can manually set up a listener. Within the *messenger* package is another package, *messagelet.model*, that enables the developer to define a subscription list using a configuration file. If you combine the subscription file with the *Command* best practice, you create the basics of a workload-balancing system. Especially useful is that a configuration file defines the workload that is processed. [Listing 6.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) shows an example.

Larger View

Add a note hereAdd a note here**Listing 6.26**

Add a note hereAdd a note here<?xml version="1.0" encoding="UTF-8"?>

<subscriptions>

<subscription connection="queue" subject="test.queue">

<listener

className="com.devspace.jseng.asynchronous.SomeMessage"/>

</subscription>

</subscriptions>

Add a note hereAdd a note hereThe subscription configuration file is defined by a number of XML subscription elements. [Listing 6.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) shows the essentials of what is necessary to define a subscription. The XML element subscription has two attributes: connection and subject. The attribute connection references a connection identifier from the *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml) file used when the subscription configuration file is processed. The attribute subject is the name of the topic or queue that the message will be read from. Within the XML element subscription can be the XML elements listener, servlet, or bridge. In [Listing 6.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), the XML element listener is used to indicate that when a message arrives, you should call the object defined by the attribute className. The XML element servlet allows the client to read from a servlet, which is also part of the *messenger* framework (and is beyond the scope of this book). The XML element bridge is used to redirect a message from one queue to another, as shown in [Listing 6.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585).

Add a note hereAdd a note here**Listing 6.27**

Add a note hereAdd a note here<subscription connection="queue" subject="foo.input">

<bridge outputConnection="queue"

outputSubject="foo.output"/>

</subscription>

Add a note hereAdd a note hereIn [Listing 6.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), the XML element bridge has two attributes: outputConnection and outputSubject. The attribute outputConnection defines the *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml) connection, which is used to connect to the JMS server. The attribute outputSubject is the name of the queue or topic where the incoming message will be sent.

Add a note hereAdd a note hereOnce you have defined the subscription configuration, you need to create the listener defined in [Listing 6.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585). A sample listener that incorporates the *Command* best practice is shown in [Listing 6.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585).

Larger View

Add a note hereAdd a note here**Listing 6.28**

Add a note hereAdd a note herepublic class MyMessageHandler implements MessageListener {

private Command convertXMLToBean( String buffer)

throws Exception {

BeanReader reader = new BeanReader();

reader.registerBeanClass( MessageBean.class);

return (Command)reader.parse(new StringReader(buffer));

}

public void onMessage(Message message) {

try {

TextMessage messageText = (TextMessage)message;

Command command = convertXMLToBean(

messageText.getText());

command.execute();

}

catch( Exception ex) {

System.out.println( ex.getMessage());

ex.printStackTrace();

}

}

}

Add a note hereAdd a note hereIn [Listing 6.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), the class MyMessageHandler implements the interface MessageListener, which only has one method. The method onMessage is the callback method that is called when a message arrives. In the implementation of the onMessage method, the variable message is cast to the type TextMessage. The contained text message is then converted into a Command interface instance using the method convertXMLToBean. To be able to process the message, the class method command.execute is called. This implementation is very similar to [Listing 6.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585), except that the class is called automatically by the subscription mechanism in the *Messenger* package.

Add a note hereAdd a note hereAll of the configuration files are wired together so that everything happens automatically. [Listing 6.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) shows how to wire everything together.

Add a note hereAdd a note here**Listing 6.29**

Add a note hereAdd a note hereMessengerManager manager = MessengerManager.getInstance();

SubscriptionManager subscriber = new SubscriptionManager();

subscriber.setMessengerManager( manager);

SubscriptionDigester digester = new SubscriptionDigester();

SubscriptionList subscriptionList =

(SubscriptionList)digester.parse(

"/src/common/subscriptions.xml");

subscriber.setSubscriptionList(subscriptionList);

subscriber.setServletContext( null);

subscriber.subscribe();

for( Iterator iter = manager.getMessengerNames();

iter.hasNext(); ) {

String name = (String) iter.next();

Messenger messenger = manager.getMessenger( name );

messenger.getConnection().start();

}

Add a note hereAdd a note here[Listing 6.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) contains two main sections. The [first section](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) is the one that uses the words "subscription" and "subscriber" in their identifier. The second section consists of the for loop. The [first section](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) uses the class SubscriptionManager, which is a class responsible for putting all of the pieces together and creating a subscription mechanism. The file *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml) is associated with the subscription using the method setMessengerManager. The class SubscriptionDigester is responsible for parsing and loading the subscription configuration file. To make the subscription list do something useful, you must associate the list with the class SubscriptionManager using the method setSubscriptionList. The method setServletContext is not used in this case since the application is running on its own. However, if [Listing 6.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) were running in the context of a servlet, then we would have to use the method setServletContext. You assemble all the pieces by calling the method subscribe. In the implementation of the method subscribe, the subscription list is the iterator and the listeners are created and associated as per the subscription configuration file.

Add a note hereAdd a note hereIn the second section, we start the various connections defined in the *Image from book*[*messenger.xml*](http://cdcontent.books24x7.com/id_7265/messenger.xml) file by iterating them. The iterator is retrieved from the method getMessengerNames. The iterator contains a list of String-based names, used to load an individual Messenger configuration. Once the messenger has been retrieved, you need to create the connection needs using the method getConnection().start().

Add a note hereAdd a note hereThe code to manage the subscriptions is that simple, but there is a hidden catch. It is the threading. [Listing 6.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=400314585) should run in its own thread. Or the subscriptions should run in their own thread. This is called using a JMS message consumer thread, which is the class ConsumerThread. You can associate the class ConsumerThread with the class Subscription by using the class method Subscription.setConsumerThread. However, in any case, the exact nature of how to manage the threading is an application-implementation detail.

## **Creating Services**

Add a note hereAdd a note hereWhen you write asynchronous applications, the receiver has to be started manually. In most asynchronous applications, there is no automatic mechanism to start a service. A *service* is an application whose sole purpose is to wait for an event and then process the event. A good service infrastructure decouples the subsystem that processes the request and the subsystem that initializes the subsystem that processes the request. Relating this to the Commons Bridge best practice, the service subsystem is the interface that defines some kind of intention.

Add a note hereAdd a note hereThe [*Services*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=801#801) package is a Jakarta Commons project that allows a developer to create a plug-in architecture. The service subsystem could be some program that initializes some state and then needs to start the various components. The components that are started are defined in a configuration file. This allows an administrator to configure which subsystem is instantiated, and in which order the instantiation occurs.

### Add a note hereAdd a note hereTechnical Details for the *Services* Package

Add a note hereAdd a note here[Tables 6.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730) and [6.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730) contain the abbreviated details necessary to use the [*Services*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=801#801) package.

**Add a note hereAdd a note hereTable 6.5:** Repository details for the *Services* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[Jakarta-commons-sandbox](http://cdcontent.books24x7.com/id_7265/jakarta-commons-sandbox.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note hereservices |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.services |

**Add a note hereAdd a note hereTable 6.6:** Package and class details (legend: [services] = org.apache.commons.services).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[services].ServiceManager | Add a note hereAdd a note hereThe main class that is used as a façade to control the configuration file contents. Essentially, this class is used to load a configuration and then execute the configuration. |
| Add a note hereAdd a note here[services].Service | Add a note hereAdd a note hereA base class used when you are creating custom services. |
| Add a note hereAdd a note here[services].EventRegistration | Add a note hereAdd a note hereA base class used to define a registration class that is used to reference an actual event. |
| Add a note hereAdd a note here[services].Event | Add a note hereAdd a note hereA base class used to define a custom event. |

### Add a note hereAdd a note hereConfiguring a Service

Add a note hereAdd a note hereThe foundation of the [*Services*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=801#801) package is an XML-defined service configuration file. The service configuration file is shown in [Listing 6.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730) (note that for space reasons, the package identifier com.devspace.jseng.asynchronous is replaced with com).

Larger View

Add a note hereAdd a note here**Listing 6.30**

Add a note hereAdd a note here<service-manager>

<event-module>

<event

name="someEvent"

type="com.CustomEventRegistration"/>

<event

name="anotherEvent"

type="org.apache.commons.services.EventRegistration"/>

</event-module>

<service-module>

<service name="firstService"

type="com.ServiceOutput" />

<service name="secondService"

type="com.ServiceProcessor" />

</service-module>

<queue-module>

<queue

name="testQueue1"

type="org.apache.commons.services.SequenceQueue"

sequence="secondService,firstService"/>

<queue

name="testQueue2"

type="org.apache.commons.services.SequenceQueue"

sequence="firstService,secondService"/>

</queue-module>

</service-manager>

Add a note hereAdd a note hereThe service configuration file defined in [Listing 6.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730) has three sections: events, services, and queues. The events section is defined by the XML element event-module. Contained within the events section are individual events that can be sent to various services. It is important to realize that events are not events in the asynchronous messaging sense. In the [*Services*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=801#801) package, events are used as context objects defined in the *Command* best practice. The idea behind the event object is to be able to dynamically configure a context object that will be consumed by the individual services. The XML element event is defined by two attributes: name and type. The attribute name defines an easy-to-understand identifier used to uniquely identify the event. The attribute type is a class that the [*Services*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=801#801) package instantiates to generate an event object.

Add a note hereAdd a note hereThe services section is defined by the XML element service-module. Contained with the services section are individual services used to process some type of program logic. The XML element service has two attributes: name and type. The attribute name defines a unique identifier used for reference purposes. The attribute type is a classname that is instantiated when the service is started.

Add a note hereAdd a note hereThe queues section is defined by the XML element queue-module. Contained with the queues section are individual queues. However, as with the events section, the individual queues are not queues in the messaging sense, but more like batch processes that involve the execution of multiple services. The idea of service queues is to enable you to define a batch process that allows the definition of a lifecycle of service calls.

### Add a note hereAdd a note hereExecuting a Service

Add a note hereAdd a note hereIn [Listing 6.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730), two services were defined. To be able to execute those services, they have to be implemented as in [Listing 6.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730).

Add a note hereAdd a note here**Listing 6.31**

Add a note hereAdd a note herepublic class ServiceOutput extends Service {

public Object execute( Event event ) {

System.out.println( "Output from ServiceOutput");

return null;

}

}

public class ServiceProcessor extends Service {

public Object execute( Event event ) {

System.out.println( "Output from ServiceProcessor");

return null;

}

}

Add a note hereAdd a note hereIn [Listing 6.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730), the custom services are classes that subclass the class Service. Each of the services implements one method, execute. The method execute is called when the service is to execute its programming logic. A null is returned, because in the [*Services*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=801#801) package the return value of a service is not inspected. The execute method has one parameter, which is an event object. To execute these two services, [Listing 6.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730) is used.

Add a note hereAdd a note here**Listing 6.32**

Add a note hereAdd a note hereServiceManager serviceManager = new ServiceManager();

serviceManager.init( "file:///src/common/service.xml");

serviceManager.execute();

Add a note hereAdd a note here[Listing 6.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730) is extremely small and simple. The class ServiceManager is instantiated. The method init loads and parses a service configuration file like one defined in [Listing 6.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730). Once the service configuration file is processed, you can execute the services using the method execute.

Add a note hereAdd a note hereWhen you call the method execute without any parameters, all of the services defined in the service configuration file will be called. The order in which the services are called is not predetermined. The order depends on the iterator used within the [*Services*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=801#801) implementation. At the time of this writing, the iterator used was based on the class HashMap.

### Add a note hereAdd a note hereExecuting a Queue and Event

Add a note hereAdd a note hereTo have more control over what service is executed in which context, you can execute a specific queue, as shown in [Listing 6.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730).

Add a note hereAdd a note here**Listing 6.33**

Add a note hereAdd a note hereServiceManager serviceManager = new ServiceManager();

serviceManager.init( "file:///src/common/service.xml");

serviceManager.execute( "testQueue1");

Add a note hereAdd a note hereIn [Listing 6.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730), the only change from [Listing 6.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730) is the addition of a parameter to the method execute. The additional parameter specifies the queue that will be executed. This, in turn, causes the execution of a single or multiple services as per the comma-separated attribute sequence of [Listing 6.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730). Each service is executed in order that is referenced in the sequence attribute.

Add a note hereAdd a note hereEach time thus far when the service was executed, the event object passed to the service was a default Event class instance. The service cannot do anything constructive with the event because it does not contain any useful information. To make the event more interesting and meaningful, we need to create two new classes. The first class is a class based on the class EventRegistration, as shown in [Listing 6.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730).

Larger View

Add a note hereAdd a note here**Listing 6.34**

Add a note hereAdd a note herepublic class CustomEventRegistration extends EventRegistration {

public String getType() {

return "com.devspace.jseng.asynchronous.CustomEvent";

}

}

Add a note hereAdd a note hereThe class CustomEventRegistration in [Listing 6.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730) is like a factory. This class does not actually instantiate the event class; rather, it provides a reference to the event class that will be instantiated. The reason for this has to do with how the services are called. The event class is instantiated and used only for the scope of the service call. Because the service call could occur thousands of times, the event class is allocated from a pool of objects (as we discussed in [Chapter 4](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=250#250)). The second new class, which is the class that is the event, is defined in [Listing 6.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730).

Add a note hereAdd a note here**Listing 6.35**

Add a note hereAdd a note herepublic class CustomEvent extends Event {

private String \_data;

public CustomEvent() {

}

}

Add a note hereAdd a note hereThe class CustomEvent in [Listing 6.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730) is not defined in any particular fashion. The only requirement is that the event class subclass the class Event. [Listing 6.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730) uses the event class in conjunction with the queue used in [Listing 6.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730).

Add a note hereAdd a note here**Listing 6.36**

Add a note hereAdd a note hereServiceManager serviceManager = new ServiceManager();

serviceManager.init( "file:///src/common/service.xml");

Event event =

serviceManager.getEventModule().getEvent( "someEvent");

serviceManager.execute( "testQueue1", event);

Add a note hereAdd a note hereIn [Listing 6.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730), the event instance is not directly instantiated, but retrieved. The class ServiceManager has references to the individual sections described in the service configuration file. You retrieve a specific event by referencing the event registration identifier of the service configuration file. In [Listing 6.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730), this means using the identifier someEvent because in [Listing 6.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=173171730), this was cross-referenced with the class CustomEventRegistration. In addition, the class CustomEventRegistration references the class CustomEvent, which means that the event variable will reference an instance of the class CustomEvent. Then, the method execute is called with the event instance as the second parameter.

## **Summary**

Add a note hereAdd a note hereAsynchronous and message-based programming is different from traditional programming because you cannot know when something is going to occur. A method call that adds two numbers is going to return a result. A method call will never decide to maybe return a number or execute some other instructions before returning the number. This might be obvious, but that is how asynchronous and message-based programming works.

Add a note hereAdd a note hereIn the beginning of this chapter, we showed you the simplest kind of asynchronous programming by using the interface technique. The interface technique is very simple, but the Java libraries created the Observer and Observable interfaces

## **Questions**

Add a note hereAdd a note here**6.1:** Outline the different types of messaging and define a scenario where each would be useful.

Add a note hereAdd a note here**6.2:** Extend the extension framework such that when a calculation is executed a message "Come back later" is returned. Also add to the extension a message "More data later." Neither message needs to be text based; both could be numerical. The objective is to be able to call an extension, get a message, and then come back to that session. Note the use of the word "session," which is a hint on how to solve the problem.

Add a note hereAdd a note here**6.3:** Implement the asynchronous messages using the Publish and Subscribe framework. Add an operation of your choosing that would ideally illustrate an asynchronous calculation.

Add a note hereAdd a note here**6.4:** Modify the calculator and associated extensions to use the stack principle of the HP calculator called HP Reverse Polish Notation (RPN). Hint: use the *Command* best practice and remember to clean up the entire calculator component (e.g., history, etc.).

**Chapter 7: Collections and Searching for Objects**

**Add a note hereAdd a note here****Overview**

Add a note hereAdd a note hereIn this chapter, we will do the following:

* Add a note hereAdd a note hereManipulate Java Beans dynamically
* Add a note hereAdd a note hereMake dynamic method and constructor calls
* Add a note hereAdd a note hereManage large collections of objects using specialized collection classes
* Add a note hereAdd a note hereExecute queries on collections of objects using an XML XPath notation

## **The Purpose of This Chapter**

Add a note hereAdd a note hereWhen you're writing a program, the likelihood that there will be only a single object referencing another object is nearly zero. When you write programs using a language such as Java, object hierarchies will be created. There might be a root-level object referencing another object, which references a group of objects, each referencing multiple objects. When you consider such an architecture from a high-level point of view, the hierarchy can become very daunting.

Add a note hereAdd a note hereThe solution to such a hierarchy is to consider the hierarchy as a set of data that can be filtered, manipulated, collected, and moved. The hierarchy needs higher-level classes that allow a developer to treat the hierarchy as a dynamic data set that can be dynamically manipulated. Overall, this means that Java programs must be programmed using dynamic programming techniques. Of course, this refers only to the hierarchies, not to the entire program.

Add a note hereAdd a note hereSome people may comment that writing dynamic code is not a good idea because it causes problems when a program is executed. People who like strong data typing say that a compiler will find errors ahead of time and hence will make an application more stable and robust. The programming language Python is an excellent example of how to manage dynamic data types. In Python, all data is dynamic but strongly typed. Python variables, once assigned, cannot be converted willy-nilly to another data type. That type of dynamic programming is the task of this chapter.

## ***The Beanutils* Package**

Add a note hereAdd a note hereThe *Beanutils* package is used to dynamically manipulate a Java Bean. Typically, when referencing other classes, most classes will use a hard-coded identifier. Using the *Beanutils* package, you can dynamically choose a method and then use it when the program is executing.

Add a note hereAdd a note hereLet's take the example of building a serialization framework. [Listing 7.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) defines a very simple bean (some parts have been omitted for simplicity).

Add a note hereAdd a note here**Listing 7.1**

Add a note hereAdd a note hereclass SimpleBean {

public void setValue( int value) { }

public int getValue() { }

}

Add a note hereAdd a note hereIn [Listing 7.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the class SimpleBean exposes the property value. In a scripting language the bean could be assigned using a notation similar to [Listing 7.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Add a note hereAdd a note here**Listing 7.2**

Add a note hereAdd a note hereSimpleBean bean;

bean.value = 1234;

Add a note hereAdd a note hereIn [Listing 7.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the bean is assigned as if the bean getters and setters were a data member. The problem with [Listing 7.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) is that it does not work in Java. If we compiled [Listing 7.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), a missing declaration error would be generated since the Java compiler would explicitly look for a data member with an identifier value. In a dynamic solution, the property could be assigned as shown in [Listing 7.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Add a note hereAdd a note here**Listing 7.3**

Add a note hereAdd a note hereassignPropertyValue( "value", 1234);

Add a note hereAdd a note hereThe advantage of the solution proposed by [Listing 7.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) is that specific properties could be assigned using configuration files. Alternatively, we could load any generic bean and assign a property without knowing the interface or class definition details.

Add a note hereAdd a note hereThe sole purpose of the *Beanutils* package in the Jakarta Commons project is to make dynamic programming much simpler. In fact, many of the packages described in this book depend on the *Beanutils* package.

### Add a note hereAdd a note hereTechnical Details for the *Beanutils* Package

Add a note hereAdd a note here[Tables 7.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) and [7.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) contain the abbreviated details necessary to use the *Beanutils* package.

**Add a note hereAdd a note hereTable 7.1:** Repository details for the *Beanutils* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note herebeanutils |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.beanutils |

**Add a note hereAdd a note hereTable 7.2:** Package and class details (legend: [utils] = org.apache.commons.beanutils).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[utils].BeanUtils | Add a note hereAdd a note hereA static class used to manipulate a bean and perform dynamic operations such as retrieving and assigning property values |
| Add a note hereAdd a note here[utils].ConvertUtils | Add a note hereAdd a note hereA static class used to perform conversions from one data type to another |
| Add a note hereAdd a note here[utils].Converter | Add a note hereAdd a note hereAn interface that is implemented when custom data conversion implementations are written |
| Add a note hereAdd a note here[lang].ConstructorUtils | Add a note hereAdd a note hereA static class used to call a constructor dynamically, which returns a newly instantiated object |
| Add a note hereAdd a note here[lang].MethodUtils | Add a note hereAdd a note hereA static class used to call a specific method dynamically |
| Add a note hereAdd a note here[lang].DynaBean | Add a note hereAdd a note hereAn interface used to manage a property bag of items stored in key value pairs |

### Add a note hereAdd a note hereSetting and Retrieving Bean Properties

Add a note hereAdd a note hereFor the next set of demos, we will use the class BeanToWrite from [Chapter 5](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=293#293). [Listing 7.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) will refresh your memory on the implementation details of the class BeanToWrite.

Add a note hereAdd a note here**Listing 7.4**

Add a note hereAdd a note herepublic class BeanToWrite implements java.io.Serializable {

private int \_iValue;

private String \_strValue;

public BeanToWrite() {

}

public BeanToWrite( int ival, String sval) {

\_iValue = ival;

\_strValue = sval;

}

public int getIntegerValue() {

return \_iValue;

}

public void setIntegerValue( int val) {

\_iValue = val;

}

public String getStringValue() {

return \_strValue;

}

public void setStringValue( String val) {

\_strValue = val;

}

public String toString() {

return "\_iValue: [" + \_iValue +

"] \_strvalue: [" + \_strValue + "]";

}

}

Add a note hereAdd a note hereThe BeanUtils package has two classes called BeanUtils and BeanUtilsBean. The class BeanUtils is a static class that delegates all of its method implementations to a Bean-UtilsBean singleton. If desired, you can instantiate the class BeanUtilsBean and call each of the methods directly. The class BeanUtilsBean is a simplification of the Java Reflection API, which allows you to dynamically assign properties or call methods.

Add a note hereAdd a note here[Listing 7.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) starts with a simple example of assigning a property with a specific value.

Larger View

Add a note hereAdd a note here**Listing 7.5**

Add a note hereAdd a note hereBeanToWrite bean = new BeanToWrite();

BeanUtils.setProperty( bean, "integerValue", new Integer( 1));

BeanUtils.setProperty( bean, "integerValue", "2");

BeanUtils.setProperty( bean, "stringValue", "something else");

Add a note hereAdd a note hereIn [Listing 7.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the bean BeanToWrite is instantiated and assigned using the method setProperty. The method setProperty has three parameters. The first parameter is the bean that will be modified, the second is the property that is going to be updated, and the last is an object used to assign the property.

Add a note hereAdd a note hereIn [Listing 7.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the property integerValue was assigned twice. We did this on purpose to illustrate a very important point. The property integerValue is a primitive data type and is not an object. The *Beanutils* package can transform an object into a primitive data type, as shown by the two property assignments in [Listing 7.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359). In the first assignment example, the Integer object is instantiated, which is an easy conversion. However, the second assignment example of a string containing the value "2" is a more complex conversion. In either case, the assignment is successful, which indicates that the *Beanutils* package will attempt a data type transformation in the best way possible.

Add a note hereAdd a note hereIn contrast to assigning values to a bean, you can dynamically retrieve the values of a bean, as shown in [Listing 7.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), which is a continuation of [Listing 7.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Add a note hereAdd a note here**Listing 7.6**

Add a note hereAdd a note hereSystem.out.println( "integerValue " +

BeanUtils.getProperty( bean, "integerValue"));

System.out.println( "stringValue " +

BeanUtils.getProperty( bean, "stringValue"));

Add a note hereAdd a note hereIn [Listing 7.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the properties are retrieved using the method getProperty. The method getProperty has two parameters: bean, which has the properties, and the property to retrieve the value. Returned is a string buffer that would need to be converted to the desired data type.

Add a note hereAdd a note hereSetting and retrieving bean properties using this technique is an interesting technique, but not the most efficient one. It's inefficient because the property has to be searched for, and then the data types are converted to the required data types. Performing those operations takes time, and in performance-critical applications, it might not be the best way to solve problems. Note, though, that it's sometimes not possible to use another technique, as in serialization.

### Add a note hereAdd a note hereCloning and Assigning Another Bean

Add a note hereAdd a note hereThe Java specification defines that when an object is to be cloned or copied, an object should implement the clone method. Implementing the clone method can be tedious, but the class BeanUtils does make this simpler by exposing clone methods. A sample clone usage is shown in [Listing 7.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Larger View

Add a note hereAdd a note here**Listing 7.7**

Add a note hereAdd a note hereBeanToWrite bean = new BeanToWrite( 1234, "hello world");

BeanToWrite cloned = (BeanToWrite)BeanUtils.cloneBean( bean);

Add a note hereAdd a note hereIn [Listing 7.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the class BeanToWrite is instantiated and then passed to the method cloneBean. The method cloneBean will instantiate a new class instance and will then copy all of the properties. The BeanUtil class becomes very useful because it can copy properties that exist on different objects, as shown in [Listing 7.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Add a note hereAdd a note here**Listing 7.8**

Add a note hereAdd a note hereBeanToWrite srcBean = new BeanToWrite( 1234, "hello world");

BetwixtBean destBean = new BetwixtBean();

BeanUtils.copyProperties( destBean, srcBean);

Add a note hereAdd a note hereIn [Listing 7.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), there are two bean classes: BeanToWrite and BetwixtBean, both of which we discussed in [Chapter 5](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=293#293). In addition, both classes have the exact same properties. Each class is a different type, so a normal property copy or clone is not possible. The method copyProperties can read the properties from one bean and then set the properties of the other bean. In the case of the method copyProperties, the first parameter is the bean that will have its properties set; the second parameter is the bean that has its properties retrieved.

Add a note hereAdd a note hereThere is a catch to the clone and copy methods: they are shallow clones or copies. We made them shallow to mimic how the Java clone method operates. However, it is inconvenient because it does make for more complicated coding when you need to perform a deep clone. At the time of this writing, this issue was being resolved. Visit the author's Web site ([*www.devspace.com*](http://www.devspace.com)) for the latest information on this issue and on where to get the updates.

### Add a note hereAdd a note hereCloning and Assigning Only Specific Properties

Add a note hereAdd a note hereIn [Listing 7.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), we copied the instance properties of one bean type to an instance of another bean type. When the properties were copied, it was an all-or-nothing situation. This meant that either all of the property values were copied or none were. For a bean cloning solution, this is acceptable, but for a property value transfer, it might not be so acceptable. For instance, the property title might be identified as a title of a book on one bean. On another bean, title might refer to a title given to a person. On an identifier level, they are identical, but logically they are not identical and copying a title of a person on a title of a book would be incorrect. You can solve this problem by selectively removing specific property values, as shown in [Listing 7.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Larger View

Add a note hereAdd a note here**Listing 7.9**

Add a note hereAdd a note hereBeanToWrite sourceBean = new BeanToWrite( 1234, "hello world");

BetwixtBean destBean = new BetwixtBean();

Map properties = BeanUtils.describe( sourceBean);

properties.remove( "stringValue");

BeanUtils.populate( destBean, properties);

Add a note hereAdd a note hereLike in previous listings, [Listing 7.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) starts by instantiating the beans that will be used as source and destination. The method describe is used to extract the property descriptors from the bean instance that will be copied. The resultant Map-based variable properties contains the property descriptors and the associated values. This means that we could serialize or modify the variable properties before assigning them to the destination bean instance. Using the class method Map.remove, a specific property is removed from the collection. We then assign the leftover properties to the destination bean instance using the method populate.

### Add a note hereAdd a note hereAccessing Arrays and Maps

Add a note hereAdd a note hereThe *Beanutils* package can deal with more complex data types such as arrays and maps. [Listing 7.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) is a sample bean that the BeanUtils class will use to transfer properties.

Add a note hereAdd a note here**Listing 7.10**

Add a note hereAdd a note herepublic class BeanProperties {

private int \_intArray[] = new int[ 3];

private String \_strArray[] = new String[ 3];

private Map \_mapProperty = new HashMap();

public BeanProperties() {

\_intArray[ 0] = 1;

\_intArray[ 1] = 3;

\_intArray[ 2] = 9;

\_strArray[ 0] = "hello";

\_strArray[ 1] = "guten tag";

\_strArray[ 2] = "bonjour";

\_mapProperty.put( "first", "hello");

\_mapProperty.put( "second", "guten tag");

\_mapProperty.put( "third", "bonjour");

}

public int[] getIntArray() {

return \_intArray;

}

public void setIntArray( int[] value) {

\_intArray = value;

}

public String[] getStringArray() {

return \_strArray;

}

public void setStringArray( String[] value) {

\_strArray = value;

}

public Map getMapProperty() {

return \_mapProperty;

}

public void setMapProperty(Map value) {

\_mapProperty = value;

}

}

Add a note hereAdd a note hereIn [Listing 7.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the class BeanProperties has two properties that expose an int and String based array. There is also a property that exposes a Map type. In the constructor of the class BeanProperties, the array properties are assigned default values. The getters and setters expose the properties. The experiment is to copy the values of the property intArray to the property stringArray and vice versa. The focus of the experiment is to figure out how the *Beanutils* package handles this kind of property transfer. [Listing 7.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) is an implementation of the experiment.

Add a note hereAdd a note here**Listing 7.11**

Add a note hereAdd a note hereBeanProperties bean = new BeanProperties();

String arrayString[] = BeanUtils.getArrayProperty(

bean, "stringArray");

String arrayInt[] = BeanUtils.getArrayProperty(

bean, "intArray");

BeanUtils.setProperty( bean, "stringArray", arrayInt);

BeanUtils.setProperty( bean, "intArray", arrayString);

Add a note hereAdd a note hereIn [Listing 7.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the class BeanProperties is instantiated. Using the method getArrayProperty, an array representing the contents of the property is retrieved. The return value is an array of String values. An array of String values is returned regardless of the array type. This means that the intArray property is converted into an array of string values. The resulting array of string values can then be assigned to the other property using the method setProperty.

Add a note hereAdd a note hereThe class BeanUtils does not expose the method setArrayProperty. It is expected that the method setProperty is used and that the class BeanUtils figures out how the data needs to be converted. This introduces a question. What happens if the property is not a standard array type like, int, Long, or String? If a property exposes a method based on an object like the bean in [Listing 7.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), BeanUtils converts the object into a string. The conversion performed is based on calling the method toString. Implementing a meaningful toString implementation in this context is a bit tricky. In classical implementations, the toString method should return a string representation of the object, which should be concise and easy to read for the person who has to read the information. The original information of the object is lost; hence, a one-way serialization has occurred. Therefore, when you use the method getArrayProperty, do not consider it as serialization but as a way of getting a textual description of the property. The more accurate way to retrieve properties without destroying the value of the property is to use the method describe, as we saw in [Listing 7.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Add a note hereAdd a note hereYou can also retrieve an individual value from an array index, as shown in [Listing 7.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Add a note hereAdd a note here**Listing 7.12**

Add a note hereAdd a note hereBeanProperties bean = new BeanProperties();

String value = BeanUtils.getIndexedProperty(

bean, "intArray[2]");

value = BeanUtils.getIndexedProperty(bean, "intArray", 2);

Add a note hereAdd a note hereIn [Listing 7.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the array index is referenced in two different ways, but each time uses the same method: getIndexedProperty. In the first example of the method getIndexedProperty, the array index is referenced using the string intArray[2]. The string is parsed for a property and index identifier. The text is structured in the same way as a programmer would structure a Java index reference. However, what is important is that there cannot be any space between the square brackets because that would generate an exception.

Add a note hereAdd a note hereThe second way to reference an array index is the last line of [Listing 7.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359). In that example, the second parameter of the method represents an identifier of the array, and the third parameter represents the index of the array element being referenced. Using the latter way to reference an array index is simpler when you're using a program to read or write the individual elements of the array.

Add a note hereAdd a note hereIn [Listing 7.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the property mapProperty exposes a Map type. As with the array type, the class BeanUtils can manage the Map type. [Listing 7.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) shows how to access a mapped data type.

Add a note hereAdd a note here**Listing 7.13**

Add a note hereAdd a note hereBeanProperties bean = new BeanProperties();

String value = BeanUtils.getMappedProperty(

bean, "mapProperty(first)");

value = BeanUtils.getMappedProperty(

bean, "mapProperty", "first");

Add a note hereAdd a note hereIn [Listing 7.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the mapped property is accessed using a similar notation as we saw in [Listing 7.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359). The mapped property is accessed using the method getMappedProperty, which is shown in two different examples. In the first example of using the method getMappedProperty, the index of the map is accessed using a bracket notation. The notation is very similar to the array notation. And like the array notation, the index notation does not allow for extra spaces between the bracket and the identifier.

Add a note hereAdd a note hereThe second example of using the method getMappedProperty is similar to referencing an individual array index like shown in [Listing 7.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359). The second parameter of the method getMappedProperty represents the identifier of the mapped property, and the third parameter represents the key of the mapped value that should be retrieved. As with referencing the array index, using the latter approach to referencing a map is simpler in a programming context.

Add a note hereAdd a note hereIt is important to remember the title of this section (*Accessing Arrays and Maps*) when you use code demonstrated from this section. The title started with the word "accessing," which means that the techniques shown are good for extracting values. The techniques shown are not very helpful when you're transferring data from one bean to another source.

### Add a note hereAdd a note hereConverting Data Types

Add a note hereAdd a note hereThe class BeanUtilsBean uses conversion routines that convert a data type from one to another. In all of the listings so far in this chapter, the conversion happened automatically. To do manual conversions from one data type to another, we can use the class ConvertUtils. [Listing 7.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) is an example where a string is converted into an integer.

Add a note hereAdd a note here**Listing 7.14**

Add a note hereAdd a note hereString strValue = "1234";

Object value = ConvertUtils.convert( strValue, Integer.TYPE);

assertTrue(value instanceof Integer);

int val = ((Integer)value).intValue();

Add a note hereAdd a note hereIn [Listing 7.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the method convert is used to convert data from one type to another. The first parameter to the method convert is the value to be converted. The second parameter to the method convert represents destination conversion type. In [Listing 7.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the second parameter is the type Integer.TYPE, which is a class identifier for the type int. The second parameter of the method convert is the class Class. In addition, in [Listing 7.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) the variable value is tested to see if it is a class of type Integer, which in this case would be correct. The variable value is typecast to a primitive of type int and the method intValue is called to retrieve an int value.

Add a note hereAdd a note hereIt's a bit odd to use the class Class to identify the destination data type because we could have used an enumeration. However, by using the class Class, which every object references, it is easy to remember conversions. This solution also makes it simpler to extend the class ConvertUtils because the unique identifier is a class and not numeric identifier or string identifier. The standard conversions available include int, Integer, long, array, and other standard Java data types.

### Add a note hereAdd a note hereCreating a Custom Converter

Add a note hereAdd a note hereIn [Listing 7.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), a custom conversion destination is defined.

Add a note hereAdd a note here**Listing 7.15**

Add a note hereAdd a note hereObject beanValue = ConvertUtils.convert(

objValue, BeanToWrite.class);

assertTrue(beanValue instanceof BeanToWrite);

Add a note hereAdd a note here[Listing 7.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) calls the method convert. The difference between [Listing 7.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) and [Listing 7.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) is that here the second parameter is the class identifier of the class BeanToWrite. The idea is to convert a String into the class BeanToWrite. With the code in [Listing 7.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) as is, when the method convert returns the variable, beanValue does reference a valid object. In the default case, the assert method call will fail because the variable beanValue will not reference a class of type BeanToWrite. This is because the conversion routines do not have a converter destination for the class BeanToWrite. In that case, the class ConvertUtils will use a default string conversion.

Add a note hereAdd a note hereTo get around this problem, the developer would have to write a custom converter based on the interface Convert, as shown in [Listing 7.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Add a note hereAdd a note here**Listing 7.16**

Add a note hereAdd a note hereclass BeanToWriteConverter implements Converter {

public Object convert(Class type, Object value) {

try {

if( value instanceof String) {

BeanToWrite bean =

(BeanToWrite)type.newInstance();

bean.setStringValue((String)value);

return bean;

}

}

catch( Exception ex) {

throw new ConversionException(

"Attempted conversion, but failed cause "

+ ex.toString());

}

throw new ConversionException(

"BeanToWriteConverter cannot convert type "

+ type.getName());

}

}

Add a note hereAdd a note hereIn [Listing 7.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the class BeanToWriteConverter implements the interface Converter, which has only one method convert. Considering the implementation details, it would appear that the conversion is very simple to implement. However, quite a bit more is going on. The method convert has two parameters and expects a return value. The first parameter type is the class descriptor passed in originally to the class method ConvertUtils.convert. The second parameter value is the object value passed to the class method ConvertUtils.convert.

Add a note hereAdd a note hereThe fact that the class descriptors and value are passed in is important. This means when that when you perform a conversion, the converter class and descriptor class are loosely coupled. Consider the class name BeanToWriteConverter, which would indicate that all conversions would result in an object instance of the type BeanToWrite. Therefore, a small subsection of [Listing 7.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) could have been rewritten as in [Listing 7.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Add a note hereAdd a note here**Listing 7.17**

Add a note hereAdd a note hereif( value instanceof String) {

return new BeanToWrite( 0, (String)value);

}

Add a note hereAdd a note here[Listing 7.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) is simpler and more concise than the same lines shown in [Listing 7.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), but ConvertUtils wants to avoid that conciseness. The objective of implementing the interface Converter is to be able to provide conversion groups when more complex objects are involved. In the ideal case, the class BeanToWrite in [Listing 7.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) would be an interface, and the class descriptor passed in would be some class implementation. The class ConvertUtils is giving the end developer the maximum amount of flexibility, which is always a good thing.

Add a note hereAdd a note hereGoing back to [Listing 7.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the conversion desired depends on the if statement that sees which object type the second parameter value is. In the case of [Listing 7.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the only object type that is supported is the class String. If the object type is a string, then the conversion begins by instantiating the destination class type and assigning the property stringValue. The instantiated object type is then returned to the caller.

Add a note hereAdd a note hereIf the conversion type is not supported, then the interface Converter implementation must throw a ConversionException exception to indicate that something went wrong. A null object should not be returned.

Add a note hereAdd a note here[Listing 7.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) uses the custom converter.

Add a note hereAdd a note here**Listing 7.18**

Add a note hereAdd a note hereString strValue = "1234";

ConvertUtils.register(

new BeanToWriteConverter(), BeanToWrite.class);

Object beanValue = ConvertUtils.convert(

strValue, BeanToWrite.class);

assertTrue(beanValue instanceof BeanToWrite);

Add a note hereAdd a note here[Listing 7.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) looks very similar to [Listing 7.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), except that an additional method is called. The additional method register is used to register the converter class and associate it with a class descriptor, which for these examples would be the class BeanToWrite. Once the registration has been completed, the remaining lines after the register method call in [Listing 7.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) are identical to [Listing 7.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359). Then, the assertion to test the variable beanValue, which is an instance of the class BeanToWrite, would not fail, as it did in [Listing 7.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Add a note hereAdd a note hereIt would seem to be a good thing to be able to convert an object from one type to another. And generally it is a good and necessary thing. However, the implementations have to be mapped out carefully. Writing converters should not be approached willy-nilly or as needed because the wrong converter could be called at the wrong time. When writing converters, a developer should consider which combinations of conversions could occur. Once the combinations have been defined, they should be implemented, and if a mapping cannot be performed, an exception should be generated. In this case, exceptions are good things because a conversion should fail loudly; it shouldn't be silently converted into a string because that could mask potential bugs.

### Add a note hereAdd a note hereCalling Constructors Dynamically

Add a note hereAdd a note hereIt would seem that calling constructors using reflection should not be such a difficult thing. And generally it is not, if you know what you are instantiating. More often than not, though, especially when you're developing more complex frameworks, it's not always obvious what you are calling. This could be due to things like legacy code. The *Beanutils* package's class ConstructorUtils helps the developer call constructors that instantiate objects. The interesting thing about the class ConstructorUtils is that it helps the developer figure out what he wants to call. [Listing 7.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) shows a sample object instantiation.

Add a note hereAdd a note here**Listing 7.19**

Add a note hereAdd a note hereBeanToWrite bean = (BeanToWrite)

ConstructorUtils.invokeConstructor(

BeanToWrite.class, null);

Add a note hereAdd a note hereIn [Listing 7.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the class ConstructorUtils has the method invokeConstructor, which has two parameters. The first parameter, BeanToWrite.class, represents the class descriptor used to instantiate the class. The second parameter, null, represents the constructor parameters, which in [Listing 7.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) means call the constructor without any parameters. When the method returns the object instance, a type cast converts the object instance to the class BeanToWrite. At that point, you can use the object instance as you do an object instance that's instantiated using the new keyword.

Add a note hereAdd a note hereGoing back to the BeanToWrite constructor in [Chapter 5](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=293#293), we used a second constructor that looked similar to [Listing 7.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Add a note hereAdd a note here**Listing 7.20**

Add a note hereAdd a note here public BeanToWrite( int ival, String sval) {

\_iValue = ival;

\_strValue = sval;

}

Add a note hereAdd a note hereIn [Listing 7.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the constructor has two parameters, an int and a String value. To be able to call this constructor, the other variant of the method invokeConstructor has to be called, as demonstrated in [Listing 7.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Add a note hereAdd a note here**Listing 7.21**

Add a note hereAdd a note hereObject args[] = new Object[ 2];

args[ 0] = new Integer( 9876);

args[ 1] = "goodbye";

bean = (BeanToWrite)ConstructorUtils.invokeConstructor(

BeanToWrite.class, args);

Add a note hereAdd a note hereIn [Listing 7.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), you specify constructor arguments by using an array of Object objects. The constructor in [Listing 7.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) requires two arguments, which in [Listing 7.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) are allocated as an array of the class type Object. The first array element is an Integer class instance. This is necessary even though the constructor in [Listing 7.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) is the primitive int. The Integer class instance is converted to the primitive one. The second array element is a string, which is the second parameter of the constructor in [Listing 7.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359). Then, in [Listing 7.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the method invokeConstructor, where the third parameter represents the constructor arguments, is called. An instance to the class BeanToWrite will be returned.

Add a note hereAdd a note hereThe power of the class ConstructorUtils is not that it simplifies calling and instantiating classes, but that it provides you with flexibility. When you use the method invokeConstructor, the class ConstructorUtils will attempt to find the best-matched constructor. It is also possible to exactly match the constructor arguments using the method invokeExactConstructor. Invoking this method can be at times too picky when you're choosing constructors and, when a constructor cannot be found, errors will be generated even though a constructor should have been found. For most cases, though, invokeConstructor will be used.

Add a note hereAdd a note hereAfter all is done and said, there is one final catch-22. If the class that is being instantiated does not have an explicit constructor, then the method invokeConstructor will fail with the error that it could not find a constructor. We should state here that the invokeConstructor is not a replacement for a factory.

### Add a note hereAdd a note hereCalling Methods Dynamically

Add a note hereAdd a note hereWhen the method invokeConstructor is called, the underlying implementation searches for a specific method, which is a specific constructor. The specific constructor has to match the calling sequence specified by the user. The class MethodUtils is used to find the closest-matching constructor. The class MethodUtils can also be used to invoke methods dynamically, much like the class ConstructorUtils. [Listing 7.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) shows the class MethodUtils.

Add a note hereAdd a note here**Listing 7.22**

Add a note hereAdd a note hereBeanToWrite bean = new BeanToWrite( 1234, "hello");

MethodUtils.invokeMethod( bean, "setStringValue",

"goodbye");

Add a note hereAdd a note hereIn [Listing 7.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the class MethodUtils has the method invokeMethod, which has three parameters. The first parameter, bean, represents the object instance, where the method will be called. The second parameter, "setStringValue", is a string that represents the name of the method that will be called. The last parameter, "goodbye", is the parameter used to call the method. If the method had multiple arguments, then the arguments would be created in the same way as shown in [Listing 7.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) for the multi-parameter constructor.

Add a note hereAdd a note hereNow we need to consider two other issues: overloaded methods and static methods. *Overloaded methods* are similar methods with different parameters, whereas *static methods* are methods that do not require an object instance. As in the constructor calls, the *Bean-utils* package will find the method that most closely matches the method to be called. When a method is overloaded, some odd things could happen, as shown in [Listing 7.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Larger View

Add a note hereAdd a note here**Listing 7.23**

Add a note hereAdd a note here public void overloadedMethod( int val) {

System.out.println( "Overloaded method with int");

}

public void overloadedMethod( Integer val) {

System.out.println( "Overloaded method with Integer");

}

Add a note hereAdd a note hereGoing back to [Listing 7.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the arguments to the constructor or method are stored in an array of objects. Primitives cannot be stored in object arrays, so to get around the problem, the class Integer is used. However, in [Listing 7.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), there is a method that uses both the class Integer and the primitive int. Now there is a dilemma, because the method overloadedMethod with the primitive int will not be accessible using the method invokeMethod. There is no solution other than retrieving the specific method manually and then calling that method explicitly (we will discuss how to do this shortly). If we wanted to call the overloaded method with the primitive, we would have to change the other method with the class Integer to another data type like Long.

Add a note hereAdd a note hereCalling static methods can also cause a bit of confusion, as shown in [Listing 7.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Add a note hereAdd a note here**Listing 7.24**

Add a note hereAdd a note here public static void staticMethod() {

System.out.println( "Static method");

}

Add a note hereAdd a note hereIn [Listing 7.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the static method is just another method except that the keyword static is used. This means that the method staticMethod does not require an instance when the method is called. Therefore, in the simplest case, [Listing 7.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) can be used to call the method.

Add a note hereAdd a note here**Listing 7.25**

Add a note hereAdd a note hereBeanMethods methods = new BeanMethods();

MethodUtils.invokeMethod( methods, "staticMethod", null);

Add a note hereAdd a note hereLooking at [Listing 7.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), it is obvious that everything will work. However, what is also obvious is that it might not be the most efficient piece of code. Ideally, a static method, like a static method where the class does not have to be initialized, should be called. The solution lies in [Listing 7.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Larger View

Add a note hereAdd a note here**Listing 7.26**

Add a note hereAdd a note hereMethod method =

MethodUtils.getMatchingAccessibleMethod(

BeanMethods.class, "staticMethod", new Class[ 0]);

method.invoke( null, null);

Add a note hereAdd a note hereIn [Listing 7.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the class method MethodUtils.getMatchingAccessibleMethod is used to retrieve the static method from the BeanMethods class descriptor. Returned is a Method class instance. The class Method is part of the java.lang.reflect package. Next, the method invoke is used to call the static method. There are two parameters to the method invoke. The first parameter is the object instance, which in [Listing 7.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) is null. The second parameter is the array of objects used as arguments, which is also null because the static method has no arguments. The method getMatchingAccessibleMethod is used to retrieve the hidden overloaded method.

### Add a note hereAdd a note hereUsing DynaBeans

Add a note hereAdd a note hereOne of the last interesting bits about the *Beanutils* package is the interface DynaBean. A DynaBean is a bean that acts like a property bag. A DynaBean is not like a coded bean, but is more like a class that has a number of methods to store key value pairs. In earlier chapters, we used the DynaBean interface as part of another Commons package. Some may question why there is a need for something like a DynaBean when there are Map implementations like HashMap. The answer was given in the Commons mailing list, available at the Apache Web site ([*http://commons.apache.org*](http://commons.apache.org)), and it was for simplicity and ease of use. DynaBeans support setters and getters, and that is about it. There is no logic involved. [Listing 7.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) shows an example of using a DynaBean.

Add a note hereAdd a note here**Listing 7.27**

Add a note hereAdd a note hereDynaClass dynaClass = new BasicDynaClass( "MyClass",

null, new DynaProperty[] {

new DynaProperty("stringValue", String.class) });

DynaBean dynaBean = dynaClass.newInstance();

dynaBean.set( "stringValue", "hello");

System.out.println( "Greeting is " +

dynaBean.get( "stringValue"));

Add a note hereAdd a note hereThe *Beanutils* package contains the basic DynaBean class called BasicDynaBean and the factory class BasicDynaClass, as are illustrated in [Listing 7.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359). To create a DynaBean interface instance, the BasicDynaClass is instantiated and described using a number of properties called DynaProperties. The properties identify the supported properties that can be set and retrieved from the DynaBean instance. The method newInstance instantiates a new DynaBean. To assign and retrieve properties, the methods set and get are used, respectively.

Add a note hereAdd a note here[Listing 7.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) looks entirely uninteresting because the same can be achieved with the class HashMap. In fact, if we look at the implementation of BasicDynaBean, we see that a hash map is used. So, the question still persists why we use a DynaBean. To understand the DynaBean, let's focus on [Listing 7.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), which is a small piece of [Listing 7.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Add a note hereAdd a note here**Listing 7.28**

Add a note hereAdd a note heredynaBean.set( "stringValue", "hello");

System.out.println( "Greeting is " +

dynaBean.get( "stringValue"));

Add a note hereAdd a note here[Listing 7.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) uses only the interface DynaBean. This means that properties from the DynaBean are set and retrieved using the methods set and get. The class BeanUtils does the same thing, except more typing is involved. DynaBean comes into its own when you use the class WrapDynaBean like in [Listing 7.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359).

Add a note hereAdd a note here**Listing 7.29**

Add a note hereAdd a note hereBeanToWrite bean = new BeanToWrite( 1234, "hello");

DynaBean dynaBean = new WrapDynaBean( bean);

Add a note hereAdd a note hereIn [Listing 7.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the class BeanToWrite is instantiated. Then, the class WrapDynaBean is also instantiated, and the constructor parameter is the instance of class BeanToWrite. The returned value is an interface instance to DynaBean. Looking at [Listing 7.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) again, we'll notice that something really interesting has occurred. [Listing 7.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) was initially used to set and get properties from a BasicDynaBean instance defined in [Listing 7.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359). However, now with [Listing 7.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359), the exact same DynaBean code is used to retrieve and set properties in an actual coded bean. The difference is that [Listing 7.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=900210359) does not care where the DynaBean comes from.

Add a note hereAdd a note hereIn other words, we have applied the Commons Bridge pattern. This is the power of the DynaBean, which we can use to define beans regardless of the origin of the data. The data could be XML, SQL, another bean, or simply some coded routine. In all cases, the consumer of the DynaBean sees the exact same thing. This could have been applied using a Map as well, except that a Map has many more methods to implement and the logic is more complicated. In addition, a Map is intended for collections, whereas a DynaBean is intended to represent a bean of some sort. The DynaBean interface is simple and solves the problem at hand.

## **Managing Object Collections**

Add a note hereAdd a note hereIn the Java library, a number of classes—such as HashMap and Vector—support collections. Although these classes do their job, they are meant to be generic and solve a wide range of problems. The Java library, on the other hand, is meant to be generic. The Jakarta Commons project *Collections* contains all of the specialized collection classes.

Add a note hereAdd a note hereThe *Collections* package contains many classes. The book will not outline every collection class in detail because that would simply take up too much space. Instead, we explain the main concepts and provide some examples. We will not discuss the remaining classes; it's up to you to investigate those.

### Add a note hereAdd a note hereTechnical Details for the *Collections* Package

Add a note hereAdd a note here[Tables 7.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) contains the abbreviated details necessary to use the *Collections* package.

**Add a note hereAdd a note hereTable 7.3:** Repository details for the *Collections* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note hereCollections |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.collections (all packages below this package are detailed implementations that support the classes in the higher-level package) |

### Add a note hereAdd a note hereDefining the Basis

Add a note hereAdd a note hereAll collections in the *Collections* package are based on some interface defined in the *Collections* package. The *Collections* package contains the following interfaces:

* Add a note hereAdd a note here**Bag**: This is a collection that can have multiple instances of the same item in the collection.
* Add a note hereAdd a note here**BoundedCollection**: This is a collection that has a maximum number of elements that can be managed.
* Add a note hereAdd a note here**Buffer**: This collection is not just a random collection of objects, but a collection where the order of the objects is very important. For example, a Stack requires that you respect the order in which the objects were added.
* Add a note hereAdd a note here**Closure**: This represents a interface implementation that has an associated action. This is similar to the *Command* best practice, which we discussed in [Chapter 6](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=392#392).
* Add a note hereAdd a note here**Factory**: This represents an interface used to instantiate a specific class. This is used by some collections and was explained in [Chapter 3](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=164#164).
* Add a note hereAdd a note here**MultiMap**: This represents an interface that is similar to a Java library-defined Map. The difference is the interface MultiMap can associate multiple items with a single key.
* Add a note hereAdd a note here**Predicate**: This represents an interface implementation that is used to test an object resulting in either a true or false value.
* Add a note hereAdd a note here**PriorityQueue**: This defines an interface that represents a heap where there are no maximum or minimum values.
* Add a note hereAdd a note here**SortedBag**: This is an interface similar to the interface Bag, except that the Bag implementation can be sorted.
* Add a note hereAdd a note here**Transformer**: This represents an interface implementation that performs a cloned conversion of a specific object.

Add a note hereAdd a note hereDon't worry if the purpose of the various interfaces is still a puzzle. At this point, we're just introducing you to the interfaces and are highlighting the fact that there are two types of interfaces in the *Collections* package. The first type of interface is a typical collection interface like Bag. This type of interface manages actual collections of objects. The second type of interface is an action interface. This type of interfaces performs generic operations on a collection. For example, a collection filtering action would implement the interface Closure.

### Add a note hereAdd a note hereUsing Bag

Add a note hereAdd a note hereOne of the simplest classes to use is HashBag, which implements the interfaces Bag and Collection. The implementation of the interface Collection highlights the fact that most collection classes implement the interface Collection. [Listing 7.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) shows a simple usage of the class HashBag.

Add a note hereAdd a note here**Listing 7.30**

Add a note hereAdd a note hereBag bag = new HashBag();

BeanToWrite bean1 = new BeanToWrite( 1234, "hello");

BeanToWrite bean2 = new BeanToWrite( 2345, "goodbye");

bag.add( bean1);

bag.add( bean2, 2);

Iterator iter = bag.iterator();

while( iter.hasNext() == true) {

BeanToWrite bean = (BeanToWrite)iter.next();

System.out.println( bean.toString());

}

Add a note hereAdd a note hereIn [Listing 7.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the variable bag is an instantiated HashBag. To add elements to the HashBag class, you use the method add. In the second example of using the method add, there is a second parameter. The second parameter is a number that defines how many references of the object will be added to the Bag. In [Listing 7.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the number 2 means that two references of the object will be added to the Bag. To iterate the bag, an Iterator class instance is retrieved using the method iterator.

Add a note hereAdd a note hereYou can count individual object elements using the method getCount, as shown in [Listing 7.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800).

Add a note hereAdd a note here**Listing 7.31**

Add a note hereAdd a note hereSystem.out.println( "bean1 count: " + bag.getCount( bean1));

System.out.println( "bean2 count: " + bag.getCount(bean2));

System.out.println( "total count: " + bag.size());

Add a note hereAdd a note hereThe method getCount has one parameter, which is the bean instance that is to be counted in the bag. To find the total number of elements the method, size is used.

Add a note hereAdd a note hereTo remove elements from the bag, the method remove is used. This is illustrated in [Listing 7.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800).

Add a note hereAdd a note here**Listing 7.32**

Add a note hereAdd a note herebag.remove( bean1);

bag.remove( bean2, 1);

Add a note hereAdd a note hereIn [Listing 7.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the method remove is used in two different variations. In the first variation, the method remove removes all instances of the object from the bag. In the second variation, the method remove removes the number of instances, as per the second parameter, of the object from the bag. After [Listing 7.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) and 7.32 have executed, only one element would be left in the bag.

Add a note hereAdd a note hereFinally, the methods removeAll, retainAll, and containsAll expect a collection, which the bag then acts upon. The problem with these methods is that they are currently in violation of how a bag should operate. The violations involve implementing the method incorrectly. When the methods implement the functionality correctly, all backward compatibility will be broken. We won't demonstrate the code here. If you want to see if the methods have been updated, we suggest that you read the Java Docs.

Add a note hereAdd a note hereFinally, the method uniqueSet is used to generate a unique collection of objects where there are no doubles. An example of using uniqueSet is [Listing 7.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800).

Add a note hereAdd a note here**Listing 7.33**

Add a note hereAdd a note herejava.util.Set set = bag.uniqueSet();

### Add a note hereAdd a note hereUsing SortedBag

Add a note hereAdd a note hereAnother variation of the Bag is the SortedBag. The purpose of the SortedBag is to organize data in a collection according to some sorting algorithm. An implementation of the interface SortedBag is the class TreeBag. You can just use the TreeBag like the HashBag without any additional work. However, that would be incorrect since the sorting algorithm may not be correct. Ideally, you should develop a comparator and associate it with the SortedBag. An example of the sorting algorithm is [Listing 7.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800).

Larger View

Add a note hereAdd a note here**Listing 7.34**

Add a note hereAdd a note hereclass BeanToWriteComparator implements java.util.Comparator {

public int compare(Object o1, Object o2) {

BeanToWrite bean1 = (BeanToWrite)o1;

BeanToWrite bean2 = (BeanToWrite)o2;

if( bean1.getIntegerValue() <

bean2.getIntegerValue()) {

return -1;

}

else if(

bean1.getIntegerValue() > bean2.getIntegerValue()) {

return 1;

}

else {

return 0;

}

}

}

Add a note hereAdd a note hereIn [Listing 7.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the class BeanToWriteComparator implements the interface Comparator. The interface Comparator has two methods, but the only method you're required to implement is compare. The purpose of the method compare is to compare two objects against each other and check if one object is less than, greater than, or equal to the other object. The exact reason why an object is greater than another object depends on the implementation of the method. In [Listing 7.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), a comparison means casting the individual objects to the class BeanToWrite and then comparing the integerValue properties. Three possible values can be returned. A negative integer is returned if the object o1 is less than the object o2. A positive integer is returned if the object o1 is greater than o2. A zero is returned if object o1 equals object o2.

Add a note hereAdd a note hereOnce you've created the comparator, the TreeBag can use it, as shown in [Listing 7.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800).

Add a note hereAdd a note here**Listing 7.35**

Add a note hereAdd a note hereSortedBag bag = new TreeBag( new BeanToWriteComparator());

bag.add( new BeanToWrite( 3456, "hello"));

bag.add( new BeanToWrite( 2345, "goodbye"), 2);

Iterator iter = bag.iterator();

while( iter.hasNext() == true) {

BeanToWrite bean = (BeanToWrite)iter.next();

System.out.println( bean.toString());

}

Add a note hereAdd a note hereIn [Listing 7.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the class TreeBag is instantiated where the constructor has the additional parameter of the comparator instance. The comparator needs to be associated when the bag is instantiated because otherwise you can't associate a comparator. Once the bag has been instantiated, you can retrieve the comparator using the method comparator. At that point, you could change some parameters to alter the sorting algorithm. However, that is not a good idea because there is no method to re-sort the objects in the bag afterwards.

Add a note hereAdd a note hereOnce a TreeBag class instance is available, you can add elements and iterate through the objects like in [Listing 7.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800). The difference with the iteration in [Listing 7.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) is that the individual objects will be sorted in increasing integerValue property values, instead of being in the order they were in when they were added to the TreeBag class.

### Add a note hereAdd a note hereUsing Decorators on a Bag Collection

Add a note hereAdd a note hereWhen you first hear the term *decorator*, a cake decorator may come to mind, but of course, that's not what we're talking about. The problem with the term decorator is that the Java Docs don't explicitly explain what a decorator does. Essentially, the Java Docs say that the decorators decorate another collection. To understand a decorator, we need to look at the *Decorator* pattern.

Add a note hereAdd a note hereThe purpose of the *Decorator* pattern is to add additional functionality to an object without actually changing the code of the object. Think of a decorator as a team of objects that all look the same except that each helps the other solve a bigger problem.

Add a note hereAdd a note hereIn this context, the decorator is understandable and refers to the definition of a set of classes that augment the capabilities of the individual collection type. In the case of the interface Bag, the class BagUtils is used to create decorators. An example of using the class BagUtils is [Listing 7.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800).

Add a note hereAdd a note here**Listing 7.36**

Add a note hereAdd a note hereHashBag bag = new HashBag();

bag.add( new BeanToWrite( 1234, "hello"));

bag.add( new BeanToWrite( 2345, "goodbye"), 2);

Bag syncBag = BagUtils.synchronizedBag( bag);

Add a note hereAdd a note hereIn [Listing 7.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), it is still necessary to allocate a base collection like we did in previous listings. The decorator is added by calling the static method synchronizedBag. The method synchronizedBag instantiates the class SynchronizedBag, which exposes a set of synchronized methods. The synchronized methods operate individually on the HashBag instance allocated at the beginning of the [Listing 7.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800). The decorator used in [Listing 7.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) adds synchronization capabilities to the class HashMap.

Add a note hereAdd a note hereThe following sections outline all of the available decorator methods. When the decorator identifier includes the word "sorted," then that decorator works like the decorator identifier without the sorted identifier. The decorated identifier with the "sorted" word includes the sorting capabilities. Decorators can be applied multiple times. For example, if you go back to [Listing 7.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the variable syncBag can be used as an input parameter for another decorator.

#### **predicatedBag, predicatedSortedBag**

Add a note hereAdd a note hereThe decorator methods predicatedBag and predicatedSortedBag are used to manage a predicated collection. A *predicated collection* is a collection that performs some type of validation before the item is added to the collection. The validation will either accept or reject the object to be added. [Listing 7.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) is a sample predicate implementation.

Add a note hereAdd a note here**Listing 7.37**

Add a note hereAdd a note hereclass BeanToWritePredicate implements Predicate {

public boolean evaluate(Object object) {

BeanToWrite bean = (BeanToWrite)object;

if( bean.getIntegerValue() != 1234) {

return false;

}

return true;

}

}

Add a note hereAdd a note hereIn [Listing 7.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the class BeanToWritePredicate implements the interface Predicate, which requires the implementation of the method evaluate. The method evaluate has one parameter, which is the object to be validated. If the object is validated and acceptable, then the method evaluate returns a value of true. Otherwise, a value of false is returned. [Listing 7.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) shows how to use the predicate class.

Add a note hereAdd a note here**Listing 7.38**

Add a note hereAdd a note hereBag bag = new HashBag();

Bag decoratedBag = BagUtils.predicatedBag( bag,

new BeanToWritePredicate());

decoratedBag.add( new BeanToWrite( 1234, "hello"));

try {

decoratedBag.add( new BeanToWrite( 2345, "goodbye"), 2);

}

catch( IllegalArgumentException ex) {

System.out.println( "oops could not add to bag");

}

Add a note hereAdd a note hereIn [Listing 7.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the first step is to instantiate the underlying bag implementation. The predicated bag collection is instantiated using the method predicatedBag. The second parameter of the method predicatedBag is the class that implements the interface Predicate. Then, whenever a consumer uses the method add of the decorated bag, the class method BeanToWrite.evaluate is called. In the example of the second add method call, a failure will occur because the property integerValue is not 1234, as required by [Listing 7.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800). This means the validation will fail and an IllegalArgumentException error will be generated. Therefore, to let the application continue, the second add method call is wrapped in an exception block to catch the exception.

#### **synchronizedBag, synchronizedSortedBag**

Add a note hereAdd a note hereThe methods synchronizedBag and synchronizedSortedBag have already been illustrated and are shown in [Listing 7.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800). If you missed them, the purpose of synchronizedBag and synchronizedSortedBag is to be able synchronize access to the data members, which is very useful in a multi-threaded scenario.

#### **transformedBag, transformedSortedBag**

Add a note hereAdd a note hereThe method transformedBag is similar to the method predicatedBag in that another class has to be defined. The classes transformedBag and transformedSortedBag are operation classes that clone and manipulate the object to be added. The full intention of the interface Transformer is that an object is transformed from one object into another and that the original object is left untouched. [Listing 7.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) is a sample class that implements a transformation.

Add a note hereAdd a note here**Listing 7.39**

Add a note hereAdd a note hereclass BeanToWriteTransformed implements Transformer {

public Object transform( Object value) {

try {

return BeanUtils.cloneBean( value);

}

catch( Exception ex) {

return null;

}

}

}

Add a note hereAdd a note hereIn [Listing 7.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the class BeanToWriteTransformed implements the interface Transformer. The interface Transformer has only one method to implement, transform. The purpose of the method transform is to create a new object based on the original input object represented by the parameter value. In [Listing 7.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), there is no transformation, just a cloning of the object using the class method BeanUtils.cloneBean. This is the absolute minimum that should occur. If there are any errors to report, exceptions should be generated. The following exceptions can be generated:

* Add a note hereAdd a note here**ClassCastException**: This means that the input class was not of the correct type.
* Add a note hereAdd a note here**IllegalArgumentException**: This means that the input class is not in a correct state or valid.
* Add a note hereAdd a note here**FunctorException**: This means that the class could not be transformed.

Add a note hereAdd a note hereOnce the transformer class has been implemented, it can be used like in [Listing 7.40](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800).

Add a note hereAdd a note here**Listing 7.40**

Add a note hereAdd a note hereBag bag = new HashBag();

Bag decoratedBag = BagUtils.transformedBag( bag,

new BeanToWriteTransformed());

decoratedBag.add( new BeanToWrite( 1234, "hello"));

Add a note hereAdd a note here[Listing 7.40](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) is virtually identical to [Listing 7.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800). The difference is that the method transformedBag is used here.

Add a note hereAdd a note hereWhen implementing the interface Transformer, the method transform could have returned the original object. However, that would be violating the contract. There is nothing in the sources to prevent the implementation from returning the original. It is important that the contract be kept up, because further down in the program, there might be a piece of code that manipulates the objects in the bag. That would cause the original objects to be altered.

#### **typedBag, typedSortedBag**

Add a note hereAdd a note hereThe methods typedBag and typedSortedBag are used to create a typed bag, which is a bag that allows only specific types of objects in the bag. [Listing 7.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) is an example of how to use a typed bag.

Larger View

Add a note hereAdd a note here**Listing 7.41**

Add a note hereAdd a note hereBag bag = new HashBag();

Bag decoratedBag = BagUtils.typedBag( bag, BeanToWrite.class);

decoratedBag.add( new BeanToWrite( 1234, "hello"));

try {

decoratedBag.add( new BetwixtBean( 2345, "goodbye"));

}

catch( IllegalArgumentException ex) {

System.out.println( "oops cannot add");

}

Add a note hereAdd a note hereIn [Listing 7.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the method typedBag has two parameters. The first parameter is the bag to decorate. The second parameter is a class descriptor that specifies the class that can be added to the bag. The first instance of the method add is acceptable because the class that is added is the right class type. The second instance of the method add attempts to add the class type BetwixtBean. The second instance of the method does not succeed because it is the incorrect type and generates an IllegalArgumentException.

#### **unmodifiableBag, unmodifiableSortedBag**

Add a note hereAdd a note hereThe methods unmodifiableBag and unmodifiableSortedBag return a bag that cannot be modified. The bag essentially becomes readonly. Any attempt to modify the bag will result in an UnsupportedOperationException exception.

### Add a note hereAdd a note hereMore Default Predicates

Add a note hereAdd a note hereIn [Listing 7.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), we created a user-defined predicate. Within the Collections package are a number of default predicates that you can use to carry out default actions. Alternatively, you can combine predicates to create another predicate, as shown in [Listing 7.42](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800).

Add a note hereAdd a note here**Listing 7.42**

Add a note hereAdd a note hereclass AndPredicate implements Predicate {

private Predicate \_p1, \_p2;

public AndPredicate( Predicate p1, Predicate p2) {

\_p1 = p1;

\_p2 = p2;

}

public boolean evaluate(Object object) {

if( \_p1.evaluate(object) && \_p2.evaluate(object)) {

return true;

}

else {

return false;

}

}

}

Add a note hereAdd a note hereIn [Listing 7.42](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the class AndPredicate, which requires two other predicates when instantiated, is created. In the implementation of the method evaluate, the evaluation is handed off to the other two predicates. In addition, if the other two predicates return a value of true, then the class AndPredicate returns true as well. Notice in [Listing 7.42](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) that predicates can be decorated like collections. The class PredicateUtils is a wrapper that can create many types of predicates that could be useful to a programmer. We will now discuss the predicate methods that the class PredicateUtils exposes to create specific predicate instances.

|  |  |  |
| --- | --- | --- |
|  | **Note** | Add a note hereAdd a note hereNote that each method returns a decorated predicate, which will be called "decorated predicate" for the scope of the following list. When the word "predicate/s" is used, it refers to the predicate/s passed in as a parameter/s to the method. When the word "predicate method" is used, it refers to the method that the class PredicateUtils exposes. Finally, when the term "input object" or "added object" is used, it refers to an object being added to a collection, like in [Listing 7.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800). |

Add a note hereAdd a note hereSome of the purposes of the predicates may seem a bit puzzling; for example, the predicate method nullIsTruePredicate seems unnecessary. In fact, the idea is not to use this predicate method on its own. The idea is to use this decorated predicate in conjunction with another decorated predicate. For example, imagine when a null object is passed in; generally, you would not want to generate an exception. Using the predicate method nullIsTruePredicate, an exception would not be generated. The idea is to chain decorated predicates together to follow a defined ruleset.

#### **invokerPredicate**

Add a note hereAdd a note hereThere are two variations to this method. The first variation has one parameter, which is a method name. The second variation has three parameters. The first parameter is the method name, and the second parameter is an array of class descriptors that describe the third parameter, which is an array of objects.

Add a note hereAdd a note hereThe idea of this decorated predicate is that whenever you attempt to add an object to the collection, a method is called. The method is the name of the method passed to the predicate method. An example of a bean that supports an invoked method is [Listing 7.43](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800).

Add a note hereAdd a note here**Listing 7.43**

Add a note hereAdd a note herepublic class CollectionBean {

private int \_value;

public CollectionBean( int value) {

\_value = value;

}

public int getValue() {

return \_value;

}

public void setValue( int value) {

\_value = value;

}

public boolean validate() {

return true;

}

}

Add a note hereAdd a note hereIn [Listing 7.43](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the class CollectionBean exposes the method validate. The method that is called must return either a boolean primitive or a Boolean class instance without any parameters. If the method returns false, then an IllegalArgumentException is generated. [Listing 7.44](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) shows how to use the invokerPredicate.

Add a note hereAdd a note here**Listing 7.44**

Add a note hereAdd a note hereBag bag = new HashBag();

Bag decoratedBag = BagUtils.predicatedBag( bag,

PredicateUtils.invokerPredicate( "validate"));

decoratedBag.add( new CollectionBean( 1234));

Add a note hereAdd a note hereIn [Listing 7.44](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the class PredicateUtils.invokerPredicate is used to create the instantiated predicate. Then, when the method add is called, the predicate will invoke the validate method of the class CollectionBean defined in [Listing 7.43](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800). In [Listing 7.44](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the usage of the PredicateUtils class is a prototype for all the predicate methods that are going to be defined.

#### **allPredicate**

Add a note hereAdd a note hereThere are two variations to this predicate method. The first variation accepts a Collection class instance, and the second variation accepts an array of Predicate implementations. The decorated predicate returns a true value if all of the results in from the contained predicate collection return a true value.

#### **andPredicate**

Add a note hereAdd a note hereThis predicate method has two parameters, which are two predicate instances. The decorated predicate returns true if the two predicate instances return true. The decorated predicate is identical in functionality to [Listing 7.43](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800).

#### **anyPredicate**

Add a note hereAdd a note hereThere are two variations to this predicate method. The first variation accepts a Collection class instance, and the second variation accepts an array of Predicate implementations. The result is a decorated predicate that returns true if any of the predicates in the collection return true.

#### **asPredicate**

Add a note hereAdd a note hereThis predicate method takes as a parameter a Transformer instance implementation. This predicate method is a bit odd because whenever an object is added, that object is passed to the Transformer instance. It is expected that the Transformer instance accepts the input object and then returns a Boolean instance.

#### **eitherPredicate**

Add a note hereAdd a note hereLike the andPredicate predicate method, the eitherPredicate predicate method accepts two parameters, which are predicates. If either of the two predicates return true, then the decorated predicate returns true. However, if both predicates return true, then a false is returned by the decorated predicate.

#### **equalPredicate**

Add a note hereAdd a note hereThis predicate method takes one parameter, which is a reference object. The decorated predicate will compare every object and check if it is equal to the reference object by calling the identity object's equals method.

#### **exceptionPredicate**

Add a note hereAdd a note hereThis predicate method generates an exception whenever the decorated predicate is called.

#### **falsePredicate**

Add a note hereAdd a note hereThis predicate method returns an instantiated predicate that returns a value of true every time.

#### **identityPredicate**

Add a note hereAdd a note hereThis predicate method takes one parameter, which is an identity object. The returned predicate instance will check every input object to see if it is the same object. What this predicate essentially does is manage a collection where only one object can be added.

#### **instanceofPredicate**

Add a note hereAdd a note hereThis predicate method takes one parameter, which is a class descriptor. The decorated predicate uses the class descriptor to check if every added object is of a specific type. In a sense, this is like using the typed bag decorator described earlier in this chapter.

#### **neitherPredicate**

Add a note hereAdd a note hereThis predicate method accepts two predicates as parameters. The decorated predicate returns true if neither of the predicates returns true.

#### **nonePredicate**

Add a note hereAdd a note hereThere are two variations to this predicate method. One variation is where the predicate method accepts a single parameter of the class type Collection. The second variation is where the predicate method accepts a single parameter of an array of class type Predicate. For either variation, the decorated predicate returns true if none of the predicates returns true.

#### **notNullPredicate**

Add a note hereAdd a note hereThis predicate function has no parameters. It's a bit peculiar in that the decorated predicate returns true if the object to be added is not null.

#### **notPredicate**

Add a note hereAdd a note hereThis predicate function has one parameter, which is a predicate. The decorated predicate inverts the answer the predicate returns. Therefore, if the predicate returns true, then the decorated predicate returns false.

#### **nullIsExceptionPredicate**

Add a note hereAdd a note hereThe decorated predicate throws an exception if the input object is null.

#### **nullIsFalsePredicate**

Add a note hereAdd a note hereThe decorated predicate returns false if the input object is null.

#### **nullIsTruePredicate**

Add a note hereAdd a note hereThe decorated predicate returns true if the input object is null.

#### **nullPredicate**

Add a note hereAdd a note hereThe decorated predicate returns true if the input object is true.

#### **onePredicate**

Add a note hereAdd a note hereThis predicate method has one parameter, which is an array of predicates. The decorated predicate returns true if any of the predicates returns true.

#### **orPredicate**

Add a note hereAdd a note hereThis predicate method has two parameters, which are predicates. If any or both predicates return true, then the decorated predicated returns true.

#### **truePredicate**

Add a note hereAdd a note hereThis predicate method has no parameters. The decorated predicate always returns true.

#### **uniquePredicate**

Add a note hereAdd a note hereThis predicate method has no parameters. The decorated predicate always returns true the first time an instance of an object is passed in. If the same object instance is passed in again, then a false is returned by the decorated predicate.

### Add a note hereAdd a note hereMore Default Transformers

Add a note hereAdd a note hereAs with the class PredicateUtils, you can chain Transformers together. The class TransformerUtils exposes a number of methods that provide default Transformer interface-based functionality. We will now discuss the transformer methods that the class TransformerUtils exposes.

|  |  |  |
| --- | --- | --- |
|  | **Note** | Add a note hereAdd a note hereNote that each method returns a decorated transformer, which will be called "decorated transformer" for the scope of the following list. When the word "transformer(s)" is used, it refers to the transformer(s) passed in as a parameter(s) to the method. When the word "transformer method" is used, it refers to the method that the class TransformerUtils exposes. Finally, when the term "input object" or "added object" is used, it refers to an object being added to a collection, like in [Listing 7.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800). |

#### **asTransformer**

Add a note hereAdd a note hereThere are three variations to this transformer method. The first variation has a single parameter, which is an implementation of the interface Closure. The interface closure is like the *Command* best practice. It is expected that the Closure implementation perform a transformation implementation. The second variation has a single parameter, which is an implementation of the interface Factory. The odd part of this implementation is that the factory does not process the input object, and the input object could be gibberish. The last variation has a single parameter and is an implementation of the Predicate interface.

#### **chainedTransformer**

Add a note hereAdd a note hereThere are three variations to this transformer method. In all three variations, the decorated transformer manages a Collection of transformers, arrays, or two transformers. Whenever the decorated transformer is called, the transformers are called, and the input to the transformer is the output of a previous transformer.

#### **cloneTransformer**

Add a note hereAdd a note hereHere, the decorated transformer clones the input object, which is identical in behavior to [Listing 7.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800).

#### **constantTransformer**

Add a note hereAdd a note hereThis transformer method has one parameter, which is an input object. Each time the decorated transformer is called, the input object is returned.

#### **exceptionTransformer**

Add a note hereAdd a note hereWhen this decorated transformer is called, an exception is generated.

#### **instantiateTransformer**

Add a note hereAdd a note hereThis transformer method has two variations. In the first variation, there are no parameters. With this decorated transformer, it is expected that the input object be a class descriptor that will be instantiated to produce an object. In a sense, this variation works in a similar fashion as a factory, except that the class descriptor is passed to the decorated transformer. In the second variation, there are two parameters. The first parameter is an array of class descriptors. The second parameter is an array of objects. These two parameters are used as constructor parameters for the input object.

#### **invokerTransformer**

Add a note hereAdd a note hereThis transformer method has two variations. In the first variation, there is only one parameter, which is a string name for a method. In the second variation, there are three parameters. The first parameter is a string that defines a method name. The second parameter is an array of class descriptors. The third parameter is an array of objects. In any of these variations, the name of the method represents a method that is invoked on the input object. In the second variation, the second and third parameters identify parameters on the invoked method.

#### **mapTransformer**

Add a note hereAdd a note hereThis transformer method has one parameter, which is a Map interface implementation instance. The idea behind this transformer is that the input object represents a key in the Map, which is associated with a value. The associated value is then returned as a transformed object by the decorated transformer.

#### **nopTransformer**

Add a note hereAdd a note hereThis decorated transformer returns the input object as a transformed object. You should generally not use this transformer implementation since it does not exactly adhere to the specification of the Transformer interface.

#### **nullTransformer**

Add a note hereAdd a note hereThis decorated transformer does nothing and always returns null.

#### **switchMapTransformer**

Add a note hereAdd a note hereThis transformer method has one parameter, which is a Map interface implementation instance. The Map contains a number of key value pairs, where the key is an object and the value a transformer. When called with an input object, the decorated transformer will use the input object to cross reference to a key. The key references a transformer, which will be called, and the input object will be the parameter.

#### **switchTransformer**

Add a note hereAdd a note hereThis transformer method has various implementations, but in every implementation there are two common parameters: a collection of predicates and a matching number of transformers. When the decorated transformer is called, the input object is called for all of the predicates. For those predicates that returned a value of true, the predicate associated transformer is called with the input object. This transformer method is very versatile because it enables you to create a ruleset using predicates, which then activate transformers.

### Add a note hereAdd a note hereMore Default Closures

Add a note hereAdd a note hereThe last utility class that we will cover is the interface Closure, which we have mentioned a few times. [Listing 7.45](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) defines the Closure interface.

Add a note hereAdd a note here**Listing 7.45**

Add a note hereAdd a note herepublic interface Closure {

public void execute(Object input);

}

Add a note hereAdd a note hereIn [Listing 7.45](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), the interface Closure has only one method, execute. The method execute has only one parameter, which represents some type of object. You can define custom Closure implementations, but the class ClosureUtils has already implemented some default functionalities, which we will now discuss.

|  |  |  |
| --- | --- | --- |
|  | **Note** | Add a note hereAdd a note hereNote that each method returns a decorated closure, which will be called "decorated closure" for the scope of the following list. When the word "closure(s)" is used, it refers to the closure(s) passed in as a parameter(s) to the method. When the word "closure method" is used, it refers to the method that the class ClosureUtil exposes. Finally, when the term "input object" or "added object" is used, it refers to an object being added to a collection, as in [Listing 7.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800). |

#### **asClosure**

Add a note hereAdd a note hereThis closure method has one parameter, which is a Transformer implementation. The decorated closure will call the Transformer implementation. The problem, though, is that a Transformer is supposed to operate on a input object and not touch it; however, the Closure interface does not return a value. The result is that the Transformer creates a new object that is not saved anywhere.

#### **chainedClosure**

Add a note hereAdd a note hereThis closure method has three similar variations. The idea is to combine a number of Closure implementations into a collection, which are all called when the decorated closure is called.

#### **doWhileClosure**

Add a note hereAdd a note hereThis closure method has two parameters. The first parameter is the closure itself, and the second parameter is a predicate. When the decorator closure is called, the closure is called first. Then, the predicate is called; if the predicate returns true, the closure is called again. This process repeats until the predicate returns a value of false. The input object is passed to both the closure and the predicate and is considered the shared object.

#### **exceptionClosure**

Add a note hereAdd a note hereThis decorated closure generates an exception every time.

#### **forClosure**

Add a note hereAdd a note hereThis closure method has two parameters. The first parameter is a count, and the second parameter a closure. The decorated closure then loops, and for each loop calls the closure until the count has been reached.

#### **invokerClosure**

Add a note hereAdd a note hereThis closure method has two variations. In the first variation, there is only one parameter, which is a string name for a method. In the second variation, there are three parameters. The first parameter is a string that defines a method name. The second parameter is an array of class descriptors. The third parameter is an array of objects. In any of these variations, the name of the method represents a method that is invoked on the input object. In the second variation, the second and third parameters identify parameters on the invoked method.

#### **nopClosure**

Add a note hereAdd a note hereThe decorated closure does nothing and is meant as an empty placeholder that could be used a default value.

#### **switchClosure**

Add a note hereAdd a note hereThis closure method has various implementations, but in every implementation there are two common parameters: a collection of predicates and a matching number of transformers. When the decorated closure is called, the input object is called for all of the predicates. For those predicates that returned a value of true, the associated closure is called. This closure method is very versatile because it enables you to create a ruleset using predicates, which then activate transformers.

#### **switchMapClosure**

Add a note hereAdd a note hereThis closure method has one parameter, which is a Map interface implementation instance. The Map contains a number of key value pairs, where the key is an object and the value a closure. When called with an input object, the decorated closure will use the input object to cross-reference to a key. The key references a closure, which will be called, and the input object will be the parameter.

#### **whileClosure**

Add a note hereAdd a note hereThis closure method has two parameters. The first parameter is the predicate itself, and the second parameter is a closure. When the decorator closure is called, the predicate is called, which starts a loop. The loop continues if the predicate returns a value of true. Otherwise, the loop is stopped and the decorated closure is exited. If the loop continues, the closure is executed and the loop starts again.

### Add a note hereAdd a note hereUsing Fast Lists and Maps

Add a note hereAdd a note hereWriting multi-threading code can sometimes be frustrating; what would seem to make an application faster either slows the applications down or brings it to a halt. Multi-threaded code needs to have synchronization at certain places in the program. We discussed all of these issues in [Chapter 4](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=250#250).

Add a note hereAdd a note hereWithin the *Collections* package are the classes FastArrayList, FastHashMap, and FastTreeMap. These classes were programmed for a multi-threaded environment. [Listing 7.46](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) shows how to use the class FastArrayList.

Add a note hereAdd a note here**Listing 7.46**

Add a note hereAdd a note hereFastArrayList list = new FastArrayList();

list.add( new BetwixtBean( 1234, "hello"));

list.setFast( true);

list.add( new BetwixtBean( 2345, "goodbye"));

Add a note hereAdd a note hereThe class FastArrayList in [Listing 7.46](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800) is similar to all of the collection classes that have been shown thus far. However, there is a very big implementation difference, which is activated by the method setFast.

Add a note hereAdd a note hereAll of the fast collection classes are based on the premise that sometimes a collection is read and written, and sometimes a collection is read mostly. Read mostly is when the data in the collection is mainly queried and iterated, and not altered. You'll use different synchronization depending on which situation you're in. When an object is constantly modified using read and write operations, synchronization is the only option because the changes affect read operations. However, when an object is read mostly, synchronization is not as necessary.

Add a note hereAdd a note hereWe defined the concept of an immutable object in [Chapter 4](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=250#250). An immutable object needs no synchronization, because once the object has been created it is not modified. The problem with an immutable object is that to make changes, you need to clone and then modify the immutable object. Immutable objects are not used in every multi-threaded situation because cloning can be expensive. Now, imagine if the object was read mostly; then, using the immutable concept would be very practical because for the few times the object is cloned, performance is not affected. This is the strategy used by the fast collection classes. An example of how this strategy is implemented is [Listing 7.47](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800).

Add a note hereAdd a note here**Listing 7.47**

Add a note hereAdd a note here public boolean add(Object element) {

if (fast) {

synchronized (this) {

ArrayList temp = (ArrayList) list.clone();

boolean result = temp.add(element);

list = temp;

return (result);

}

} else {

synchronized (list) {

return (list.add(element));

}

}

}

Add a note hereAdd a note hereIn [Listing 7.47](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800), there is a decision block that decides whether or not the fast flag is set. The *fast flag* defines whether fast mode or slow mode is activated. *Fast mode* is when the collection is considered immutable, and *slow mode* is when the object uses synchronization. When the fast flag has a value of true, then fast mode is activated. In slow mode, the lower part of the decision block is executed. In the lower part of the decision block, a synchronization is executed on the list object. This means that when an element is added, a synchronization is executed, which will stop any type of operation whether it be read or write.

Add a note hereAdd a note hereLet's go back to the upper part of the decision block in [Listing 7.47](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=719625800). Here, the synchronization is only on the this object. While this would seem like synchronization based on the slow mode, there is a big difference. In slow mode, the synchronization is on the list itself, which means any references to the list result in a block. However, in fast mode, there is no synchronization. The synchronization on the this object is necessary because once the list has been cloned, the old list reference has to be replaced with the new one. Doing this would require synchronization so that only one thread can modify the list variable.

Add a note hereAdd a note hereThe fast collection classes are defined as follows:

* Add a note hereAdd a note here**FastArrayList**: This is an implementation of the class java.util.ArrayList.
* Add a note hereAdd a note here**FastHashMap**: This is an implementation of the class java.util.HashMap.
* Add a note hereAdd a note here**FastTreeMap**: This is an implementation of the class java.util.TreeMap.

Add a note hereAdd a note hereThere is a catch to the fast collections approach, however. These classes may cause unexpected errors on specific platforms. The cause of the errors is a common problem that depends on the optimization in performance on some platforms. For more details, see [*www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html*](http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html).

### Add a note hereAdd a note hereOther Classes and the Details

Add a note hereAdd a note hereWe'll now move on to discussing some other classes that are available in the *Collections* package.

|  |  |  |
| --- | --- | --- |
|  | **Tip** | Add a note hereAdd a note hereWe recommend that you do a quick read of these classes. These classes are extensions of the concepts we've already covered, so we don't allocate that much space to them. However, we still want you to know how to use these classes appropriately. |

* Add a note hereAdd a note here**ArrayStack**: This class implements a stack. This means that the last added item to the list is also the first item removed. The class ArrayStack is based on the class java.util.ArrayList. When iterating using an Iterator, the iterator starts at the bottom of the stack.
* Add a note hereAdd a note here**BeanMap**: This class implements the interface java.util.Map. The main purpose of this map is to be able to manipulate and iterate a Java Bean using a collection mech-anism. The Java Bean is set and retrieved using the methods setBean and getBean, respectively.
* Add a note hereAdd a note here**BinaryHeap**: This is a binary implementation of the interface PriorityQueue and Buffer. Simply put, the class BinaryHeap is an implementation of a stack like the class ArrayStack. The difference, though, is that the addition and removal order is determined by an ordering. The ordering is determined by the Comparator interface implementation.
* Add a note hereAdd a note here**BoundedFifoBuffer**: This is an efficient implementation of a buffer that always maintains a constant buffer size. The first element added is always the first element removed from the list (Fifo = first in first out). Iterating the elements occurs in the same order.
* Add a note hereAdd a note here**CircularFifoBuffer**: This is fixed length buffer that can always be written to. As with the class BoundFifoBuffer, if the buffer becomes full, the oldest element is overridden. This class is extremely useful in live-feed-type situations, where old data is not useful. With this class, a window of data across time is typically stored.
* Add a note hereAdd a note here**CursorableLinkedList**: This is a double linked list that implements the List interface.
* Add a note hereAdd a note here**ExtendedProperties**: This is an implementation of the class java.util.HashMap. If an element already exists in the HashMap, then the already existing contents are concatenated with the new content. Usually, the HashMap implementation simply overwrites the old data.
* Add a note hereAdd a note here**LURMap**: This is an implementation of the Map interface. The class has a maximum number of elements. When the limited is reached, an addition of a new element will remove the least recently used map entry.
* Add a note hereAdd a note here**NodeCachingLinkedList**: This is a double linked list implementation that caches the added and deleted items. This class is very useful in situations where a list is constantly reorganized by the addition, removal, or reordering of the same elements. Otherwise, this class can entail using a large amount of resources.
* Add a note hereAdd a note here**ReferenceMap**: This is a reference map that allows the use of soft references that can be garbage collected by the JVM.
* Add a note hereAdd a note here**SequencedHashMap**: This is a class where the addition of the elements produces an ordered sequence of items added list.
* Add a note hereAdd a note here**StaticBucketMap**: This is an implementation of the interface Map, which can be used very efficiently in multi-threaded environments.
* Add a note hereAdd a note here**SynchronizedPriorityQueue**: This is a thread safe implementation of the interface PriorityQueue.
* Add a note hereAdd a note here**UnboundedFifoBuffer**: This very efficient implementation of the Buffer interface can be used instead of the class ArrayList.

## **The *JXPath* Package: Referencing Objects Using XPath**

Add a note hereAdd a note hereXPath is a language used in the XML environment to reference elements within an XML document. The power of XPath is that it is a compact language that makes it very simple to generate complex queries. In XML speak, XPath is executed on the XML Document Object Model (DOM). Let's take a step back to consider this for a moment; it is evident that XPath is executed against objects. The original developers of the *JXPath* package noticed that XML searches object instances, which is very similar to what generic programs create, so they created a set of classes that use XPath as a generic querying language to navigate complex Java object hierarchies.

Add a note hereAdd a note hereWith *JXPath*, it's not essential to know the classes within the *JXPath* package, but it is imperative to know XPath. However, to understand XPath, we need to show a simple mapping between XML and Java. [Listing 7.48](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) is a sample XML document that will be converted into a Java object hierarchy.

Add a note hereAdd a note here**Listing 7.48**

Add a note hereAdd a note here<data>

<elements>hello</elements>

</data>

Add a note hereAdd a note here[Listing 7.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) is the mapped Java class.

Add a note hereAdd a note here**Listing 7.49**

Add a note hereAdd a note herepublic class Data {

private String \_elements;

public String getElements() { return \_elements; }

}

Add a note hereAdd a note hereIn [Listing 7.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) is the root XML element data, with a child element elements. This parent-child relationship is mapped as a Java Bean in [Listing 7.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577). The root XML element would be the class Data, and the child element would be the bean property elements. In [Listing 7.49](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), elements could have also represented another Java Bean object.

Add a note hereAdd a note hereGranted, there are limits to comparing XML with Java objects. For example, XML attributes and XML documents that represent HTML are not easily represented. However, this is OK, because *JXPath* is not meant to complement XML or provide XML serialization patterns. *JXPath* uses the XPath language to simplify object referencing. Therefore, it's not necessary to do referencing like that shown in [Listing 7.50](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577).

Add a note hereAdd a note here**Listing 7.50**

Add a note hereAdd a note heredata.getMyProperty().getFromCollection(

123).getAnotherProperty(

).getYetAnotherProperty().getMyValue();

Add a note hereAdd a note here[Listing 7.50](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) shows a very long and tedious reference. If you use *JXPath*, such a reference is not necessary anymore. In addition, using *JXPath*, you can add filtering code, which simply is not possible in [Listing 7.50](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577).

### Add a note hereAdd a note hereTechnical Details for the *JXPath* Package

Add a note hereAdd a note here[Tables 7.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) and [7.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) contain the abbreviated details necessary to use the *JXPath* package.

**Add a note hereAdd a note hereTable 7.4:** Repository details for the *JXPath* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note herejxpath |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.jxpath |

**Add a note hereAdd a note hereTable 7.5:** Package and class details (legend: [jxpath] = org.apache.commons.jxpath).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[jxpath].JXPathContext | Add a note hereAdd a note hereThe core class of the entire *JXPath* package. From this class all other operations are possible. This class is also the class that most developers will solely interact with. |
| Add a note hereAdd a note here[jxpath].Pointer | Add a note hereAdd a note hereA class that stores a reference to the absolute XPath expression for found elements. |
| Add a note hereAdd a note here[jxpath].CompiledExpression | Add a note hereAdd a note hereA compiled XPath class that is used for XPath expressions that are constantly used. |

### Add a note hereAdd a note hereStarting with the Basics of *JXPath*

Add a note hereAdd a note hereIf you want to be able to use the *JXPath* package, there has to be a hierarchy of objects. Granted, the samples in this chapter will only extend to a couple of layers of object hierarchy; more complexity is assumed in your programs. Otherwise, using the *JXPath* package is adding complexity when it is not necessary.

Add a note hereAdd a note hereLet's look at [Listing 7.51](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577). This consists of two Java Beans that we used in [Chapter 5](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=293#293) (parts have been deleted for simplicity).

Add a note hereAdd a note here**Listing 7.51**

Add a note hereAdd a note herepublic class BeanToWrite implements java.io.Serializable {

private int \_iValue;

private String \_strValue;

public BeanToWrite( int ival, String sval) {

\_iValue = ival;

\_strValue = sval;

}

public int getIntegerValue() {

return \_iValue;

}

public String getStringValue() {

return \_strValue;

}

}

public class ParentBean implements java.io.Serializable {

private String \_dataMember;

private BeanToWrite \_bean;

public ParentBean() {

\_bean = new BeanToWrite( 1234, "hello");

\_dataMember = "parent";

}

public String getDataMember() {

return \_dataMember;

}

public BeanToWrite getMyReferenceToAnotherBean() {

return \_bean;

}

}

Add a note hereAdd a note hereIn [Listing 7.51](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), the class BeanToWrite represents a very simple bean that exposes two properties. The class ParentBean is a simple bean that exposes two properties also; one is myReferenceToAnotherBean, which is a reference to the BeanToWrite class instance. The other is the propery getDataMember, which returns a String object instance.

Add a note hereAdd a note hereTo retrieve the value of the class property BeanToWrite.integerValue, use *JXPath*. [Listing 7.52](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) is an implementation using *JXPath*.

Add a note hereAdd a note here**Listing 7.52**

Add a note hereAdd a note hereBeanToWrite bean = new BeanToWrite( 1234, "hello");

JXPathContext ctxt = JXPathContext.newContext( bean);

Integer value = (Integer)ctxt.getValue( "/integerValue");

Add a note hereAdd a note hereIn [Listing 7.52](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), the class JXPathContext exposes the static method newContext, which is used to generate a query tree. The first parameter of the method newContext is a reference to the Java Bean instance that will be queried. The query tree based on the Java Bean instance is stored in the variable ctxt. Then, to query the Java Bean instance, the method getValue is used. The method getValue has one parameter, which is the XPath statement. In [Listing 7.52](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), the XPath statement is used to find the value of the property integerValue. Let's look to [Listings 7.48](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) and 7.49; there, the property integerValue is mapped as a child element for the XPath parser. The return value from the method getValue is an object that represents what is being searched. In the case of the XPath query, it is safe to cast to the class Integer because the property integerValue is an integer value. Otherwise, the cast might cause an exception.

Add a note hereAdd a note here[Listing 7.53](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) is a bit more complex in that the class ParentBean is used as the basis of a query.

Add a note hereAdd a note here**Listing 7.53**

Add a note hereAdd a note hereParentBean bean = new ParentBean();

JXPathContext ctxt = JXPathContext.newContext( bean);

Integer value = (Integer)ctxt.getValue(

"/myReferenceToAnotherBean/integerValue");

Add a note hereAdd a note hereIn [Listing 7.53](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), the same code as in [Listing 7.52](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) is executed, except that the XPath query is referencing the properties myReferenceToAnotherBean.integerValue. If you look back at [Listing 7.51](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), the same value could be returned using a property reference notation, as shown in [Listing 7.54](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577).

Add a note hereAdd a note here**Listing 7.54**

Add a note hereAdd a note hereParentBean bean;

int value =

bean.getMyReferenceToAnotherBean().getIntegerValue();

Add a note hereAdd a note hereXPath, for those of you who do not know anything about it, is a querying language that can be frustrating at times because of its syntax. The simplest way to understand XPath is to consider an XPath expression like a directory location. For example, [Listing 7.55](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) is a command to move up the directory.

Add a note hereAdd a note here**Listing 7.55**

Add a note hereAdd a note herecd ..

Add a note hereAdd a note hereAnother example, shown in [Listing 7.56](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), is a command to navigate into a subdirectory.

Add a note hereAdd a note here**Listing 7.56**

Add a note hereAdd a note herecd items

Add a note hereAdd a note hereUsing XPath is just like navigating a directory, except that XPath has more sophisticated navigation commands. Each time you try to navigate an XPath query, the context or location is altered. The difference in using XPath is that you do not need to use the cd command when navigating directories. [Listing 7.57](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) shows how complicated XPath can get.

Add a note hereAdd a note here**Listing 7.57**

Add a note hereAdd a note herechild::elements[ child::items != "me"]

Add a note hereAdd a note hereIn [Listing 7.57](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), the XPath is saying to make sure that all XML child element items are not equal to the value of dumdum. [Listing 7.57](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) is an example of using concepts such as axis and predicates.

### Add a note hereAdd a note hereNavigating the Hierarchy

Add a note hereAdd a note hereWhen you reference nodes in an XPath query, the position in the hierarchy changes. In XPath, you typically navigate by using one of the notations shown in [Listing 7.58](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577).

Add a note hereAdd a note here**Listing 7.58**

Add a note hereAdd a note here\* or /item or

/\*[ 1] or //item[ 1]

Add a note hereAdd a note hereThe XPath notation examples defined in [Listing 7.58](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) are called abbreviated XPath syntax. The abbreviated XPath syntax notations shown are explained as follows:

* Add a note hereAdd a note here**\***: This matches all subelements specified at the current context.
* Add a note hereAdd a note here**/**: This selects a new context. In [Listing 7.52](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) and 7.53, the slash was used to select both the root context and the child context. If the XPath query does not start with a slash, this implies that the current context is selected. In the case of [Listing 7.52](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) and 7.53, this would mean the root context.
* Add a note hereAdd a note here**//**: This selects all nodes in the current context. In the case of the class ParentBean, this would mean listing all of the properties that the classes ParentBean and BeanToWrite expose.
* Add a note hereAdd a note here**[ ]**: This specifies a predicate from where the thus-far selected nodes are filtered according to the rules indicated. inside the brackets
* Add a note hereAdd a note here**.**: This selects the current context (this is usually not referenced).
* Add a note hereAdd a note here**..**: This selects the parent element instead of changing the location to a subelement.

Add a note hereAdd a note hereInstead of using the abbreviated XPath syntax, [Listing 7.59](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) illustrates standard XPath syntax using axis specifiers.

Add a note hereAdd a note here**Listing 7.59**

Add a note hereAdd a note herechild::\* or child::item or

child::\*[ 1] or descendant::item[ 1]

Add a note hereAdd a note hereIn XPath terms, the axis specifier is the identifier before the double colon in [Listing 7.59](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577). Like in the abbreviated syntax specification, the wild cards and predicates still apply. The full axis specifiers are defined as follows:

* Add a note hereAdd a note here**child**: This references all of the children of the current context.
* Add a note hereAdd a note here**descendant**: This references all of the children of the current context (except children of children are included).
* Add a note hereAdd a note here**parent**: This references the parent of the current context.
* Add a note hereAdd a note here**ancestor**: This references the parents of the current context and which includes the parent of the parent.
* Add a note hereAdd a note here**ancestor-or-self**: This references the current context and the parents of the current context, which includes the parent of the parent.
* Add a note hereAdd a note here**following-sibling**: This references all of the XML nodes that follow the current context.
* Add a note hereAdd a note here**preceding-sibling**: This references all of the XML nodes that precede current context.
* Add a note hereAdd a note here**following**: This references all of the XML nodes after the current context.
* Add a note hereAdd a note here**preceding**: This references all of the XML nodes before the current context.
* Add a note hereAdd a note here**attribute**: This references a specific attribute from the current context.
* Add a note hereAdd a note here**namespace**: This contains the namespace nodes of the current context.
* Add a note hereAdd a note here**self**: This references the current context.
* Add a note hereAdd a note here**descendant-or-self**: This references the current context and all descendants.
* Add a note hereAdd a note here**ancestor-or-self**: This references the current context and all ancestors.

Add a note hereAdd a note hereThe XPath queries illustrated in [Listing 7.53](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) and 7.54 could be rewritten as shown in [Listing 7.60](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577).

Add a note hereAdd a note here**Listing 7.60**

Add a note hereAdd a note here/child::integerValue

/child::myReferenceToAnotherBean/child::integerValue

### Add a note hereAdd a note hereDefining the Classes to Be Searched

Add a note hereAdd a note hereThe XPath searches referenced here are based on [Listing 7.61](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), which are a number of classes that reference other classes.

|  |  |  |
| --- | --- | --- |
|  | **Note** | Add a note hereAdd a note hereDon't worry about the naming convention too much. The classes Data, Child, Elements, and Sub are not logical identifiers in usage terms. They are named as such to illustrate specific XPath constructs. |

Add a note hereAdd a note here**Listing 7.61**

Add a note hereAdd a note herepublic class Data {

private Child \_child;

private Elements \_elements;

private String \_unique;

public Data() {

\_child = new Child();

\_elements = new Elements();

\_unique = "a unique string property";

}

public Child getChild() {

return \_child;

}

public Elements getElements() {

return \_elements;

}

public String getUnique() {

return \_unique;

}

}

public class Child {

private String \_elements;

public Child() {

\_elements = "a string in Child";

}

public String getElements() {

return \_elements;

}

}

public class Elements {

private String \_elements;

private Sub \_sub;

public Elements() {

\_elements = "a string in Elements";

\_sub = new Sub();

}

public String getElements() {

return \_elements;

}

public Sub getSub() {

return \_sub;

}

}

public class Sub {

private String \_elements;

public Sub() {

\_elements = "a string in Sub";

}

public String getElements() {

return \_elements;

}

}

Add a note hereAdd a note hereBased on [Listing 7.61](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), a generic XPath query and display routine are shown in [Listing 7.62](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577).

Larger View

Add a note hereAdd a note here**Listing 7.62**

Add a note hereAdd a note here private void iterate( JXPathContext ctxt, String xpath) {

Iterator iter = ctxt.iterate( xpath);

while( iter.hasNext()) {

Object obj = iter.next();

System.out.println( "Object is " + obj.toString());

}

}

Add a note hereAdd a note hereIn [Listing 7.62](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), the JXPathContext class instance would contain a parse tree of an instance of the class Data from [Listing 7.61](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577). An XPath query is executed using the method iterate. However, a single object is not returned from the method; rather, an instance of an iterator is returned. The iterator can be used to go through the collection and inspect each individual found element.

Add a note hereAdd a note hereThe next set of examples will show a query and will display the output from [Listing 7.62](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) in one listing context. [Listing 7.63](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) is a simple example.

Add a note hereAdd a note here**Listing 7.63**

Add a note hereAdd a note hereQuery: child::\*

Object is com.devspace.jseng.collections.Child@89cf1e

Object is com.devspace.jseng.collections.Elements@982589

Object is a unique string property

Add a note hereAdd a note hereIn [Listing 7.63](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), the query is a simple iteration of all the child elements of the class Data. Notice that the results contain a reference to the object exposed by the properties. The last output is a text and not an object reference because the object is a string. The same result would be returned if the XPath queries were changed to the various combinations shown in [Listing 7.64](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577).

Add a note hereAdd a note here**Listing 7.64**

Add a note hereAdd a note here/child::\*

Add a note hereAdd a note here[Listing 7.65](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) illustrates a query that iterates the child nodes and then finds specific child properties.

Add a note hereAdd a note here**Listing 7.65**

Add a note hereAdd a note hereQuery: /child::\*/child::elements

Object is a string in Child

Object is a string in Elements

Add a note hereAdd a note hereThe XPath illustrated in [Listing 7.65](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) is drill-down example query. The first part of the query (between the two slashes) instructs the XPath query processor to generate a collection of all properties from the class Data. Looking back to the results of [Listing 7.63](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), this would mean that the collection has three elements. Then, the second part of the query (after the second slash) instructs the XPath query processor to retrieve all properties that are identified by the name elements. The result would be two string objects. It is important to remember that the XPath always returns a reference to an object that represents the value of the property. This means that if an object is returned, it is not possible to know who the parent or owner of the object is, since it was picked from the hierarchy.

Add a note hereAdd a note here[Listing 7.66](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) shows a similar query to [Listing 7.65](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577).

Add a note hereAdd a note here**Listing 7.66**

Add a note hereAdd a note hereQuery: /child::\*/descendant::elements

Object is a string in Child

Object is a string in Elements

Object is a string in Sub

Add a note hereAdd a note hereThe queries in [Listing 7.65](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) and 7.66 look similar, but there is a big difference in selection. The axis specifier child in [Listing 7.65](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) says to search only the objects in the collection. On the other hand, the axis specifier descendant in [Listing 7.66](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) says to search the objects in the collection and the objects that they reference. The descendent specifier is a sort of recursion that says, "In the selected directories, find something including all subdirectories." This accounts for the difference because the class Data references the class Elements, which references the class Sub.

Add a note hereAdd a note here[Listing 7.67](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) is a query where mentally you expect one answer, but in fact another answer is output.

Add a note hereAdd a note here**Listing 7.67**

Add a note hereAdd a note hereQuery: /child::\*/child::\*[1]

Object is a string in Child

Object is a string in Elements

Object is 97

Add a note hereAdd a note hereIn [Listing 7.67](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), the query says, for the main class Data select all properties, which resulted in the output of [Listing 7.63](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577). Then, from that collection return the first element, which, according to [Listing 7.64](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), would be the properties of the class Child. [Listing 7.67](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) outputs the desired properties of the class Child, but it also outputs some other pieces of data because the XPath does not say what is requested. The XPath says, get all of the child properties and from those properties retrieve the first property of each element. The number 97 would seem a bit out of place, but it is the first property of the class String. [Listing 7.68](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) is the query that was originally requested.

Add a note hereAdd a note here**Listing 7.68**

Add a note hereAdd a note hereQuery: /child::child/child::\*[1]

Object is a string in Child

Add a note hereAdd a note hereAnother interesting query is shown in [Listing 7.69](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577). This shows how to select one set of properties, but display the values that follow the selection.

Add a note hereAdd a note here**Listing 7.69**

Add a note hereAdd a note hereQuery: /child::child/following::elements

Object is a string in Elements

Object is a string in Sub

Object is com.devspace.jseng.collections.Elements@982589

Add a note hereAdd a note hereIn [Listing 7.69](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), the query selects the class Child, but the axis specifier following says select the properties following the class Child. That would then select the property elements from the classes Data, Element, and Sub. The axis specifiers descendant, preceding, and following are powerful specifiers used to select properties located on another property. If you're unsure what each specifier selects, experiment and go back to the [*Navigating the Hierarchy*](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) section for details.

### Add a note hereAdd a note hereFiguring Out the Location

Add a note hereAdd a note hereOne of the problems with XPath is that it's difficult to figure out what the parent node of a result is. Using the XML DOM, this is not that difficult because each resulting node has references to its parent, child, and siblings. To get around this problem, the *JXPath* package has exposed the notion of pointers. *Pointers* are XPath references that specify the location of a found element. Using [Listing 7.61](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) as a basis, [Listing 7.70](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) shows how to find the paths of found element properties.

Add a note hereAdd a note here**Listing 7.70**

Add a note hereAdd a note hereData bean = new Data();

JXPathContext ctxt = JXPathContext.newContext( bean);

Pointer ptr = ctxt.getPointer( "//elements");

System.out.println( "Pointer: " + ptr.asPath());

Iterator iterPtr = ctxt.iteratePointers( "//elements");

while( iterPtr.hasNext()) {

Object obj = iterPtr.next();

System.out.println( "Pointer is " + obj.toString());

}

Add a note hereAdd a note hereIn [Listing 7.70](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), two methods are used to retrieve the found references from the XPath query. The method getPointer retrieves an individual reference regardless of the number of references that actually exist. If no references exist, then a null is returned. If one or more references exist, then the first reference in the collection is returned. The returned value is a Pointer class instance. Using the method asPath, the textual value of the XPath reference is retrieved. If there are multiple found references, an iterator is retrieved using the method iteratorPointers. Using the iterator, you can manually go through each found XPath reference.

### Add a note hereAdd a note hereCollections and XPath Referencing

Add a note hereAdd a note hereAll of the *JXPath* package sample queries shown used a simple notation where the object hierarchy was completely composed of single objects referencing other single objects. This is not the typical case; more likely, the object hierarchies will include references to arrays and collections. When using *JXPath* to query object hierarchies that include arrays and collections, you don't need to change the program structure. It is only necessary to change the query as shown in [Listing 7.71](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577).

Add a note hereAdd a note here**Listing 7.71**

Add a note hereAdd a note hereBeanProperties bean = new BeanProperties();

JXPathContext ctxt = JXPathContext.newContext( bean);

Object item = ctxt.getValue("/child::beanArray[ 1]");

Add a note hereAdd a note hereIn [Listing 7.71](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), the array property beanArray is referenced using the array predicate in the XPath expression. The XPath expression will return only one object, and that is assigned to the variable item. *JXPath* supports the types Vector, Collection, List, and Set.

### Add a note hereAdd a note hereAssigning Properties

Add a note hereAdd a note hereYou can also assign properties using *JXPath*. The code used to assign a property is similar to that for getting the properties. [Listing 7.72](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) shows how to assign a property that was retrieved in the [Listing 7.53](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577).

Add a note hereAdd a note here**Listing 7.72**

Add a note hereAdd a note hereParentBean bean = new ParentBean();

JXPathContext ctxt = JXPathContext.newContext( bean);

ctxt.setValue( "/myReferenceToAnotherBean/integerValue",

new Integer( 4567));

Add a note hereAdd a note hereIn [Listing 7.72](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), the property is assigned using the method setValue, which has two parameters. The first parameter is the XPath query used to find the individual properties. The second parameter identifies the object that will be assigned to the properties found. This second parameter is part of a potential problem. If an XPath query finds multiple items, then each of those found items will be assigned the same object. The problem with this is that the program might assume that each assigned property has a unique value. In the case of [Listing 7.72](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), this is not a problem because the property is a primitive.

### Add a note hereAdd a note hereCompiling an XPath Expression

Add a note hereAdd a note hereIn all of the examples shown thus far, the XPath query was dynamically compiled. It is possible with *JXPath* to precompile the expression for use at a later time. [Listing 7.73](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577) shows how to create and use a precompiled XPath query.

Add a note hereAdd a note here**Listing 7.73**

Add a note hereAdd a note hereData bean = new Data();

JXPathContext ctxt = JXPathContext.newContext( bean);

CompiledExpression compiled = ctxt.compile( "//elements");

Iterator iter = compiled.iterate( ctxt);

Add a note hereAdd a note hereIn [Listing 7.73](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=813739577), the XPath is compiled using the method compile. The returned CompiledExpression class instance exposes the same method as the class JXPathContext. The difference, though, is that the parse tree is already compiled. To execute a query, you need to associate the compiled expression with a bean context. (Note that what was compiled was not the bean context, but the XPath expression.) The method iterate creates an iterator that contains the found items.

## **Summary**

Add a note hereAdd a note hereThis chapter tackled the topic of complex object manipulation. This is an important topic because of the code that business programmers will create. Business programmers, or application programmers, write applications that solve specific business purposes. Often, that means developing large and complex data structures. Ideally, a developer would develop a simple object structure. However, when you're dealing with business applications, that is simply not possible. Business applications have large amounts of data, and that means large objects. If you hadn't read all the way up to this chapter, this would mean that as a developer, you would have to manually iterate and search objects.

Add a note hereAdd a note hereWe learned in this chapter that you can dynamically manipulate those objects. The advantage of dynamic manipulation is that the objects could change, but you wouldn't have to change the business application code. For example, the *Beanutils* package can clone two entirely different class types. When you move from a legacy system to a new system, the objects will change but the identifiers most likely will not. This means that a developer can code logic without having to remember the types. The same could be said for the *JXPath* package, which allows you to dynamically resolve properties and their values. These two packages solve many problems related to manipulating complex data structures.

Add a note hereAdd a note hereWe also discussed the *Collections* package, which rounds out the complex data. This package offers very sophisticated collection structures. Collections are a complex topic because they need to hold objects that obey some rules. Typically, those rules are programmed on top of the collection. However, for the *Collections* package those rules are part of the collection, which allows a developer to use a collection in any context without having to know the rules. The rules and circumstances are defined externally to the collection manipulation. This follows the Commons Bridge intention.

Add a note hereAdd a note here**On the CD** The sources to the concepts presented in this chapter are located under the directory *[CDROM]/jseng/src/com.devspace.jseng.collections*.

**Questions**

Add a note hereAdd a note here**7.1:** Explain how a dynamic scripting language could be implemented using the *Beanutils* package.

Add a note hereAdd a note here**7.2:** The batch file extension used to add or manipulate single lined numbers is to be discarded and replaced with a simple single command calculator language. The RPN notation will still exist, but the commands used by the RPN notation will be textual and still operate using the stack principle. The objective is to rewrite the calculator so all commands are dynamic and no command is "hard-coded." Each command consists of key value pairs (e.g., ADD 1234). For the beginning, support the following commands (note that all commands should be stored and read using XML serialization):

Add a note hereAdd a note hereADD: Add the number from the stack.

Add a note hereAdd a note hereSUBTRACT: Subtract the number from the stack.

Add a note hereAdd a note hereMULTIPLY: Multiply the number from the stack.

Add a note hereAdd a note hereDIVIDE: Divide the number from the stack.

Add a note hereAdd a note hereSTORE: Store the stack value at the identifier specified by the key.

Add a note hereAdd a note hereRETRIEVE: Retrieve the value specified by the stack and push it onto the stack.

Add a note hereAdd a note hereCLEAR: Clear the stack of all values if the value is –1; otherwise, only clear as many stack positions as indicated by the number.

Add a note hereAdd a note here**7.3:** Add the instructions to the calculator defined as follows:

Add a note hereAdd a note hereBOOKMARKSTART: Defines a starting point for a bookmark.

Add a note hereAdd a note hereBOOKMARKEND: Defines an ending point for a bookmark.

Add a note hereAdd a note hereSEARCH (e.g., SEARCH COMMANDS\HISTORY\ADD, COMMANDS\ BOOKMARK\SOMETHING): The exact search path used depends on the locally defined object hierarchy. The objective with the SEARCH command is to be able to search a single or multiple commands and then execute those commands in the calculator. Consider this command as a sort of script filter used to execute commands.

**Chapter 8: Configuration and Logging**

**Add a note hereAdd a note here****Overview**

Add a note hereAdd a note hereIn this chapter, we will do the following:

* Add a note hereAdd a note hereLearn how to manage the command line effectively
* Add a note hereAdd a note hereLearn how to write a configuration file
* Add a note hereAdd a note hereWrite applications that use configuration files
* Add a note hereAdd a note hereLog meaningful messages

## **The Purpose of This Chapter**

Add a note hereAdd a note hereThere are two kinds of applications: end-user applications and server or custom applications. End-user applications are applications that people use to solve a specific task in their job. Examples include a word processor or spreadsheet. Although this chapter does not contain solutions that would be of general interest to end-user client applications, it does not mean that some parts of it cannot be used.

Add a note hereAdd a note hereThe other kind of application is a server or custom application. These kinds of applications might be written in-house or by a software vendor. Regardless, these kinds of applications are used by people who are trying to solve business problems, but are managed by people called administrators. Administrators are professionals who might not program, but are very well versed in computer software.

Add a note hereAdd a note hereIn the server and custom application world, administrators have certain needs that the application's programmer often has not fulfilled. A programmer will often write an application and instruct the administrator to install it and distribute it. The problem often is that there might be problems that have to be diagnosed. Perhaps the administrator cannot diagnose the problem because the application does not have log files, or maybe it cannot be installed in the manner desired by the administrator. In the end, it is like having the administrator trying to fix a car when the hood is welded shut. The solution to this dilemma is for the developer to add certain features that make an administrator's life simpler. The task in this chapter, therefore, is how to make an application administrator friendly.

## **Understanding the Administrator**

Add a note hereAdd a note hereAdministrators are people who tend to be picky, or at least they're picky if they do their job well. Being detail-oriented is part of the administrator's job description because his task is to keep things running smoothly. In essence, a good administrator makes his job look easy. The reality, however, is that being a good administrator is hard. And developers do not make the administrator's job any easier. Many developers consider that an installation manual and an install file are good enough for an administrator. However, the installation manual will typically be half-written and will often be inaccurate. The result is that the administrator has to learn about the system. Until he learns it, the administrator will not be keen on letting more pieces of the application be installed. As a result, conflicts and disasters between the developers and administrator will inevitably happen.

Add a note hereAdd a note hereNow there is an irony. Open Source, which is complex, command line-oriented, and compile-it-yourself, is preferable to Closed Source GUI-based applications. The reason why is simple. The program RedHat™, which is a Linux distribution with Release 8.0, for example, made huge strides in ease of use as compared to previous versions of RedHat. The newest release of RedHat is very GUI friendly and allows a normal user to use Linux without having to know all of the configuration files and configuration programs. However, an administrator would reply to this information about RedHat by saying, "That sounds good, but it's not for me." Administrators are generally not interested in GUIs because GUIs are useful only when a single person is involved. When administrators are responsible for about 100 desktops, GUIs make it extremely difficult to manage the configurations on the different machines. As a result, administrators generally prefer using the command line and scripts, where everything can be automated. So, while learning the command line and script is initially more complicated than learning a GUI, in the long run, using the command line is simpler.

Add a note hereAdd a note hereOpen Source administrators are not programmers, even though they might write some shell scripts. However, Open Source administrators can configure, tweak, and debug applications because Open Source programs are tuned for that purpose. The programs are tuned towards helping the administrator do their job.

### Add a note hereAdd a note hereA Checklist of What an Administrator Wants

Add a note hereAdd a note hereAn administrator's wish list is simple: give me something that works. That is a typical and very vague response. However, with some prodding, you can get a more accurate list of things that administrators want in a program. They are:

1. Add a note hereAdd a note here**Good documentation**: The administrator does not just want documentation that explains the terms like a dictionary; he wants a cookbook. Included in this category is documentation that defines where further help, in terms of a FAQ, mailing list, or support, is available.
2. Add a note hereAdd a note here**Tweaking abilities, using either configuration items or command line options**: The tweaks must also be documented in terms of task-based solutions. For example, the documentation should use the notation "To solve problem X, use option Y," and not "Option Y is used to do task X." The difference lies in how the administrator can solve problems; the documentation must describe the situation so the administrator doesn't have to read each option.
3. Add a note hereAdd a note here**A guide to set the program up quickly**: This is for the impatient. The administrator would like to know how to set the program up quickly, although it might not be the ideal configuration, because the administrator may need other settings.
4. Add a note hereAdd a note here**An overview of all the files required in an installation:** The administrator doesn't need to know what the application files do; rather, he needs to know which configuration requires which files.
5. Add a note hereAdd a note here**A modular application**: The administrator needs the application to be modular so that different configurations can be installed and maintained. The modules should be created in layers. An example of a simple and modular application is Apache Web Server. This is popular because non-programmers can set up a Web Server. Modules can be easily added using configuration definitions or compile time options. For an administrator, a reliable system is a simple system that can be understood. That way, if anything goes wrong, the administrator can figure out what is wrong on his own on.
6. Add a note hereAdd a note here**Definition of the build process**: This build process should be kept simple.
7. Add a note hereAdd a note here**Not too many features**: The administrator prefers a minimalist approach here.

Add a note hereAdd a note hereAdministrators have simple requests. Administrators are not developers, and this has a very important ramification. The administrator does not really care why program A exists, nor what business need it solves. It is as if you took your car to the mechanic and explained to him that the car is used to drive along a road with five curves and two stop signs. The mechanic wants to know only what noises are made in what context. Likewise, the administrator wants to know only when the application is needed and in which context. This is a hard thing for developers to swallow. Developers take pride and joy in watching the program run, and they want to talk about how well it works and solves the business process. The administrator could not care less, since in his eyes, it's just another application to keep running.

### Add a note hereAdd a note herePlug-ins Equal a Happy Administrator

Add a note hereAdd a note hereWhen you write programs that use configuration items, you will most likely include the ability to dynamically load plug-ins. Plug-ins are formalized Commons Bridge implementations. The main difference between a plug-in and a Commons Bridge implementation lies in the distribution. A plug-in implements an interface defined in another package. The implementation of the interface is stored in the plug-in package. This means that any application contains three packages: application, plug-in definitions, and plug-in implementations. To realize such a solution, you can use the *Discovery* package, which we discussed in [Chapter 3](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=164#164). We didn't, however, address the details of implementing a plug-in framework. A sample plug-in framework is shown in [Listing 8.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=329302239).

Larger View

Add a note hereAdd a note here**Listing 8.1**

Add a note hereAdd a note hereURL url = new URL( Configuration.getClassPath());

URL[] urls = new URL[] { url };

ClassLoader classloader = new URLClassLoader( urls);

ClassLoaders classloaders = new ClassLoaders();

classloaders.put( classloader);

DiscoverClass discoverClass = new DiscoverClass( classloaders);

Properties props = new Properties();

props.setProperty( InterfaceToBeShared.class.getName(),

Configuration.getDefaultImplementation());

InterfaceToBeShared interf = (InterfaceToBeShared)discoverClass.newInstance(

InterfaceToBeShared.class, props);

interf.availability();

Add a note hereAdd a note hereThe code from [Listing 8.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=329302239) is similar to code shown in [Chapter 3](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=164#164). The small differences involve the use of a class loader and a configuration class. The code in [Listing 8.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=329302239) would be located in the application package. It would be assumed that the plug-in definitions package would be referenced statically on the class path, like the application package. What is dynamically referenced and loaded is the plug-in implementation package. The class Configuration contains the information on what classes and paths the plug-in implementation package requires.

Add a note hereAdd a note hereThe class Configuration exposes two static methods: getClassPath and getDefaultImplementation. The method getClassPath returns a path or jar filename that contains the plug-in implementation package. To load the plug-in implementation package, you use the class ClassLoader. The class ClassLoader is part of the Java runtime and allows a developer to dynamically load Java classes on the fly. To make use of the class ClassLoader in the *Discovery* package, you add the class ClassLoader to the class ClassLoaders. This, in turn, is added to the class DiscoverClass.

Add a note hereAdd a note hereThe method getDefaultImplementation returns a class name that implements the plug-in definitions package. It is added to the class Properties, which is then referenced to find a default implementation for the plug-in. In [Listing 8.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=329302239), the plug-in definition is the interface InterfaceToBeShared.

### Add a note hereAdd a note hereWhat Should Be Configured

Add a note hereAdd a note here[Listing 8.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=329302239) contains the basics of what is needed to implement a plug-in architecture. However, a good application is not only about converting an application into a plug-in architecture. A good application is an application that can be tuned and managed. The following are items that you can tune:

* Add a note hereAdd a note here**Database connection**: This specifies who the provider is, as well as what connection settings are necessary to connect to the database.
* Add a note hereAdd a note here**Pool size**: This is the maximum and minimum size of the objects that are kept in the collection, and the length of time before an object is collected.
* Add a note hereAdd a note here**Threads**: This is how many threads the application will start.
* Add a note hereAdd a note here**Factory**: This is a dynamic specification of the objects that will be instantiated.
* Add a note hereAdd a note here**Serialization**: This specifies where the files will be saved and how the objects will be read and written.
* Add a note hereAdd a note here**Messaging**: This specifies which objects will receive messages, as well as which queues and topics are used.
* Add a note hereAdd a note here**Collections**: This is the type of collection used and the maximum and minimum size of the collections.
* Add a note hereAdd a note here**Logging**: This defines the levels of errors and warnings that need to be logged and written to some storage device.

Add a note hereAdd a note hereThe list presented is only a small example of what could be configured. If you consider Open Source programs like Apache HTTP Server or Jakarta Tomcat Server, it is very obvious that many additional items can be configured.

Add a note hereAdd a note hereSo, now comes the question that all developers need to ask themselves: what things should be configurable? Some administrators say that everything should be, so that the application is as flexible as possible. Other administrators say that everything that has affects how the application acts should be configurable. For example, allowing only three threads has a different reaction than allowing as many threads as necessary. Finally, other administrators say that developers should sit down with the administrator and ask him what he wants to be able to do.

Add a note hereAdd a note hereIn this last case, if the developer sits down with the administrator, he can extract the administrator's requirements, which are not user requirements. Administrator requirements are related to how to make the application run as efficiently as possible on a given set of hardware. The administrator wants to control the application, and hence those controls are specific to the administrator. The developer of the application may not know what the controls of the administrator are if he doesn't sit down with the administrator.

### Add a note hereAdd a note hereCommand Line Management

Add a note hereAdd a note hereMany people have a hard time with command line parameters. To many people, using the command line is nonintuitive, complex, and downright confusing. This may be true. Command line parameters are not intuitive and they are hard to learn. However, command line parameters are still necessary. Command line parameters are used to bootstrap a process.

Add a note hereAdd a note hereFor example, when a process starts, a configuration file needs to be read. However, how does the process know where the configuration file is? Some operating systems use strategies like managing a registry of configuration settings. The process uses method calls to hook into the registry and retrieve the necessary configuration items. A registry is OK, but the problem with registries is that it is extremely difficult to manage multiple configurations. A registry is a central location for all items, which also makes it a central place for failure. Time has shown that there are advantages and disadvantages to both a registry and configuration files.

Add a note hereAdd a note hereIf a registry is not available, then a command line parameter can reference the location of the configuration file. A command line parameter can also define certain items relating to how the application should execute. Overall, a command line parameter should not replace a configuration file but complement it.

### Add a note hereAdd a note hereTechnical Details for the *cli* Package

Add a note hereAdd a note here[Tables 8.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=329302239) and [8.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=329302239) contain the abbreviated details necessary to use the *cli* package.

**Add a note hereAdd a note hereTable 8.1:** Repository details for the *cli* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note herecli |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.cli |

**Add a note hereAdd a note hereTable 8.2:** Package and class details (legend: [cli] = org.apache.commons.cli).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[cli].Options | Add a note hereAdd a note hereA class that contains a collection of acceptable command line options represented by the class Option. |
| Add a note hereAdd a note here[cli].Option | Add a note hereAdd a note hereA class that defines an option. This class is a core class that defines an individual command line option and is used to fine-tune the definition. |
| Add a note hereAdd a note here[cli].CommandLineParser | Add a note hereAdd a note hereAn interface used to define a specific type of command line parser. Available implementations are BasicParser, GnuParser, and PosixParser. |
| Add a note hereAdd a note here[cli].CommandLine | Add a note hereAdd a note hereA class that is returned when the parser has processed the command line arguments. Contained with the class are the processed options and arguments. |
| Add a note hereAdd a note here[cli].OptionBuilder | Add a note hereAdd a note hereA helper class to assist creating an Option class instance. |
| Add a note hereAdd a note here[cli].HelpFormatter | Add a note hereAdd a note hereA class used to output a help message that defines the available options and command line usage. |
| Add a note hereAdd a note here[cli].PatternOptionBuilder | Add a note hereAdd a note hereA class used to retrieve available supported class types for type-specific arguments. |

## **A Command Line Parameter Example for Referencing Configuration Files**

Add a note hereAdd a note hereIn the simplest case of a command line argument, you define a reference to a configuration file. For example, [Listing 8.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) could be a prototype command line argument.

Add a note hereAdd a note here**Listing 8.2**

Add a note hereAdd a note herejava com.devspace.jseng.configuration.Main -l somefile.txt

Add a note hereAdd a note hereIn [Listing 8.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), there are two command line parameters: -l and somefile.txt. The two command line parameters will be passed to the Java program. [Listing 8.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) is an implementation of how to process the arguments.

Larger View

Add a note hereAdd a note here**Listing 8.3**

Add a note hereAdd a note hereOptions options = new Options();

options.addOption( "l", "location", true, "location of configuration file");

CommandLineParser parser = new BasicParser();

CommandLine cmd = parser.parse( options, args);

System.out.println( "location of the file is (" +

cmd.getOptionValue( "l") + ")");

Add a note hereAdd a note hereIn [Listing 8.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), three basic classes are needed so that you can use the *cli* package: Options, CommandLineParser, and CommandLine. The class Options contains a collection of defined options. In the case of [Listing 8.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), one required option would be the option -l. You define an option by using the method addOption. There is a difference between the definition of the option -l in [Listing 8.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) and its actual usage in [Listing 8.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512). The reason for this difference is that the minus sign is not part of the option definition. The command line parser assumes that the minus sign indicates that an option is about to be defined. Hence, in the definition of the option there is no need to explicitly define the minus sign.

Add a note hereAdd a note hereThe method addOption has four parameters:

* Add a note hereAdd a note here**l**: Defines the short form of the option
* Add a note hereAdd a note here**location**: Defines the long form of the option
* Add a note hereAdd a note here**true**: Specifies if there is an associated value with the option
* Add a note hereAdd a note here**"location of configuration file"**: Defines a textual description of the option

Add a note hereAdd a note hereIn [Listing 8.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), the class BasicParser is an implementation of the interface CommandLineParser. You can use several different implementations to parse the command line. (We will give more details about the different parsers in the [next section](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=660#660) of this chapter.) The interface method CommandLineParser.parse is used to combine the parser with the options and then to parse the two parts. Returned is a CommandLine class instance. If there are any problems, such as an option that has no declaration, an exception is thrown. The class CommandLine contains the parsed parameters. To retrieve a parameter, the method getOptionValue is used.

### Add a note hereAdd a note hereExplaining Command Line Parser Implementations

Add a note hereAdd a note hereThere are three different command line parsers: BasicParser, GnuParser, and PosixParser. All three deal differently with how specific versions of command lines are parsed. If you look at the Java Docs of the *cli* package, you will see the term bursttoken. However, the Java Docs explanations, while correct, are written using a specification-type language. [Table 8.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) shows some command line examples, and how they are handled by each of the three parsers (BasicParser, GnuParser, and PosixParser).

**Add a note hereAdd a note hereTable 8.3:** Command line option syntax and whether or not it can be parsed by a specific parser (**BasicParser**, **GnuParser**, and **PosixParser**).

| **Add a note hereAdd a note hereExample** | **Add a note hereAdd a note hereBasicParser** | **Add a note hereAdd a note hereGnuParser** | **Add a note hereAdd a note herePosixParser** |
| --- | --- | --- | --- |
| Add a note hereAdd a note here-l | Add a note hereAdd a note hereWorks | Add a note hereAdd a note hereWorks | Add a note hereAdd a note hereWorks |
| Add a note hereAdd a note here--location | Add a note hereAdd a note hereWorks | Add a note hereAdd a note hereWorks | Add a note hereAdd a note hereWorks |
| Add a note hereAdd a note here-l=file.txt | Add a note hereAdd a note hereError | Add a note hereAdd a note hereWorks, but gives an incorrect value | Add a note hereAdd a note hereWorks, but gives an incorrect value |
| Add a note hereAdd a note here--location=file.txt | Add a note hereAdd a note hereError | Add a note hereAdd a note hereError | Add a note hereAdd a note hereWorks |
| Add a note hereAdd a note here-lfile.txt | Add a note hereAdd a note hereError | Add a note hereAdd a note hereWorks | Add a note hereAdd a note hereWorks |

### Add a note hereAdd a note hereDetailed Option Definitions

Add a note hereAdd a note hereA typical command line will appear like [Listing 8.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512). [Listing 8.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) shows a more complicated one.

Add a note hereAdd a note here**Listing 8.4**

Add a note hereAdd a note herejava Main -l somefile.txt -f -d value1 value2 "left over"

Add a note hereAdd a note hereIn [Listing 8.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), the command line has more command line options and different variations than it did in [Listing 8.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512). (For the moment, disregard how you could rewrite an individual option as per the examples shown in [Table 8.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512).) [Listing 8.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) is not clear on what the individual options are. All this is clarified in [Listing 8.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), where the square brackets are used to separate the individual options.

Larger View

Add a note hereAdd a note here**Listing 8.5**

Add a note hereAdd a note herejava Main [-l somefile.txt] [-f] [-d value1 value2] ["left over"]

Add a note hereAdd a note hereIn [Listing 8.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), there are three options and some leftover arguments. The first option is a simple option. The second option has no associated value and is optional. The third option has two associated values. Finally, there are some leftover arguments that have no associated option. To be able to parse the three options, you can't use the method addOption as illustrated in [Listing 8.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512). Instead, you need to use the class Option. [Listing 8.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) can parse the command line in [Listing 8.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512).

Larger View

Add a note hereAdd a note here**Listing 8.6**

Add a note hereAdd a note here String args[] = generateMultipleArguments();

Options options = new Options();

options.addOption( "l", "location", true, "location of configuration file");

Option option = new Option( "f", "flag", false,

"this is a flag");

option.setOptionalArg( true);

options.addOption( option);

option = new Option( "d", "define", true,

"definition declaration");

option.setArgs( 2);

options.addOption( option);

CommandLineParser parser = new BasicParser();

CommandLine cmd = parser.parse( options, args);

String results[] = cmd.getOptionValues( "d");

if( results != null) {

for( int c1 = 0; c1 < results.length; c1 ++) {

System.out.println( "Option value (" + results[ c1] + ")");

}

}

List list = cmd.getArgList();

if( list != null) {

Iterator iter = list.iterator();

while( iter.hasNext()) {

System.out.println( "Left over: " + iter.next().toString());

}

}

Add a note hereAdd a note hereIn [Listing 8.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), the first three blocks of code are used to define the three different options. The first block defines an option using the addOption method. The second block still uses the addOption method, but instead of being passed four parameters, it's passed only one parameter. The single parameter is an Option class instance. When defining an option using the class Option, you can fine-tune the definition of the option. The second option is optional, and you can define that using the class method Option.setOptionalArg only. When you create the Option class instance, the constructor options are identical in purpose to the arguments of the previous block's addOption method. The third option has two associated arguments, which are assigned in the third block using the method setArgs. And, as with the second block, the newly created option is added to the list of possible options using the method addOption.

Add a note hereAdd a note hereOnce the options have been added, you can parse the command line arguments using the parse method. This time, instead of retrieving a single option argument, you need to retrieve a list of arguments using the method getOptionValues, with the associated command line argument. The method returns an array of string values. If the option has only one associated value, an array of length one is returned. If the option does not exist, a null is returned and not an array of length zero.

Add a note hereAdd a note hereIn [Listings 8.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) and 8.6, the method getOptionValue(s) had only one argument, which was the specific command line argument. In each of the method calls, the short form of the command line argument was used. You can also use the long form of the command line argument. The class CommandLine will automatically retrieve the associated short form. It's up to the programmer to decide which form to use; however, the long form might be easier to maintain because the long form is explicit and does not leave the programmer guessing what that command line argument does.

Add a note hereAdd a note hereIn [Listing 8.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), we used the method getArgList to retrieve the leftover arguments that have no associated options. Returned is a List class instance. If there are no leftover arguments, then the List is a null reference. To iterate the individual arguments, an Iterator class instance is used.

Add a note hereAdd a note hereAnother way to create the options defined in [Listing 8.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) is to use the class OptionBuilder. Using this class has a catch, however: there is a static class instance definition within the class OptionBuilder. This will cause problems if there are multiple program instances or multiple threads within the same JVM trying to create options. Generally this is not an issue, but remember it just in case. [Listing 8.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) is the same code as [Listing 8.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) except that the options have been created using the class OptionBuilder.

Larger View

Add a note hereAdd a note here**Listing 8.7**

Add a note hereAdd a note here Options options = new Options();

Option option = OptionBuilder.withLongOpt( "location")

.hasArg()

.isRequired()

.withDescription( "location of configuration file" )

.create( 'l' );

options.addOption( option);

option = OptionBuilder.withLongOpt( "flag")

.isRequired( false)

.withDescription( "this is a flag" )

.create( 'f' );

options.addOption( option);

option = OptionBuilder.withLongOpt( "define")

.isRequired( true)

.hasArgs( 2)

.withDescription( "definition declaration" )

.create( 'd' );

options.addOption( option);

Add a note hereAdd a note hereIn [Listing 8.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), a unique feature of the *cli* package is used. The methods are strung together using an object reference notation. This is fine because each OptionBuilder class method returns an instance to OptionBuilder. Each method is used to manipulate an option instance setting. When they are strung together, a valid option is created. It is important to call the method create at the end of the strung-together methods. The method create is the method that instantiates a new Option class instance.

Add a note hereAdd a note hereThe option methods are fairly self-explanatory. What is not self-explanatory is what the defaults are. Consider, for example, the second option block in [Listing 8.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), where the method isRequired with a parameter of false is called. This method call is not necessary since the class OptionBuilder defaults to a value of false. We added it because, for maintenance purposes, it is simpler for the developer to know what is going on. It is assumed that the defaults are ignored and that each characteristic of the option is defined explicitly.

Add a note hereAdd a note here[Listing 8.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) tends to be cleaner and simpler to understand than [Listing 8.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512). You can clean up [Listing 8.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), but even then it is not as self-explanatory as [Listing 8.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512). Therefore, most of the time it is preferable to use the class OptionBuilder instead of instantiating the option classes manually.

### Add a note hereAdd a note hereWhen Something Fails or Help Is Needed

Add a note hereAdd a note hereCommand line options are not self-explanatory, so you will sometimes need help making the application do what is desired. Most applications will output a help message to guide the user on what the possible options are. You typically invoke this by using the -h or —help option. [Listing 8.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) shows how you could rewrite [Listing 8.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) to be more user-friendly.

Larger View

Add a note hereAdd a note here**Listing 8.8**

Add a note hereAdd a note herepublic class MainClass {

private void printHelpMessage( Options options) {

HelpFormatter help = new HelpFormatter();

String cmdLine = "command [OPTIONS] LEFT\_OVER";

String descriptor = "Where [OPTIONS] can be one or more of the following";

String moreHelp = "email:you@somecompany.com for more help";

help.printHelp( cmdLine, descriptor, options, moreHelp);

}

private Options generateOptions() {

Options options = new Options();

Option option = OptionBuilder.withLongOpt( "location")

.hasArg()

.isRequired( false)

.withDescription( "location of configuration file" )

.create( 'l' );

options.addOption( option);

option = OptionBuilder.withLongOpt( "flag")

.isRequired( false)

.withDescription( "this is a flag" )

.create( 'f' );

options.addOption( option);

option = OptionBuilder.withLongOpt( "define")

.isRequired( false)

.hasArgs( 2)

.withDescription( "definition declaration" )

.create( 'd' );

options.addOption( option);

option = OptionBuilder.withLongOpt( "help")

.isRequired( false)

.withDescription( "displays this help message" )

.create( 'h' );

options.addOption( option);

return options;

}

public static void main( String args[]) {

Options options = generateOptions();

try {

CommandLineParser parser = new BasicParser();

CommandLine cmd = parser.parse( options, args);

if( cmd.hasOption( "h") == true) {

printHelpMessage( options);

return;

}

// Validate the arguments...

}

catch( ParseException ex) {

System.out.println( "Error: " + ex.getMessage());

printHelpMessage( options);

}

// Let the program run as normal...

}

}

Add a note hereAdd a note here[Listing 8.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) has been broken apart into three blocks of functionality. Each block is represented by a method. The method main is used to start the program. At the start of the method, the options are created using the method generateOptions. Once the options have been created, the command line arguments can be parsed using the method parse. The next line is a decision that calls the method hasOption to test if the -h or —help option has been called. If help is requested, then the method printHelpMessage is called. Otherwise, the application continues, the arguments are validated, and the program continues.

Add a note hereAdd a note hereThe command line parsing method call is encapsulated within an exception block. The exception block is necessary because it catches any errors that relate to incorrect command line options. The exception to catch is not the generic Exception class, but the specific ParseException class. This ensures that the only exceptions caught by this block are command line parsing exceptions. If an exception occurs, then the cause of the exception is retrieved using the method getMessage. Then, as a helper the correct arguments are output using the method printHelpMessage.

Add a note hereAdd a note hereIf you look at the method generateOptions in [Listing 8.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), you'll notice one change: all options are optional. This can be problematic because it means that the parameters must be checked for correctness. If you didn't check the validity of the options, there would be parsing errors when only the -h or —help flag is available.

Add a note hereAdd a note hereThe method printHelpMessage is a generic help message approach. In fact, we recommend that you use this format. This method makes it simpler to keep track of the various options. To understand the code, look at [Listing 8.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), which is an output of the help message.

Add a note hereAdd a note here**Listing 8.9**

Add a note hereAdd a note hereusage: command [OPTIONS] LEFT\_OVER

Where [OPTIONS] can be one or more of the following

-d,—define <arg> definition declaration

-f,—flag this is a flag

-h,—help displays this help message

-l,—location location of configuration file

email:you@somecompany.com for more help

Add a note hereAdd a note hereIn [Listing 8.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), the first line of text represents the variable cmdLine from [Listing 8.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512). The first line is supposed to be a command line usage of the application. In other examples of this line, many people specify the various options. The problem with this approach is due to maintenance. If a specific option is referenced in the first line, it has to be checked for existence against the method generateOptions. If you use a generic reference [OPTIONS], the class HelpFormatter will generate the actual available options. You don't need to reference correctly the leftover arguments that do not have an available option.

Add a note hereAdd a note hereThe second line of [Listing 8.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) represents the variable descriptor from [Listing 8.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512). The second line represents a header, which is a filler that ties the command line to the generated options. Typically, the second line will indicate how the generated options are used.

Add a note hereAdd a note hereThe third, fourth, fifth, and sixth lines in [Listing 8.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) represent the output of the available options defined in [Listing 8.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512). These lines are automatically generated when the method printHelp in [Listing 8.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) is called. In [Listing 8.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), the method printHelp has four parameters, which represent the command line, header, options, and footer.

Add a note hereAdd a note hereThe last line in [Listing 8.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) represents the variable moreHelp in [Listing 8.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512). The last line could be any information, but we recommend that it be where the developer can find further help if he has no idea what the options mean.

Add a note hereAdd a note hereWe recommend that you follow this approach for parsing command line arguments. If you use this approach we have shown, the correct things will happen. For example, if there are incorrect parameters, the error will be dealt with correctly. In addition, in this approach, the individual blocks of functionality are separated, which makes maintenance and debugging simpler than when everything is put together in one block.

### Add a note hereAdd a note hereUsing Specific Data Types

Add a note hereAdd a note hereAll of the examples shown thus far assume that the option is string value based. You can, however, define specific object types. For example, a command line argument could be a number, and therefore it would be better to return an integer value. [Listing 8.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) is example of where an integer command line argument is converted into a number class instance.

Larger View

Add a note hereAdd a note here**Listing 8.10**

Add a note hereAdd a note here Options options = new Options();

Option option = OptionBuilder.withLongOpt( "integer")

.hasArg()

.isRequired( false)

.withDescription( "integer option type" )

.withType( PatternOptionBuilder.NUMBER\_VALUE)

.create( 'i' );

options.addOption( option);

CommandLineParser parser = new BasicParser();

CommandLine cmd = parser.parse( options, args);

Integer value = (Integer)cmd.getOptionObject("i");

System.out.println( "Value is " + value.toString());

Add a note hereAdd a note hereIn [Listing 8.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), the type of the option is set using the method withType. The method withType has only one parameter, which is the class descriptor of a supported type. The class descriptor has to be one of the types exposed by the class PatternOptionBuilder.

Add a note hereAdd a note hereThe class PatternOptionBuilder is a class that helps a developer define options that are type specific. However, it's not that useful, as shown in [Listing 8.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512).

Larger View

Add a note hereAdd a note here**Listing 8.11**

Add a note hereAdd a note hereOptions options = PatternOptionBuilder.parsePattern("a:b@cde>f+n%t/");

Add a note hereAdd a note hereIn [Listing 8.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), the method parsePattern parses a string of characters and returns a list of options. However, looking at the string in [Listing 8.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), you can see that it's not even close to apparent what the individual option types are. Hence, using the syntax in [Listing 8.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512) is much more useful and maintainable than that in [Listing 8.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512).

Add a note hereAdd a note hereTo return a specific data type, you use the method getOptionObject, which searches for a specific option. If the option is found, the associated argument value is converted into the correct data type. In [Listing 8.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=103735512), the returned type is cast into an Integer class instance. If the cast were incorrect, a cast exception would be generated.

## **Managing Configurations Using *Digester***

Add a note hereAdd a note hereIn the past, configuration files have been stored in a Windows ini-type format, which defines sections with key value pairs. Many UNIX formats that predate the Windows ini-type format are similar. Regardless of the format types, the future of applications lies in the usage of XML files as configuration formats. XML is a standard, and it is easy to use. The *Digester* package is used to read XML-based configuration files.

Add a note hereAdd a note hereIn [Chapter 5](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=293#293), we used the *betwixt* package to serialize objects to and from XML. The *Digester* package, which is a dependency of the *betwixt* package, is an XML serialization package. You may ask why we didn't cover the *Digester* package in [Chapter 5](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=293#293). Well, from day one, the *betwixt* package was intended to serialize Java objects to XML, whereas the *Digester* package was originally intended for reading configuration files. However, either one could be used for either task. After all, reading a configuration file is nothing more than de-serializing a set of configuration objects.

### Add a note hereAdd a note hereTechnical Details for the *Digester* Package

Add a note hereAdd a note here[Tables 8.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) and [8.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) contain the abbreviated details necessary to use the *Digester* package.

**Add a note hereAdd a note hereTable 8.4:** Repository details for the *Digester* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note heredigester |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.digester (contains the core classes); org.apache.commons.[digester.xml](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053)rules (contains the classes used to parse XML digester configuration files) |

**Add a note hereAdd a note hereTable 8.5:** Package and class details (legend: [digester] = org.apache.commons.digester).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[digester].Digester | Add a note hereAdd a note hereThe core class that manages the rules for the *Digester* package. You will use this class the most often. |
| Add a note hereAdd a note here[digester].Rule | Add a note hereAdd a note hereAn abstract class that might be extended when you attempt to define custom actions and rules. |
| Add a note hereAdd a note here[digester].\*Rule | Add a note hereAdd a note hereCore classes that implement the various actions and rules. When the Digester class is used to add rules, these classes are created automatically. |
| Add a note hereAdd a note here[digester].xmlrules.DigesterLoader | Add a note hereAdd a note hereA core class used to instantiate the Digester class instance from an XML digester configuration file. |

### Add a note hereAdd a note hereReading a Simple Configuration

Add a note hereAdd a note hereIn this chapter, we will be using the class BeanToWrite, which originates from [Chapter 5](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=293#293) and was also used in [Chapter 7](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=468#468). As a quick refresher, the class BeanToWrite has two properties: integerValue and stringValue. In this chapter, the object is to be able to read a simple configuration file, as shown in [Listing 8.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317). The configuration file data is used to populate the class BeanToWrite.

Add a note hereAdd a note here**Listing 8.12**

Add a note hereAdd a note here<body>

<element1>hello</element1>

<element2>1234</element2>

</body>

Add a note hereAdd a note hereThe XML document in [Listing 8.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) is processed by [Listing 8.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317).

Larger View

Add a note hereAdd a note here**Listing 8.13**

Add a note hereAdd a note hereDigester digester = new Digester();

digester.addObjectCreate( "body", BeanToWrite.class);

digester.addCallMethod( "body/element1", "setStringValue", 0);

digester.addBeanPropertySetter( "body/element2", "integerValue");

BeanToWrite bean = (BeanToWrite)digester.parse( generateSimpleConfiguration());

Add a note hereAdd a note hereIn [Listing 8.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), the main class used to process a configuration file is the class Digester. If we want to cross-reference the class BeanToWrite to individual XML tags in [Listing 8.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), we create digester rules. The concept of a *digester rule* is that if a specific XML tag is encountered, some action should be taken. The action to be taken could be to create a class, call a method, or assign a bean property.

Add a note hereAdd a note hereTo create a rule, an XML tag is needed. In [Listing 8.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), there are three XML tags. Therefore, in the simplest case, the rules could be defined as follows:

* Add a note hereAdd a note hereWhen the XML tag body is encountered, create a BeanToWrite class instance.
* Add a note hereAdd a note hereWhen the XML tag element1 is encountered, assign the value to the property stringValue.
* Add a note hereAdd a note hereWhen the XML tag element2 is encountered, assign the value to the property integerValue.

Add a note hereAdd a note hereThe definition of the rules is a bit vague, but a computer needs precision. The problem is that the definition of the tag is vague. Let's say there is an XML tag title. The XML tag title occurs at multiple places in the XML document. Using the rules defined in the previous list would cause problems because the same rule could be triggered even though the XML tag title might be located in different places. The resolution is to define the location of the XML tag by providing the context of the location. Therefore, we need to rewrite the rules with a more precise location and action, as shown in this list of rules:

* Add a note hereAdd a note hereWhen the XML tag body is encountered, call the method addObjectCreate that creates a BeanToWrite class instance.
* Add a note hereAdd a note hereWhen the XML tag body/element1 is encountered, call the method addCallMethod, where the method to call is setStringValue.
* Add a note hereAdd a note hereWhen the XML tag body/element2 is encountered, call the method addBeanPropertySetter, where the property to set is integerValue.

Add a note hereAdd a note hereOnce all of the rules have been defined, the XML file can be parsed using the method parse. Returned will be an object instance of the class BeanToWrite. Of course, this assumes that no errors occurred while the XML file was processed.

### Add a note hereAdd a note hereDefining the Matching Patterns

Add a note hereAdd a note hereThe *matching patterns* define a location where an action should occur. In [Listing 8.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), three rules were defined when the *Digester* object was used to call a method that included the word add. The parameters of the methods are contexts that trigger a specific action. The way that the actions are triggered is related to how the XML configuration file is processed. The configuration file is processed using the Simple API for XML (SAX). The SAX processing model is based on events firing whenever an XML element is encountered. The events are used to provide a location definition using a stack (more about stacks soon). This is why a location of an XML element is defined in absolute terms. [Listing 8.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) converts [Listing 8.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) to a stack of locations from the *Digester* package's point of view.

Larger View

Add a note hereAdd a note here**Listing 8.14**

Add a note hereAdd a note here<body> body

<element1>hello</element1> body/element1

<element2>1234</element2> body/element2

</body>

Add a note hereAdd a note hereThe side effect of using absolute positions to define a location is that it can get tedious. For example, the location body/something/somewhere/and/more/elements is a bit long. This is why the *Digester* package is better as a configuration tool than a generic serialization framework like the *betwixt* package. The *Digester* assumes that the XML file is easy to read and understand.

Add a note hereAdd a note hereThere is one exception to the absolute path reference rule, which is shown in [Listing 8.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317).

Add a note hereAdd a note here**Listing 8.15**

Add a note hereAdd a note here\*/element1

Add a note hereAdd a note hereIn [Listing 8.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), the asterisk represents a wildcard. This wildcard indicates that XML element element1 can be matched anywhere in the XML document. In contrast, the patterns in [Listing 8.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) are not the same elements in a logical sense.

Add a note hereAdd a note here**Listing 8.16**

Add a note hereAdd a note herebody/element1

body/element1/element1

body/elsewhere/child/element1

Add a note hereAdd a note hereIn [Listing 8.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), the matched element is element1, but it exists in three different locations. And for each of the locations, a different rule is created.

### Add a note hereAdd a note hereDefining the Rules

Add a note hereAdd a note hereIt's important to remember what the stack concept means. The stack concept involves pushing things on a stack, like one would put a piece of paper on top of a stack of paper. The top of the stack is always the last elements pushed. The rules created by the consumer have the ability to manipulate the stack. For example, if the matching pattern body is found, then the class BeanToWrite is instantiated. The action responsible for instantiating the class pushes the instantiated class onto a stack. Multiple actions can be executed for each matching pattern. This can result in multiple instantiated objects being pushed on the stack. However, due to the current architecture, only one object can be manipulated at a time.

Add a note hereAdd a note hereTo understand the fundamental premise of the action, [Listing 8.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) shows a class that parses the configuration file defined in [Listing 8.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317).

Larger View

Add a note hereAdd a note here**Listing 8.17**

Add a note hereAdd a note hereclass MyRule extends Rule {

private String \_bodyText;

public void begin(String namespace, String name, Attributes attributes)

throws Exception {

if( name.compareTo( "body") == 0) {

digester.push( new BeanToWrite());

}

}

public void body(String namespace, String name, String text) throws Exception {

\_bodyText = text.trim();

}

public void end( String namespace, String name) throws Exception {

if( name.compareTo( "element2") == 0) {

BeanToWrite bean = (BeanToWrite)digester.peek();

bean.setIntegerValue( new Integer( \_bodyText).intValue());

}

else if( name.compareTo( "element1") == 0) {

BeanToWrite bean = (BeanToWrite)digester.peek();

bean.setStringValue( \_bodyText);

}

}

}

Add a note hereAdd a note hereIn [Listing 8.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), the class MyRule subclasses the class Rule. The abstract class Rule is the basis of all actions. When you're extending the class Rule, the methods that are implemented are based on the events that are captured. The events relate to the SAX XML event-processing model. Specifically, [Listing 8.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) illustrates the events that are generated as per the XML document from [Listing 8.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317).

Add a note hereAdd a note here**Listing 8.18**

Add a note hereAdd a note herebegin( "", "body", attributes);

begin( "", "element1", attributes);

body( "", "element1", "hello");

end( "", "element1");

begin( "", "element2", attributes);

body( "", "element2", "1234");

end( "", "element2");

end( "", "body");

Add a note hereAdd a note hereIn [Listing 8.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), three major method calls can be considered as events: when an element begins (begin), the element body text (body), and when an element ends (end). The class in [Listing 8.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) implements these methods and hence will capture these events. For example, in [Listing 8.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), the first line is the begin method call, which instantiates the class BeanToWrite when cross-referenced to the method begin in [Listing 8.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317). In the implementation of the method begin in [Listing 8.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), the instantiated class is pushed on the stack by the class method digester.push.

Add a note hereAdd a note hereThen, to fill in the properties of the class BeanToWrite, the event when an element ends is used. This event is used because only this event will ensure that all configuration data has been captured. The configuration data is stored as an XML key value pair. This means that the configuration data triggers the event element body text, which is stored temporarily in the variable \_bodyText of [Listing 8.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317). The BeanToWrite class instance, which was allocated and pushed onto the stack in a previous event, is peeked at from the stack using the method peek. In [Listing 8.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), the events are then generated and the class of [Listing 8.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) captures these events; when the two listings are assembled together they make it possible to populate a complex object hierarchy.

Add a note hereAdd a note here[Listing 8.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) shows how to make use of the user-defined rule.

Larger View

Add a note hereAdd a note here**Listing 8.19**

Add a note hereAdd a note hereDigester digester = new Digester();

MyRule rule = new MyRule();

digester.addRule( "body", rule);

digester.addRule( "body/element1", rule);

digester.addRule( "body/element2", rule);

BeanToWrite bean = (BeanToWrite)digester.parse( generateSimpleConfiguration());

Add a note hereAdd a note hereIn [Listing 8.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), the code has been written in an unusual manner. Instead of having a unique rule instance for every matched pattern, we use the same rule instance. The reason for this is simple, because the action class MyRule can process all elements of the configuration file. Of course, writing all applications using this manner is not that useful.

Add a note hereAdd a note hereThe *Digester* package defines a number of smaller rules, which are instantiated using methods on the Digester class. For example, the method addObjectCreate instantiates and adds the rule ObjectCreateRule. In the *Digester* package, all of the classes that end with the word "Rule" extend the Rule class. Following is a list of Digester class methods and what rules are instantiated:

* Add a note hereAdd a note here**addBeanPropertySetter**: Instantiates the class BeanPropertySetterRule. This is used to assign a specific property on the current object in the *Digester* stack.
* Add a note hereAdd a note here**addCallMethod**: Instantiates the class CallMethodRule. This is used to call a method on the current object in the *Digester* stack.
* Add a note hereAdd a note here**addCallParam**: Instantiates the class CallParamRule. This is used in conjunction with the class CallMethodRule to establish a more complex method call that has parameters.
* Add a note hereAdd a note here**addFactoryCreate**: Instantiates the class FactoryCreateRule. This is like the class ObjectCreateRule, except that a factory is used to instantiate the class instance that is pushed on the *Digester* stack.
* Add a note hereAdd a note here**addObjectCreate**: Instantiates the class ObjectCreateRule. This class creates a class instance based on a class descriptor. The instantiated class is pushed on the *Digester* stack.
* Add a note hereAdd a note here**addSetNext**: Instantiates the class SetNextRule. When executed, this class pops an object from the digester stack. The popped object is then assigned to the next available object on the *Digester* stack using a specified method.
* Add a note hereAdd a note here**addSetProperties**: Instantiates the class SetPropertiesRule. This set of properties will be assigned to the current object on the stack.
* Add a note hereAdd a note here**addSetProperty**: Instantiates the class SetPropertyRule. A single property is assigned to the current object on the stack.
* Add a note hereAdd a note here**addSetRoot**: Instantiates the class SetRootRule. The implementation of this action is like the implementation of the method addSetNext. The difference is that, here, the child instance is assigned to the bottom object instance from the *Digester* stack.
* Add a note hereAdd a note here**addSetTop**: Instantiates the class SetTopRule. This class is like the class SetNextRule except the order of assignment. This class does not pop anything from the *Digester* stack. The purpose of this class is to assign a parent to a child class. You create the relationship by retrieving the second-from-the-top object instance from the *Digester* stack and then assigning it using a specified method to the top object instance of the *Digester* stack.

### Add a note hereAdd a note hereCalling a Method with Parameters

Add a note hereAdd a note hereIn [Listing 8.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), we used the method addCallMethod. In that case, the method called was a property method. There was a single parameter, which was a string. We used a method with multiple parameters because the Java configuration class will not always expose Java Bean properties that can easily be assigned. Instead, the state of the Java Configuration class needs to be assigned using a method. This would usually happen when the configuration file references Java classes that might not be a Java Bean or might be a business process class, or even a legacy class that cannot be rewritten. For example, maybe the contents of element1 and element2 of [Listing 8.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) are parameters for a method, which usually are assigned to Java properties. The solution to this type of problem is shown in [Listing 8.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317). It is assumed that the XML configuration file in [Listing 8.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) is attempted to be assigned to a class that exposes a method with two string parameters, for example, void assignBothValues ( String param1, String param2).

Larger View

Add a note hereAdd a note here**Listing 8.20**

Add a note hereAdd a note hereDigester digester = new Digester();

MyRule rule = new MyRule();

digester.addRule( "body", new ObjectCreateRule( BeanToWrite.class));

digester.addRule( "body", new CallMethodRule( "assignBothValues", 2));

digester.addRule( "body/element2", new CallParamRule(0));

digester.addRule( "body/element1", new CallParamRule(1));

BeanToWrite bean = (BeanToWrite)digester.parse( generateSimpleConfiguration());

Add a note hereAdd a note hereIn [Listing 8.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), the method addRule was used and the classes were instantiated directly. We chose to use this technique in this context to show how it would be done, and it allows a developer to extend the *Digester*-provided classes. To make a multiple parameter method call, the class CallMethodRule has to be associated with a pattern that is located at a higher level than the parameters. The class CallParamRule is used to associate an XML element value with a parameter of the method call. The integer passed to the constructor of the class CallParamRule indicates the order of the parameter in the method call. In [Listing 8.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), this means that the first parameter is identified by the zero and the second parameter identified by the one. [Listing 8.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) shows how to use the rules defined in [Listing 8.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) (note that the events shown are the actual important events, and not the initialization of the individual classes).

Add a note hereAdd a note here**Listing 8.21**

Add a note hereAdd a note hereObjectCreateRule.begin( "", "body", attributes);

CallParamRule.body( "", "element1", "hello");

CallParamRule.body( "", "element2", "1234");

CallMethodRule.end( "", "body");

Add a note hereAdd a note hereNotice in [Listing 8.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) that the object is created correctly at the beginning of the events. Then, when the method body is called, the parameters are saved to a temporary location. Finally, the method end, which calls the method assignToBothValues, is called.

Add a note hereAdd a note hereThe method assignToBothValues is defined in [Listing 8.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) (note that the class has been abbreviated for clarity).

Larger View

Add a note hereAdd a note here**Listing 8.22**

Add a note hereAdd a note herepublic class BeanToWrite implements java.io.Serializable {

private int \_iValue;

private String \_strValue;

public void assignBothValues( String ival, String sval) {

\_iValue = new Integer( ival).intValue();

\_strValue = sval;

}

}

Add a note hereAdd a note hereIn [Listing 8.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), the method assignBothValues has two methods, where both parameters are strings. This is the default format for all parameters when you make method calls without specifying the types. A variation of the Digester class method addCallMethod accepts an array of class descriptors as a parameter. The class descriptors define the data type of the individual parameters.

### Add a note hereAdd a note hereEstablishing Parent-Child Relationships

Add a note hereAdd a note hereMost configurations are not based on a single class. In most configuration cases, multiple objects are involved. This means that some parts of the configuration file will instantiate an object that needs to be added to a previously created object. [Listing 8.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) is a more complicated configuration file than [Listing 8.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317).

Add a note hereAdd a note here**Listing 8.23**

Add a note hereAdd a note here<parent>

<dataMember>hello</dataMember>

<myReferenceToAnotherBean>

<stringValue>hello</stringValue>

<integerValue>1234</integerValue>

</myReferenceToAnotherBean>

</parent>

Add a note hereAdd a note hereIn [Listing 8.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), there are two classes. The tag parent identifies the first class ParentBean, and the tag myReferenceToAnotherBean identifiers the other class, BeanToWrite. [Listing 8.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) illustrates how to parse the multiple class configuration file.

Larger View

Add a note hereAdd a note here**Listing 8.24**

Add a note hereAdd a note hereDigester digester = new Digester();

digester.addObjectCreate( "parent", ParentBean.class);

digester.addBeanPropertySetter( "parent/dataMember", "dataMember");

digester.addObjectCreate( "parent/myReferenceToAnotherBean", BeanToWrite.class);

digester.addBeanPropertySetter(

"parent/myReferenceToAnotherBean/stringValue", "stringValue");

digester.addBeanPropertySetter(

"parent/myReferenceToAnotherBean/integerValue", "integerValue");

digester.addSetNext(

"parent/myReferenceToAnotherBean ", "setMyReferenceToAnotherBean");

ParentBean bean = (ParentBean)digester.parse( generateParentConfiguration());

Add a note hereAdd a note here[Listing 8.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) shows the essentials of how to string together multiple classes and assign Java properties. For every tag that has been identified as a class, which in our case are the tags parent and myReferenceToAnotherBean, there is an associated create object method call. The method addObjectCreate with the correct class descriptor is used to create the object. To assign the bean properties, the method addBeanPropertySetter is used.

Add a note hereAdd a note hereTo assign the child BeanToWrite class instance to the parent ParentBean class instance, the method addSetNext is used. The method addSetNext has two parameters. The first parameter is the pattern match, but the second parameter is the method that is used to assign the child class to the parent class. The method used to assign the child object in [Listing 8.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) is a bean property. This is not a recommended practice because a configuration file may have multiple setMyReferenceToAnotherBean elements. In those cases, the method used to assign a child object to a parent object should be a method to add the object to a list. However, in this case, it is acceptable to use a bean property since there is only one setMyReferenceToAnotherBean element.

Add a note hereAdd a note hereIf you need to have a reference of the ParentBean class instance within the BeanToWrite class instance, you need to use the method addSetTop. [Listing 8.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) is a sample implementation.

Larger View

Add a note hereAdd a note here**Listing 8.25**

Add a note hereAdd a note heredigester.addSetTop( "parent/myReferenceToAnotherBean", "setParent");

Add a note hereAdd a note hereGenerally speaking, wherever the method addSetNext is used, the method addSetTop can be used.

### Add a note hereAdd a note hereUsing XML Attributes

Add a note hereAdd a note hereIn all of the configuration examples so far in this chapter, we used XML configurations without any attributes. *Digester* does, however, allow the use of attributes, which can be assigned to bean properties like the method addBeanPropertySetter. [Listing 8.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) is a modified configuration file that uses only attributes.

Add a note hereAdd a note here**Listing 8.26**

Add a note hereAdd a note here<body integerValue='1234' stringValue='hello'

Add a note hereAdd a note here[Listing 8.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) is a program that parses the configuration into the object BeanToWrite.

Larger View

Add a note hereAdd a note here**Listing 8.27**

Add a note hereAdd a note hereDigester digester = new Digester();

digester.addObjectCreate( "body", BeanToWrite.class);

digester.addSetProperties( "body", "integerValue", "integerValue");

digester.addSetProperties( "body", "stringValue", "stringValue");

BeanToWrite bean = (BeanToWrite)digester.parse( generateAttributeConfiguration());

Add a note hereAdd a note hereIn [Listing 8.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), the method addSetProperties is used to assign an attribute to a specific property. The method addSetProperties has three parameters. The first parameter is the pattern match, as in previous examples. The second parameter is the attribute identifier, where an attribute value is obtained. The third parameter represents the bean property that will be assigned with the obtained attribute value.

### Add a note hereAdd a note hereUsing XML Digester Configuration Files

Add a note hereAdd a note hereThus far, we have specified all the rules using a piece of code. For added flexibility, you can assign the rules using an XML file. In effect, this is a configuration file that parses a configuration file. The advantage of using this approach is that you don't have to change the program if either the configuration file or the configuration objects change. We will convert [Listing 8.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) into a configuration file, where the class ParentBean references the class BeanToWrite. [Listing 8.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) is the implementation of the XML *Digester* configuration file.

Larger View

Add a note hereAdd a note here**Listing 8.28**

Add a note hereAdd a note here<digester-rules>

<pattern value="parent">

<object-create-rule

classname="com.devspace.jseng.serialization.ParentBean" />

<bean-property-setter-rule pattern="dataMember" name="dataMember" />

</pattern>

<pattern value="parent/myReferenceToAnotherBean">

<object-create-rule classname="com.devspace.jseng.serialization.BeanToWrite" />

<call-method-rule pattern="stringValue" methodname="setStringValue"

paramcount="0" />

<call-method-rule pattern="integerValue" methodname="setIntegerValue"

paramcount="0" paramtypes="java.lang.Integer" />

<set-next-rule methodname="setMyReferenceToAnotherBean" />

</pattern>

</digester-rules>

Add a note hereAdd a note hereWhat should be very obvious from [Listing 8.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) is that the element names are almost identical to the names of the classes used to execute a specific action. This is not accidental, because we use the *Digester* package to read [Listing 8.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317). The details are partially managed in the class DigesterRuleParser. The difference between the class names and the XML elements is that the XML elements are all lowercase words separated by hyphens.

Add a note hereAdd a note hereOne big advantage of using configuration files over a program that manually defines the *Digester* package rules is that it is simpler to make references because the XML elements are embedded. For example, [Listing 8.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) contains the XML element pattern. The XML element pattern has an attribute value>, which has a value of parent. Notice that the child element bean-property-setter-rule pattern attribute is only dataMember and not parent/dataMember. This happens because the XML *Digester* parser can build an absolute path based on relative values. In fact, in [Listing 8.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), we could have embedded the second XML tag pattern within the first XML tag pattern. In that case, the second XML element's pattern value attribute would be myReferenceToAnotherBean instead of parent/myReferenceToAnotherBean. The items within the second XML element pattern would not have to be adapted because they are already relative to the parent. However, if it is absolutely desired to use absolute paths through all elements, you can do so by removing the XML element pattern and referencing the paths absolutely. Note that the XML element object-create-rule would need an additional pattern attribute. In [Listing 8.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), the lack of a pattern attribute means that you should use the current value.

Add a note hereAdd a note here[Listing 8.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) is used to load and parse both configuration files and generate a set of class instances.

Larger View

Add a note hereAdd a note here**Listing 8.29**

Add a note hereAdd a note hereFile rules = new File( getFilename());

Digester digester = DigesterLoader.createDigester( rules.toURL());

ParentBean bean = (ParentBean)digester.parse( generateParentConfiguration());

Add a note hereAdd a note hereIn [Listing 8.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317), the class DigesterLoad, which is located in the package org.apache.commons.[digester.xml](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053).rules, is used. The class DigesterLoader is used to instantiate a *Digester* class instance based on the XML *Digester* configuration in [Listing 8.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317). There is a special catch, though. When the class DigesterLoad attempts to parse the XML *Digester* configuration file, a reference to the org.apache.commons.[digester.xml](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053)rules.[digester-rules.dtd](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) must exist on the class path. The returned *Digester* class instance can then be used to parse a configuration file, which instantiates a ParentBean class instance. [Table 8.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317) explains the available XML elements and what they mean.

**Add a note hereAdd a note hereTable 8.6:** XML elements and the command line parsers implementations that can parse the arguments.

| **Add a note hereAdd a note hereXML Element** | **Add a note hereAdd a note hereAssociated Class** | **Add a note hereAdd a note hereAvailable Attributes** |
| --- | --- | --- |
| Add a note hereAdd a note herebean-property-setter-rule | Add a note hereAdd a note hereBeanPropertySetterRule | Add a note hereAdd a note herepattern: path propertyname: The name of the bean property. |
| Add a note hereAdd a note herecall-method-rule | Add a note hereAdd a note hereCallMethodRule | Add a note hereAdd a note herepattern: path methodname (req): The name of the method that is called. paramcount: The number of parameters that the method expects. paramtypes: A comma-separated list of data types used to define the method type. |
| Add a note hereAdd a note herecall-param-rule | Add a note hereAdd a note hereCallParamRule | Add a note hereAdd a note herepattern: path paramnumber (req): The order of the parameter in the method call. attrname: The name of the attribute if the value is retrieved from an XML attribute. (Note that attrname and from-stack cannot be combined.) from-stack: Sets the value from the stack. (Note that attrname and from-stack cannot be combined.) |
| Add a note hereAdd a note herefactory-create-rule | Add a note hereAdd a note hereFactoryCreateRule | Add a note hereAdd a note herepattern: path classname (req): The name of class that supports the interface ObjectCreationFactory. ignore-exceptions: A true or false value that indicates whether or not exceptions in object instantiation should be ignored. |
| Add a note hereAdd a note hereobject-create-rule | Add a note hereAdd a note hereObjectCreateRule | Add a note hereAdd a note herepattern: path classname (req): The name of the class to instantiate. |
| Add a note hereAdd a note hereset-properties-rule | Add a note hereAdd a note hereSetPropertiesRule | Add a note hereAdd a note hereA an attribute pattern (not usually used), but therefore child XML element alias, which has the following attributes: attr-name (req): The name of the XML attribute to retrieve the value from. prop-name: The name of the bean property to assign. |
| Add a note hereAdd a note hereset-property-rule | Add a note hereAdd a note hereSetPropertyRule | Add a note hereAdd a note herepattern: path name: The name of the XML attribute to retrieve the value from. value: The name of the bean property to assign. |
| Add a note hereAdd a note hereset-top-rule | Add a note hereAdd a note hereSetTopRule | Add a note hereAdd a note herepattern: path methodname (req): The name of the method to assign the parent. paramtype: The data type used to assign the parent. |
| Add a note hereAdd a note hereset-next-rule | Add a note hereAdd a note hereSetNextRule | Add a note hereAdd a note herepattern: path methodname (req): The name of the method to add the child. paramtype: The data type used to assign the child. |
| Add a note hereAdd a note herepattern include N/A | Add a note hereAdd a note hereN/A | Add a note hereAdd a note herevalue (req): The path in the XML document.class: The name of the class that includes a number of rules. path: The name of a file that includes a number of configuration rules. |
| Add a note hereAdd a note hereNote that “path” means either a relative or absolute XML path in the configuration file, as per our explanation in the [previous section](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=433278317). “req” means required; otherwise, the attributes are optional. | | |

## **Creating Log Output**

Add a note hereAdd a note hereAnother major item that administrators want control of is logging. *Logging* is the process of generating output when something occurs. That something could be an error, warning, or message. It's important to generate log output because it tells the administrator what the application is doing, and it saves the administrator from having to resort to using a debugger. Administrators are sophisticated computer people, but they are not programmers in the sense we use the term in this book. The closest the administrator wants to get to programming is setting the verbose flag on the application (more on this shortly).

Add a note hereAdd a note hereHere are the basic logging levels:

* Add a note hereAdd a note here**error**: Something caused an error in the application, and that causes the application to generate the reason why an error occurred.
* Add a note hereAdd a note here**warning**: Something is causing a problem, but the application can cope with the problem. An example includes having two aliases pointing to the same item. This condition is not an error but should not occur.
* Add a note hereAdd a note here**informational**: Something of importance has occurred and the application makes a note of it. An example includes identifying the database connection that the server is using. An informational-type log output is extremely important because it allows an administrator to troubleshoot and see where things are going wrong.
* Add a note hereAdd a note here**verbose**: Whatever has occurred while the program is running, including errors, warnings, messages, etc., are output. This type of log output should be used only when you're debugging very complicated problems. A verbose log output is very similar to a debug output, in that everything that could be important is logged. When the verbose mode is activated, logging can and will impair the performance of the application. It is not meant to be a general logging mode.

Add a note hereAdd a note hereThe *Logging* package in the Commons project is not an entire package; rather, it's a wrapper for other logging packages. The *Logging* package provides a wrapper for the Java logging mechanism, Avalon Framework, and the Log4J package.

### Add a note hereAdd a note hereTechnical Details for the *Logging* Package

Add a note hereAdd a note here[Tables 8.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981) and [8.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981) contain the abbreviated details necessary to use the *Logging* package.

**Add a note hereAdd a note hereTable 8.7:** Repository details for *Logging* package

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note herelogging |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.logging (contains the core logging classes); org.apache.commons.logging.impl (contains the implementations of the interface Log; more details are given in the Java Docs and later in this chapter) |

**Add a note hereAdd a note hereTable 8.8:** Package and class details (legend: [logging] = org.apache.commons.logging).

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[logging].Log | Add a note hereAdd a note hereA core interface used to generate log messages |

### Add a note hereAdd a note hereA Simple Log Example

Add a note hereAdd a note hereLogging an error can be really simple, as shown in [Listing 8.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981).

Add a note hereAdd a note here**Listing 8.30**

Add a note hereAdd a note here Log log = new SimpleLog( "myLogFile");

log.error( "some error");

Add a note hereAdd a note hereIn [Listing 8.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981), the interface Log is the core interface used to access the logging implementation. There are multiple implementations of the interface Log, which are all located in the package org.apache.commons.logging.impl. In [Listing 8.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981), the implementation SimpleLog is used. Then, to log a message, the method error is called. What the method error does is up to the implementation of the Log interface. In [Listing 8.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981), the error is output to the System.err stream.

Add a note hereAdd a note hereThe following are the available Log interface implementations:

* Add a note hereAdd a note here**AvalonLogger**: A class that accepts as a constructor parameter an Avalon Logger interface instance. The Log interface wraps the Avalon Logger interface instance, and whenever errors occur, the Avalon Logger handles them.
* Add a note hereAdd a note here**Jdk14Logger**: A class that wraps the standard JDK logging mechanisms introduced in the JDK 1.4.
* Add a note hereAdd a note here**Log4JLogger**: A class that maps the Log interface methods to the Log4J logger. Using the Log4J logger requires that the Log4J library be properly configured.
* Add a note hereAdd a note here**LogKitLogger**: A class that maps the Log interface to the Avalon Logging system. The Avalon Logging system needs to be properly configured.
* Add a note hereAdd a note here**NoOpLog**: A class that implements the Log interface but does nothing with it.
* Add a note hereAdd a note here**SimpleLog**: A class that implements the Log interface and outputs all of the log messages to the System.err stream.

### Add a note hereAdd a note hereDefining the Logging Levels

Add a note hereAdd a note hereIn [Listing 8.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981), the error that was logged would have been output. However, in the default level, any warning or debug message will be ignored. This means that debug messages will be ignored when only error messages are to be logged. For example, [Listing 8.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981) is an example of how you can use the logging interface in a production coding setting.

Add a note hereAdd a note here**Listing 8.31**

Add a note hereAdd a note here Log log = getLogImpl();

try {

log.debug( "Allocating the bean");

BeanToWrite bean = null;

bean.setIntegerValue( 1234);

log.info( "Carried out logic");

}

catch( Exception ex) {

log.error( "oops message", ex);

}

Add a note hereAdd a note hereIn [Listing 8.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981), three levels are used. The method info is used to create an informational message. In [Listing 8.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981), the important event is that the bean logic has been carried out. The method debug is used to create a verbose log output. In [Listing 8.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981), the verbose log output is the allocation of a specific bean. It is important to realize the difference between log outputs. For informational-type log outputs, the objective is to log logic, or business-related, actions. For verbose-type log outputs, the objective is to log resource or programmatic actions, like an allocation.

Add a note hereAdd a note hereFinally, when an error occurs, it is logged using the method error. Unlike the other logging methods, error has two parameters. The first parameter represents the error message. The second parameter represents an exception that has been thrown. The underlying logging implementation will then log the exception to give better details on what caused the error.

Add a note hereAdd a note hereThe Log interface has more methods than logging levels defined in the beginning of this Logging section. Here are the individual logging levels methods listed, cross-referenced with the associate level:

* Add a note hereAdd a note here**fatal**: This method is an error level that defines a fatal error, meaning that the program can no longer continue and must exit.
* Add a note hereAdd a note here**error**: This method is an error level that needs to be logged. However, you could let the program continue, albeit not optimally.
* Add a note hereAdd a note here**warn**: This method is a warning level that needs to be logged. The program can recover and still functional correctly. However, certain parts of the program may not be operational.
* Add a note hereAdd a note here**info**: This method is an informational level. The program will output business- or logic-related information, such as database server started, or mortgage completed.
* Add a note hereAdd a note here**debug**: This method is a verbose level. The program will output programmatically related issues, such as memory allocation or network connections.
* Add a note hereAdd a note here**trace**: This method is a verbose level. The method is used to output almost every detail of the program, without using a debugger. Using this method should slow down the entire program because of the number of logging calls made.

### Add a note hereAdd a note hereActivating the Logging Process

Add a note hereAdd a note hereUsing the *Logging* package is not that difficult; in fact, it can be relatively simple. However, we haven't yet explored what is going on underneath the package. In [Listing 8.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981), the method getLogImpl was a custom method that instantiates an instance of the Log interface implementations. (Of course, since you've been reading this chapter, you know that this method would be based on a configuration item.) Regardless of which implementation you use, you need to configure the underlying logging implementation properly. If the implementation is related to Log4J, the Avalon logging toolkit, or JDK logging, you can get further information from the associated logging implementation.

Add a note hereAdd a note hereIf the implementation of the method getLogImpl uses the SimpleLog class, supplied by the *Logging* package, a configuration file has to be provided. The configuration file defines how to generate the log output. When the SimpleLog class is instantiated, the file Image from book[simplelog.properties](http://cdcontent.books24x7.com/id_7265/simplelog.properties) is searched for. Typically, this file should reside in the classpath or a jar file. [Listing 8.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981) is a sample Image from book[simplelog.properties](http://cdcontent.books24x7.com/id_7265/simplelog.properties) file.

Add a note hereAdd a note here**Listing 8.32**

Add a note hereAdd a note hereorg.apache.commons.logging.simplelog.defaultlog=debug

org.apache.commons.logging.simplelog.log.myLogFile=debug

org.apache.commons.logging.simplelog.showlogname=true

org.apache.commons.logging.simplelog.showShortLogname=true

org.apache.commons.logging.simplelog.showdatetime=true

Add a note hereAdd a note hereIf you want to understand this configuration file, it is simpler to remove the package prefix and consider [Listing 8.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981). (Note that in an actual configuration file, the package prefix is required.)

Add a note hereAdd a note here**Listing 8.33**

Add a note hereAdd a note heredefaultlog=debug

log.myLogFile=debug

showlogname=true

showShortLogname=true

showdatetime=true

Add a note hereAdd a note hereWhen the class SimpleLog is instantiated, the property that defines the debug level is searched for. In [Listing 8.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981), this means the properties defaultLog and log.myLogFile. There are two properties because the property defaultLog is the default logging level. The identifier log.myLogFile is a specific logging level identifier. The identifier myLogFile is the same identifier passed to the constructor of the class SimpleLog that was demonstrated in [Listing 8.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981). The available levels are: all, trace, debug, info, warn, error, fatal, and off. The level all indicates that all logging outputs should be activated. The level off indicates that nothing should be logged. The levels of logging ordered into levels of priority are fatal, error, warn, info, debug, and trace, where fatal is first and highest priority. For example, if the level fatal is defined, then only fatal errors will be output because fatal is the first and highest priority. The lowest and last priority is the level trace, which is like the level all and logs all outputs.

Add a note hereAdd a note hereIn [Listing 8.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981), the remaining properties are defined as follows:

* Add a note hereAdd a note here**showlogname**: The default is false; if this is set to true, it outputs the specific logging identifier.
* Add a note hereAdd a note here**showShortLogname**: The default is true and outputs a short form of the specific logging identifier. This is important only if the logging identifier is separated by dots to indicate a namespace-type definition.
* Add a note hereAdd a note here**showdatetime**: The default is false; if this is set to true, it is used to output the current time and date.

### Add a note hereAdd a note hereChecking If a Level Is Available

Add a note hereAdd a note hereIn [Listing 8.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981), the individual method levels were called without regard to whether or not the debug level was enabled. The problem with doing that is evident in [Listing 8.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981).

Larger View

Add a note hereAdd a note here**Listing 8.34**

Add a note hereAdd a note here Log log = getLogImpl();

try {

log.info( "Allocating the bean");

BeanToWrite bean = null;

bean.setIntegerValue( 1234);

log.debug( "Carried out logic" + bean.isValidObject());

}

catch( Exception ex) {

log.error( "oops message", ex);

}

Add a note hereAdd a note hereIn [Listing 8.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981), look at the method debug. The output string is a concatenation of a string and the output of the method isValidObject. The problem with this form of debug logging output is that the text might be ignored. And, if the text is ignored, that would mean that the method call isValidObject would be unproductive. A more productive way of writing [Listing 8.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981) is [Listing 8.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981).

Larger View

Add a note hereAdd a note here**Listing 8.35**

Add a note hereAdd a note here Log log = getLogImpl();

try {

log.info( "Allocating the bean");

BeanToWrite bean = null;

bean.setIntegerValue( 1234);

if( log.isDebugEnabled()) {

log.debug( "Carried out logic" + bean.isValidObject());

}

}

catch( Exception ex) {

log.error( "oops message", ex);

}

Add a note hereAdd a note hereIn [Listing 8.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981), the method call debug has been encapsulated within a decision block. The decision block calls the method isDebugEnabled to query if the debug level is output. The method isDebugEnabled exists for all levels using the pattern is[mode]Enabled.

Add a note hereAdd a note hereWhen you're writing complicated logging code, enabled method calls are preferred. Enabled method calls like those shown in [Listing 8.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=954563981) are preferred because they will not affect resource-intensive or time-critical code that may or may not output an informational message. If the enabled code did not exist, the message would be output, even if the debugger will not do anything with the message. Consider it as follows: a simple process would be to send mail to somebody whenever something occurs. But if the person does not want such a level of detail, then sending the mail will cost money and time. Hence, it is better to ask the person if he wants to be informed.

## **Summary**

Add a note hereAdd a note hereWe could quickly summarize this chapter by saying, "Administrators are people too." This phrase is meant to reinforce the fact that applications need to be put into a production environment. The programmer does not manage the production environment, nor does the user. The administrator manages the production environment and will most likely not use the application being managed. Therefore, the programmer's objective should be to develop an application that the administrator can manage without too much trouble.

Add a note hereAdd a note hereA good manageable application implements three tools: plug-ins, configuration files, and logging routines. Each of these pieces makes it easier for the administrator to create an environment that will make the application manageable and robust. The plug-ins make it easy for the administrator to add in external functionality. The configuration files allow the administrator to tune the application for optimum performance. Lastly, the logging files make it possible for the administrator to figure out what went wrong. An administrator who has these tools at his disposal will have fewer questions for the developer.

Add a note hereAdd a note hereSome people may argue that plug-ins, configuration files, and logging routines make an application slower. Granted, these three do present some performance issues, but these problems are easily offset by the fact that the administrator can figure out what to do with the application. The performance issues will not be issues if the configuration files and logging routines are properly engineered into the application. The problem is that developers often add such functionality as an afterthought.

Add a note hereAdd a note here**On the CD** The sources to the concepts presented in this chapter are located under the directory *[CDROM]/jseng/src/com.devspace.jseng.configuration*. For the section [*Activating the Logging Process*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=698#698) a sample configuration file is located at *[CDROM]/jseng/src/classes/ Image from book*[*simplelog.properties*](http://cdcontent.books24x7.com/id_7265/simplelog.properties)*.*

## **Questions**

Add a note hereAdd a note here**8.1:** Add a command line option to load the calculator extensions dynamically; e.g., a command line could be —extension=myclassfile. Note that multiple extensions could be specified. You'll need to figure out the exact nature of how to specify multiple extensions.

Add a note hereAdd a note here**8.2:** Add a command line option that specifies an XML configuration file used to define which extensions should be loaded dynamically. Remember that any command line option other than the configuration option must be available as an option in the configuration file.

Add a note hereAdd a note here**8.3:** Enable logging in the entire calculator application. Remember to include different levels of logging capabilities that a command line option or a configuration file defines.

**Chapter 9: Computer Algorithms**

**Add a note hereAdd a note here****Overview**

Add a note hereAdd a note hereIn this chapter, we will do the following:

* Add a note hereAdd a note hereSolve complex formula calculations using functors
* Add a note hereAdd a note hereCreate enumerations
* Add a note hereAdd a note hereUse the helper classes from the *lang* package
* Add a note hereAdd a note hereImplement the methods toString, compareTo, equals, and hashCode using helper classes
* Add a note hereAdd a note hereValidate data passed in from a Web site

## **The Purpose of This Chapter**

Add a note hereAdd a note hereMost people who attended any type of computer programming class at school were taught how to sort data. Oddly enough, very few people have written sorting routines because a plethora of sorting implementations are already available. Computer algorithms are generally taught to solve problems in the numerical field. In the general computing field, computer algorithms revolve around the ability to use specific patterns or best practices, many of which we have already discussed.

Add a note hereAdd a note hereThis chapter is different than previous ones because the focus is not on solving generic problems, but on solving specific problems using specific tools. These tools are not generally best practice tools, but tools that implement something. And something implemented means it does not need to be implemented yet again. Therefore, the task of this chapter is to introduce several specific algorithms.

## **Defining Functors**

Add a note hereAdd a note hereA program is a collection of decisions, loops, and declarations. Sometimes, though, you need to create a program dynamically. *Functors* are special constructors in a programming environment. A functor is a function with a state. With functors, you can dynamically build up an application and then have that application execute some type of logic.

Add a note hereAdd a note hereIn general, functors are not that useful. The scope of a functor is very specific to the domain of solving logic problems. For example, let's say that a word processor application is being coded. When the document is printed, the individual elements would need to be laid out on the virtual paper. Functors would be a good way of automating the layout calculation process.

Add a note hereAdd a note hereConsider the following simple example of calculating the royalties on the book you are reading. Note that the values for the individual royalty figures are fictitious, but the formula is not. Typically, royalties are calculated using a formula that involves royalties according to specific countries and specific book sales. The formula is not complex, but it involves calculations and decisions. Such a formula cannot be represented using one single mathematical formula. In most cases, the formula would be written as shown in [Listing 9.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455).

Larger View

Add a note hereAdd a note here**Listing 9.1**

Add a note hereAdd a note here public double calculateRoyalty( long countMarket1, long countMarket2) {

double total = 0.0;

if( countMarket1 < \_pivot) {

total += (\_royalty1 \* (double)countMarket1);

}

else {

total += (\_royalty1 \* (double)\_pivot);

total += (\_royalty2 \* (double)(countMarket1 - \_pivot));

}

if( countMarket2 < \_pivot) {

total += (\_royalty1 \* (double)countMarket2);

}

else {

total += (\_royalty1 \* (double)\_pivot);

total += (\_royalty2 \* (double)(countMarket2 - \_pivot));

}

return total;

}

Add a note hereAdd a note here[Listing 9.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) illustrates the classic approach to writing business logic that involves mathematical calculations. The problem with this approach is that it's not easy to maintain the application. If a formula requires any change, you must then meticulously inspect the code and make the necessary modifications. Using functors, we can rewrite [Listing 9.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) into something that is maintainable and extendable.

### Add a note hereAdd a note hereTechnical Details for the *functor* Factory

Add a note hereAdd a note here[Tables 9.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) and [9.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) contain the abbreviated details necessary to use the *functor* package.

**Add a note hereAdd a note hereTable 9.1:** Repository details for the *functor* factory.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note herefunctor |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.lang.functor (contains the main interfaces) |
| Add a note hereAdd a note hereAll of the subpackages reference some implementation of a specific type of functor. For example, the subpackage composite contains composite-type functors. | |

**Add a note hereAdd a note hereTable 9.2:** Package and class details (legend: [functor] = org.apache.commons.functor).

| **Add a note hereAdd a note hereInterface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[functor].BinaryFunction | Add a note hereAdd a note hereA function-based functor that accepts two object parameters and returns an object. |
| Add a note hereAdd a note here[functor].BinaryPredicate | Add a note hereAdd a note hereA validator-based functor that accepts two object parameters and returns a Boolean value. |
| Add a note hereAdd a note here[functor].BinaryProcedure | Add a note hereAdd a note hereA function-based functor that accepts two object parameters but returns no values. |
| Add a note hereAdd a note here[functor].Function | Add a note hereAdd a note hereA function-based functor that has no parameters but returns an object. |
| Add a note hereAdd a note here[functor].Predicate | Add a note hereAdd a note hereA validator-based functor that has no parameters but returns a Boolean value. |
| Add a note hereAdd a note here[functor].Procedure | Add a note hereAdd a note hereA function-based functor that has no parameters and returns no values. |
| Add a note hereAdd a note here[functor].UnaryFunction | Add a note hereAdd a note hereA function-based functor that accepts an object parameter and returns an object. |
| Add a note hereAdd a note here[functor].UnaryPredicate | Add a note hereAdd a note hereA validator-based functor that accepts an object parameter and returns a Boolean value. |
| Add a note hereAdd a note here[functor].UnaryProcedure | Add a note hereAdd a note hereA function-based functor that accepts an object parameter but returns no values. |

### Add a note hereAdd a note hereDecomposing a Formula as a Functor

Add a note hereAdd a note hereThe *functor* package has many interfaces that are identical to those in the *Collections* package. For example, the interface Predicate, which exists in the *Collections* package, also exists in the *functor* package. In either package, they serve the same purpose. The *Collections* package contained decorators, which alter the characteristics of the individual collection. This is very similar to functors, except that the scope of using the concept functor was restricted.

Add a note hereAdd a note hereRewriting a mathematical formula using functors is not a simple task. It is simpler to write a mathematical formula with a few decisions than to use functors. To write maintainable and extendable functors, you need to decompose the core formula into its core parts.

Add a note hereAdd a note hereLet's start with the simplest part of the royalty formula from [Listing 9.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), which is the calculation of royalties based on the simple formula defined by [Listing 9.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455).

Add a note hereAdd a note here**Listing 9.2**

Add a note hereAdd a note hereTotal Income = Royalty \* Sales \* Price per book

Add a note hereAdd a note hereTo convert the formula in [Listing 9.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) into functors, the first step is to convert the formula into reusable calculation blocks. In the case of [Listing 9.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), the formula is a series of multiplications. Therefore, the simplest reusable calculation block would be to multiply the individual blocks together. [Listing 9.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) is an example of how this could be written.

Add a note hereAdd a note here**Listing 9.3**

Add a note hereAdd a note heredouble result = royalty( priceperbook( sales()))

Add a note hereAdd a note hereThe type of formula used in [Listing 9.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) is called an *unary function*. An unary function is a function that has one parameter and returns a value. In the *functor* package, the unary function is represented by the interface UnaryFunction.

Add a note hereAdd a note hereThe next step would be to create an implementation that derives from the UnaryFunction and makes the logic of [Listing 9.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) possible. [Listing 9.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) shows how to implement the logic of [Listing 9.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455).

Larger View

Add a note hereAdd a note here**Listing 9.4**

Add a note hereAdd a note hereabstract class Formula implements UnaryFunction {

protected abstract double calculate( double input);

public Object evaluate(Object obj) {

if( obj == null) {

return new Double( calculate( Double.NaN));

}

else {

Double input = (Double)obj;

return new Double( calculate( input.doubleValue()));

}

}

}

Add a note hereAdd a note hereIn [Listing 9.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), the abstract class Formula implements the UnaryFunction interface. The interface UnaryFunction exposes one method, evaluate. The method evaluate has both an input parameter of type Object, and a return value of type Object. The object can be anything, but in the context of the royalty formula, the object represents a Double class instance.

Add a note hereAdd a note hereConverting a Double class instance to a double value can be tedious. To simplify the individual calculations, the abstract method calculate is defined. The abstract method calculate has the input and output parameter a double value.

Add a note hereAdd a note hereLooking closer at the evaluate method implementation in [Listing 9.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), notice how if the parameter obj is null, the value Double.NaN is set as a parameter. The reason for this has to do with the problem of starting the decorator calculation. Go back to [Listing 9.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455); the method sales requires no input parameter. Therefore, to ensure that the chain of calculations starts that way, the method evaluate is called with a null object, which then generates a Double.NaN value. If the value Double.NaN is used by accident in the calculation, then the entire calculation will be incorrect.

Add a note hereAdd a note hereGoing back to [Listing 9.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), three classes need to be implemented, which represent the individual parts of the calculation. [Listing 9.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) is the class implementation for the number of books sold.

Add a note hereAdd a note here**Listing 9.5**

Add a note hereAdd a note hereabstract class Market extends Formula {

private long \_totalSold;

public Market() { }

public Market( long totalSold) {

\_totalSold = totalSold;

}

protected abstract double getFactor();

public double calculate( double input) {

return \_totalSold \* getFactor();

}

}

Add a note hereAdd a note hereThe class Market in [Listing 9.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) represents the number of books sold in a particular market. In the book industry, certain markets have certain factorizations. For example, selling an English-language book in a foreign market may involve extra costs, thus reducing the overall royalty. Therefore, the implementation of the method calculate requires that the total number of books sold be multiplied by a factor. The total number of books sold is the variable \_totalSold. And the factor is represented by the abstract method getFactor. Also notice how the parameter input is ignored in the calculation. This is per the requirements defined earlier by the formula in [Listing 9.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455).

Add a note hereAdd a note here[Listing 9.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) represents the remaining implementations of the price and royalty parts of the formula.

Add a note hereAdd a note here**Listing 9.6**

Add a note hereAdd a note hereclass Royalty extends Formula {

private double \_amount;

public Royalty( double amount) {

\_amount = amount;

}

public double calculate( double input) {

return \_amount \* input;

}

}

class Price extends Formula {

private double \_amount;

public Price( double amount) {

\_amount = amount;

}

protected double calculate(double input) {

return input \* \_amount;

}

}

Add a note hereAdd a note hereIn [Listing 9.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), the classes Price and Royalty have a calculate method implementation that calculates the input with the amount of each class.

Add a note hereAdd a note hereThe individual calculations are chained together using the CompositeUnaryFunction, as is shown in [Listing 9.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455).

Larger View

Add a note hereAdd a note here**Listing 9.7**

Add a note hereAdd a note herepublic static UnaryFunction getSimpleContract(

Market market, Price price, Royalty royalty) {

CompositeUnaryFunction calculation = new CompositeUnaryFunction();

calculation.of( price).of( royalty).of( market);

return calculation;

}

Add a note hereAdd a note hereIn [Listing 9.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), the method getSimpleContract has three parameters. Each of the parameters represents one part of the formula. The thought behind the method getSimpleContract is to create a contract by stringing together a number of classes that calculate the royalty. You can use the class CompositeUnaryFunction to string the individual calculations together using the method of.

Add a note hereAdd a note hereTo calculate the royalty of a single contract, [Listing 9.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) is used.

Add a note hereAdd a note here**Listing 9.8**

Add a note hereAdd a note hereDouble value = (Double)contract.evaluate( null);

Add a note hereAdd a note hereHowever, overall contracts are not single contracts. For example, the European market has a different contract than the North American market. To finish the entire formula, we combine everything and manage it using the class defined in [Listing 9.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455).

Larger View

Add a note hereAdd a note here**Listing 9.9**

Add a note hereAdd a note hereclass Author {

Vector \_contracts = new Vector();

private double \_total;

public void addContract( Object market) {

\_contracts.add( market);

}

class Sum implements UnaryProcedure {

public void run(Object obj) {

UnaryFunction contract = (UnaryFunction)obj;

Double value = (Double)contract.evaluate( null);

\_total += value.doubleValue();

}

}

public double calculateRoyalty() {

\_total = 0.0;

Sum sum = new Sum();

CollectionAlgorithms.foreach( \_contracts.iterator(), sum);

return \_total;

}

}

Add a note hereAdd a note hereIn [Listing 9.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), the class Author represents an individual author. The author has associated contracts and can have his royalty calculated. The contracts are stored in a Vector class instance. A contract can be added using the method addContract. The input parameter market is the class instance of the interface UnaryFunction.

Add a note hereAdd a note hereIn the method calculateRoyalty, the list is iterated using the class method CollectionAlgorithms.foreach. The first parameter is an iterator that will be iterated. The second parameter is an instance of the interface UnaryProcedure, which is the class Sum. The iterator is necessary because the individual totals from each contract are added together to create an overall total. The class Sum keeps a running total and is responsible for calling the chained contract method.

Add a note hereAdd a note hereThis is where the magic of functors comes into full view. The class method Sum.run typecasts the parameter obj to the interface UnaryFunction. This is legal because, going back to [Listing 9.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), the contract that is generated by the method getSimpleContract is an UnaryFunction interface instance. Calling the UnaryInterface interface instance calls an implementation that iterates the chained formula and generates a result. In the sample contract, the result is generated using a multiplication. However, that result could have been generated using a database lookup. In either case, the caller of the functor does not care and expects only a Double class instance result. [Listing 9.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) shows how to create the individual contracts.

Larger View

Add a note hereAdd a note here**Listing 9.10**

Add a note hereAdd a note hereAuthor author = new Author();

author.addContract(

Contract.getSimpleContract( new BasicMarket( 10), new Price( 10.00),

new Royalty( 0.35)));double value = author.calculateRoyalty();

Add a note hereAdd a note hereIn [Listing 9.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), the individual contracts are created by calling the class method Contract.getSimpleContract. Notice how the individual parts of the formula are based on the instantiation of the correct class, with the correct numeric value.

### Add a note hereAdd a note hereA More Complicated *functor*-Based Royalty Formula

Add a note hereAdd a note hereCynics might say that such a lengthy explanation to carry out a formula as simple as that represented by [Listing 9.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) is an example of complicating a simple solution. This is partially correct, but an equation often masks the complexity or incompleteness of the entire solution. Remember that when you're calculating the royalty of a book, there are different levels of royalties depending on the region and price levels sold. The same equation is used, but different numbers are inserted into the equation. Those different numbers are what makes it difficult to represent a universal equation to solve all problems.

Add a note hereAdd a note hereGoing back to [Listing 9.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), the code took into account a high level of royalties and a low level of royalties that depend on the total sales count. This means that we have to decide which formula to use to do the calculation, and then alter the calculations. The first part of the formula dictates if a sales level is higher or lower than a certain level, the class ConditionalUnaryFunctor can make a decision and then can call the correct functor. [Listing 9.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) is an example of using the class ConditionalUnaryFunction to illustrate a decision.

Larger View

Add a note hereAdd a note here**Listing 9.11**

Add a note hereAdd a note hereConditionalUnaryFunction conditional =

new ConditionalUnaryFunction( new MyTestUnaryPredicate(),

new TrueValueUnaryFunction(), new FalseValueUnaryFunction());

Add a note hereAdd a note hereIn [Listing 9.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), the decision class is the predicate class MyTestUnaryPredicate. If the predicate class returns a true value, then the unary function of the class TrueValueUnaryFunction is called. Otherwise, a false value will call the unary function in class FalseValueUnaryFunction. If we go back to the royalty formula, that means a level predicate class has to be created. This is shown in [Listing 9.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455).

Add a note hereAdd a note here**Listing 9.12**

Add a note hereAdd a note hereclass SalesBreakpoint implements UnaryPredicate {

private Market \_market;

private long \_level;

public SalesBreakpoint( Market market, long level) {

\_market = market;

\_level = level;

}

public boolean test(Object object) {

if( \_market.getTotalSold() < \_level) {

return false;

}

else {

return true;

}

}

}

Add a note hereAdd a note hereIn [Listing 9.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), the class SalesBreakpoint implements the interface UnaryPredicate. The interface UnaryPredicate has one method, test. The method test is used to compare the value and then return either a value of true or false. In the implementation of the method test in [Listing 9.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), the parameter object is not used because the value of object is a Double class instance. What needs to be compared is the total sales count in a particular market. Hence, the constructor of the class SalesBreakpoint requires a Market class instance and the level used to distinguish between the higher and lower royalty rates.

Add a note hereAdd a note hereDepending on the output of the class SalesBreakpoint, a specific calculation is performed. This specific calculation is nothing more than a specific version of a contract. For example, if the sales count is below the required level, then a contract of a lower royalty is applied. This means the method getSimpleContract can be reused, except that different initial values are defined. If a total sales count is higher than a requested level, the royalty rate is calculated based on two royalty rates: higher level and lower level. This in effect means that there are two contracts. Yet again, the method getSimpleContract can be reused. The key is in being able to string the contracts together properly so that the correct royalty is generated.

Add a note hereAdd a note hereWe also have to change the calculation of the total sales count for a particular contract. The way to handle this is to create a decorator that adjusts the sales figure, as shown in [Listing 9.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455).

Larger View

Add a note hereAdd a note here**Listing 9.13**

Add a note hereAdd a note here private static Market getDecoratorMarketLow( final Market market, final long level) {

return new Market() {

private Market \_market = market;

private long \_level = level;

public double getFactor() { return 0.0; }

public Object evaluate(Object obj) {

if( \_market.getTotalSold() < \_level) {

return new Double( \_market.getTotalSold() \* \_market.getFactor());

}

else

{

return new Double( \_level \* \_market.getFactor());

}

}

};

}

private static Market getDecoratorMarketHigh( final Market market, final long level) {

return new Market() {

private Market \_market = market;

private long \_level = level;

public double getFactor() { return 0.0; }

public Object evaluate(Object obj) {

if( \_market.getTotalSold() < \_level) {

return new Double( 0.0);

}

else {

long actualCount = \_market.getTotalSold() - \_level;

return new Double( actualCount \* \_market.getFactor());

}

}

};

}

Add a note hereAdd a note hereIn [Listing 9.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), two anonymous classes that act as decorators and adjust the total sales count are created. The adjustment depends on whether the calculation is based on the high royalty figure or a low royalty figure. [Listing 9.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) strings everything together to create a contract that generates the correct royalty figure.

Larger View

Add a note hereAdd a note here**Listing 9.14**

Add a note hereAdd a note herepublic static UnaryFunction getMinimumValueContract(Market market, Price price,

Royalty royaltyHigh, Royalty royaltyLow, long level) {

UnaryFunction calcLow = Contract.getSimpleContract(

Contract.getDecoratorMarketLow( market, level), price, royaltyLow);

UnaryFunction calcHigh = Contract.getSimpleContract(

Contract.getDecoratorMarketHigh( market, level), price, royaltyHigh);

UnaryFunction highLow = Contract.getDecoratorHighLow( calcHigh, calcLow);

ConditionalUnaryFunction condition = new ConditionalUnaryFunction(

new SalesBreakpoint(market, level), highLow, calcLow);

return condition;

}

Add a note hereAdd a note hereIn [Listing 9.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), the method getMinimumValueContract has five parameters that make up the royalty calculation. In the implementation of the method getMinimumValueContract, two contracts are generated because the low sales count contract used for the combination royalty calculation can also be used as a low sales contract. The variable calcLow represents the low royalty contract. The variable calcHigh represents the high royalty contract. The variable highLow represents the combination royalty contract. Lastly, the variable condition combines the low royalty contract with the combination royalty contract to produce an overall contract. The individual contracts were created using the method getSimpleContract.

Add a note hereAdd a note hereNext, [Listing 9.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455) shows how to create the overall generic contract that represents the author.

Add a note hereAdd a note here**Listing 9.15**

Add a note hereAdd a note hereauthor.addContract(

Contract.getMinimumValueContract(

new NorthAmericanMarket( 10), new Price( 10.00),

new Royalty( 0.35), new Royalty( 0.25), 5));

author.addContract(

Contract.getMinimumValueContract(

new EuropeanMarket( 10), new Price( 10.00),

new Royalty( 0.35), new Royalty( 0.25), 5));

double value = author.calculateRoyalty();

Add a note hereAdd a note hereIn [Listing 9.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=725044455), the method addContract is called twice because a royalty total needs to be generated for both the European and North American markets. We do not need to adapt the method calculateRoyalty for this new contract because both contracts generated a UnaryFunction interface instance.

### Add a note hereAdd a note hereDesigning Functor Chains

Add a note hereAdd a note hereThe *functor* package has many different classes that help a developer create a complicated functor chain that performs some complex operation dynamically. However, it's beyond the scope of this book to discuss these classes. In addition, the classes are well documented by the generated Java Docs. What is not even close to apparent, though, is how to make use of functors. Functors are not your everyday programming constructs. Functors can make a calculation simpler or devilishly complex and unusable. Therefore, it's important that we define a strategy to design functors:

1. Add a note hereAdd a note here**Decompose a formula into its core parts that make up all variations of the calculation known at the time.** The core parts of our royalty example are royalty rate, price of book, and sales count. All of the core parts are required in all calculations, but you may tweak and adjust them to make the overall formula work. For example, we can adjust the sales count to account for the market conditions.
2. Add a note hereAdd a note here**Create a simple version of the formula.** Do not attempt to solve the entire formula in one step. Create a simple version of the formula that uses all the core parts. The objective is to hopefully find a version of the formula that can be reused in some fashion. In our royalty example, that meant generating a simple royalty calculation regardless of market or royalty adjustments.
3. Add a note hereAdd a note here**Decompose the formula in multiple smaller formulas.** In this step, the idea is to decompose a complex formula into smaller formulas that are combined. Ideally, the smaller formulas should bear some resemblance to the simple version of the formula. In our royalty rate, the individual parts were decomposed into a simple formula, which was combined into formula blocks by decorating parts of the simple formula.
4. Add a note hereAdd a note here**Combine the smaller formulas into a big formula and see if everything works.** Once the individual formula blocks have been defined, they are combined into an overall calculation that is used to calculate a result given a set of conditions.

Add a note hereAdd a note hereIn addition, remember these notes when designing your own functors:

* Add a note hereAdd a note here**Do not be afraid to decorate components.** Decorating a component of the formula is a very effective way of altering the characteristics without actually cloning or modifying the original component.
* Add a note hereAdd a note here**Do not modify the individual components of the formula.** The components are considered immutable. Modifying them may result in a very unstable and unpredictable result because, essentially, functors are state driven; if the original state is modified during the execution of the formula, other parts of the formula might change in unforeseeable ways. These sorts of bugs are extremely difficult to locate and fix. You can add other items to the state, so long as the original is kept intact.
* Add a note hereAdd a note here**Use anonymous classes and inner classes to create your own functors.** Anonymous classes are state driven and not subject to extending or modification. In addition, anonymous classes make it very easy to implement immutable objects. They are very useful when you need specific functors that are not provided by the *functor* package.

Add a note hereAdd a note here**On the CD** It is recommended that you look at the sources provided in this book's CD-ROM because they show a working copy of the royalty functor described in this chapter. The function example is found in the file at the location *[CDROM]/jseng/src/com.devspace. jseng.algorithms/FunctorText.java.*

## **Using the *lang* Extensions**

Add a note hereAdd a note hereThe *lang* package contains a number of routines that solve a number of smaller problems. These smaller problems are convenience issues. For example, Java does not support enumerations, which allow you to define types using alphabetic identifiers; the *lang* package does. We've already talked a bit about the *lang* package. In this chapter, we will discuss only aspects of this package that we haven't discussed.

### Add a note hereAdd a note hereTechnical Details for the *lang* Package

Add a note hereAdd a note here[Tables 9.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509) and [9.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509) contain the abbreviated details necessary to use the *lang* package.

**Add a note hereAdd a note hereTable 9.3:** Repository details for the *lang* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note herelang |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.lang: Contains all the utility classes needed for the smaller, tedious tasks (all of the useful classes are defined later in the chapter). |
|  | Add a note hereAdd a note hereorg.apache.commons.lang.enum: Contains the enumeration classes used to create enumerators. |
|  | Add a note hereAdd a note hereorg.apache.commons.builder: Contains the classes used to help implement the common object methods such as toString, and equals. |

**Add a note hereAdd a note hereTable 9.4:** Package and class details (legend: [lang] = org.apache.commons.lang).

| **Add a note hereAdd a note hereInterface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[lang].enum.Enum | Add a note hereAdd a note hereA class used to implement an enumeration. |
| Add a note hereAdd a note here[lang].builder.HashCodeBuilder | Add a note hereAdd a note hereA class used to simplify the implementation of the method *hashCode*. |
| Add a note hereAdd a note here[lang].builder.CompareToBuilder | Add a note hereAdd a note hereA class used to simplify the implementation of the method *compareTo*. |
| Add a note hereAdd a note here[lang].builder.EqualsBuilder | Add a note hereAdd a note hereA class used to simplify the implementation of the method *equals*. |
| Add a note hereAdd a note here[lang].builder.ToStringBuilder | Add a note hereAdd a note hereA class used to simplify the implementation of the method *toString*. |
| Add a note hereAdd a note here[lang].builder.ToStringStyler | Add a note hereAdd a note hereA class used to define a *toString* output styling. |

### Add a note hereAdd a note hereManaging Enumerations

Add a note hereAdd a note hereEnumerations do not exist in Java. The class solution has been to create a static class containing variables that represents an enumeration. The *lang* package exposes a class called Enum that is extended and makes it possible to build an enumeration. A sample shape enumeration class is defined in [Listing 9.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509).

Larger View

Add a note hereAdd a note here**Listing 9.16**

Add a note hereAdd a note hereclass ShapeEnum extends Enum {

public static final ShapeEnum CIRCLE = new ShapeEnum( "circle");

public static final ShapeEnum SQUARE = new ShapeEnum( "square");

public static final ShapeEnum TRIANGLE = new ShapeEnum( "triangle");

public ShapeEnum( String shape) {

super( shape);

}

public static ShapeEnum getEnum(String shape) {

return (ShapeEnum) getEnum(ShapeEnum.class, shape);

}

public static Map getEnumMap() {

return getEnumMap(ShapeEnum.class);

}

public static List getEnumList() {

return getEnumList(ShapeEnum.class);

}

public static Iterator iterator() {

return iterator(ShapeEnum.class);

}

}

Add a note hereAdd a note hereIn [Listing 9.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), the class ShapeEnum defines a shape enumeration. The enumeration is created by the variables CIRCLE, SQUARE, and TRIANGLE. Each of the variables is an instance of the class ShapeEnum. The instances are stored as a key value pair within the class ShapeEnum. The advantage of using this class is that using the defined methods, you can get an Iterator, List, or Map instance. Each of the instances can then be used to iterate through the content.

Add a note hereAdd a note hereIf you want, you can extend the ShapeEnum class to define a more complex shape, as defined in [Listing 9.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509).

Larger View

Add a note hereAdd a note here**Listing 9.17**

Add a note hereAdd a note hereclass ComplexShapeEnum extends ShapeEnum {

public static final ComplexShapeEnum OCTAGON =

new ComplexShapeEnum( "octagon");

public ComplexShapeEnum( String shape) {

super( shape);

}

public static ShapeEnum getEnum(String shape) {

return (ShapeEnum) getEnum(ComplexShapeEnum.class, shape);

}

public static Map getEnumMap() {

return getEnumMap(ComplexShapeEnum.class);

}

public static List getEnumList() {

return getEnumList(ComplexShapeEnum.class);

}

public static Iterator iterator() {

return iterator(ComplexShapeEnum.class);

}

}

Add a note hereAdd a note hereIn [Listing 9.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), the class ComplexShapeEnum extends the class ShapeEnum and adds the additional shape OCTAGON. The individual Map, Iterator, and List methods return a list that includes the three shapes in the class ShapeEnum, plus the extra shape defined in the class ComplexShapeEnum.

Add a note hereAdd a note hereThe example shown in [Listing 9.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509) fulfills the need for an enumeration. However, [Listing 9.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509) shows another context of showing how to define an enumeration.

Add a note hereAdd a note here**Listing 9.18**

Add a note hereAdd a note hereclass PlainEnum extends Enum {

public PlainEnum( String name) {

super(name);

}

public static Iterator iterator() {

return iterator(ComplexShapeEnum.class);

}

}

Add a note hereAdd a note hereIn [Listing 9.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), an enumerator is created, but no enumeration values are defined. There is only one method, which returns an Iterator class instance. It is not obvious what would be returned because the class PlainEnum has no enumerator values defined. The answer lies in the method iterator, which has a single parameter. The single parameter is the class descriptor ComplexShapeEnum.class. What this parameter does is retrieve all of the enumerator values associated with the class type. The iterator method then returns the enumerator values defined in [Listing 9.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), which is a list of one shape: OCTAGON. What is important about this technique is that it is possible for one enumerator to be a specific collection of enumeration values based on other enumerators. The consumer of the specific collection will not realize that the enumeration values are from another, simpler enumerator.

### Add a note hereAdd a note hereUtility Type Classes

Add a note hereAdd a note hereImagine having a Boolean string buffer with the value "true." The boolean value object is not a string-based value but a numeric value type. What is desired is to change a string "true" to a value true. There are longer, more complex ways to do the conversion, but we want a quick and easy way to do the conversion. Well, that is the purpose of the utility data classes in the *lang* package. These classes make it simpler to do the things that you often need to do but that often require a few extra steps. For reference purposes, [Listing 9.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509) converts the buffer to a Boolean value true.

Add a note hereAdd a note here**Listing 9.19**

Add a note hereAdd a note here boolean value = BooleanUtils.toBoolean( "true");

Add a note hereAdd a note hereThe *lang* package contains not only a utility class to convert a Boolean, but classes that execute many utility-type tasks. Here are all of the available utility classes:

* Add a note hereAdd a note here**ArrayUtils**: A class that contains utilities to manage arrays. The arrays that can be managed are basic value type arrays, such as double [] or int []. Objects are managed using Object [] arrays. Some definitions represent empty arrays. This is extremely useful for those methods that return arrays. Traditionally, some methods return a null, which is incorrect (they should return an empty array). Some methods clone an array, find an element, reverse the array, or check the lengths of the array.
* Add a note hereAdd a note here**BooleanUtils**: A class used to manage Boolean values. Some routines convert a string to a Boolean value or Boolean object, or vice versa. One very useful aspect of this class is that you can convert a true/false Boolean into a yes/no or on/off Boolean value.
* Add a note hereAdd a note here**CharSetUtils**: A class used to manage a String buffer. The objective of the class is to manipulate a buffer and perform specific character manipulation operations. Operations include counting, deleting, and removing specific characters or sets of characters.
* Add a note hereAdd a note here**ClassUtils**: A class used to manipulate other classes without using reflection, which is the process of figuring out the structure of a class using methods while the program is executing. Some routines find out the package name for a class or the interfaces implemented by a specific class. This class is useful for finding out specific attributes about a class in the context of dynamic plug-ins.
* Add a note hereAdd a note here**NumberUtils**: A class that provides extra functionality that complements the Java Number classes. For example, there are some static declarations for the values -1, 1, or 0. Included are methods to convert strings into BigInteger class instances. Another example is finding the maximum, or minimum, values in a set.
* Add a note hereAdd a note here**RandomStringUtils**: A class that creates a String buffer of random characters. This class is very useful when you need to create random file names or identifiers that have a specific character length.
* Add a note hereAdd a note here**SerializationUtils**: A class that helps you create serialization routines. For example, the method clone implements a deep clone. In addition, some routines help with serialization to from an array of bytes, which could be used to save to a file or buffer stream.
* Add a note hereAdd a note here**StringEscapeUtils**: A class used to escape and unescape to and from Java, JavaScript, HTML, and XML.
* Add a note hereAdd a note here**StringPrintWriter**: A PrintWrite implementation that maintains all of the written data as a String buffer.
* Add a note hereAdd a note here**StringUtils**: A class used to manage a String buffer. The difference between this class and CharSet is that this class performs operations on the overall buffer and not character sets. Some methods capitalize or abbreviate text. For example, abbreviating text allows you to provide a predetermined length of text, without cutting out a word in the middle. Other routines center a string in a buffer, find the difference between two buffers, or pad strings with spaces. Overall, the class StringUtils is extremely useful because it helps a developer clean up a text buffer using some specific rules.
* Add a note hereAdd a note here**SystemUtils**: A class to figure out what environment the JVM is operating in. For example, it tells you if the JVM is operating in 1.3 or 1.4 JDK mode, or what the classpath is. If the property cannot be determined, then an error is generated.
* Add a note hereAdd a note here**WordWrapUtils**: A class used to help in word wrapping a buffer according to specific dimensions.

### Add a note hereAdd a note hereImplementing the Method toString

Add a note hereAdd a note hereImplementing the method toString is a good programming habit. However, the good habits tend to be tedious. Within the org.apache.commons.lang.builder package are a number of classes that help the developer implement the tedious methods.

Add a note hereAdd a note hereTo implement the toString method, the class ToStringBuilder is used as shown in [Listing 9.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509) (note that part of the class has been abbreviated for clarity).

Larger View

Add a note hereAdd a note here**Listing 9.20**

Add a note hereAdd a note herepublic class BuilderBean implements java.io.Serializable {

protected int \_iValue;

protected String \_strValue;

protected ChildBean \_beanValue = new ChildBean( 2345, "childbean");

public String toString() {

return new ToStringBuilder( this)

.append( "integerValue", \_iValue)

.append( "stringValue", \_strValue)

.append( "beanValue", \_beanValue)

.toString();

}

}

Add a note hereAdd a note hereIn [Listing 9.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), the class BuilderBean has two properties: \_iValue and \_strValue. The method toString is implemented by instantiating the class ToStringBuilder. Then, the method append is repeatedly called for each property to output. Finally, to construct the string, the method toString is called. Output should be something similar to [Listing 9.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509).

Add a note hereAdd a note here**Listing 9.21**

Add a note hereAdd a note here.com.devspace.jseng.algorithms.BuilderBean@af8358[

integerValue=1234,stringValue=hello,

beanValue=com.devspace.jseng.algorithms.ChildBean@1f4689e]

Add a note hereAdd a note hereThe generated output in [Listing 9.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509) generates the class instance identifier and the individual property values. The properties integerValue and stringValue are displayed in a clear and easy-to-read format. What is not as clear is the property value of the bean property beanValue. This is because the class ChildBean has not implemented the method toString itself. Had the class ChildBean implemented a toString method, then the output from that method would have been in [Listing 9.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509) instead of in the object instance reference.

Add a note hereAdd a note hereAnother way of generating a similar output is to use the method reflectionString, as shown in [Listing 9.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509).

Larger View

Add a note hereAdd a note here**Listing 9.22**

Add a note hereAdd a note here public String toString() {

return new ToStringBuilder( this).reflectionToString( this);

}

Add a note hereAdd a note hereIn [Listing 9.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), the implementation of the method toString using the method reflectionToString is simpler than the similar implementation in [Listing 9.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509). The method reflectionToString uses reflection to figure out what properties exist and displays them like in [Listing 9.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509).

Add a note hereAdd a note here**Listing 9.23**

Add a note hereAdd a note herecom.devspace.jseng.algorithms.BuilderBean@b61fd1[

\_iValue=1234,\_strValue=hello,

\_beanValue=com.devspace.jseng.algorithms.ChildBean@1adc30]

Add a note hereAdd a note hereThe output of [Listing 9.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509) is very similar to the output of [Listing 9.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509). The major difference is the labeling of the property identifiers, which are discovered using reflection. When you use reflection to generate the properties, it would appear that the output is identical, but there is a big difference. If any of the properties have the attribute transient, then the property value will not be output using the reflection method.

Add a note hereAdd a note hereIn the constructor of the class ToStringBuilder, you can specify how the output of the string is generated. The default is to output everything on one line. However, in [Listing 9.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), the output is generated using the multiple-line string output style.

Larger View

Add a note hereAdd a note here**Listing 9.24**

Add a note hereAdd a note here public String toString() {

return new ToStringBuilder( this, ToStringStyle.MULTI\_LINE\_STYLE)

.append( "integerValue", \_iValue)

.append( "stringValue", \_strValue)

.append( "beanValue", \_beanValue)

.toString();

}

Add a note hereAdd a note hereIn [Listing 9.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), the style type is defined using the class declaration ToString-Style.MULTI\_LINE\_STYLE. The output generated is shown in [Listing 9.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509).

Add a note hereAdd a note here**Listing 9.25**

Add a note hereAdd a note herecom.devspace.jseng.algorithms.BuilderBean@b61fd1[

integerValue=1234

stringValue=hello

beanValue=com.devspace.jseng.algorithms.ChildBean@1c99159

]

Add a note hereAdd a note hereThe style MULTI\_LINE\_STYLE is one of four default styles provided by the class ToStringStyle. You can generate custom styles by instantiating the class StandardToStringBuilder and then tweaking the individual properties. For example, you can tweak the headers and footers of the output. You can't, however, instantiate the class ToStringStyle outside of the *lang* package.

### Add a note hereAdd a note hereImplementing the Method compareTo

Add a note hereAdd a note hereYou use the interface Comparable to compare two objects for equality, less than, or greater than value. Any class that implements the interface Comparable has to implement the method compareTo. [Listing 9.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509) is an implementation of the compareTo method for the class BuilderBean defined in [Listing 9.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509).

Add a note hereAdd a note here**Listing 9.26**

Add a note hereAdd a note here public int compareTo( Object o) {

BuilderBean other = (BuilderBean)o;

return new CompareToBuilder()

//.appendSuper( super.compareTo())

.append( \_iValue, other.\_iValue)

.append( \_strValue, other.\_strValue)

.append( \_beanValue, other.\_beanValue)

.toComparison();

}

Add a note hereAdd a note hereIn [Listing 9.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), the implementation of the method compareTo is similar to the implementation of the method toString. The similarity lies in the usage of a builder-type class, and the properties are appended, with a final method to perform the actual operation. The builder-type class is the class CompareToBuilder, and the final method is the method toComparison. The individual append method calls can be called on all property types. The only catch is that, in the case of the property \_beanValue, the accompanying class must also support the interface Comparable. If not then a cast exception will be generated.

Add a note hereAdd a note hereIn [Listing 9.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), the method appendSuper is commented out because the class BuilderBean does not subclass another class. However, if the class BuilderBean did, then it would be necessary to call the method appendSuper and the method super.compareTo to do a comparison with the subclasses.

Add a note hereAdd a note hereLike the ToStringBuilder class, the class CompareToBuilder has a method that uses reflection to perform the comparison, as shown in [Listing 9.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509).

Larger View

Add a note hereAdd a note here**Listing 9.27**

Add a note hereAdd a note here public int compareTo( Object o) {

return new CompareToBuilder().reflectionCompare( this, o);

}

Add a note hereAdd a note hereIn [Listing 9.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), the method reflectionCompare performs a comparison using reflection. The same rules regarding the modifier transient applies, in that the modifier will cause a property to be ignored.

### Add a note hereAdd a note hereImplementing the Method equals

Add a note hereAdd a note hereImplementing the method equals can be a challenge because people implement it in different ways. This is why the class EqualsBuilder is all the more useful, since it implements the equals method consistently. [Listing 9.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509) is a sample implementation.

Add a note hereAdd a note here**Listing 9.28**

Add a note hereAdd a note here public boolean equals(Object o) {

BuilderBean other = (BuilderBean)o;

return new EqualsBuilder()

//.appendSuper( super.equals( o))

.append( \_iValue, other.\_iValue)

.append( \_strValue, other.\_strValue)

.append( \_beanValue, other.\_beanValue)

.isEquals();

}

Add a note hereAdd a note hereIn [Listing 9.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), the method equals is implemented almost identically to the previous methods, like compareTo or toString. The only difference is the name of the implementing classes. In [Listing 9.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), the method appendSuper has been commented out. This is because the bean class subclasses the class Object. In addition, the method Object.equals returns true only if the instance identifiers are identical. This is not correct and hence requires that the Object.equals method not be called. If, however, the bean class were to subclass another class, you would need to add the method call appendSuper.

Add a note hereAdd a note here[Listing 9.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509) is an implementation of the equals method using reflection techniques.

Larger View

Add a note hereAdd a note here**Listing 9.29**

Add a note hereAdd a note here public boolean equals( Object o) {

return new EqualsBuilder().reflectionEquals( this, o);

}

Add a note hereAdd a note hereIn [Listing 9.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), the method reflectionEquals uses reflection to test if two class instances are equal. If you are unsure at all about how to implement the equals method correctly, then using the method reflectionEquals is the simplest and most consistent way to implement the equals method. Using the available reflectionEquals method is not the fastest when you use it; however, you can expect to get the correct answer.

### Add a note hereAdd a note hereImplementing the Method hashCode

Add a note hereAdd a note hereImplementing the hashCode method for a class is as simple as it is for all of the other implementations. [Listing 9.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509) is a sample implementation.

Add a note hereAdd a note here**Listing 9.30**

Add a note hereAdd a note here public int hashCode() {

return new HashCodeBuilder( 11, 21)

//.appendSuper( super.hashCode())

.append( \_iValue)

.append( \_strValue)

.append( \_beanValue)

.toHashCode();

}

Add a note hereAdd a note hereIn [Listing 9.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509), the method hashCode is implemented using the class method HashCodeBuilder. The method appendSuper should be used only if the subclass is not the class Object. The constructor for the class HashCodeBuilder requires two integer values as parameters. The first parameter represents an initial number, and the second parameter represents a multiplier. Both numbers are used to generate a unique identifier. Both numbers should also be unique to the class and be odd, not even. This means that two different class definitions should have two different numbers.

Add a note hereAdd a note here[Listing 9.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=195385509) shows how the method hashCode can be implemented using reflection.

Larger View

Add a note hereAdd a note here**Listing 9.31**

Add a note hereAdd a note here public int hashCode() {

return new HashCodeBuilder( 11, 21).reflectionHashCode( this);

}

## **Validating End-User Data**

Add a note hereAdd a note hereOne of the problems that developers have is validating data. The problem is not that it is a difficult task, but that it is tedious. Typically, data will arrive from some source and the program that needs to process the data will have to validate it. The validation might be according to a simple rule like not being null, or it could involve a more complex rule of referencing an actual client. The validation could even be a password or security validation. In any case, validation is a step that is undertaken by the process before the actual business processing.

Add a note hereAdd a note hereThe *validator* package in the Commons Bridge is a package used to perform validation. The validation is not performed in the context of a program running, but in the context of a configuration file. Essentially what happens is that the programmer is presented with a Java bean that references some data. The Java bean is sent to the *validator* package and is associated with a specific form. The form is defined within a configuration file and references the individual fields and what their values have to be. At the end of the entire validation process, a result set, which can then be sent to the end user for verification, is created.

### Add a note hereAdd a note hereTechnical Details for the *validator* Package

Add a note hereAdd a note here[Tables 9.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685) and [9.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685) contain the abbreviated details necessary to use the *validator* package.

**Add a note hereAdd a note hereTable 9.5:** Repository details for the *validator* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereCVS repository | Add a note hereAdd a note hereImage from book[jakarta-commons](http://cdcontent.books24x7.com/id_7265/jakarta-commons.zip) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note herevalidator |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note hereorg.apache.commons.validator |

**Add a note hereAdd a note hereTable 9.6:** Package and class details (legend: [validator] = org.apache.commons.validator).

| **Add a note hereAdd a note hereInterface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[validator].ValidatorResources | Add a note hereAdd a note hereA class that represents an individual or multiple configuration file(s) used to perform validation. |
| Add a note hereAdd a note here[validator].Validator | Add a note hereAdd a note hereThe main class used to perform an individual validation based on validator form. |
| Add a note hereAdd a note here[validator].ValidatorResults | Add a note hereAdd a note hereA class that represents a collection of validation results after having executed a validation. |
| Add a note hereAdd a note here[validator].ValidatorResult | Add a note hereAdd a note hereA class that represents the validation results of various validation processes for a single property. |
| Add a note hereAdd a note here[validator].Field | Add a note hereAdd a note hereA class that represents an individual field from the validator configuration file. |
| Add a note hereAdd a note here[validator].ValidatorAction | Add a note hereAdd a note hereA class that provides the front end to a validator plug-in. |

Add a note hereAdd a note hereThe complicated part of using the validator is getting everything to work. Most examples, including the ones from this book, require references to configuration files and classes that are necessary and in different locations. This means that the directories [validator]/ conf/share and [validator]/target/test-classes have to be added to the classpath. It is also required that you properly configure the *log4j* package; if it's improperly configured, simple errors such as class loading errors will not be output.

### Add a note hereAdd a note hereDefining a Validator

Add a note hereAdd a note hereFor all validation, a configuration file is required. The configuration file has two parts: validators and forms. A partial sample configuration file is shown in [Listing 9.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685).

Larger View

Add a note hereAdd a note here**Listing 9.32**

Add a note hereAdd a note here<form-validation>

<global>

<validator name="custom"

classname="com.devspace.jseng.algorithms.MyValidatorTest"

method="validateRequired"

methodParams="java.lang.Object,org.apache.commons.validator.Field"

msg="errors.required"/>

</global>

<formset>

</formset>

</form-validation>

Add a note hereAdd a note hereIn [Listing 9.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685), the validators part is represented by the XML tag global, and the forms part is represented by the tag formset. Within the XML tag global is a number of validator XML tags. Each validator XML tag defines an action that is used when a field is validated. A configuration file does not have a predefined number of validators that can be contained. Of course, physical resource limits apply. Each validator has to be uniquely identified using the attribute name.

Add a note hereAdd a note hereThe XML tag validator has the following five attributes:

* Add a note hereAdd a note here**classname**: The name of a class that is loaded and used to perform the validation.
* Add a note hereAdd a note here**method**: The name of a method that is called to perform the validation.
* Add a note hereAdd a note here**methodParams**: The parameters that the validation method requires. We'll discuss this further a bit later in this section.
* Add a note hereAdd a note here**msg**: A reference to a message identifier that is output when there are validation errors.
* Add a note hereAdd a note here**name**: A unique identifier for the file that describes the purpose of the validation. For example, "required" could mean "required field." Or, "creditcard" could mean "credit card validation."

Add a note hereAdd a note hereThe purpose of the validator definition is to cross-reference a Java class that performs some validation. The *validator* package does not define generic validators per se. Rather, it provides the infrastructure so that developers can write their own routines. The routines might not be complicated, because the *validator* package provides the basics, such as having a required field. [Listing 9.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685) shows the implemented validator that was defined in [Listing 9.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685).

Larger View

Add a note hereAdd a note here**Listing 9.33**

Add a note hereAdd a note hereimport org.apache.commons.validator.\*;

public class MyValidatorTest {

public static boolean validateRequired(Object bean, Field field) {

return true;

}

}

Add a note hereAdd a note hereIn [Listing 9.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685), the class MyValidatorTest exposes the method validateRequired, which is called when a field is to be validated. In the current implementation, a boolean value of true is returned. This means there is no validation, just an answer that everything is OK. This implementation is done on purpose to illustrate a couple of other points.

Add a note hereAdd a note hereGoing back to [Listing 9.31](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=756#756), the attribute methodParams defined two parameters, which are of the type Object and Field. Of course, the method validateRequired in [Listing 9.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685) has the same parameters in the same order. This is not random and does follow some logic. When the *validator* package processes the data, four different variables are available. These variables reference some aspect of the *validator* package. The variables are passed as parameters using their type as an identifier. The four variables are:

* Add a note hereAdd a note here**org.commons.apache.validator.ValidatorAction**: Represents the pluggable validator defined by [Listing 9.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685). Using this class, you can figure out other pluggable validator dependencies and their associated characteristics.
* Add a note hereAdd a note here**org.commons.apache.validator.Validator**: Represents the core class used to perform a validation. This class is needed when you're doing nested bean validations, because within the validation of one bean a delegation has to be made to validate another bean.
* Add a note hereAdd a note here**org.commons.apache.validator.Field**: Represents the field that is currently being validated.
* Add a note hereAdd a note here**java.lang.Object**: Represents the Java Bean that is being validated. It is assumed that the pluggable validator will cross-reference the field with the Java Bean. The *Beanutils* package is very useful for finding out the values of individual bean properties and methods.

Add a note hereAdd a note hereOf all the variables defined in the previous list, only Field and Object provide any useful information. The other two classes are useful only for extremely complex validation, which is beyond the scope of this book.

### Add a note hereAdd a note hereDefining the Form

Add a note hereAdd a note hereOnce the validators have been defined, you can define individual forms that need to be validated. The form that will be validated is the Java Bean BeanToWrite, which we have used very often in this book. The class BeanToWrite exposes two properties: stringValue and integerValue. [Listing 9.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685) shows a full configuration file that references the individual properties.

Larger View

Add a note hereAdd a note here**Listing 9.34**

Add a note hereAdd a note here<form-validation>

<global>

<validator name="custom"

classname="com.devspace.jseng.algorithms.MyValidatorTest"

method="validateRequired"

methodParams="java.lang.Object,org.apache.commons.validator.Field"

msg="errors.required"/>

</global>

<formset>

<form name="ValidateBeanToWrite">

<field property="integerValue" depends="custom">

<arg key="ValidateBeanToWrite.integerValue.displayname"/>

</field>

<field property="stringValue" depends="custom">

<arg key="ValidateBeanToWrite.stringValue.displayname"/>

</field>

</form>

</formset>

</form-validation>

Add a note hereAdd a note hereIn [Listing 9.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685), the XML tag form defines a collection of properties that will be validated. The XML tag field defines individual properties that are to be validated. The XML tag field has two attributes: name and depends. The attribute name is the identifier of the property that will be validated on the Java Bean. The attribute depends represents a comma-separated list of validators that will be executed for the property.

Add a note hereAdd a note hereSince the configuration defines only one validator with the identifier custom, the attributes depends can have only that value. Had the configuration file had multiple validators, then the attribute depends could have contained all the validators. Going back to [Listing 9.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685), the validator cross-references the Field class instance to XML tag field in [Listing 9.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685). In actuality, the *Digester* package is used to read the configuration file defined in [Listing 9.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685). Therefore, if you want to figure out what attributes are possible, refresh your memory about the *Digester* package and then look at the class Field.

Add a note hereAdd a note hereThe child XML tag arg defines a cross-reference between the property and a properties file. The properties file defines a number of definitions used when messages are displayed. The properties file could be considered a user-friendly way of outputting a validation problem. [Listing 9.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685) shows the sample properties file used.

Add a note hereAdd a note here**Listing 9.35**

Add a note hereAdd a note hereerrors.required=The {0} field is required.

ValidateBeanToWrite.integerValue.displayname=Integer Value

ValidateBeanToWrite.stringValue.displayname=String Value

Add a note hereAdd a note hereIn [Listing 9.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685), the definitions are cross-referenced with the attributes msg and key in [Listing 9.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685). When a validation issue is raised, instead of dumping the property name integerValue or \_integerValue, the text Integer Value is used.

### Add a note hereAdd a note hereValidating the Bean

Add a note hereAdd a note hereWith the configuration file and properties file properly configured and cross-referenced, you can perform some validation. [Listing 9.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685) shows a sample program that validates the class BeanToWrite.

Larger View

Add a note hereAdd a note here**Listing 9.36**

Add a note hereAdd a note here InputStream in = null;

ValidatorResources resources = null;

try {

URL url = new URL( "file:/src/classes");

URL[] urls = new URL[] { url };

ClassLoader classloader = new URLClassLoader( urls);

in = classloader.getResourceAsStream("BeanToWriteValidator.xml");

resources = new ValidatorResources(in);

} finally {

if (in != null) {

in.close();

}

}

BeanToWrite bean = new BeanToWrite();

Validator validator = new Validator(resources, "ValidateBeanToWrite");

validator.setParameter(Validator.BEAN\_PARAM, bean);

ValidatorResults results = validator.validate();

printResults(bean, results, resources);

Add a note hereAdd a note here[Listing 9.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685) contains three parts in three separate code blocks: loading of resources, initialization, and validation. The first part is the loading of the resources. The resources to be loaded are the configuration file and the properties file. The configuration file needs to be loaded as a stream; the easiest way of doing that would be to use the class URLClassLoader. Or, if a classpath configuration has already been established, you could use the *Discovery* package. Once the resource has been loaded, it is passed as a constructor parameter to the class ValidatorResources, which represent the validator resources.

Add a note hereAdd a note hereThe class ValidatorResources is a class that you can create once and reuse in different validation schemes. However, do realize the ValidatorResources in [Listing 9.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685) references only a single configuration file. To associate multiple configuration files, you should use an array of streams that represent individual configuration files. When the configuration file is loaded, the configuration file DTD is loaded as well. Therefore, it is necessary to make sure that the files are available from the classpath.

Add a note hereAdd a note hereThe second part in [Listing 9.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685) is the initialization of the validation. In this part, the validator searches the configuration file for a specific form. The specific form represents a validator that can be executed on a bean. To create a validator, the class Validator is instantiated, with the constructor parameter being the specific form to load from the validator resources. The bean is associated with the validator using the method setParameter. The method setParameter has two parameters: Validator.BEAN\_PARAM and the bean instance.

Add a note hereAdd a note hereThe validator has to be assigned the bean instance whenever a different bean is to be validated. For example, if you know that a specific bean keeps the state of the data and is used for a long time, you don't have to constantly assign it.

Add a note hereAdd a note hereThe third part in [Listing 9.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685) is validation. The method validate returns a ValidatorResults class instance. The class ValidatorResults is a result set that contains the output of the validation. The results are then passed to the method printResults, which in a successful situation contains the text, as shown in [Listing 9.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685).

Add a note hereAdd a note here**Listing 9.37**

Add a note hereAdd a note hereValidating:

\_iValue : [0] \_strvalue : [null]

integerValue[custom] (PASSED)

stringValue[custom] (PASSED)

FORM VALIDATION PASSED

Add a note hereAdd a note hereIn [Listing 9.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685), the output of the method printResults seems to be a black box that outputs some text. In the simplest analogy, that is exactly what the method printResults does. However, iterating the validation result is more than just dumping some strings to an output buffer. Each validation item result has an associated state and descriptive information that needs to be custom-processed. In simple terms, there is no notion of a simple true or false answer.

### Add a note hereAdd a note hereGenerating the Output of a Validation

Add a note hereAdd a note hereYou generate the validation output by inspecting the results in the ValidatorResults class instance. To iterate the results, first iterate the property names and then retrieve an individual result about the property. [Listing 9.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685) shows how to iterate the validation table.

Larger View

Add a note hereAdd a note here**Listing 9.38**

Add a note hereAdd a note hereIterator iter = results.getPropertyNames().iterator();

while( iter.hasNext()) {

String propertyName = (String)iter.next();

ValidatorResult result = results.getValidatorResult(propertyName);

System.out.println( "Property " + propertyName +

" validated [" + result.isValid( "custom") + "]");

}

Add a note hereAdd a note hereIn [Listing 9.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685), the property name iterator is retrieved using the method calls getPropertyNames().iterator(). Returned is an iterator of string values that represent individual property names. Then, to retrieve an individual validation result, the method getValidatorResult is called. Returned is a ValidatorResult class instance. The class instance represents the validation results of a specific property. Going back to the configuration file defined in [Listing 9.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685), the attribute depends could have multiple validations being executed. Therefore, in [Listing 9.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685), the method isValid is used to retrieve the result of an individual validation defined in the configuration file. Returned from the method isValid is a Boolean true or false value to indicate whether or not validation was successful.

Add a note hereAdd a note hereIt is possible to pretty print the output of a validation routine. The class ResourceBundle in conjunction with a properties file can be used to pretty print the output. However, such a routine is not that simple; it requires many more steps, and as such is beyond the focus of this book. For those interested, a pretty printing routine is available on this book's CD-ROM.

### Add a note hereAdd a note hereImplementing a Functional Validator Class

Add a note hereAdd a note hereThere are different ways of performing a validation. The validator used in the beginning of this section was an overall validation. However, it is possible to validate generically using a Field object instance and a Java Bean, as shown in [Listing 9.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685).

Larger View

Add a note hereAdd a note here**Listing 9.39**

Add a note hereAdd a note herepublic class MyValidatorTest {

public static boolean validateRequired(Object bean, Field field) {

try {

String value = (String)BeanUtils.getSimpleProperty( bean, field.getProperty());

return !GenericValidator.isBlankOrNull(value);

}

catch( Exception ex) {

;

}

return false;

}

}

Add a note hereAdd a note hereIn [Listing 9.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685), the class method BeanUtils.getSimpleProperty is used to retrieve the value of a property from a Java Bean. The value retrieved is stored in the variable value, which is then a parameter for the class method GenericValidator.isBlankOrNull. The method isBlankOrNull is a test used to check if the input object instance is a blank buffer or a null object instance. The return value is inverted, because if the method isBlankOrNull then a true is returned for an empty buffer and the validateRequired should return a false.

Add a note hereAdd a note hereOne item to note in [Listing 9.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=416007685) is that the *Beanutils* package is used instead of just the *validator* package. The *Beanutils* package, introduced in [Chapter 7](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=468#468), allows a developer to use reflection on an object and read the object instance class data dynamically. The method getSimpleProperty could have been replaced with a reference to the class method ValidatorUtil.getValueAsString. However, that method was not used because the *Bean-utils* package has been explained and the class ValidatorUtil uses the *Beanutils* package itself.

Add a note hereAdd a note hereThe class GenericValidator exposes various verification routines. Note that if the verification routines fail, a false is returned. For example, if isShort is called and the value is a long, then the verification will fail. In addition, in all cases, the routines manipulate a string value.

* Add a note hereAdd a note here**isBlankOrNull**: Verifies if the value is a blank or null value.
* Add a note hereAdd a note here**isByte**: Verifies if the value is a byte.
* Add a note hereAdd a note here**isCreditCard**: Verifies if the value is a valid credit card. The credit cards that can be tested are American Express, Visa, MasterCard, and Discover. For more details about the various date formats, look at the Java Docs for the class org.apache.commons. validation.CreditCardValidator.
* Add a note hereAdd a note here**isDate**: Verifies if the value is a date. For more details about the various date formats, look at the Java Docs for the class org.apache.commons.validation.DateValidator.
* Add a note hereAdd a note here**isDouble**: Verifies if the value is double.
* Add a note hereAdd a note here**isEmail**: Verifies if the value is a valid e-mail address.
* Add a note hereAdd a note here**isFloat**: Verifies if the value is a float.
* Add a note hereAdd a note here**isInRange**: Verifies if the value, which must be some type of number and not a string, is within a specific range.
* Add a note hereAdd a note here**isInt**: Verifies if the value is an integer.
* Add a note hereAdd a note here**isLong**: Verifies if the value is a long.
* Add a note hereAdd a note here**isShort**: Verifies if the value is a short.
* Add a note hereAdd a note here**isUrl**: Verifies if the value is a valid URL.
* Add a note hereAdd a note here**matchRegexp**: Verifies if the expression matches a Perl 5 regular expression. Using this module requires that the Jakarta ORO jar file be installed, which is a regular expression library.

Add a note hereAdd a note hereThe *Beanutils* package is very useful for extracting specific values from a bean instance. However, the *lang* package and its utility classes (defined earlier in this chapter) are very useful for manipulating strings. Using the string classes in the *lang* package, you can properly trim the value for consistent validation.

## **Summary**

Add a note hereAdd a note hereIn this chapter, we described three different packages: *functor*, *lang*, and *validator*, which had absolutely nothing to do with each other. It is OK that those packages had nothing to do with each other because the techniques described in this chapter are specific routines to help solve specific types of problem. In the case of functors, the specific problem was how to create maintainable complex formulas. Using functors requires more thought than using straight formulas because designing functors is not like designing objects. Functors are about decomposing the problem into smaller pieces. Without the *functor* package, the developer could still develop maintainable reusable formulas, but they would resemble functors. Using the functors, the developer has a head start and does not need to create the infrastructure himself.

Add a note hereAdd a note hereAnother kind of problem is doing the little tedious tasks that take a long time. The problem could be a simple trimming of a string buffer or the implementation of hashCode method. In either case, it is not a difficult problem to solve. The solution just takes time. And, doing your own implementation could introduce bugs that simply do not need to exist.

Add a note hereAdd a note hereFinally, the last kind of problem is user-data validation. Validation is not difficult, but it is tedious and can be error prone. Implemented incorrectly, validation can cause more bugs since specific rules can conflict with other rules. Using the *validator* package, the developer can separate the validation and the bean being validated. This means that the administrator can tweak and tune the validation. It also makes it possible to define validations as a single task, which ensures that there will be as little feature interaction as possible.

Add a note hereAdd a note here**On the CD** The sources to the concepts presented in this chapter are located under the directory *[CDROM]/jseng/src/com.devspace.jseng.algorithms*. For the section [*Defining a Validator*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=761#761) the sample configuration files are located at *[CDROM]/jseng/src/classes/BeanToWriteValidator.xml,* and *[CDROM]/jseng/src/classes/*[*applicationResources.properties*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053).

## **Questions**

Add a note hereAdd a note here**9.1:** Create an extension that is a calculator scripting language. The language could be based on lines like the Java programming language, or the scripting language could be based on elements in an XML file. The complexity of the language and how it is defined is dependent on how energetic you want to be. The objective of this exercise is to convert the scripting language into a number of functors that are executed to produce a formula. Ideally, the scripting language should, at a minimum, be able to represent the royalty formula discussed at the beginning of this chapter.

Add a note hereAdd a note here**9.2:** As part of the ongoing calculator project, clean up the extensions and application to use wherever possible the classes ArrayUtils, CharSetUtils, BooleanUtils, ClassUtils, Enumeration, NumberUtils, RandomStringUtils, Serialization-Utils, StringEscapeUtils, SystemUtils, and WordWrapUtils. For those classes not used in the calculator project, create another project and define instances where those classes could be used.

Add a note hereAdd a note here**9.3:** As part of the ongoing calculator project, tidy up the classes and implement the methods toString, compareTo, equals, and hashCode.

**Chapter 10: Distributed Internet Services**

**Add a note hereAdd a note here****Overview**

Add a note hereAdd a note hereIn this chapter, we will do the following:

* Add a note hereAdd a note hereIntroduce SOAP, WSDL, and other Web Service acronyms
* Add a note hereAdd a note hereMake a simple send and receive message request
* Add a note hereAdd a note hereUnderstand RPC and message-based Web Services
* Add a note hereAdd a note hereSend attachments using DIME

## **The Purpose of This Chapter**

Add a note hereAdd a note here"Web Services" is a misnomer. One could suppose the term was created in the hype-heavy days of the 1990s, or perhaps it's a remnant from the dot-com heyday. The term Web Services is a misnomer because Web Services do not have anything in particular to do with a browser, or an end client in specific Web Services are used to provide structure to an inherently unstructured world. Web Services do use the HTTP protocol, but they are not limited to that protocol and could be used in an email or File Transfer Protocol (FTP) context. Web Services are useful when you are solving complex problems that involve computer-to-computer communications.

Add a note hereAdd a note hereWeb Services are ways for two different computers to exchange information across a network. Physically, the two computers could be the same computer because Web Services use an abstraction to create two computers virtually but generally a network is involved. The network could be a Local Area Network (LAN) or a Wide Area Network (WAN). The task of this chapter is to explain Web Services, the underlying protocols, how they are constructed, and why we use them.

## **An Introduction to the Theory of SOAP**

Add a note hereAdd a note hereAround December 1999, the Simple Object Access Protocol (SOAP) became official after the W3C organized an XML protocol discussion mailing list. The initial version of SOAP was meant to allow a computer to exchange data using XML. Before SOAP, those types of communications were carried out using technologies such as Distributed Component Object Model (DCOM), Common Object Request Broker Architecture (CORBA), or Java Remote Method Invocation (RMI). The problem with these protocols is that they are specific to a language, vendor, or platform. Before SOAP, there was no such thing as a universal protocol. The protocols that existed around that time were intended for Intranet consumption and not the Internet. SOAP changed all that. Using SOAP, people could exchange XML data packets and nobody cared what technology, vendor, or platform the client or server used.

Add a note hereAdd a note hereBy nature, the SOAP specification has no protocol, but it was never defined that way. SOAP could be used by any other protocol, or you could use it using raw network communications. Originally when SOAP was developed, the specification used HTTP as the protocol. In the SOAP 1.2 specification, SOAP has been disentangled from an underlying protocol, and bindings to protocols have been defined. There are specifically defined bindings, but a SOAP message could be sent using the Simple Mail Transfer Protocol (SMTP). SOAP defines in an abstract sense a sender and receiver. The terms "client" and "server" are not used because in SOAP a server could send a message to the client without warning. In that scenario, the server would be the sender and the client the receiver.

### Add a note hereAdd a note hereThe SOAP Specification

Add a note hereAdd a note hereIn the simplest case, a SOAP message is identical to [Listing 10.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488).

Add a note hereAdd a note here**Listing 10.1**

Add a note hereAdd a note here<Envelope>

<Header>...</Header>

<Body>...</Body>

</Envelope>

Add a note hereAdd a note hereThe SOAP message has the following two parts:

* Add a note hereAdd a note here**Header**: Is an XML tag used to define how to describe the packet of data. This is similar to an address on an envelope you are mailing. Within the XML tag Header you can have multiple child XML tags, which depend on the SOAP infrastructure used.
* Add a note hereAdd a note here**Body**: Is an XML tag used to define the payload of the package of data. Every SOAP message must have a SOAP Body XML tag. This is similar to the letter within the envelope you are mailing. Within the XML tag Body you can have a single child XML tag, or multiple ones. Only the SOAP message processor is interested in how many child elements there are. The available tags are dependent on the data exchanged between the client and the server, or vice versa.

Add a note hereAdd a note hereThe content of the SOAP packet is stored within the XML tag Body. Within the contents of the SOAP Body tag you can have any valid XML tags that are namespace identified. Abstractly, this is like embedding an XML document within another XML document. A Web Service client or server would generate the data within the body. The Web Service infrastructure would generate the data around the body, which includes the XML tags Envelope and Header.

Add a note hereAdd a note here[Listing 10.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488) is a simple SOAP message. However, most SOAP messages would be structured like [Listing 10.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488).

Larger View

Add a note hereAdd a note here**Listing 10.2**

Add a note hereAdd a note here<?xml version="1.0" ?>

<env:Envelope xmlns:env=" http://www.w3.org/2003/05/soap-envelope ">

<env:Header>

<th:transaction xmlns:th="http://www.transaction.org/2001/12/transaction">

123

</th:transaction>

</env:Header>

<env:Body>

<math:add xmlns:math="http://www.devspace.com/2003/6/math">

<math:num>1</math:num>

<math:num>1</math:num>

</math:add>

</env:Body>

</env:Envelope>

Add a note hereAdd a note here[Listing 10.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488) is a typical SOAP message because XML namespaces are used. The SOAP 1.2 specification explicitly defines that you should use XML namespaces. XML namespaces prevent the various XML document chunks from being confused with each other.

Add a note hereAdd a note hereIn [Listing 10.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488), there are three different namespaces: env, th, and math. The env namespace is part of the SOAP specification. The th namespace is a fictitious namespace that references a transaction specification. The math namespace references a namespace defined by the author to perform mathematical additions. In each of the examples, the namespace references a URL. You might be tempted to believe that if you typed the URL into the browser something would be returned. You would be sadly mistaken, because a namespace identifier is just that, an identifier, even if it resembles a URL.

### Add a note hereAdd a note hereSOAP Headers

Add a note hereAdd a note hereIn [Listing 10.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488), the SOAP message had a SOAP Header tag. Contained within the SOAP Header tag was a child transaction XML tag. The SOAP Header tag is optional. When a SOAP Header tag is present, the contents within are translated as infrastructure data. Examples of defining a header include defining a transaction or routing information. If a particular embedded XML tag means nothing to the SOAP infrastructure, then the embedded XML tag is ignored.

Add a note hereAdd a note here[Listing 10.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488) is an example where the embedded XML tag cannot be ignored.

Larger View

Add a note hereAdd a note here**Listing 10.3**

Add a note hereAdd a note here<th:transaction

xmlns:th="http://www.transaction.org/2001/12/transaction"

env:mustUnderstand="1">

Add a note hereAdd a note hereIn [Listing 10.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488), the XML tag transaction has an attribute called mustUnderstand. Notice how the XML tag transaction and attribute mustUnderstand are associated with different namespaces. This is because the attribute mustUnderstand is part of the SOAP specification. The attribute mustUnderstand with a value of 1 indicates that the SOAP infrastructure must understand the tag, which in the case of [Listing 10.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488) is the XML tag transaction. If the tag cannot be understood, then an error is generated and returned to the sender of the message. This is akin to addressing a letter using a zip code that does not exist.

Add a note hereAdd a note hereThe sender or receiver process does not need to process the SOAP header. This is because the SOAP message could be routed or proxied using an intermediary process. The intermediary process could add, remove, or manipulate SOAP headers. However, the contents of the SOAP message aren't usually manipulated or processed.

Add a note hereAdd a note hereIf a SOAP header has to be processed by a specific SOAP intermediary, then the attribute role can be assigned as is shown in [Listing 10.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488).

Larger View

Add a note hereAdd a note here**Listing 10.4**

Add a note hereAdd a note here<th:some\_action

xmlns:th="http://www.devspace.com/2001/12/something"

env:role="http://www.devspace.com/some\_logical\_machine" ..>

Add a note hereAdd a note hereIn [Listing 10.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488), the attribute role references the name of some machine. A SOAP intermediary may process the header information, or it may even generate the header and role information. The attribute role forces particular pieces of SOAP infrastructure into particular roles. Roles could be message logging, or transaction management applications.

Add a note hereAdd a note hereThe SOAP 1.2 specification introduced the notion that SOAP headers are removed when they are processed. For example, if a SOAP intermediary processed the header in [Listing 10.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488), the header would have to be removed. This helps keep the SOAP header infrastructure simple. For example, imagine sending a SOAP message to five different computers that are connected via chained SOAP processes. This might mean that you could have excess baggage because all of the computers are adding and not removing headers. If the SOAP infrastructure explicitly removes a header when it is processed, the SOAP message is kept lean. If, however, the SOAP header needs to be kept and sent to further computers, then you could use the attribute relay as shown in [Listing 10.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488).

Add a note hereAdd a note here**Listing 10.5**

Add a note hereAdd a note here<th:some\_action

xmlns:th="http://www.devspace.com/2001/12/something"

env:role="http://www.devspace.com/some\_logical\_machine"

env:relay="true" >

Add a note hereAdd a note hereIn [Listing 10.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488), the attribute relay is assigned a value of true, which means that the header is removed when processed.

### Add a note hereAdd a note hereSOAP Encodings

Add a note hereAdd a note hereIn the SOAP 1.2 specification, the way the data is encoded is very important. In general, XML and SOAP allow the data to be encoded using XML rules. However, there are other encoding schemes, such as the SOAP encoding format. This means that you must structure the content within the SOAP Body tag using a specific format. The encoding schemes are purely optional and are defined using namespace identifiers. The purpose of an encoding is to make it simpler to translate a data type from one application platform to another. We will discuss the specifics of an encoding format later in this chapter.

### Add a note hereAdd a note hereVariations in the SOAP Response

Add a note hereAdd a note hereWhen the SOAP message causes the receiver to generate a reply, the reply must take a specific form. The reply has the exact same format as the request, except that the generated content is a response. With SOAP 1.2, a reply is the same as a message send.

Add a note hereAdd a note hereThe only special type of SOAP message is a SOAP error message, as shown in [Listing 10.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488).

Larger View

Add a note hereAdd a note here**Listing 10.6**

Add a note hereAdd a note here<?xml version="1.0" ?>

<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">

<env:Body>

<env:Fault>

<env:Code>

<env:Value>env:Receiver</env:Value>

</env:Code>

<env:Reason>

<env:Text xml:lang="en">Something happened</env:Text>

</env:Reason>

<env:Detail>

<err:message xmlns:err="http:www.devspace.com/errors">

Here is some error

</err:message>

</env:Detail>

</env:Fault>

</env:Body>

</env:Envelope>

Add a note hereAdd a note hereIn [Listing 10.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488), the SOAP error is a document within the SOAP Body tag, as specified by the Fault XML tag. The Fault tag is part of the env namespace, which is part of the SOAP specification. [Listing 10.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=461802488) contains three child XML elements that are usually there: Code, Reason, and Detail. The XML element Code defines the specific fault code of the SOAP message, which is a predefined type. The XML element Reason defines the reason of the error as a short title. You can define multiple reasons that represent different languages. Lastly, the XML element Detail represents a detailed error message that can contain other XML elements. The detailed error message should indicate what went wrong and how to correct it.

## **The WSDL Specification**

Add a note hereAdd a note hereSOAP works well in an informal context. It's as if two friends meet and talk about a cool new Web site using slang and informal lingo. However, these friends couldn't talk as informally to a judge in a courtroom. Saying, "Hey, judge. How's it hanging?" wouldn't go over too well. Likewise, when strangers exchange SOAP messages a formalized approach is required: Web Service Definition Language (WSDL).

### Add a note hereAdd a note hereWSDL Basics

Add a note hereAdd a note hereA WSDL document has the following five types:

* Add a note hereAdd a note here[**XML types**](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775): You can define the XML type of the various elements of the SOAP message. This allows you to verify the correctness of the data sent and/or received.
* Add a note hereAdd a note here**Message description**: A SOAP message is a concatenation of various logical message sections. Consider an individual SOAP message as a chapter in a book and the logical part as a paragraph within the chapter.
* Add a note hereAdd a note here**Port type**: Programmers might think that port means logical device port, but it doesn't. The port type means a combination of input and output messages. If we use a book metaphor, a port is like a part of a book that contains different chapters.
* Add a note hereAdd a note here[**Binding**](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775): This defines the specific format of the port type, including things like a protocol or data [binding](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775). If we use the book metaphor, a [binding](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775) is the entire book, including pictures, chapters, etc.
* Add a note hereAdd a note here**Service**: This defines the service, which implies that the Service tag is the entry point and puts everything together for processing.

Add a note hereAdd a note hereA WSDL document defines some type of abstract business logic that is associated with a number of SOAP messages. WSDL is the metadata and object-oriented capabilities of Web Services.

### Add a note hereAdd a note hereXML Types

Add a note hereAdd a note hereThe first step of the WSDL definition process is to define the various XML types that make up the SOAP message. This means describing the SOAP message using either an encoding or an XML Schema (the preferred way). A sample WSDL-type declaration that describes the data types in [Listing 10.2](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=782#782) is shown in [Listing 10.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775).

Larger View

Add a note hereAdd a note here**Listing 10.7**

Add a note hereAdd a note here<types>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.devspace.com/2002/1/math"

elementFormDefault="qualified">

<xs:element name=" add">

<xs:complexType>

<xs:sequence>

<xs:element name="num" ref="num" minOccurs="1"

maxOccurs="unbounded"/>

</xs:sequence>

<xs:attribute name=" math" type="xs:anyURI" use="required"/>

</xs:complexType>

</xs:element>

<xs:element name=" num">

<xs:simpleType>

<xs:restriction base="xs:byte">

<xs:enumeration value="1"/>

<xs:enumeration value="2"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

</xs:schema>

</types>

Add a note hereAdd a note hereIn [Listing 10.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775), the XML element types defines the section where the data types of the SOAP message are defined. The types are defined using the XML tag schema which means all of the embedded XML tags are XML Schema-related items.

### Add a note hereAdd a note hereMessages

Add a note hereAdd a note hereOnce you've defined the types, you need to define the message, which defines individual SOAP messages that are sent. Note that a message is a one-way message. Jumping a bit ahead of ourselves, an RPC-type SOAP call would involve two messages: from sender to receiver and from receiver to sender. [Listing 10.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775) is a sample definition of two SOAP messages.

Add a note hereAdd a note here**Listing 10.8**

Add a note hereAdd a note here<wsdl:message name="msgAdd">

<wsdl:part name="main" element="mathxsd:add"/>

</wsdl:message>

<wsdl:message name="msgAddResponse">

<wsdl:part name="main" element="mathxsd:addResponse"/>

</wsdl:message>

Add a note hereAdd a note hereIn [Listing 10.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775), the XML tag message identifies a message and has as a child element the XML tag part The XML tag part has two attributes: name and element The attribute name defines the name of the SOAP message. The attribute element references an XML element that represents the message contents that are sent. Note that in most likelihood, the attribute element references a type defined in the XML types section. [Listing 10.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775) only has one child XML tag part However, in a more complicated message-based SOAP message, there could be multiple part XML tags.

### Add a note hereAdd a note herePort Types

Add a note hereAdd a note hereA WSDL port type has absolutely nothing to do with a socket port. A WSDL port type is a way of describing an interaction between the sender and receiver of the SOAP message. The SOAP specification outlines specific ways of doing a message exchange. A response to a message just happens to be another message exchange. Using WSDL, you can define exactly what will be sent and received. This is very important; consider the case of sending a SOAP message using SMTP. With SMTP, you can't get an answer right away and hence a waiting SOAP sender will wait indefinitely. If you use WSDL, the infrastructure can determine if a response will be sent, and when. However, do not consider the port type as a physical definition of a SOAP message exchange. The port type defines an abstract message exchange mechanism. The specifics are defined in the bindings, which we will discuss later in this chapter.

Add a note hereAdd a note hereIn the WSDL messages, the messages are used to reference a conversation, as shown in [Listing 10.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775).

Larger View

Add a note hereAdd a note here**Listing 10.9**

Add a note hereAdd a note here<wsdl:portType>

<wsdl:operation name="portAddingNumbers">

<wsdl:input name="Adding" message="msgAdd"/>

<wsdl:output name="Result" message="msgAddResponse"/>

</wsdl:operation>

</wsdl:portType >

Add a note hereAdd a note hereIn [Listing 10.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775), the port is defined using the XML tag portType. Within the XML tag portType are one or multiple child operation XML tags. Each XML tag operation defines a SOAP conversation, which is defined using child input and output XML tags. An input message is a message sent from sender to receiver. An output is a message sent from receiver to sender. The XML tags input and output have two attributes: name and message. The attribute name defines the message type. The attribute message references a message from the WSDL message section. In theory, WSDL could define a whole host of conversations that have multiple inputs and multiple outputs. However, WSDL defines the following four logical conversation types (note that the XML child tag combinations of input and output are defined in the brackets):

* Add a note hereAdd a note here**One Way Operation (input)**: The sender sends a SOAP message to the receiver and does not expect a response.
* Add a note hereAdd a note here**Request Response Operation (input-output)**: The sender sends a SOAP message to the receiver, and the receiver will send a response to the message.
* Add a note hereAdd a note here**Solicit Response Operation (output)**: The receiver sends a SOAP message to the sender, and the sender will send a response. If this sounds confusing, consider this nothing more than an asynchronous callback using SOAP.
* Add a note hereAdd a note here**Notification Operation (output - input)**: The receiver sends a SOAP message to the sender, and the receiver does not expect an answer. This could also be called a "targeted broadcast" in SOAP terms.

Add a note hereAdd a note here[Listing 10.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775) is a sample Request Response-type conversation.

### Add a note hereAdd a note hereBinding

Add a note hereAdd a note hereBinding is a bit more complicated to understand than defining messages or types in a WSDL file because a binding mixes together multiple XML elements that have nothing to do with each other. Binding is a generic way of defining an actual exchange of messages. It is important to remember that the exchange of messages may or may not be SOAP related. For example, the messages that are exchanged could be HTTP based, or even XML RPC based. In the simplest case, a binding is defined in [Listing 10.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775).

Add a note hereAdd a note here**Listing 10.10**

Add a note hereAdd a note here<binding name="BindingAddition" type="portAddingNumbers">

<operation name="AddSomeNumbers">

<input message="add">

</input>

</operation>

</binding>

Add a note hereAdd a note here[Listing 10.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775) looks very similar to [Listing 10.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775), which defines a port. This occurs because the binding is fine-tuning the conversation. You can overload and override the individual conversations. [Listing 10.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775) is a typical binding definition.

Larger View

Add a note hereAdd a note here**Listing 10.11**

Add a note hereAdd a note here<binding name="bindingAddition" type="portAddingNumbers">

<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>

<operation name="AddSomeNumbers">

<soap:operation soapAction="http://www.devspace.com/add"/>

<input message="add">

<soap:body use="encoded"

namespace="http://example.com/stockquote"

encodingStyle="http://schemas.xmlsoap.org/soap/ encoding/"/>

</input>

</operation>

</binding>

Add a note hereAdd a note hereIn [Listing 10.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775), the elements associated with the namespace soap have been added. The added elements are used to define the various SOAP-specific attributes that relate to a binding. [Listing 10.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775) is saying that there is a conversation, but it is using SOAP as the messaging technique. Specifically, the XML tag soap:binding is saying that an RPC-type conversation is desired. The XML tag soap:body is saying that the SOAP content is SOAP encoded.

### Add a note hereAdd a note hereServices

Add a note hereAdd a note hereThe last item to define in a WSDL file is the service itself. The service provides the entry point to the WSDL file. From the service definition you can figure out what is being defined and how it is to be used. [Listing 10.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775) shows how to define a service for the bindings defined in the [previous section](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775).

Larger View

Add a note hereAdd a note here**Listing 10.12**

Add a note hereAdd a note here<service name="MathService">

<documentation>Want to add numbers? Here it is</documentation>

<port name="AddNumbersPort" binding="tns:bindingAddition">

<soap:address location="http://www.devspace.com.com/addnums"/>

</port>

</service>

Add a note hereAdd a note hereIn [Listing 10.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775), the service is defined by the service XML tag, with the name attribute being a unique name to identify the Web Service. Newly used is the documentation XML tag. We have avoided using this tag, but it is optional with every major WSDL item type. The documentation tag makes it possible to provide generic documentation that can be displayed using a tool. Don't confuse the documentation tag with the XML comment tag. The documentation tag is for application processing and display. Within the WSDL service tag is the port tag. The port tag is responsible for providing a physical binding to the binding defined by bindingAddition. In our binding example, SOAP and HTTP were used, but the address where the service was located was not provided. The SOAP address\_location tag provides the physical address. Again, this is an abstraction to make it simpler to separate and reuse various WSDL definitions.

### Add a note hereAdd a note hereWrapping It Up

Add a note hereAdd a note hereThe individual pieces of the WSDL have been defined, so now it is time to put everything together into one WSDL file, as shown in [Listing 10.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775).

Add a note hereAdd a note here**Listing 10.13**

Add a note hereAdd a note here<definitions name="MyAdditionExample"

targetnamespace="http://www.devspace.com/addition.wsdl"

xmlns:tns=" http://www.devspace.com/addition.wsdl"

xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"

xmlns:xsd=" http://www.devspace.com/2002/1/math"

xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

xmlns="http://schemas.xmlsoap.org/wsdl/">

...<!—Put everything into here—>

</definitions>

Add a note hereAdd a note hereWe can simply cut and paste the individual elements that were defined into the area where the XML comment says "Put everything into here." The only issue is to make sure that the namespaces are correct. WSDL and SOAP are very namespace-heavy and hence require careful attention.

Add a note hereAdd a note hereTo keep things simple, you should cut and paste the namespaces defined in [Listing 10.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775) and make appropriate changes for the namespaces targnetnamespace, tns, and xsd1. The changes in each namespace are defined by the following list:

* Add a note hereAdd a note here**targetnamespace**: This defines the namespace of your WSDL file locally, and hence should be a unique identifier.
* Add a note hereAdd a note here**xmlns:tns**: This is the same namespace of the targetnamespace except that the namespace is being assigned to a prefix.
* Add a note hereAdd a note here**xmlns:xsd**: This is the namespace definition of your types defined in terms of a Schema file, or it should be the same namespace as defined in the types section of the WSDL file.

Add a note hereAdd a note hereAnother thing you should remember when creating WSDL files is to stick with the same namespace prefixes. This makes it simpler to create and debug the various WSDL files.

Add a note hereAdd a note hereTo keep smaller WSDL files and make it possible to reuse different files, you can import sections into your WSDL file (so that you don't have to recreate them). An example of importing XML content into a WSDL is [Listing 10.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116094775).

Larger View

Add a note hereAdd a note here**Listing 10.14**

Add a note hereAdd a note here<definitions name="MyAdditionExample">

<import namespace=" http://www.devspace.com/2002/1/math"

location="http://www.devspace.com/math/math.xsd"/>

...

</definitions>

Add a note hereAdd a note hereOne last question to ask yourself when you're working with WSDL: do you create WSDL files by hand or do you have a tool to generate them? The answer is that it doesn't matter. WSDL files are going to created once and kept forever. The one problem with Web Services is that once deployed, they cannot be changed. Imagine a popular SOAP service where changing the WSDL once could cause all clients to crash. A WSDL file has to be defined once, and you need to take great care so that it is defined properly.

## **Axis Technical Details**

Add a note hereAdd a note hereThe Axis toolkit is a Java implementation of the SOAP and WSDL specifications. Using Axis, a developer can create or consume Web Services. Because of the inherent cross-platform and cross-toolkit nature of Web Services, a Web Service created by Axis can be consumed by a toolkit other than Axis running on a different platform, and vice versa.

### Add a note hereAdd a note hereDetails of Building and Configuring Axis

Add a note hereAdd a note hereWe don't recommend that you build the Axis sources because the CVS sources could contain bugs or problems. The CVS sources may be too recent and unstable, or their fixes could cause other problems. We strongly urge you to download stable builds, or if necessary stable beta or release candidate builds.

Add a note hereAdd a note hereIt's beyond the scope of this chapter to discuss the development configuration details for the Axis toolkit. However, in [Chapter 11](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=859#859), we discuss the details of managing a Web Services project.

### Add a note hereAdd a note hereHow Axis Works

Add a note hereAdd a note hereAxis is not that complicated in essence. However, there are few pieces that you have to properly organize. You can download it from [*http://ws.apache.org/axis/index.html*](http://ws.apache.org/axis/index.html). When you download a binary archive, within the archive is the directory *Image from book*[*webapps*](http://cdcontent.books24x7.com/id_7265/webapps.zip). Beneath the *Image from book*[*webapps*](http://cdcontent.books24x7.com/id_7265/webapps.zip) directory is the directory *axis*. The *axis* directory contains the Axis Web Service application. Following the installation instructions, we recommend that you copy the entire *axis* directory to a particular servlet container (this will make for an easy installation).

Add a note hereAdd a note here[Table 10.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=205815504) defines some of the major files found in the axis Web Service application.

**Add a note hereAdd a note hereTable 10.1:** Major Axis Web Service application files and purpose (legend: [axis] axis web application installation directory).

| **Add a note hereAdd a note hereFile** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note here[axis]/EchoHeaders.jws | Add a note hereAdd a note hereA sample Web Service that returns the various headers. |
| Add a note hereAdd a note here[axis]/fingerprint.jsp | Add a note hereAdd a note hereA Web page that loads, iterates, and displays the available jar files to the Axis application. |
| Add a note hereAdd a note here[axis]/happyaxis.jsp | Add a note hereAdd a note hereA Web page that should be loaded first to verify the installation of Axis. Loading this page will help you find the reason for any problems. |
| Add a note hereAdd a note here[axis]/[index.html](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) | Add a note hereAdd a note hereThe welcome page for the Axis toolkit. This page should be referenced first because the page provides links to the other pieces of functionality. |
| Add a note hereAdd a note here[axis]/web-inf/lib/[axis.jar](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) | Add a note hereAdd a note hereA jar file that contains the entire Axis application. |
| Add a note hereAdd a note here[axis]/web-inf/lib/[jaxrpc.jar](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) | Add a note hereAdd a note hereA jar file that contains the JAX RPC API definitions. |
| Add a note hereAdd a note here[axis]/web-inf/lib/[saaj.jar](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) | Add a note hereAdd a note hereA jar file that contains the SOAP and Attachments API definitions. |
| Add a note hereAdd a note here[axis]/web-inf/lib/[wsdl4j.jar](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) | Add a note hereAdd a note hereA jar file that contains the SOAP WSDL API definitions. |
| Add a note hereAdd a note here[axis]/web-inf/[web.xml](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) | Add a note hereAdd a note hereA configuration file that contains the axis details. Modify this file when you’re integrating Axis with already-existing applications. |
| Add a note hereAdd a note here[axis]/web-inf/perms.lst | Add a note hereAdd a note herePermissions of specific users using Axis-specific security. |
| Add a note hereAdd a note here[axis]/web-inf/users.lst | Add a note hereAdd a note hereA security list that contains users using Axis-specific security. |

Add a note hereAdd a note hereAxis can be integrated into any other Web application so long as you follow these rules:

1. Add a note hereAdd a note hereAxis must execute within a servlet container.
2. Add a note hereAdd a note hereAll of the jar files in the directory *[axis]/web-inf/lib* must exist in the other Web application.
3. Add a note hereAdd a note hereThe settings found in the Image from book[*web.xml*](http://cdcontent.books24x7.com/id_7265/web.xml) file must be applied in the other Web application's Image from book[*web.xml*](http://cdcontent.books24x7.com/id_7265/web.xml) file.

Add a note hereAdd a note hereThese rules provide a basic minimum Axis Web Services infrastructure. The other files and settings are optional and do not need to exist in the other Web application.

## **An Echo Web Service**

Add a note hereAdd a note hereThe impatient can very quickly create and call a generic Web Service by creating a *.jws* file. A *.jws* file is a file that represents a Web Service but that is compiled and deployed automatically. A sample *.jws* file is [Listing 10.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002).

Add a note hereAdd a note here**Listing 10.15**

Add a note hereAdd a note herepublic class EchoService {

public String doEcho( String input) {

return "Hello world " + input;

}

}

Add a note hereAdd a note here[Listing 10.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) looks just like a Java class file, and indeed it is. The extension *.jws* is used to define a compilable file, which is a Java class file. You can reconfigure the extension using the Web application configuration file *Image from book*[*web.xml*](http://cdcontent.books24x7.com/id_7265/web.xml).

Add a note hereAdd a note hereThe Web Service is called using a client application like [Listing 10.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002).

Larger View

Add a note hereAdd a note here**Listing 10.16**

Add a note hereAdd a note herepublic class Client {

static String \_endPointJWS = "http://localhost:8080/axis/EchoService.jws";

public void doEchoServiceJWS() throws Exception{

Service service = new Service();

Call call = (Call)service.createCall();

call.setTargetEndpointAddress( new URL( Client.\_endPointJWS));

call.setOperationName( new QName(

"urn:echoservice.localhost", "doEcho"));

call.addParameter( "input", XMLType.XSD\_STRING,

ParameterMode.IN);

call.setReturnType( XMLType.XSD\_STRING);

String sendText = "hello world";

System.out.println((String)call.invoke(

new Object[] { sendText }));

}

public static void main(String[] args) {

try {

Client client = new Client();

client.doEchoServiceJWS();

}

catch( Exception ex) {

System.out.println( "oops error " + ex.toString());

}

}

}

Add a note hereAdd a note here[Listing 10.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) is a sample of a manual Web Service request. The code is manual because the Web Service client request is created manually using the appropriate object and method call. When you use the code generation tools, all of the manual steps in [Listing 10.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) are encapsulated in generated easy-to-use classes.

Add a note hereAdd a note hereTo call a Web Service, you need to instantiate the class Service. This class provides the base infrastructure for all client requests. To create a Web Service and call a Call class instance, use the method createCall. Once the Call class instance has been created, you can specify various parameters that make up the Web Service request.

Add a note hereAdd a note hereThe method setTargetEndpointAddress specifies the URL where the Web Service is located. Because the server Web Service is a *.jws* file, a direct reference to the *.jws* is defined. The method setOperationName accepts a QName class instance as a single parameter. A QName is a legitimate XML term signifying an XML element and its associated namespace. Remember when we talked about the SOAP specification that namespaces have to be used. Therefore, to be able to specify an XML element and its associated namespace, the class QName is used. The only issue is that in [Listing 10.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) the namespace is not relevant because a *.jws* file is called.

Add a note hereAdd a note hereLooking back at [Listing 10.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002), the method doEcho has one parameter and a return value. Therefore, in [Listing 10.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) the client has to allocate space for both a parameter and return value. The method addParameter has three parameters: "input", XMLType. XSD\_STRING, and ParameterMode.In. The first parameter, "input", specifies the name of the parameter and must match the name of the parameter in [Listing 10.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002). The second parameter, XMLType.XSD\_String, specifies the parameter type, which in [Listing 10.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) is a string data type. It is important to match the parameter types because the marshalling code, which converts Java types to XML and vice versa, on the client and server side uses the specified parameter data types to format the data correctly. The third and last parameter, ParameterMode.In, specifies the direction of the parameter. The direction of the parameter is used to optimize network traffic. In the case of [Listings 10.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) and 10.16, the parameter input is sent from the sender to the receiver only. If the parameter were ParameterMode.INOUT, then the data would travel from the sender to the receiver and back to the sender. If the parameter were ParameterMode.OUT, then the data would travel from the receiver to the sender. To specify a return value, you use the method setReturnType. The method setReturnType has only one parameter, which specifies the return value data type.

Add a note hereAdd a note hereOnce the method call has been set up, the call to the Web Service can be made using the method call invoke. The method call invoke accepts as a parameter an array of objects that represent the individual parameters defined earlier in the program. When the invoke method completes, an object representing the returned message and associated data is automatically generated and returned to the caller of the method.

### Add a note hereAdd a note hereWatching the Data

Add a note hereAdd a note hereThe simple Web Service will do what is required of it. The simple Web Service demo is about as exciting as watching paint dry because most of the action about the Web Service is happening at a lower level. To be able to watch this lower level, you need to start a SOAP protocol spy. Within the Axis toolkit is an application called TcpMon. The application TcpMon is a port sniffer that captures requests and then delegates the requests to a destination port. TcpMon could be considered a sort of proxy, except it does nothing that useful other than sniff SOAP requests. The details of running TcpMon are outlined in [Chapter 11](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=859#859). [Listing 10.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) is the request for the simple Web Service, and [Listing 10.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) is the response from the Web Service.

Larger View

Add a note hereAdd a note here**Listing 10.17**

Add a note hereAdd a note herePOST /axis/EchoService.jws HTTP/1.0

Content-Type: text/xml; charset=utf-8

Accept: application/soap+xml, application/dime, multipart/related, text/\*

User-Agent: Axis/1.1RC2

Host: 127.0.0.1

Cache-Control: no-cache

Pragma: no-cache

SOAPAction: ""

Content-Length: 478

<?xml version="1.0" encoding="UTF-8"?>

<soapenv:Envelope

xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<soapenv:Body>

<ns1:doEcho

soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

xmlns:ns1="urn:echoservice.localhost ">

<input xsi:type="xsd:string">hello world</input>

</ns1:doEcho>

</soapenv:Body>

</soapenv:Envelope>

Larger View

Add a note hereAdd a note here**Listing 10.18**

Add a note hereAdd a note hereHTTP/1.1 200 OK

Set-Cookie: JSESSIONID=F042CD79C0C7E00ABDA5D2CF07D028CD; Path=/axis

Content-Type: text/xml; charset=utf-8

Date: Fri, 13 Jun 2003 13:25:04 GMT

Server: Apache Coyote/1.0

Connection: close

<?xml version="1.0" encoding="UTF-8"?>

<soapenv:Envelope

xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<soapenv:Body>

<ns1:doEchoResponse

soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

xmlns:ns1="urn:echoservice.localhost ">

<doEchoReturn xsi:type="xsd:string">Hello world hello world</doEchoReturn>

</ns1:doEchoResponse>

</soapenv:Body>

</soapenv:Envelope>

Add a note hereAdd a note hereIn [Listings 10.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) and 10.18, we kept the HTTP headers so that we could illustrate the information that is sent between the server and client. Notice how, in both listings, the SOAP message is wrapped in namespaces and Schema information. The soapenv namespace identifier is used to identify the SOAP message specification type, which is not SOAP 1.2, but SOAP 1.1. Also notice that the attribute soapenv:encodingStyle is used to indicate a SOAP-encoded format for the parameters. We implicitly chose this format because the Web Service is an RPC-style Web Service. The HTTP header Content-Type refers to a content payload of text/xml, which is the correct Multipurpose Internet Mail Extensions (MIME) encoding. The HTTP header Accept refers to various formats that can be generated by a Web Service. Specifically, the MIME encoding application/dime is a format of SOAP that supports attachments. One of the last items to notice is how the content data that is encapsulated by the XML tag soapenv:Body has its own custom namespace that references a namespace identifier. The namespace identifier is identical to the QName defined in [Listing 10.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002).

### Add a note hereAdd a note hereAuto-Generating the Client Stub

Add a note hereAdd a note hereWhen you're writing professional Web Service applications, the client application in [Listing 10.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) involves too much coding. Within the Axis toolkit is a tool to generate WSDL from a Web Service, and a tool to generate client or server stubs from a WSDL. For example, on the client side, the stubs contain the manual steps defined in [Listing 10.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002). Since the Web Service is already deployed and working, the objective would be to use a WSDL file to generate a client implementation. To generate a WSDL file, you can use the built-in WSDL generator. To generate a WSDL file, use the Web Service URL endpoint and add the text *?wsdl*. For example, in [Listing 10.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) the endpoint URL was *http://localhost:8080/ xis/EchoService.jws;* therefore, the WSDL URL would be *http://localhost:8080/axis/EchoService.jws?wsdl*. The generated WSDL will be returned as an XML response, which could be saved. Typically, the end user would use a browser when generating the WSDL and then save the content, which is shown in [Listing 10.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) (the details to notice are in bold).

Larger View

Add a note hereAdd a note here**Listing 10.19**

Add a note hereAdd a note here<wsdl:definitions

targetNamespace="http://localhost:8080/axis/EchoService.jws"

xmlns="http://schemas.xmlsoap.org/wsdl/"

xmlns:apachesoap="http://xml.apache.org/xml-soap"

xmlns:impl="http://localhost:8080/axis/EchoService.jws"

xmlns:intf="http://localhost:8080/axis/EchoService.jws"

xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"

xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:message name="doEchoResponse">

<**wsdl:part name="doEchoReturn" type="xsd:string"/**>

</wsdl:message>

<wsdl:message name="doEchoRequest">

<**wsdl:part name="input" type="xsd:string"/**>

</wsdl:message>

<wsdl:portType name="EchoService">

<**wsdl:operation name="doEcho" parameterOrder="input"**>

<**wsdl:input message="impl:doEchoRequest" name="doEchoRequest"/**>

<**wsdl:output message="impl:doEchoResponse" name="doEchoResponse"/**>

<**/wsdl:operation**>

</wsdl:portType>

<wsdl:binding name="EchoServiceSoapBinding" type="impl:EchoService">

<**wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/**>

<wsdl:operation name="doEcho">

<**wsdlsoap:operation soapAction=""/**>

<wsdl:input name="doEchoRequest">

<**wsdlsoap:body**

**encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"**

**namespace="http://localhost:8080/axis/EchoService.jws" use="encoded"/**>

</wsdl:input>

<wsdl:output name="doEchoResponse">

<wsdlsoap:body

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

namespace="http://localhost:8080/axis/EchoService.jws" use="encoded"/>

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

<wsdl:service name="EchoServiceService">

<wsdl:port binding="impl:EchoServiceSoapBinding" name="EchoService">

<**wsdlsoap:address location="http://localhost:8080/axis/EchoService.jws"/**>

</wsdl:port>

</wsdl:service>

</wsdl:definitions>

Add a note hereAdd a note hereIn the WSDL file defined by [Listing 10.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002), there is no WSDL types section. This is done on purpose because the types are so simple that there does not need to be a WSDL types section. The WSDL tag wsdl:message defines two SOAP messages. The identifiers are generated using the name of the Web Service method appended with the direction of the SOAP message, which is either response or request. The Web Service method is called doEcho, and hence the names of WSDL messages are doEchoRequest and doEchoResponse. The do-EchoResponse message corresponds to the return value of the initial EchoService.jws file defined in [Listing 10.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002).

Add a note hereAdd a note hereJava does not label its return parameters, so the return value is generated using a combination of the method name and the text "Return." The doEchoRequest message labels its parameter according to the identifier of the parameter in the EchoService.jws file defined in [Listing 10.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002). In each of these messages, the parameter is a string, which corresponds to the XML Schema data type xsd:string.

Add a note hereAdd a note hereFor the WSDL operation, the messages are combined into a request response message conversation. What is special is the attribute parameterOrder. This attribute is optional and specifies the order of the parameters in the SOAP method calls.

Add a note hereAdd a note hereThe WSDL tag wsdlsoap:binding tag specifies that the SOAP conversation is an RPC type using the transport specified by the HTTP [binding](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=798#798). The more dangerous tags in the sense of configuration issues are the tags wsdlsoap:body and wsdlsoap:address. These tags are problematic for configuration because they reference absolute addresses where the Web Service exists. The URL references the localhost address. For testing purposes, this is fine, but for Internet or Intranet access, this is a problem. The client-generated code will attempt to make a request on the localhost Web server, which most likely will not work. Therefore, before deploying the WSDL file, you need to update these settings. This means for development purposes that it is fine to dynamically generate a WSDL file. In production, though, all WSDL files should be made available statically.

Add a note hereAdd a note hereOnce the WSDL file has been generated, you can use the Axis tool that converts the WSDL to a number of Java files. The generated Java files perform the exact same operations as the manual client did. The difference is that the code is generated and presents an easy-to-use interface, as shown in [Listing 10.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002).

Larger View

Add a note hereAdd a note here**Listing 10.20**

Add a note hereAdd a note hereEchoServiceServiceLocator locator = new EchoServiceServiceLocator();

EchoService service = locator.getEchoService( new URL( Client.\_endPointJWS));

System.out.println( service.doEcho( "hello world"));

Add a note hereAdd a note hereThe code in [Listing 10.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) has been shrunk down to three lines of Java source code. All of the classes used in [Listing 10.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) are generated classes. The classes do follow a naming convention in that the name of the Web Service is combined with the generated class name. The class EchoServiceServiceLocator is used to locate the Web Service. To retrieve an actual instance of the Web Service, the method getEchoService is used. This method has one parameter. The single parameter represents the URL of the endpoint of the Web Service. The single parameter is optional and, if it's not specified, the WSDL-defined URL is used. The method getEchoService returns an interface instance that is identical in method signature to the Web Service implementation defined in [Listing 10.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002).

Add a note hereAdd a note hereThe Axis Web Service implements the Commons Bridge best practice when generating the Web Service implementations. This is good because it means that a factory can decide on the fly to return an actual implementation of the Web Service or a proxy, which delegates the call to the network.

### Add a note hereAdd a note hereFormal Deployment of the Web Service

Add a note hereAdd a note hereFor simple Web Services, the *.jws* extension technique we've just shown is very effective. However, it assumes that the Web Service is RPC based and has standard parameters. For anything more complex or special, it is not possible to use that type of technique. In those other situations, the Web Service has to be formally deployed. The Web Service that will be deployed is based on [Listing 10.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) and is shown in [Listing 10.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002).

Add a note hereAdd a note here**Listing 10.21**

Add a note hereAdd a note herepublic class Echo {

public String doEcho( String input) {

return "Your Message: (" + input + ")";

}

}

Add a note hereAdd a note hereTo formally deploy a Web Service, you need to create a *.wsdd* file. In the SOAP and WSDL specification, there is a hole: the definition of the implementation details. For example, the WSDL file specifies the parameters, messages, and URL of the Web Service. However, it doesn't specify how to cross-reference the Web Service WSDL file with the Java class files. When the *.jws* file was deployed and copied to the server, the cross-reference between the WSDL file and the implementation happened automatically. [Listing 10.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002) is a sample *Image from book*[*deploy.wsdd*](http://cdcontent.books24x7.com/id_7265/deploy.wsdd) file.

Larger View

Add a note hereAdd a note here**Listing 10.22**

Add a note hereAdd a note here<deployment name="Echo"

xmlns="http://xml.apache.org/axis/wsdd/"

xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"

xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">

<service name="Echo" provider="java:RPC">

<parameter

name="className" value="com.devspace.jseng.ws.simple.Echo"/>

<parameter name="allowedMethods" value="doEcho"/>

</service>

</deployment>

Add a note hereAdd a note hereIn [Listing 10.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002), the deployment file is defined using the XML tag deployment Within the XML tag deployment can be one or multiple XML service tags. Each XML tag service represents an individual Web Service that is to be deployed. A basic Web Service is defined using the attributes name and provider The attribute name defines the basic identifier for the Web Service catalog on the Servlet container. The attribute provider defines the type of Web Service, which can either be java:RPC or java:MSG. Since the Web Service being deployed is RPC based, the java:RPC identifier would be appropriate.

Add a note hereAdd a note hereWithin the XML tag service can be multiple parameter XML tags. Each parameter XML tag has a name and value attribute. The name attribute references an identifier parameter. In [Listing 10.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002), two parameter identifiers are used. The parameter identifier className defines the class that is loaded and represents an XML RPC Web Service. The parameter identifier allowedMethods defines a list of comma-separated methods that accept Web Service requests.

Add a note hereAdd a note hereTo remove a deployed Web Service, you need to create an undeployment file like the one shown in [Listing 10.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002).

Larger View

Add a note hereAdd a note here**Listing 10.23**

Add a note hereAdd a note here<undeployment name="Echo" xmlns="http://xml.apache.org/axis/wsdd/">

<service name="Echo"/>

</undeployment>

Add a note hereAdd a note hereIn [Listing 10.23](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002), the XML tag undeployment is used to undeploy a file. Within the XML tags can be multiple service XML tags. Each service XML tag represents a service that is to be undeployed. The deployment and undeployment files are manipulated using a program called the *AdminClient*, which is a Java class file. It is invoked using a command line similar to [Listing 10.24](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=485357002).

Larger View

Add a note hereAdd a note here**Listing 10.24**

Add a note hereAdd a note herejava org.apache.axis.client.AdminClient [deployment file or undeployment file]

Add a note hereAdd a note hereProgress messages are output, and a successful deployment or undeployment will result in a final success message.

## **Extending the Web Service to Include Bean References**

Add a note hereAdd a note hereWhen creating an Axis Web Service, you don't have to rely on the base data types like an integer, double, or string. Axis Web Services can also expose more complicated Bean-type structures. Bean-type structures can be defined using WSDL, but a simpler way is to define the methods and beans using Java, generate a WSDL file, and then generate the Java bindings. In all frankness, this sounds like an extremely long-winded way to create a Web Service. However, there is reason behind this madness, so to speak.

### Add a note hereAdd a note hereGenerating the WSDL File

Add a note hereAdd a note hereAs already mentioned, the Axis toolkit contains a WSDL-to-Java generator, whose purpose is to read a number of compiled reference Java classes that define a Web Service and generate a WSDL file. You use Java classes to define a WSDL file because it is simpler to express a Web Service using Java. It is not beyond the realm of possibilities to define a Web Service using WSDL, but it does require you to understand WSDL and how to convert Java types to XML schema types.

Add a note hereAdd a note hereUsing this process of separating the design of the Web Service from the implementation is a design and implementation decision and is very useful. A developer can first develop the Web Service and ensure a good design before any implementation is started. In fact, you can use another toolkit called *gSoap* to generate the WSDL files. The *gSoap* toolkit gives the developer good control over the generated WSDL files without actually coding in WSDL. The tool provided within Axis that converts Java to WSDL is useful, but it doesn't allow as much fine tuning as *gSoap*. Working with Web Services in a production scenario and using only the Axis provided tool requires tweaking to make the Web Service do what you want. You might therefore ask, why not code directly in WSDL? The answer is that coding in WSDL without an appropriate editor can be very tedious and error-prone. The Axis WSDL or the *gSoap* WSDL generator tool allows developers to focus on the design issue in a programming language that they familiar with.

Add a note hereAdd a note hereWe will now program a sample Web Service, a Calculator service that adds and subtracts two numbers. The numbers are stored in a Java Bean. In a specific version of the Web Service that will be defined, the result of the operation is also stored in the Java Bean. [Listing 10.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) is the Java Bean.

Add a note hereAdd a note here**Listing 10.25**

Add a note hereAdd a note herepublic class Number {

private long \_value1;

private long \_value2;

private long \_result;

public long getValue1() {

return \_value1;

}

public void setValue1( long value) {

\_value1 = value;

}

public long getValue2() {

return \_value2;

}

public void setValue2( long value) {

\_value2 = value;

}

public long getResult() {

return \_result;

}

public void setResult( long value) {

\_result = value;

}

}

Add a note hereAdd a note hereIn [Listing 10.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the class Number has three properties: value1, value2 and result. The properties value1 and value2 are used to store the two numbers that will either be added or subtracted. The property result is used to store the value of either the addition or subtraction. [Listing 10.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) is the definition of the Web Service.

Add a note hereAdd a note here**Listing 10.26**

Add a note hereAdd a note herepublic class Calculator {

public long add( Number num) {

}

public void subtract( Number num) {

return;

}

}

Add a note hereAdd a note hereIn [Listing 10.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the class Calculator defines a Web Service that has two methods: add and subtract. In the Axis documentation, the recommendation is to define the Web Service using interfaces, but [Listing 10.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) shows it is not necessary. The method add adds the properties value1 and value2 and returns the result as a long value. The method subtract subtracts the properties value1 from value2 and stores the result in the property result.

Add a note hereAdd a note hereTo use the WSDL generator, you have to compile [Listings 10.25](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) and 10.26. Then, to generate the WSDL file, the WSDL generator consumes the compiled class Calculator. The class Calculator manipulates the class Number, so the WSDL generator will trace the class dependencies and generate the necessary dependent class descriptors. [Listing 10.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) shows the WSDL Schema type that is generated from the class Number.

Larger View

Add a note hereAdd a note here**Listing 10.27**

Add a note hereAdd a note here<wsdl:types>

<schema

targetNamespace="calculator.devspace.ws"

xmlns="http://www.w3.org/2001/XMLSchema">

<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>

<complexType name="Number">

<sequence>

<element name="result" type="xsd:long"/>

<element name="value1" type="xsd:long"/>

<element name="value2" type="xsd:long"/>

</sequence>

</complexType>

</schema>

</wsdl:types>

Add a note hereAdd a note hereIn [Listing 10.27](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the class Number is converted into the XML Schema complex type Number. The XML tag element represents an individual property from the Number class. Had any of the properties been a reference to another bean, then that bean would have been added to the types list. Essentially what happens is that Axis will convert the bean class definition into a standard XML Schema declaration.

Add a note hereAdd a note hereHowever, there is a problem with the generation, because [Listing 10.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) defines a class method that retrieves the input from the same object that is returned to the caller. The WSDL generator does not understand this logic and hence the generated WSDL has an error, as shown in [Listing 10.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167).

Add a note hereAdd a note here**Listing 10.28**

Add a note hereAdd a note here <wsdl:message name="subtractRequest">

<wsdl:part name="num" type="impl:Number"/>

</wsdl:message>

<wsdl:message name="addResponse">

<wsdl:part name="addReturn" type="xsd:long"/>

</wsdl:message>

<wsdl:message name="addRequest">

<wsdl:part name="num" type="impl:Number"/>

</wsdl:message>

<wsdl:message name="subtractResponse">

</wsdl:message>

Add a note hereAdd a note hereIn [Listing 10.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), four messages are defined. Going back to [Listing 10.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the class Calculator exposed two methods. Remembering our method of WSDL mapping, a SOAP conversation is the exchange of two SOAP messages. Therefore, [Listing 10.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) is correct in defining four messages, since two methods times two is four.

Add a note hereAdd a note hereLet's focus on Listing 10.26's method add. The input parameter was a Number and the return value was a long. In [Listing 10.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the request message addRequest had one part to the message, which was the XML Schema type impl:Number. The response message addResponse has one part, which is the XML Schema type long. The method add is converted correctly.

Add a note hereAdd a note hereWhat is incorrectly converted is the method subtract from [Listing 10.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167). The logic used for this method is that the response is stored in the incoming object. The input message subtractRequest described in [Listing 10.28](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) defines the XML Schema type impl:Number, which is correct. However, what is incorrect is that the message subtractResponse has no message parts. This means that the WSDL generator did not pick up on the logic. This is logical because the WSDL generator cannot assume incoming objects will always be returned. Such an assumption could be a performance bottleneck. To solve the problem of the missing return object, add a message part to the WSDL file. This is shown in [Listing 10.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167).

Add a note hereAdd a note here**Listing 10.29**

Add a note hereAdd a note here <wsdl:message name="subtractResponse">

<wsdl:part name="num" type="impl:Number"/>

</wsdl:message>

Add a note hereAdd a note hereIn [Listing 10.29](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the wsdl message part that is added is identical to the message subtractRequest. This small fix might make you think that the Axis toolkit is not as useful as it could be. This is extremely far from the truth. The problem of the missing returned object is not due to the Axis toolkit, but due to the fact that the Java programming language is limited in its Web Service expressions. The Axis toolkit is much better at generating Java code from a WSDL file than generating a WSDL file from Java code. Other languages are not better because Web Services have very rich expression capabilities, like multiple return data types, that traditional programming languages cannot express. Now you'll see why we referred to the Open Source *gSoap* Web Service toolkit earlier. *gSoap* is a C and C++ Web Service toolkit and is beyond the scope of this book. However, the WSDL generator for the *gSoap* toolkit is a special language that is based on a superset of the C header syntax. Using this special language, you can very easily define any kind of Web Service construct, without having to manually write WSDL code.

### Add a note hereAdd a note hereGenerating the Java Stubs

Add a note hereAdd a note hereOnce the WSDL file has been generated, you generate the Java class stubs using the same process as described in the [*Generating the WSDL file*](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) section earlier in this chapter. However, this time we'll use the server stubs as well, instead of just using the client-side files. [Listing 10.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) shows the generated server stub file with implemented logic.

Larger View

Add a note hereAdd a note here**Listing 10.30**

Add a note hereAdd a note herepublic class MathSoapBindingImpl implements

ws.devspace.calculator.Calculator

{

public long add(ws.devspace.calculator.Number num)

throws java.rmi.RemoteException

{

return num.getValue1() + num.getValue2();

}

public void subtract(ws.devspace.calculator.holders.NumberHolder num)

throws java.rmi.RemoteException

{

num.value.setResult( num.value.getValue1() - num.value.getValue2());

}

}

Add a note hereAdd a note hereIn [Listing 10.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the generated server stub is a concatenation of the Web Service name and the identifier SoapBindingImpl that generates the class name MathSoapBindingImpl. The methods add and subtract have been implemented as a simple addition and subtraction operation except the results in the subtraction are assigned to an object.

Add a note hereAdd a note here[Listing 10.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) is also a round trip in Java programming terms. [Listing 10.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) represents the Java class file that generates the WSDL file that generates the Java class file in [Listing 10.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167). Using logic, you have to functionally define the classes in [Listings 10.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) and 10.30.

Add a note hereAdd a note hereThe classes defined in [Listings 10.26](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) and 10.30 are not functionality identical because the method subtract uses a different data type in [Listing 10.30](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167). The reason has to do with our modification of the original WSDL file and the inability of programming languages like Java to cope without parameters.

Add a note hereAdd a note hereWhen a parameter is passed to a method in Java, the original value of the parameter can be read and modified, but the original value will not be passed back to the caller. Consider for example [Listing 10.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), which allocates an object and attempts to return the object instance on the calling stack.

Add a note hereAdd a note here**Listing 10.31**

Add a note hereAdd a note here public void subtract( Number num) {

num = new Number();

return;

}

Add a note hereAdd a note hereIn [Listing 10.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the parameter num is an input parameter that is assigned a new object instance. After the method subtract returns, the object will be automatically ready for garbage collection, because Java does not have a notion of an output parameter. The parameter num is input only on the calling stack and cannot be reassigned. However, there is a trick in that a holder class is used to return the object instance. Consider [Listing 10.32](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), which defines the class NumberHolder.

Add a note hereAdd a note here**Listing 10.32**

Add a note hereAdd a note herepublic class NumberHolder {

public Number reference;

}

Add a note hereAdd a note hereThe class NumberHolder contains a public data member, which references an object of data type Number If we modify [Listing 10.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the allocator in [Listing 10.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) results.

Add a note hereAdd a note here**Listing 10.33**

Add a note hereAdd a note here public void subtract( NumberHolder holder) {

holder.num = new Number();

return;

}

Add a note hereAdd a note hereIn [Listing 10.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the input parameter is now the data type NumberHolder. This time, when the class Number is allocated, it is assigned to a data member. This means that the calling stack is not modified and the allocated object instance is automatically returned.

Add a note hereAdd a note hereNow, let's go back to [Listing 10.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167). The data type NumberHolder is in essence identical to the class NumberHolder defined in [Listing 10.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167). The term Holder is used to denote that the class "holds" a reference to another class, which is the identifier before the term Holder. In [Listing 10.33](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), this means it is the class Number The idea of a "holder" class is not new and has been used in the Object Management Group (OMG) bindings for Java and the Java XML RPC bindings.

### Add a note hereAdd a note hereDeploying the Web Service

Add a note hereAdd a note hereDeploying a generated Web Service is no different than deploying a hand-coded Web Service, as we've previously shown. The major difference is that more generated items are written to the deployment file. [Listing 10.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) shows a simple deployment file generated by the WSDL-to-Java generator.

Larger View

Add a note hereAdd a note here**Listing 10.34**

Add a note hereAdd a note here<deployment

xmlns="http://xml.apache.org/axis/wsdd/"

xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

<service name="Math" provider="java:RPC" style="rpc" use="encoded">

<parameter name="wsdlTargetNamespace" value="calculator.devspace.ws"/>

<parameter name="wsdlServiceElement" value="CalculatorService"/>

<parameter name="wsdlServicePort" value="Math"/>

<parameter name="className"

value="ws.devspace.calculator.MathSoapBindingSkeleton"/>

<parameter name="wsdlPortType" value="Calculator"/>

<parameter name="allowedMethods" value="\*"/>

<parameter name="scope" value="Session"/>

<typeMapping

xmlns:ns="calculator.devspace.ws"

qname="ns:Number"

type="java:ws.devspace.calculator.Number"

serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

/>

</service>

</deployment>`

Add a note hereAdd a note hereIn [Listing 10.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the number of parameter XML elements have increased; there is also an additional typeMapping XML element. The added parameter XML elements all relate to the WSDL mapping when SOAP messages are received and sent. The XML element typeMapping is used to automatically serialize and deserialize the bean using the standard Axis serializers. Consider this as an Axis *betwixt*-type mapping.

Add a note hereAdd a note hereThe deployment file in [Listing 10.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) does not contain any configuration information regarding method names or data type mappings. This is because the class referenced by the parameter class, ws.devspace.calculator.MathSoapBindingSkeleton is a generated delegator class that provides the mapping information. [Listing 10.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) shows the static constructor of the class MathSoapBindingSkeleton that maps the methods.

Larger View

Add a note hereAdd a note here**Listing 10.35**

Add a note hereAdd a note here static {

org.apache.axis.description.OperationDesc \_oper;

org.apache.axis.description.FaultDesc \_fault;

org.apache.axis.description.ParameterDesc [] \_params;

\_params = new org.apache.axis.description.ParameterDesc [] {

new org.apache.axis.description.ParameterDesc(

new javax.xml.namespace.QName("", "num"),

org.apache.axis.description.ParameterDesc.IN,

new javax.xml.namespace.QName("calculator.devspace.ws", "Number"),

ws.devspace.calculator.Number.class, false, false),

};

\_oper = new org.apache.axis.description.OperationDesc("add", \_params,

new javax.xml.namespace.QName("", "addReturn"));

\_oper.setReturnType(

new javax.xml.namespace.QName(

"http://www.w3.org/2001/XMLSchema", "long"));

\_oper.setElementQName(

new javax.xml.namespace.QName("calculator.devspace.ws", "add"));

\_oper.setSoapAction("");

\_myOperationsList.add(\_oper);

if (\_myOperations.get("add") == null) {

\_myOperations.put("add", new java.util.ArrayList());

}

((java.util.List)\_myOperations.get("add")).add(\_oper);

\_params = new org.apache.axis.description.ParameterDesc [] {

new org.apache.axis.description.ParameterDesc(

new javax.xml.namespace.QName("", "num"),

org.apache.axis.description.ParameterDesc.INOUT,

new javax.xml.namespace.QName("calculator.devspace.ws", "Number"),

ws.devspace.calculator.Number.class, false, false),

};

\_oper = new org.apache.axis.description.OperationDesc("subtract", \_params, null);

\_oper.setElementQName(

new javax.xml.namespace.QName("calculator.devspace.ws", "subtract"));

\_oper.setSoapAction("");

\_myOperationsList.add(\_oper);

if (\_myOperations.get("subtract") == null) {

\_myOperations.put("subtract", new java.util.ArrayList());

}

((java.util.List)\_myOperations.get("subtract")).add(\_oper);

}

Add a note hereAdd a note hereIn [Listing 10.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the methods used look extremely similar to those used in [Listing 10.16](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=812#812), where the Web Service parameters had to be manually coded. The reason for this is that in this case the generated code uses the exact same way of defining parameters and methods as the manual code. The difference is that [Listing 10.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) is automatically generated based on the WSDL file.

Add a note hereAdd a note hereHowever, there is a problem. A bit further in the MathSoapBindingSkeleton, you typically find an instantiation to the MathSoapBindingImpl class like that shown in [Listing 10.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167).

Larger View

Add a note hereAdd a note here**Listing 10.36**

Add a note hereAdd a note here public MathSoapBindingSkeleton() {

this.impl = new ws.devspace.calculator.MathSoapBindingImpl();

}

Add a note hereAdd a note hereThe problem with [Listing 10.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) is that a hard-coded MathSoapBindingImpl object allocation has been defined. [Listing 10.31](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) added code to the generated file MathSoapBindingImpl. This is a problem because it means that when a WSDL change requires a new generation of the sources, one of two things can happen. Either the WSDL-to-Java generator can overwrite the already existing MathSoapBindingImpl class and force a deletion of all modifications, or the WSDL-to-Java generator knows not to overwrite and generates new class files for all of the other generated files. While the second solution seems better, the problem is that the developer will have to manually add or modify the methods in the MathSoapBindingImpl class. Either case requires tweaking of the class files. A better solution would be to remap the class reference in [Listing 10.36](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) to a custom Calculator interface implementation instance. The problem of the WSDL-to-Java generation wiping out the changes still exists, but at least the modification is a single line of code and nothing is lost in the update. The advantage of this approach is that it entirely separates the Axis-generated files and custom-coded files. This also allows you to reuse the custom-coded files in other contexts.

Add a note hereAdd a note hereThe WSDL-to-Java generator can also generate a different stub binding. In this modified stub binding, the details of the binding are stored in the deployment file, as shown in [Listing 10.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167).

Larger View

Add a note hereAdd a note here**Listing 10.37**

Add a note hereAdd a note here<service name="Math" provider="java:RPC" style="rpc" use="encoded">

<parameter name="wsdlTargetNamespace" value="calculator.devspace.ws"/>

<parameter name="wsdlServiceElement" value="CalculatorService"/>

<parameter name="wsdlServicePort" value="Math"/>

<parameter name="className"

value="ws.devspace.calculator.MathSoapBindingImpl"/>

<parameter name="wsdlPortType" value="Calculator"/>

<operation name="add" qname="operNS:add"

xmlns:operNS="calculator.devspace.ws" returnQName="addReturn"

returnType="rtns:long" xmlns:rtns="http://www.w3.org/2001/XMLSchema" >

<parameter name="num" type="tns:Number"

xmlns:tns="calculator.devspace.ws"/>

</operation>

<operation name="subtract" qname="operNS:subtract"

xmlns:operNS="calculator.devspace.ws" >

<parameter name="num" type="tns:Number"

xmlns:tns="calculator.devspace.ws" mode="INOUT"/>

</operation>

<parameter name="allowedMethods" value="add subtract"/>

<parameter name="scope" value="Session"/>

<typeMapping

xmlns:ns="calculator.devspace.ws"

qname="ns:Number"

type="java:ws.devspace.calculator.Number"

serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

/>

</service>

Add a note hereAdd a note hereIn [Listing 10.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the additional XML elements operation define a method and its associated parameters. The child XML element parameter identifies a parameter for the method. The definitions created by the XML elements operation replace the declarations created in [Listing 10.35](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167). In the default mode of the WSDL-to-Java generator, the definitions of operations and parameters are stored in the deployment file.

Add a note hereAdd a note hereThe deployment file of [Listing 10.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) has a parameter name className that references the class ws.devspace.calculator.MathSoapBindingImpl. The other deployment file defined in [Listing 10.34](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), in contrast, references a delegator that references the class MathSoapBindingImpl. When the deployment file generates the operation and parameter description information, the class MathSoapBindingSkeleton is not generated. This then raises the problem of adding implementation code to a generated piece of code. The solution is to change the deployment parameter classname to the custom interface implementation.

Add a note hereAdd a note hereRegardless of the generated deployment file's form, the registration of the Web Service is carried out the same way and uses the *AdminClient* tool.

### Add a note hereAdd a note hereApplying Scope to a Web Service Request

Add a note hereAdd a note hereMaking a Web Service request can involve a stateless call like HTTP. This means a request is made, some data is processed, and a response is received. If another request were made, then the Web Service would by default not realize that the same client is making a second call. This results in the inability to manage the state on the server. To get around this problem, Axis uses the concept of scope. A scope is a way of defining how objects and states will be remembered. Axis uses the following three scopes defined:

|  |  |  |
| --- | --- | --- |
|  | **Note** | Add a note hereAdd a note hereNote that the term "Web Service object" refers to the parameter name className in the deployment file, which in [Listing 10.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) is ws.devspace.calculator.MathSoapBindingImpl. |

* Add a note hereAdd a note here**Request**: In this scope, every request causes a new Web Service object to be instantiated. This type of scope is identical to the HTTP stateless protocol, where no state is saved on the server side.
* Add a note hereAdd a note here**Session**: In this scope, every request from the same instantiated client-side object that is returned from the locator object is sent to the same object. If the locator object instantiates two different client-side object instances, then there will be two different Web Service object conversations.
* Add a note hereAdd a note here**Application**: In this scope, the Web Service object is instantiated only once and all requests, regardless of from which client, communicate to that single instantiated object. This scope could also be considered as a singleton.

Add a note hereAdd a note hereThe scope of the object is defined in the deployment file, which is a result of the WSDL-to-Java generator. Referring back to [Listing 10.37](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the parameter name scope has a value of Session. Using the Session or Application scope enables a developer to create private data members that could represent a database connection or running total type data.

### Add a note hereAdd a note hereConfiguration Data

Add a note hereAdd a note hereIf you use the information you've learned from previous chapters, it would be easy to construct a singleton object using a generic factory or a discovery mechanism. The singleton would contain the configuration information necessary for any further configuration. However, there is a way of getting configuration information from the deployment file. [Listing 10.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167) shows a modified version of the deployment file that contains a configuration item (note that the actual deployment file has been abbreviated for clarity).

Larger View

Add a note hereAdd a note here**Listing 10.38**

Add a note hereAdd a note here <service name="Echo" provider="java:RPC">

<parameter name="myProperty" value="Here is what you said "/>

</service>

Add a note hereAdd a note hereIn [Listing 10.38](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), a custom parameter name myProperty has been defined. This custom parameter has a notation like the rest of the parameters. The difference is that the custom parameter has a user-defined identifier. To be able to read this parameter or any other parameter, use [Listing 10.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167).

Larger View

Add a note hereAdd a note here**Listing 10.39**

Add a note hereAdd a note herepublic class Echo {

public String doEcho( String input) {

MessageContext msgCtxt = MessageContext.getCurrentContext();

return msgCtxt.getStrProp( "myProperty") + input;

}

}

Add a note hereAdd a note hereIn [Listing 10.39](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=724857167), the class method Echo.doEcho has been extended to read the initial buffer string using the configuration file. The class method MessageContext.getCurrentContext is used to retrieve the current context object, which is the data type MessageContext. The class MessageContext references the current SOAP message in its raw form and other infrastructure bits. To retrieve the property value, the method getStrProp is used.

## **Using Web Service Attachments**

Add a note hereAdd a note hereUsing attachments on a Web Service is not that complicated because it only involves defining a new data type. We can best illustrate the new data type by looking at the deployment configuration file shown in [Listing 10.40](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=459562119).

Larger View

Add a note hereAdd a note here**Listing 10.40**

Add a note hereAdd a note here<deployment xmlns="http://xml.apache.org/axis/wsdd/"

xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

<service name="File" provider="java:RPC">

<parameter name="className" value="com.devspace.jseng.ws.file.File"/>

<parameter name="allowedMethods" value="\*"/>

<operation name="upload" qname="operNS:upload"

xmlns:operNS="urn:file.devspace.ws"

returnQName="returnqname" returnType="rtns:long"

xmlns:rtns="http://www.w3.org/2001/XMLSchema">

<parameter name="dh" type="ns1:DataHandler"

xmlns:ns1="urn:file.devspace.ws"/>

</operation>

<typeMapping xmlns:ns1="urn:file.devspace.ws"

deserializer="org.apache.axis.encoding.ser.JAFDataHandler-DeserializerFactory"

languageSpecificType="java:javax.activation.DataHandler"

qname="ns1:DataHandler"

serializer="org.apache.axis.encoding.ser.JAFDataHandler-SerializerFactory"

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>

</service>

</deployment>

Add a note hereAdd a note hereIn [Listing 10.40](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=459562119), the deployment file has a different definition of the method parameter than when using method calls that use standard Java data types. The parameter name dh has a value of ns1:DataHandler. The qname of the XML element typeMapping is identical to the parameter name dh. This means that when the parameter dh is serialized to and from the SOAP message, the serialization used is the factories JAFDataHandlerDeserializerFactory and JAFDataHandlerSerializerFactory. The factories instantiate classes that serialize Direct Internet Message Encapsulation (DIME) encoded data.

Add a note hereAdd a note hereWhat is fundamental about this change in data mapping is that in theory you can process any kind of data by simply defining the correct mapping. And, in this context, the XML element typeMapping resembles the *betwixt* mapping file because it specifies how to read and write the data.

Add a note hereAdd a note hereTo support the reading and writing of the DIME encoded data, the server has to use a special class that contains the data. [Listing 10.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=459562119) is the Web Service implementation that can process an attachment.

Larger View

Add a note hereAdd a note here**Listing 10.41**

Add a note hereAdd a note herepublic class File {

public long upload( DataHandler dh) throws Exception {

if( dh != null) {

InputStream iis = (InputStream)dh.getContent();

if( iis != null) {

FileOutputStream fos;

ObjectOutputStream oos;

fos = new FileOutputStream( "/home/cgross/temp/build.xml");

byte rawData[] = new byte[ iis.available()];

iis.read( rawData);

fos.write( rawData);

fos.close();

return 0;

}

else {

System.out.println( "Ooops not a stream");

}

}

return -1;

}

}

Add a note hereAdd a note hereIn [Listing 10.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=459562119), the class File has a single parameter, which is the data type DataHandler. The class DataHandler is from the activation package and contains the attachment data as a stream. The stream is then read as an input stream using the method getContent. From there, the input stream is converted to an array of bytes that are written to a file. The stream can represent a file, a serialized Java object, or a picture. The stream can be any stream of bytes.

Add a note hereAdd a note here[Listing 10.41](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=459562119) expects a stream of bytes, which is a file sent by the client as shown in [Listing 10.42](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=459562119).

Larger View

Add a note hereAdd a note here**Listing 10.42**

Add a note hereAdd a note here private static void sendFile( String filename, String url) throws Exception {

DataHandler dhSource = new DataHandler( new FileDataSource( filename));

org.apache.axis.client.Service service = new org.apache.axis.client.Service();

org.apache.axis.client.Call call = (org.apache.axis.client.Call) service.createCall();

call.setTargetEndpointAddress( new URL(url));

call.setOperationName(

new QName("file.devspace.ws","upload") );

QName qnameAttachment =

new QName("file.devspace.ws", "DataHandler");

call.registerTypeMapping(dhSource.getClass(),

qnameAttachment, JAFDataHandlerSerializerFactory.class,

JAFDataHandlerDeserializerFactory.class);

call.addParameter( "dh", qnameAttachment,

ParameterMode.IN );

call.setReturnType( new QName(

"http://www.w3.org/2001/XMLSchema", "long"));

call.invoke( new Object[] { dhSource } );

}

Add a note hereAdd a note hereThe client code in [Listing 10.42](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=459562119) uses a coding notation that is similar to the manual coding example ([Listing 10.16](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=812#812)). There are a couple of differences, though. The first difference is the usage of the class DataHandler to load a file. Another difference is what is new in this manual coding example: the usage of the method registerTypeMapping. In the deployment file of the server, DIME encoded data is managed using some special classes. On the client side, though, there is no deployment file to define a mapping. Therefore, the mapping needs to be defined using code like that in [Listing 10.42](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=459562119). Notice that the parameters of the method registerTypeMapping use the same data types as the deployment files JAFDataHandlerSerializerFactory and JAFDataHandlerDeserializerFactory. Then, to invoke the Web Service, the method invoke is called using an array of objects.

Add a note hereAdd a note hereWhen using attachments, you need to remember that they use special processors. DIME attachments are a standard, but they need special processing, which is not always supported. Using custom serialization works if the developer controls the sender and receiver. However, custom serialization may not work if the sender and receiver have small differences in implementation. It is similar to when a British English speaker and American English speaker hear the word "subway." Yes, both know the word, but there is a difference in what it represents to each person. A British person will consider a subway to be an underpass where pedestrians can walk underneath some object. An American will consider a subway to be a form of city transportation, or what a British person considers the "tube" or "underground." It's therefore very important when you're developing Web Services that you never ignore the topic of interoperability. Doing so could result in huge integration problems at a later point in time

## **Writing Messaging-Type Applications**

Add a note hereAdd a note hereMessaging applications are entirely different from RPC-type messages. Messaging applications are based on the processing of XML documents. This means that there are no parameters and no object mappings. The developer is left to his own devices and needs to think of a different processing strategy. This is not a bad thing because messaging applications are better able to deal with variable data. A messaging application is coded in a similar way as a manually coded Web Service. One big difference that distinguishes a message-based Web Service and the Web Services defined thus far in the chapter is the deployment file.

Add a note hereAdd a note here[Listing 10.43](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039) is a sample messaging deployment file.

Larger View

Add a note hereAdd a note here**Listing 10.43**

Add a note hereAdd a note here<deployment name="test" xmlns="http://xml.apache.org/axis/wsdd/"

xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"

xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">

<service name="Message" provider="java:MSG">

<parameter name="className" value="com.devspace.jseng.ws.msg.Msg"/>

<parameter name="allowedMethods" value="echoElements"/>

</service>

</deployment>

Add a note hereAdd a note hereIn [Listing 10.43](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039), the deployment file is the smallest yet, and has only two lines at the basic minimum. This is entirely acceptable because messaging-based Web Services are very easy to configure. What is different in this deployment file is the attribute provider, which has a value of java:MSG The java:MSG value means that the Web Service is messaging- based and that all of the methods exposed by the Web Service have a specific method signature. The two parameter XML elements are similar to those used in previous deployment files in this chapter.

Add a note hereAdd a note here[Listing 10.44](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039) is the implementation of the messaging Web Service (note that the class has been abbreviated for clarity).

Add a note hereAdd a note here**Listing 10.44**

Add a note hereAdd a note herepublic class Msg {

public Element[] echoElements(Element[] elems) {

Element[] result = new Element[ elems.length];

for ( int i = 0 ; i < elems.length ; i++ ) {

Element bodyNode = elems[ i];

contentString = "";

parseCurrentNode( bodyNode);

result[ i] = XMLUtils.StringToElement(

"urn:courseJWS", "result", contentString);

}

return( result );

}

}

Add a note hereAdd a note hereIn [Listing 10.44](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039), the method echoElements has an input parameter that is an array of the Element, and an output that is an array of the Element. The data type was not declared in the deployment file; in addition, there is no descriptor, because when a messaging Web Service is written using Axis, Axis expects that the Web Service consumes XML nodes. The class Element is an XML node with child nodes and attributes. In terms of an RPC processing-type architecture, the raw XML is the step before an RPC Web Service converts the data to Java objects. The individual Element class instances can be iterated and manipulated like another XML document. If desired, the developer could use the *betwixt* or *digester* package to process the XML nodes. In [Listing 10.44](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039), the individual XML documents are processed using the method parseCurrentNode. The method parseCurrentNode looks at the individual nodes and their content and returns a new XML document.

Add a note hereAdd a note hereThe client application is shown in [Listing 10.45](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039).

Larger View

Add a note hereAdd a note here**Listing 10.45**

Add a note hereAdd a note hereclass Client {

public Element buildNode( String namespace, String text) throws Exception {

Element embedded = XMLUtils.StringToElement( namespace, "greeting", text);

Document doc = embedded.getOwnerDocument();

Element element = doc.createElementNS( "default", "wrapper");

element.appendChild( embedded);

return element;

}

public Element buildNode2( String identifier) throws Exception {

Document doc = XMLUtils.newDocument();

Element element = doc.createElementNS( "default", "usertext");

Text textElement = doc.createTextNode( identifier);

element.appendChild( textElement);

return element;

}

public String doit(String[] args) throws Exception {

Options opts = new Options(args);

opts.setDefaultURL("http://localhost:8081/axis/services/Message");

Service service = new Service();

Call call = (Call) service.createCall();

call.setTargetEndpointAddress( new URL(opts.getURL()) );

SOAPBodyElement[] input = new SOAPBodyElement[3];

input[0] = new SOAPBodyElement( buildNode( "urn:English", "Hello"));

input[1] = new SOAPBodyElement( buildNode( "urn:French", "Bonjour"));

input[2] = new SOAPBodyElement( buildNode2( "Some text"));

Vector elems = (Vector) call.invoke( input );

SOAPBodyElement elem = null ;

Element e = null ;

elem = (SOAPBodyElement) elems.get(0);

e = elem.getAsDOM();

String str = "Res elem[0]=" + XMLUtils.ElementToString(e) + "\n";

elem = (SOAPBodyElement) elems.get(1);

e = elem.getAsDOM();

str = str + "Res elem[1]=" + XMLUtils.ElementToString(e) + "\n";

elem = (SOAPBodyElement) elems.get(2);

e = elem.getAsDOM();

str = str + "Res elem[2]=" + XMLUtils.ElementToString(e) + "\n";

return( str );

}

public static void main(String[] args) throws Exception {

String res = (new Client()).doit(args);

System.out.println(res);

}

}

Add a note hereAdd a note here[Listing 10.45](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039) is a bit longer than the code for coding the Web Service client manually, shown in [Listing 10.16](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=812#812). [Listing 10.45](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039) contains some old methods and some new ones. What's old is the preparation of the method call using the class Service and the method createCall. What's new is the use of the class SOAPBodyElement. In the specifications of the SOAP message, it was said that SOAP messages could contain multiple content elements. With RPC, there is only one content element. The class SOAPBodyElement represents an individual content element. In [Listing 10.45](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039), three body elements are created. The constructor to the class SOAPBodyElement accepts an XML document. The XML document is created using the methods buildNode and buildNode2. The individual SOAPBodyElement class instances are stored in an array that is passed to the method invoke. The method invoke takes the individual content elements and generates a SOAP message that is sent to the Web Service provider. [Listing 10.46](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039) illustrates the sent SOAP message.

Larger View

Add a note hereAdd a note here**Listing 10.46**

Add a note hereAdd a note here<?xml version="1.0" encoding="UTF-8"?>

<soapenv:Envelope

xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<soapenv:Body>

<ns1:wrapper xmlns:ns1="default">

<ns2:greeting xmlns:ns2="urn:English">Hello</ns2:greeting>

</ns1:wrapper>

<ns3:wrapper xmlns:ns3="default">

<ns4:greeting xmlns:ns4="urn:French">Bonjour</ns4:greeting>

</ns3:wrapper>

<ns5:usertext xmlns:ns5="default">Some text</ns5:usertext>

</soapenv:Body>

</soapenv:Envelope>

Add a note hereAdd a note hereIn [Listing 10.46](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039), notice how each of the XML elements wrapper and usertext are wrapped in some namespace. The namespaces are created when the XML DOM method createElementNS is used. The Web Service server defined in [Listing 10.44](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039) would iterate the three documents and generate three new XML documents as a response. The response is shown in [Listing 10.47](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039).

Larger View

Add a note hereAdd a note here**Listing 10.47**

Add a note hereAdd a note here<?xml version="1.0" encoding="UTF-8"?>

<soapenv:Envelope

xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<soapenv:Body>

<ns1:result xmlns:ns1="urn:courseJWS">

Node name is (ns2:greeting)Text value is (Hello)

</ns1:result>

<ns2:result xmlns:ns2="urn:courseJWS">

Node name is (ns4:greeting)Text value is (Bonjour)

</ns2:result>

<ns3:result xmlns:ns3="urn:courseJWS">

Text value is (Some text)

</ns3:result>

</soapenv:Body>

</soapenv:Envelope>

Add a note hereAdd a note hereIn [Listing 10.47](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039), the returned XML documents are represented by the XML tag result. The results bear no resemblance to the input XML documents. That is OK and indicates the freeform nature of the XML documents. However, this does a raise a problem in the types of WSDL files, as shown in [Listing 10.48](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039).

Larger View

Add a note hereAdd a note here**Listing 10.48**

Add a note hereAdd a note here<wsdl:types>

<schema targetNamespace="">

<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>

<element name="echoElements" type="xsd:anyType"/>

<element name="echoElementsReturn" type="xsd:anyType"/>

</schema>

</wsdl:types>

Add a note hereAdd a note hereIn [Listing 10.48](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039), the schema XML tag references two XML tags element. The XML tag element is the definition of the input and output data types. In [Listing 10.48](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=822780039), the data types are defined to be xsd:anyType, which means that the SOAP message can contain any kind of data. This is a problem because different Web Service implementations will translate the WSDL types differently. It does not mean that messaging Web Services are incompatible across implementations. What it means is that messaging Web Services require more tweaking to make the sender and receiver be able to communicate and exchange messages. You can use an XML Schema to define specific data types, but that would require manual definition. At that point, it would be very beneficial to use an XML editor like XMLSpy. XMLSpy is an excellent tool to manipulate XML Schemas and WSDL files using an XML IDE. However, XMLSpy is not Open Source and costs money.

### Add a note hereAdd a note hereWhen to Use Messaging or RPC

Add a note hereAdd a note hereIt is absolutely important to remember that both messaging Web Services and RPC techniques are useful. To illustrate the differences, consider the processing of a mortgage application. A mortgage application involves the buyer finding a home, contacting a bank, filling out a mortgage application, making a bid, getting the mortgage application approved, and signing the house purchase contract.

Add a note hereAdd a note hereFinding a home could be as simple as looking in a newspaper or surfing the Web. In the case of surfing the Web, the homebuyer would define a query (such as price restrictions or location), submit the query, and get a result set. In this case, the best type of Web Service to use would be an RPC Web Service. The query and response type of Web Service, which depends only on the parameters sent by the query, is an easy way to manage RPC Web Service requests. No external information is required by the Web Service Server to process the query. Therefore, the infrastructure required to process the Web Service request is simple and manageable.

Add a note hereAdd a note hereOnce the buyer has found a house, he will make an appointment with his bank and discuss the possibility of purchasing the house. At that point, the banker will start a mortgage application and will enter the specific details regarding the house. The banker will enter details about the person (or persons) applying for the mortgage as well as details about the house. At this point, the banker is creating a portfolio of information that may or may not be identical for another mortgage application. In classical terms, the banker creates a folder with the name of the application and then adds the papers related to the mortgage. The banker and others who approve or inspect the mortgage add their pieces of paper and increase the complexity of the mortgage folder. At this point, a messaging Web Service would be better to use because the XML folder could be sent from one Web Service to another. Each Web Service would serve as a workflow station and perform some logic on the XML document. Of course, when the data moves between the workflow stations it would be stored in an XML database.

## **Summary**

Add a note hereAdd a note hereWeb Services are an interesting new technology that is used to process XML documents and perform XML RPC techniques. Using Web Services makes it possible to build a more sophisticated distributed processing architecture that can cope with the old and the new techniques. This chapter is only the beginning; you are strongly urged to read more information about Web Services and what they offer.

Add a note hereAdd a note hereWow! You have almost made it to the end of this book. This means you have digested all of the software engineering principles and programming techniques presented. It also means that our discussion of Axis ends our discussion of programming.

Add a note hereAdd a note here**On the CD** The sources to the concepts presented in this chapter are located under the directories *[CDROM]/jseng/src/com.devspace.jseng.ws.file, [CDROM]/jseng/src/com.devspace.jseng.ws.msg, [CDROM]/jseng/src/com.devspace.jseng.ws.simple,* and *[CDROM]/jseng/src/com.devspace.jseng. ws.store.* To be able to run the programs the Tomcat server has to be installed *([CDROM]/ downloads/tomcat-4.1.24.tar.gz)*, and the Axis toolkit *([CDROM]/downloads/axis-1.1.tar.gz)* has to be added to the Tomcat installation as a Web Application. To make things simple with respect to the installation of Axis, the directory *[CDROM]/axis* represents a fully functional Web Application.

## **Questions**

Add a note hereAdd a note here**10.1:** Define a WSDL file that corresponds to the same method and parameter signature as the calculator extensions defined in previous questions. The trick in this question is that not only should the extension be defined, but a mechanism of connecting to a specific Web Service extension implementation also has to be defined. Consider this extra trick as being some type of factory method for Web Services. The objective of the WSDL definition is to be able to use the Web Service as if it were a calculator, where a properly programmed client could assemble and execute the questions dynamically, like the calculator application currently can. The difference is that a network exists between the consumer and implementation.

Add a note hereAdd a note here**10.2:** Implement the WSDL file defined in question 10.1. For the time being, keep the different operations simple, like adding and subtracting.

Add a note hereAdd a note here**10.3:** Generate the client stubs of the WSDL file defined in question 10.1. Of all questions in the entire book, this is the trickiest. If the WSDL is properly defined, then the generated stubs should be identical to the extensions defined in the calculator application. The objective of this question is to structure the calculator application such that whenever a calculation is to be performed, the extension is dynamically retrieved.

Add a note hereAdd a note hereThe application will have a configuration, batch file processing, etc., that stays the same. What changes is how the implementation of the calculator operation is executed. In this question, the implementation could be local, it could be a library that needs to be defined remotely, or it could be based on a Web Service defined remotely. In the configuration file, the ability to define a factory for other Web Services is then dynamically assigned.

Add a note hereAdd a note hereIt is important that you spend some time designing this part of the application because after this step, your calculator will be complete and should be fully operational. Then, in the implementation, anybody could use your WSDL file to define and integrate their Web Service.

Add a note hereAdd a note here**10.4 (Optional)** If these exercises are part of a bigger group effort, then exchange WSDL files and attempt to use other Web Service implementations. The key here is not to update your WSDL file, but rather figure out a strategy of how to integrate, which may or may not include updating the WSDL file.

**Chapter 11: Project Management**

**Add a note hereAdd a note here****Overview**

Add a note hereAdd a note hereIn this chapter, we will do the following:

* Add a note hereAdd a note hereDefine how to manage a project
* Add a note hereAdd a note hereTest an application using *JUnit*
* Add a note hereAdd a note hereManage the build process using Ant
* Add a note hereAdd a note hereUse a version control system to manage source code

**The Purpose of This Chapter**

Add a note hereAdd a note hereWriting software is not that complicated, because all that is needed is an editor and a compiler. The editor is used to write Java source code, and the compiler is used to convert the source code into a Java byte code. A problem is when multiple Java source code files are involved because that introduces complexity. Complexity begets bugs and longer development times, which means some mechanism needs to be developed to stop the bugs and manage the development process.

Add a note hereAdd a note hereWhen we move the software development process into a more abstract concept, the word "project" comes to mind. Managing a project involves managing the build process, managing sources, and defining a way to test the project. Getting those three parts of the project under control makes it much simpler to determine the state of the project. Therefore, the task of this chapter is to explain how to manage a project.

**Managing a Project**

Add a note hereAdd a note hereWhen you're managing a software project, you need to keep the following three parts of the project under control:

* Add a note hereAdd a note here**Test management**: The process of managing the test programs that execute routines to validate or invalidate the correctness of the software being constructed.
* Add a note hereAdd a note here**Build management**: The process of managing how a piece of software is compiled, distributed, and tested. This is the overall process that brings everything together.
* Add a note hereAdd a note here**Source code management**: The process of managing the software sources using a program that allows multiple developers to concurrently update and manipulate individual source code files without major conflict issues.

Add a note hereAdd a note hereThe tools used to manage these three tasks are:

* Add a note hereAdd a note here***JUnit***: A test management library.
* Add a note hereAdd a note here**Ant**: A build management tool.
* Add a note hereAdd a note here**CVS**: A source code management tool.

Add a note hereAdd a note hereYou could also use other tools such as Maven (build tool) or Subversion (source code tool), but because they are less widely used, we will not discuss them here.

Add a note hereAdd a note hereThe objective of project management is to make the entire build and test processes painless. The QA team and auditor should be able to control every aspect of the project and see when things are going out of control. Developers can define their own processes for individual parts of the software, but these processes need to fit into the overall project.

Add a note hereAdd a note hereA good project management system is one that can run on autopilot. The build file should be able to control all aspects without any intervention by the developer. This will allow a code auditor to create a build server and track the stability of the project. A *build server* is a computer or sets of computers that do nothing more than manage the entire build and source processes.

## **Test Management Using *JUnit***

Add a note hereAdd a note hereThe package *JUnit* is a tool used to perform *unit tests*, which are tests executed on a piece of code that test an individual piece of functionality. The problem with generating unit tests is that it takes time to generate the individual pieces of code. Therefore, many people do not end up writing any tests. This is a shame and leads to buggy code. Alternatively, some people do implement unit tests but write their own framework. That framework might not be comprehensive enough and will still lead to buggy code. The *JUnit* package is a framework used to write unit tests. The *JUnit* package does not write the tests for you, but it provides an infrastructure to write the tests.

Add a note hereAdd a note hereDevelopers often wonder how many tests to write. Some people think that the design should be driven by the tests, whereas other people have other ideas. The opinion of the author is that tests should not drive the design of the software. Tests are used to validate design ideas.

Add a note hereAdd a note hereThe engineering industry and especially the car industry understand proper testing of designs. Consider a car. A car has thousands of parts. A car and its associated design are not driven by the tests because complete testing would result in a car that would not be affordable. Instead, the car designers rely on the car engineers and statistics to validate the quality of the car. Yes, as a result, recalls sometimes happen, and some people end up with lemons. But history has shown it is an effective way of weighing the costs of design and production versus the quality of the car.

Add a note hereAdd a note hereThe engineering world copes with how much testing has to be carried by complying with standards and defining contexts. For example, a water pump used to pump water to your lawn most likely will not have undergone that much testing. However, a water pump used to pump coolant liquids for the space shuttle will undergo much more testing. Even though the design of the water pump for the lawn and the shuttle is probably identical, the pump for the shuttle is absolutely critical. Of course, even the shuttle might suffer catastrophic failures; however, those failures are millions of factors lower than the failures of the water pump for the lawn. The difference in engineering levels is not comparable. Therefore, when constructing your own tests, consider the context. Will your software be executing as a water pump for the lawn or for the space shuttle? If you know which pump your software is, then you will also know how exhaustive your tests have to be. Remember also that the more tests there are, the less flexibly and more slowly your development will proceed. This is because you need time to digest the results of the various tests and to design new tests for new pieces of functionality.

### Add a note hereAdd a note hereTechnical Details for *JUnit*

Add a note hereAdd a note here[Tables 11.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=167024619) and [11.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=167024619) contain the abbreviated details necessary to use the *JUnit* package

**Add a note hereAdd a note hereTable 11.1:** Repository details for the *JUnit* package.

| **Add a note hereAdd a note hereItem** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note hereDownload repository | Add a note hereAdd a note hereDetails found at [*www.junit.org*](http://www.junit.org) |
| Add a note hereAdd a note hereDirectory within repository | Add a note hereAdd a note herejunit |
| Add a note hereAdd a note hereMain packages used | Add a note hereAdd a note herejunit.framework (the core classes and infrastructure); junit.awtui, junit.swingui, and junit.textui (contain the control programs that kickstart the individual tests) |

**Add a note hereAdd a note hereTable 11.2:** Package and class details of the *JUnit* toolkit.

| **Add a note hereAdd a note hereClass/Interface** | **Add a note hereAdd a note hereDetails** |
| --- | --- |
| Add a note hereAdd a note herejunit.framework.TestCase | Add a note hereAdd a note hereA neutral interface used to instantiate objects |
| Add a note hereAdd a note herejunit.framework.TestSuite | Add a note hereAdd a note hereThe main class that contains all of the functions used to instantiate the different implementations of the Factory interface |
| Add a note hereAdd a note here[lang].functor.FactoryException | Add a note hereAdd a note hereThe exception thrown if the Factory classes have encountered a problem |

### Add a note hereAdd a note hereBuilding a Simple Test Case

Add a note hereAdd a note hereWhen you're developing test cases, the objective is to split the design into a main test program, modules that are tested, and then individual tests to be executed. Listing 11.1 is the main test program.

Add a note hereAdd a note here**Listing 11.1**

Add a note hereAdd a note hereclass MyRunner extends TestRunner {

public void startTests() {

TestSuite suite = new TestSuite();

suite.addTest( SampleTest.suite());

doRun(suite, false);

}

}

public class ProjectManagement {

public static void main(String args[]) {

MyRunner runner = new MyRunner();

runner.startTests();

}

}

Add a note hereAdd a note hereIn Listing 11.1, the class ProjectManagement is some main class that is executed when the program starts. The class ProjectManagement instantiates the class MyRunner. The class MyRunner subclasses the class TestRunner. In most source code examples of *JUnit*, very few cases subclass the class TestRunner. Instead, the class is instantiated directly. The reason for the subclassing is that it is easier to tweak certain parameters and settings (which will be shown later) than to do a direct instantiation.

Add a note hereAdd a note hereIn the implementation of the method startTests the class TestSuite is instantiated. The class TestSuite references all of the tests that will be executed, and monitors the individual progress of the tests. To add a test, the method addTest is called. The single parameter of the method addTest is a Test interface instance. Once all of the tests have been added, you can execute the tests by calling the method doRun. The method doRun has two parameters: suite and false. The parameter suite is the reference object instance to the suite of tests to run. The flag false indicates whether or not the test application should wait until after the tests have been executed.

Add a note hereAdd a note hereRemember back to [Chapter 9](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=706#706), where we discussed the concept of a functor. The functor had the ability to string together a number of classes that represent a single formula. The interface Test that was passed to the method addTest does the same thing as a functor, but instead strings tests that need to be executed together. In abstract terms, when a test is in fact a delegation to other tests, then that test is called a *module*. Listing 11.2 is the module test implementation.

Add a note hereAdd a note here**Listing 11.2**

Add a note hereAdd a note herepublic class SampleTest extends TestCase {

public static Test suite() {

TestSuite suite= new TestSuite();

suite.addTest( SampleTestPiece.suite());

return suite;

}

}

Add a note hereAdd a note hereIn Listing 11.2, the class SampleTest extends the class TestCase. The class TestCase represents the test case infrastructure. The method suite returns a collection of tests that are added to the main test program in Listing 11.1. The class SampleTest is a module-level test and references other tests. The final individual tests are shown in Listing 11.3.

Add a note hereAdd a note here**Listing 11.3**

Add a note hereAdd a note herepublic class SampleTestPiece extends TestCase {

public void testRoutine() throws Exception {

System.out.println( "Something is tested");

}

protected void setUp() throws Exception {

System.out.println( "Setting up test");

}

protected void tearDown() throws Exception {

System.out.println( "Removing test resources");

}

public static Test suite() {

return new TestSuite( SampleTestPiece.class);

}

}

Add a note hereAdd a note hereIn Listing 11.3, the class SampleTestPiece like the module level class SampleTest extends the class TestCase. The difference with the SampleTestPiece class is that instead of performing another delegation to other tests, there are tests to execute. The method suite here is like the method suite in Listing 11.2, which adds a number of tests to the main test application. However, the difference is that the individual tests are extracted dynamically using the class TestSuite. The class TestSuite constructor parameter is a class descriptor to a specific class. The class TestSuite will dynamically iterate all of the methods of the class and extract the test methods. Test methods have the identifier test prepended to the method name, like the method testRoutine. To add a suite dynamically, you could have also used the class method TestSuite.addTestSuite.

Add a note hereAdd a note hereThe method setUp is an overridden method to initialize the class before the tests start. Some initialization operations could be the creation of database connections or the preloading of a configuration file. The method tearDown is also an overridden method, but it is used to undo and destroy the resources created in the method setUp.

### Add a note hereAdd a note hereDefining Tests Statically

Add a note hereAdd a note hereIn Listing 11.3, the tests routines were dynamically extracted. You can also statically define a test, as is shown in Listing 11.4.

Add a note hereAdd a note here**Listing 11.4**

Add a note hereAdd a note herepublic class AnotherTestPiece extends TestCase {

public AnotherTestPiece( String test) {

super( test);

}

public void testRoutine() throws Exception {

System.out.println( "A test");

}

public static Test suite() {

TestSuite suite= new TestSuite();

suite.addTest( new AnotherTestPiece( "testRoutine"));

return suite;

}

}

Add a note hereAdd a note hereIn Listing 11.4, the class AnotherTestPiece has a single parameter constructor. The parameter is then passed to the base class TestCase. This is necessary because the parameter represents the identifier of the test. The individual test methods are added by calling the method addTest, with the instantiated AnotherTestPiece class. The parameter passed to the AnotherTestPiece class constructor is the name of the method that is supposed to be executed in the text.

### Add a note hereAdd a note hereCreating Individual Tests

Add a note hereAdd a note hereNow that we've defined the technical details of setting up the test cases, our next topic is how to create a test. In Listing 11.4, the method testRoutine is used to execute some test. This test could be the instantiation of a class and then calling a method. If all goes well, then the test did its job. The focus of the test is to test an individual aspect of the program. The individual aspect could be whether two numbers were added correctly, or whether the end- user validated correctly. However, the tests test only one aspect, not multiple aspects. Listing 11.5 is a sample test.

Add a note hereAdd a note here**Listing 11.5**

Add a note hereAdd a note herepublic void testRoutine() throws Exception {

BusinessLogic logic = new BusinessLogic();

assertTrue( "Could not add user",

logic.addUser( "something"));

}

Add a note hereAdd a note hereIn Listing 11.5, the class BusinessLogic is being tested. The specific aspect that is being tested is if the addUser method works properly. The method assertTrue is used to test if the method addUser worked successfully. There are two parameters to the method assertTrue. The first parameter is a message to output if the assert fails. The second parameter is an assert value. The method assertTrue was used, so a failure will be generated if the second parameter has a value of false.

Add a note hereAdd a note hereThe method assertTrue is part of the following set of methods to indicate the state of the tests:

* Add a note hereAdd a note here**assertTrue**: Tests if the value of an assertion is true; if not, a failure is generated.
* Add a note hereAdd a note here**assertFalse**: Tests if the value of an assertion is false; if not, a failure is generated.
* Add a note hereAdd a note here**assertEquals**: A more complex assertion that has various overloaded implementations. The assertion tests if two objects equal each other. One of the two objects is the object to be tested and the other is a reference object determined to be correct. The two objects are compared using the equals method. If the two objects do not equal each other, then a failure is generated.
* Add a note hereAdd a note here**assertNotNull**: Tests if the object reference value of an assertion is not null; if not, a failure is generated.
* Add a note hereAdd a note here**assertNull**: Tests if the object reference value of an assertion is null; if not, a failure is generated.
* Add a note hereAdd a note here**assertSame**: Tests if two objects are the same value. The two objects are compared using the equality operator. If the objects are not the same value, a failure is generated.
* Add a note hereAdd a note here**assertNotSame**: Tests if two objects are the same value. The two objects are compared using the equality operator. If the objects are the same value, a failure is generated.
* Add a note hereAdd a note here**fail**: A simple fail method, not an assertion. This method is useful when the test routine provides its own way of validating the state.

Add a note hereAdd a note hereWriting test scripts is not that simple. In the case of Listing 11.5, it was easy because a simple operation was executed. However, in reality, testing means writing a sequence of test routines, not testing individual routines. Listing 11.6 is a sample implementation of a more complex scenario.

Larger View

Add a note hereAdd a note here**Listing 11.6**

Add a note hereAdd a note herepublic class AnotherTestPiece extends TestCase {

private User[] \_users;

private Mortgage \_mortgage;

boolean \_isRuntime;

public AnotherTestPiece( String test, boolean isRuntime) {

super( test);

\_isRuntime = isRuntime;

}

protected void setUp() throws Exception {

if( \_isRuntime) {

\_users = Configuration.getAllUsers();

\_mortgage = new Mortgage();

}

}

public void testConfigurationGeneration() throws Exception {

User original = new User();

original.setFirstName( "christian");

original.setLastname( "gross");

StringWriter stringWriter = new StringWriter();

BeanWriter beanWriter = new BeanWriter(stringWriter);

beanWriter.write(original);

stringWriter.flush();

String xml = "<?xml version='1.0'?>"

+ stringWriter.toString();

BeanReader reader = new BeanReader();

reader.registerBeanClass( User.class );

User compareTo = (User) reader.parse(

new StringReader(xml));

assertEquals( "Serialization does not work",

original, compareTo);

}

public static Test suiteGeneration() {

TestSuite suite= new TestSuite();

suite.addTest( new AnotherTestPiece(

"testConfigurationGeneration", false));

return suite;

}

public void testAddUser() throws Exception {

for( int c1 = 0; c1 < \_users.length; c1 ++) {

assertTrue( "Could not add user: " +

\_users[ c1].toString(),

\_mortgage.addUser( \_users[ c1]));

}

}

public void testRemoveUser() throws Exception {

for( int c1 = 0; c1 < \_users.length; c1 ++) {

assertTrue( "Could not add user: " +

\_users[ c1].toString(),

\_mortgage.addUser( \_users[ c1]));

}

}

public static Test suiteRuntime() {

TestSuite suite= new TestSuite();

suite.addTest( new AnotherTestPiece(

"testAddUser", true));

suite.addTest( new AnotherTestPiece(

"testRemoveUser", true));

return suite;

}

}

Add a note hereAdd a note hereIn Listing 11.6, the class AnotherTestPiece has been expanded to test two classes: User and Mortgage. The class User is a pure data class that contains data related to the identification of a particular user. The class Mortgage is an operations class used to process mortgages. Both of these classes have been dramatically abbreviated, even though it does not seem so, for illustration purposes.

Add a note hereAdd a note hereIn Listing 11.6, two static methods return a Test interface instance: suiteRuntime and suiteGeneration. There are two test methods because there are two sets of tests. Consider the problem that we need to create a comprehensive test framework that manipulates one user as well as multiple users. The easiest way to define multiple users is not by using code but by using configuration files. However, the problem of the configuration files is that tests have to be used to verify that configuration file routines work correctly. Think of it in the following way: how can tests be created when the testing routines might themselves have bugs? The easiest way to ensure that the testing routines themselves do not have errors is to use serialization to generate a file populated with an initial test data set. The file is then copied and populated with different data. Granted, the different data also has to be tested, but at least the structure of the file containing the data set will not be buggy. The only test to write is the testing of the serialization routines that generate the initial data set.

Add a note hereAdd a note hereThe method suiteGeneration calls a single method testConfigurationGeneration. The method testConfigurationGeneration creates a single User class instance. The single User class instance is a serialization to an XML string, which is then serialized to an object instance. The original object instance original should be equal to the object instance compareTo. If you use the assertEquals method, a successful validation will generate no error. Therefore, you can generate a runtime test that reads in a number of serialized files to test the actual object implementations.

Add a note hereAdd a note hereThe generation validation is an important step. When you create tests for complex systems, the operations for initializing the objects need to be working correctly. If the initialization has bugs, then the runtime could either hide bugs or generate unexplainable ones.

Add a note hereAdd a note hereThe method suiteRuntime generates the tests for runtime validation. The runtime test procedures are testAddUser and testRemoveUser. The implementation of each of the runtime test procedures iterates the individual users and adds them to the mortgage or removes them from it. All of these methods are so-called business logic test routines. Typically, these kinds of methods will call multiple methods based on an initial and final state. What is tested is the final state using a series of asserts.

Add a note hereAdd a note hereNotice in Listing 11.6 how the AnotherTestPiece constructor accepts two parameters. The second parameter is used to indicate in which mode the test process is executing. In Listing 11.6, the second parameter is a boolean, but it also could be a long value or an enumeration. The mode is required when you're setting up the object values in method setUp.

Add a note hereAdd a note hereFor the method setUp, a static class method, Configuration.getAllUsers is called. The method Configuration.getAllUsers is a routine that reads in the serialized user data generated by the generation routines. It is important to realize that any step in the method setUp cannot fail. An exception could be thrown to indicate incorrect user data. However, under no circumstances can an error be generated because the data is assumed correct. Of course, errors do occur. The objective is to minimize errors or the possibility of error.

### Add a note hereAdd a note hereCreating Tests That Fail

Add a note hereAdd a note hereGenerating test cases that report only failures is not a good idea because these tests describe only half the story. Typically, if an error occurs in good data, it either means that the data was wrong or something went wrong in the processing of the good data. Here is the problem: the premise that there could be serialization data errors has been established. If some of those errors are not caught, then the test cases will have bugs since bad data can be processed. Therefore, for all of the good tests, there need to be an equal number of bad tests. The bad tests make sure that all error handling routines are in place to catch bad data. The solution is to add another mode to the test cases. In Listing 11.6, there were two different implementations of the suite type method: suiteRuntime and suiteGeneration. We would have to add another suite type method for the bad data set. That mode would call a set of methods similar to the good data set, but the mode expects errors to be generated.

## **Project Management Using Ant**

Add a note hereAdd a note hereIn the Java world, Ant has established itself as the premier project management tool. To put a context to the situation, before Ant there was Make. Make is a traditional build tool. However, Make is a very complicated piece of software that is not very simple to use. Over time, many add-on utilities like configure and autoconf complicated the project management process even more.

Add a note hereAdd a note hereAnt does away with all that by creating a new kind of project management infrastructure. Central to Ant is an XML file. The XML file contains a number of XML elements that represent instructions to do something. Ant has an XML configuration file in the *Digester* package that loads a number of properties and attributes. The properties and attributes are instructions to compile a Java file, to copy files, or to do any other task that is integrated as an instruction to the Ant infrastructure. The core of Ant is relatively small. The large number of instructions is what gives Ant its power and flexibility.

### Add a note hereAdd a note hereAn Ant Hello World Example

Add a note hereAdd a note hereListing 11.7 shows a very simple Ant script that prints out "Hello World." Typically, Ant scripts are saved to the name Image from book[build.xml](http://cdcontent.books24x7.com/id_7265/build.xml), which is the default name.

Add a note hereAdd a note here**Listing 11.7**

Add a note hereAdd a note here<project name="Hello World" default="build" basedir=".">

<target name="build">

<echo message="Hello World"/>

</target>

</project>

Add a note hereAdd a note hereListing 11.7 contains three XML elements: project, target and echo. Each Ant script needs a root XML element project. The XML element project usually has three attributes: name, default, and basedir. The attribute name is the friendly descriptor of the project. The attribute default defines the default target of the project. The attribute basedir defines the initial starting directory of the project. The value for the attribute basedir is a single period, which means the current directory where the Ant file is located.

Add a note hereAdd a note hereThe attribute default references the default target of the application. A *target* is a reference to a set of build instructions. Consider a target like a method in a class file. When a default target is specified, it is like saying, for this program, please first call the default method. In Listing 11.7, the default target references the target build. The target build is identified by the XML element target, with a name attribute value of build. Once the target has been found, all of the child elements within the target are executed like a script. In Listing 11.7, the XML element echo is an instruction to output some text. The text that is output is defined by the value of the attribute message. The instruction in Ant terminology is called an Ant task.

### Add a note hereAdd a note hereUsing Variables

Add a note hereAdd a note hereAn Ant file can reference variables, which can be used like variables in a Java program. Ant variables can be created within the Ant project file, within a properties file, or within the environment variables like PATH. Listing 11.8 is an example that defines and outputs a variable.

Add a note hereAdd a note here**Listing 11.8**

Add a note hereAdd a note here<project name="Hello World" default="build" basedir=".">

<property name="variable" value="hello" />

<target name="build">

<echo message="variable value=${variable}" />

</target>

</project>

Add a note hereAdd a note hereIn Listing 11.8, a child XML element property is below the XML element project. The XML element property is a task used to define a property, which is akin to a variable. The attribute name defines the variable identifier. The attribute value defines the value of the variable. The XML element echo references the variable using the identifier within the curly brackets. The dollar sign in front of the open curly bracket indicates that a variable is being defined. The variable reference can be used only in the context of an attribute value, as is shown in Listing 11.8.

Add a note hereAdd a note hereA property can be used to reference the environment variables, as shown in Listing 11.9.

Add a note hereAdd a note here**Listing 11.9**

Add a note hereAdd a note here<project name="Hello World" default="build" basedir=".">

<property environment="env" />

<target name="build">

<echo message="Path=${env.PATH}" />

</target>

</project>

Add a note hereAdd a note hereIn Listing 11.9, the XML element property has only one attribute: environment The attributes name and value are not applicable. The attribute environment's value env is prepended to whenever a specific environment variable is referenced. The attribute message has a contained reference to the environment variable PATH. The identifier env.PATH would denote that variables can be object-oriented since the period is used to reference methods or properties. However, that is not correct since a variable could be defined with a period, like the example module.variable. When the environment variable uses the period, it is an exception to the rule.

Add a note hereAdd a note hereAnother way of referencing variables is to define the individual variables in a properties file and then include the properties file in the Ant build file. Listing 11.10 is a sample properties file.

Add a note hereAdd a note here**Listing 11.10**

Add a note hereAdd a note hereMyVariable = This is my value

Add a note hereAdd a note hereListing 11.10 is included in the Ant build file defined in Listing 11.11.

Add a note hereAdd a note here**Listing 11.11**

Add a note hereAdd a note here<project name="Hello World" default="build" basedir=".">

<property file="sample.properties" />

<target name="build">

<echo message="Property Variable =${MyVariable}" />

</target>

</project>

Add a note hereAdd a note hereIn Listing 11.11, the XML element property has only one attribute: file. The attribute file references the name of a file that is to be included. The properties file defined in Listing 11.10 contains one property, MyVariable, which is assigned a buffer of "This is my value". When the property is referenced, it is referenced like any other Ant variable, as shown by the attribute message.

### Add a note hereAdd a note hereIncluding Other XML Elements

Add a note hereAdd a note hereWhen you're programming using the C or C++ programming language, there is an instruction called #include. Using the #include instruction, you can include the contents of one file in another file. Java does not have that ability, which is generally not a problem. However, with Ant files, and XML files in particular, this is never a problem. Using XML Document Type Definitions (DTDs), you can define generic targets in another file and then have then included for further referencing. Listing 11.12 defines a fragment of an XML document.

Add a note hereAdd a note here**Listing 11.12**

Add a note hereAdd a note here<property name="dtd.identifier" value="value in the DTD"/>

<property name="another.identifier" value="other value"/>

Add a note hereAdd a note hereIn Listing 11.12, an XML fragment is defined because there is no single root XML element. Listing 11.13 is an XML document that includes the XML fragment defined in Listing 11.12.

Add a note hereAdd a note here**Listing 11.13**

Add a note hereAdd a note here<!DOCTYPE project [

<!ENTITY common SYSTEM "file:XmlFragment.xml">

]>

<project name="Hello World" default="build" basedir=".">

&common;

<target name="build">

<echo message="DTD Value is ${dtd.identifier}"/>

</target>

</project>

Add a note hereAdd a note hereIn Listing 11.13, the DTD is defined using the <! and ]> characters. Within the DTD declaration is an ENTITY declaration. The ENTITY declaration is called Image from book[common](http://cdcontent.books24x7.com/id_7265/common.zip) and references the file "file:XmlFragment.xml". This declaration means that whenever the identifier &Image from book [common;](http://cdcontent.books24x7.com/id_7265/common.zip) is used in the XML document, the contents of the file XmlFragment.xml are inserted. This means that in Listing 11.13, even though the &commons; identifier exists, when the XML document is processed, the XML fragment is inserted and made part of the parent XML document as if it had been typed directly into the XML document. No other XML technique available allows a developer to inject one piece of XML into another.

### Add a note hereAdd a note hereTask Definition: Compiling Java Files

Add a note hereAdd a note hereThus far in our examples, the Ant build script file has done nothing interesting other than output text to the console. Originally, Ant was used to compile Java class files, which is shown in Listing 11.14.

Larger View

Add a note hereAdd a note here**Listing 11.14**

Add a note hereAdd a note here <target name="default-compile" depends="init">

<javac classpath="${standard-jars}:${build}"

srcdir="${src}" destdir="${build}" debug="${debug}"

deprecation="on" >

</javac>

</target>

Add a note hereAdd a note hereListing 11.14 contains the XML element javac, which is the Ant task for the tool javac (the Java compiler). The javac task is an example of how a tool such as the Java compiler command line tool can be converted into an Ant task. A specific Ant task implements the Ant-required interfaces and bridges the XML attributes to the Java compiler. The Ant manual that comes with every distribution of Ant contains specifics of the javac target. What should be noticed is that almost every attribute is defined by some variable. This is typically the case because it allows a developer to custom-define the actual values.

Add a note hereAdd a note hereIn the Ant infrastructure are tasks to do basically whatever is necessary. The simplest is compiling the Java class files. However, there are tasks to create an archive, or tasks to generate a jar file. The types of task range from core to optional to user-added tasks. *Core tasks* are tasks that are distributed with the Ant distribution. *Optional tasks* are those that are distributed in a jar that can be download and installed where the Ant distribution resides. *User-added tasks* are tasks that are defined somewhere on the Java classpath and then defined as additional tasks in the Ant build file.

### Add a note hereAdd a note hereHandling a Set of Files

Add a note hereAdd a note hereWhen the javac task executes, the source files defined by the attribute srcdir in Listing 11.14 are compiled to the directory defined by the attribute destdir. In a traditional build environment, you compile the files by specifying them individually and compiling each file. Ant is more clever in that files are sets that operations are executed on. Compare this to how the programming languages Java and SQL handle data. Java references another object individually, even if there is a collection. SQL, on the other hand, references data as a set. Referencing an individual record is more complicated in SQL than referencing an entire set. The same could be said about Ant. In Ant, data is referenced using a set. Consider Listing 11.15, which copies an individual file to a specific directory.

Add a note hereAdd a note here**Listing 11.15**

Add a note hereAdd a note here<target name="deploy">

<copy file="afile.txt" todir="${axisDirectory}"/>

</target>

Add a note hereAdd a note hereIn Listing 11.15, the XML element copy has two attributes: file and todir. The attribute file selects a specific file. The attribute todir defines the directory to where the file is to be copied. To copy two files, another XML element copy would have to be added. In the Ant paradigm, the attribute file can be replaced with a fileset as defined in Listing 11.16.

Add a note hereAdd a note here**Listing 11.16**

Add a note hereAdd a note here<target name="deploy">

<copy todir="${axisDirectory}">

<fileset dir="${build}">

<exclude name="\*.class" />

</fileset>

</copy>

</target>

Add a note hereAdd a note hereIn Listing 11.16, the XML element fileset is not a task but an inherent operation that exists within Ant. The XML element fileset has an attribute dir, which specifies the directory. The directory, including any subdirectories that are encountered, is iterated. Whatever is found is added into the file set. A child XML element exclude exists, so the files defined by the attribute name are removed from the file set. Notice how the name attribute has a value of \*.class, which uses an asterisk wildcard character to match anything. When the name attribute is evaluated, all files that end with the extension class are removed from the file set. The remaining files are then passed to the copy command.

Add a note hereAdd a note hereThe file set defined in Listing 11.16 added all files that were encountered. You can specifically include files, as shown in Listing 11.17.

Add a note hereAdd a note here**Listing 11.17**

Add a note hereAdd a note here<fileset dir="${build}">

<include name="\*\*/\*.java"/>

<exclude name="\*\*/\*Main\*"/>

</fileset>

Add a note hereAdd a note hereIn Listing 11.17, the XML element include selects all files that have a Java extension and creates a file set. Then, the XML element exclude removes from the file set all of the files that contain the text Main. There are other types of sets, such as directory sets. However, for the vast majority of tasks, the file set is the most useful construct. A file set is used in three different contexts. The first context, which we have already covered, is the copying, moving, or deleting of files.

Add a note hereAdd a note hereAnother context is definition of the classpath when you're executing any task that requires a classpath. The task junit shown in Listing 11.18 is an example.

Larger View

Add a note hereAdd a note here**Listing 11.18**

Add a note hereAdd a note here <junit dir="./" printSummary="yes"

fork="true" haltonerror="true">

<sysproperty key="basedir" value="src/test"></sysproperty>

<formatter type="xml"></formatter>

<formatter usefile="true" type="plain"></formatter>

<classpath>

<fileset dir="lib">

<include name="\*.jar"></include>

</fileset>

<pathelement location="target/${final.name}.jar" />

<pathelement path="${testclassesdir}" />

</classpath>

<batchtest todir="${testreportdir}">

<fileset dir="src/test">

<include name="\*\*/\*Test\*.java" />

</fileset>

</batchtest>

</junit>

Add a note hereAdd a note hereListing 11.18 is an excellent example of using fileset to reference paths that cannot be defined in the attributes. The XML element junit is a task used to run *JUnit* tasks defined in *JUnit* section earlier in this chapter. In Listing 11.18, are two XML elements that use file sets: classpath and batchtest. The XML element classpath defines a Java classpath for the task junit to use. The classpath has been split into three parts. In command line terms, this would be the same as putting three pieces of the Java classpath into one overall path. The XML element fileset, which we have already explained, shows how to include a number of jar files that exist in a specific directory. The XML element pathelement is an individual reference to either a jar file or a path that contains a number of classes. For the XML element batchtest, a file set is used to reference the Java test files.

Add a note hereAdd a note hereAnother way of referencing a Java classpath is to use a reference ID, a file set identified somewhere in the build script that is referenced by a classpath or other task that requires a file set. Listing 11.19 shows how to reference file sets.

Add a note hereAdd a note here**Listing 11.19**

Add a note hereAdd a note here<project name="Hello World" default="build" basedir=".">

<path id="compile.classpath">

<fileset dir="lib">

<include name="\*.jar"></include>

</fileset>

<pathelement location="${build.home}/classes"/>

</path>

<target name="build">

<javac srcdir="${source.home}"

destdir="${build.home}/classes"

debug="${compile.debug}"

deprecation="${compile.deprecation}"

optimize="${compile.optimize}">

<classpath refid="compile.classpath"/>

</javac>

</target>

</project>

Add a note hereAdd a note hereIn Listing 11.19, the file set is defined by the XML element path, which has one attribute, id. The attribute id is the identifier that is used to uniquely identify the file set. Then, to use the file set, it is referenced using the attribute refid. In the case of Listing 11.19, the reference is part of the XML element classpath. The advantage of using file sets is that you can reference a file set multiple times without having to explicitly type out the file set. When the file set changes, so do all of the places where the reference is used.

Add a note hereAdd a note hereWe used the wildcard asterisk character to select whatever files are found. However, you can select the files using various techniques. [Table 11.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=392444298) shows the different way matches can be made.

**Add a note hereAdd a note hereTable 11.3:** Different ways of making a selection in a file set.

| **Add a note hereAdd a note herePattern** | **Add a note hereAdd a note hereFile Set That Is Selected** |
| --- | --- |
| Add a note hereAdd a note here\*.java | Add a note hereAdd a note hereSelect all the files from the current directory that have the extension java. Matches: something.java Fails: something.javanext |
| Add a note hereAdd a note here??.java | Add a note hereAdd a note hereSelect all the files from the current directory that have exactly two letters for a name and have an extension of Java. Matches: a[a.java](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) Fails: aa[a.java](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) |
| Add a note hereAdd a note here\*\*/\*.java | Add a note hereAdd a note hereSelect all files from the current directory and subdirectories that have the extension java. Matches: /something/dir/something.java Fails: /something/another.txt |
| Add a note hereAdd a note here\*\*/src/\*.java | Add a note hereAdd a note hereFrom the current directory find all of the directories that have src child directory. And from the src directory select all of the files that have an extension of java. Matches: /something/dir/src/something.java Fails: /something/dir/src/dir/something.java |
| Add a note hereAdd a note here\*\*/src/\*\*/\*.java | Add a note hereAdd a note hereFrom the current directory find all of the directories that have src child directory. From that directory select all of the child directories that contain java files Matches: /something/src/dir/something.java Fails: /something/dir/something.java |

### Add a note hereAdd a note hereDefining Custom Tasks and Paths

Add a note hereAdd a note hereIn the Axis toolkit are a number of helper routines to generate WSDL and to generate Java files from WSDL. The functionality is exposed from the *axis-*[*ant.jar*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) file. To use the additional Ant tasks, you have to add the jar file to the classpath and then reference it in the build script, as shown in Listing 11.20.

Larger View

Add a note hereAdd a note here**Listing 11.20**

Add a note hereAdd a note here<taskdef

name="wsdl2java"

classname="org.apache.axis.tools.ant.wsdl.Wsdl2javaAntTask"

classpath="${standard-jars}:${build}" />

<taskdef

name="java2wsdl"

classname="org.apache.axis.tools.ant.wsdl.Java2WsdlAntTask"

classpath="${standard-jars}:${build}" />

Add a note hereAdd a note hereIn Listing 11.20, there are two XML elements with the same name, taskdef. The task taskdef is used to load an Ant-compliant task and make it available as another task. In the case of Listing 11.20, the names of the other tasks are identified by the attribute name. The attribute classname identifies the class that implements the Ant interfaces for Ant task purposes. The attribute classpath identifies the Java classpath used when loading the value of the attribute classname.

Add a note hereAdd a note hereListing 11.21 shows how to use either of the tasks defined in Listing 11.20.

Larger View

Add a note hereAdd a note here**Listing 11.21**

Add a note hereAdd a note here <wsdl2java

output="${project-src}/${package}/gen-wsdl-2-java"

deployscope="application"

testcase="false"

verbose="true"

skeletondeploy="false"

serverside="true"

helpergen="true"

url="${project-src}/${package}/webservice.wsdl" >

</wsdl2java>

<java2wsdl

classname="${classname}"

output="${project-src}/${package}/webservice.wsdl"

location="${location}"

namespace="${namespace}"

>

<mapping

namespace="${namespace}"

package="${package}.def" />

</java2wsdl>

### Add a note hereAdd a note hereCalling Other Ant Files or Targets

Add a note hereAdd a note hereWhen setting up an Ant build file, you need to be able to execute multiple targets. The reason is that, for most build processes, there is an initialization, compilation, and packaging phase. Maybe all phases will be called; maybe not. Regardless, the idea is to be able to split up the build process and be able to call individual targets. There are three ways of calling another target: using dependencies, making a target method call, and calling a target into another file.

Add a note hereAdd a note hereThe easiest way of chaining targets together is to use the dependency attribute shown in Listing 11.22.

Larger View

Add a note hereAdd a note here**Listing 11.22**

Add a note hereAdd a note here<project name="Hello World" default="compile" basedir=".">

<target name="init">

<echo message="initialization" />

</target>

<target name="compile" depends="init">

<javac classpath="${standard-jars}:${build}"

srcdir="${src}" destdir="${build}" debug="${debug}"

deprecation="on" >

</javac>

</target>

</project>

Add a note hereAdd a note hereIn Listing 11.22, the project has a default target of compile. The Ant process will execute the compile target; however, there is the attribute depends with a value of init. The attribute depends tells the Ant process to first process the target init before processing the target compile. The attribute depends is a list of comma-separated targets that are executed first.

Add a note hereAdd a note hereThe second way to execute an Ant target is to call it like a method call, as shown in Listing 11.23.

Larger View

Add a note hereAdd a note here**Listing 11.23**

Add a note hereAdd a note here<target name="default-java-to-wsdl">

<antcall target="compile">

<param name="src"

value="${project-src}/${package}/def"/>

</antcall>

<java

classpath="${standard-jars}:${build}"

classname="org.apache.axis.wsdl.Java2WSDL"

fork="on" dir=".">

<arg

value="-o${project-src}/${package}/webservice.wsdl"/>

<arg value="-l${location}"/>

<arg value="-n${namespace}"/>

<arg value="-p${package}.def" />

<arg value="${namespace}" />

<arg value="-wAll"/>

<arg value="${classname}"/>

</java>

</target>

<target name="java2wsdl">

<antcall target="default-java-to-wsdl" >

<param name="package" value="${package-name}" />

<param name="namespace" value="${namespace-name}" />

<param name="classname" value="${gen-classname}" />

<param name="location" value="${endpoint-name}" />

</antcall>

</target>

Add a note hereAdd a note hereIn Listing 11.23, there are two targets: java2wsdl and default-java-to-wsdl. The idea behind this strategy is to define a generic target that compiles a number of Java class files, which are then used to generate a WSDL file. This is the strategy we defined in [Chapter 10](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=776#776). The target java2wsdl found at the bottom of Listing 11.23 can execute another target by using the XML element antcall, where the attribute target represents the target to execute. Within the XML element antcall are a number of XML param elements. The XML element param represents a parameter. A parameter is an incorrect term to use here, though, since Ant does not understand the concept of parameters. Instead, consider the XML element param as the definition of properties that are valid only for the scope of the Ant call. We could consider the variables defined in Listing 11.2 to be global variables, and the properties defined by the XML element param to be local variables.

Add a note hereAdd a note hereNotice how when the target default-java-to-wsdl is executed an Ant call is made to the compile target. We could have achieved this same effect by setting a dependency to the compile target for the target default-java-to-wsdl. There is a big difference between making an Ant call and using dependencies. In [Chapter 10](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=776#776), we used classes to define what the Web Service should look like. Those classes were compiled and a WSDL file generated. The actual implementation of the Web Service would be similar classes, but not the same classes. Therefore, the Web Service definition classes and Web Service implementation classes are located in two entirely different directories. When the Ant process started, some properties were set, including the main sources. The secondary sources, which are the Web Service definition files, would not be set. Therefore, the only solution is to call the compile target explicitly and redefine the location of the sources using an Ant call parameter.

Add a note hereAdd a note hereAlso shown in Listing 11.23 is the use of the java task. Notice how parameters can be passed to the Java command line using the XML element arg.

Add a note hereAdd a note hereThe last way of calling another target is to explicitly call a target in another file, as shown in Listing 11.24.

Add a note hereAdd a note here**Listing 11.24**

Add a note hereAdd a note here<target name="GenerateWSDLStub">

<ant antfile="build.xml" dir="design" target="deploy"/>

<ant antfile="build.xml" dir="server"

target="GenerateWSDLStub"/>

<ant antfile="build.xml" dir="client"

target="GenerateWSDLStub"/>

</target>

Add a note hereAdd a note hereIn Listing 11.24, the target GenerateWSDLStub has three child XML ant elements, which are used to execute three different Ant tasks in the same file. Listing 11.24 is shown for illustrational purposes. The XML element ant is a task that calls a target in another file. The ant task has three attributes: antfile, dir, and target. The attribute antfile defines the name of the Ant file to be loaded. The attribute dir defines the directory where the Ant file to be loaded is located. The attribute target defines the target that is called on the Ant file to be loaded. Once the target destination has been called, the target destination is subject to all target-defined dependencies and Ant calls.

### Add a note hereAdd a note hereSome Tips for When You Use Ant

Add a note hereAdd a note hereHere are some tips on building good Ant files:

* Add a note hereAdd a note here**Structure your code so that it is logical.** This means that sources should be stored in the source directory, tests in the test directory, and configuration information in the conf directory. You can see multiple good examples of how to do this right in the Jakarta Commons and Commons Sandbox projects.
* Add a note hereAdd a note here**Use property files or Ant properties to define the location of specific jar files.** In addition, when defining them, use an abstract syntax similar to that used in Listing 11.25. (Note that Listing 11.25 uses the properties file notation, which could also have been the Ant property task.)

Add a note hereAdd a note here**Listing 11.25**

Add a note hereAdd a note herexml-rpc.jar=${jars-dir}/xmlrpc-1.1.jar

Add a note hereAdd a note hereIn Listing 11.25, the variable xml-rpc.jar is defined and references the specific file *Image from book*[*xmlrpc-1.1.jar*](http://cdcontent.books24x7.com/id_7265/xmlrpc-1.1.jar). This is very useful because it allows a developer to abstractly define the jar files that are being used, without defining a specific version number. When a new version of the jar file arrives, only one location needs updating.

* Add a note hereAdd a note here**Define abstract targets that are very generic and that are then referenced using specific dependencies or Ant calls.** The abstract target should be defined in a file called *targ*[*et.xml*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) and included into main *Image from book*[*build.xml*](http://cdcontent.books24x7.com/id_7265/build.xml) file using the DTD technique described earlier.
* Add a note hereAdd a note here**Use obvious identifiers, even if the identifiers are verbose.** Using obvious identifiers make it simpler to debug a problem in an Ant script.
* Add a note hereAdd a note here**If there are problems when you are running the Ant program, run the program with the –verbose or –debug flag.** In those modes, a large amount of information that can help a developer figure out what is not working is generated. For example, finding dead classpath references is very simple using these modes.

## **Project Management Using Ant**

Add a note hereAdd a note hereIn the Java world, Ant has established itself as the premier project management tool. To put a context to the situation, before Ant there was Make. Make is a traditional build tool. However, Make is a very complicated piece of software that is not very simple to use. Over time, many add-on utilities like configure and autoconf complicated the project management process even more.

Add a note hereAdd a note hereAnt does away with all that by creating a new kind of project management infrastructure. Central to Ant is an XML file. The XML file contains a number of XML elements that represent instructions to do something. Ant has an XML configuration file in the *Digester* package that loads a number of properties and attributes. The properties and attributes are instructions to compile a Java file, to copy files, or to do any other task that is integrated as an instruction to the Ant infrastructure. The core of Ant is relatively small. The large number of instructions is what gives Ant its power and flexibility.

### Add a note hereAdd a note hereAn Ant Hello World Example

Add a note hereAdd a note hereListing 11.7 shows a very simple Ant script that prints out "Hello World." Typically, Ant scripts are saved to the name Image from book[build.xml](http://cdcontent.books24x7.com/id_7265/build.xml), which is the default name.

Add a note hereAdd a note here**Listing 11.7**

Add a note hereAdd a note here<project name="Hello World" default="build" basedir=".">

<target name="build">

<echo message="Hello World"/>

</target>

</project>

Add a note hereAdd a note hereListing 11.7 contains three XML elements: project, target and echo. Each Ant script needs a root XML element project. The XML element project usually has three attributes: name, default, and basedir. The attribute name is the friendly descriptor of the project. The attribute default defines the default target of the project. The attribute basedir defines the initial starting directory of the project. The value for the attribute basedir is a single period, which means the current directory where the Ant file is located.

Add a note hereAdd a note hereThe attribute default references the default target of the application. A *target* is a reference to a set of build instructions. Consider a target like a method in a class file. When a default target is specified, it is like saying, for this program, please first call the default method. In Listing 11.7, the default target references the target build. The target build is identified by the XML element target, with a name attribute value of build. Once the target has been found, all of the child elements within the target are executed like a script. In Listing 11.7, the XML element echo is an instruction to output some text. The text that is output is defined by the value of the attribute message. The instruction in Ant terminology is called an Ant task.

### Add a note hereAdd a note hereUsing Variables

Add a note hereAdd a note hereAn Ant file can reference variables, which can be used like variables in a Java program. Ant variables can be created within the Ant project file, within a properties file, or within the environment variables like PATH. Listing 11.8 is an example that defines and outputs a variable.

Add a note hereAdd a note here**Listing 11.8**

Add a note hereAdd a note here<project name="Hello World" default="build" basedir=".">

<property name="variable" value="hello" />

<target name="build">

<echo message="variable value=${variable}" />

</target>

</project>

Add a note hereAdd a note hereIn Listing 11.8, a child XML element property is below the XML element project. The XML element property is a task used to define a property, which is akin to a variable. The attribute name defines the variable identifier. The attribute value defines the value of the variable. The XML element echo references the variable using the identifier within the curly brackets. The dollar sign in front of the open curly bracket indicates that a variable is being defined. The variable reference can be used only in the context of an attribute value, as is shown in Listing 11.8.

Add a note hereAdd a note hereA property can be used to reference the environment variables, as shown in Listing 11.9.

Add a note hereAdd a note here**Listing 11.9**

Add a note hereAdd a note here<project name="Hello World" default="build" basedir=".">

<property environment="env" />

<target name="build">

<echo message="Path=${env.PATH}" />

</target>

</project>

Add a note hereAdd a note hereIn Listing 11.9, the XML element property has only one attribute: environment The attributes name and value are not applicable. The attribute environment's value env is prepended to whenever a specific environment variable is referenced. The attribute message has a contained reference to the environment variable PATH. The identifier env.PATH would denote that variables can be object-oriented since the period is used to reference methods or properties. However, that is not correct since a variable could be defined with a period, like the example module.variable. When the environment variable uses the period, it is an exception to the rule.

Add a note hereAdd a note hereAnother way of referencing variables is to define the individual variables in a properties file and then include the properties file in the Ant build file. Listing 11.10 is a sample properties file.

Add a note hereAdd a note here**Listing 11.10**

Add a note hereAdd a note hereMyVariable = This is my value

Add a note hereAdd a note hereListing 11.10 is included in the Ant build file defined in Listing 11.11.

Add a note hereAdd a note here**Listing 11.11**

Add a note hereAdd a note here<project name="Hello World" default="build" basedir=".">

<property file="sample.properties" />

<target name="build">

<echo message="Property Variable =${MyVariable}" />

</target>

</project>

Add a note hereAdd a note hereIn Listing 11.11, the XML element property has only one attribute: file. The attribute file references the name of a file that is to be included. The properties file defined in Listing 11.10 contains one property, MyVariable, which is assigned a buffer of "This is my value". When the property is referenced, it is referenced like any other Ant variable, as shown by the attribute message.

### Add a note hereAdd a note hereIncluding Other XML Elements

Add a note hereAdd a note hereWhen you're programming using the C or C++ programming language, there is an instruction called #include. Using the #include instruction, you can include the contents of one file in another file. Java does not have that ability, which is generally not a problem. However, with Ant files, and XML files in particular, this is never a problem. Using XML Document Type Definitions (DTDs), you can define generic targets in another file and then have then included for further referencing. Listing 11.12 defines a fragment of an XML document.

Add a note hereAdd a note here**Listing 11.12**

Add a note hereAdd a note here<property name="dtd.identifier" value="value in the DTD"/>

<property name="another.identifier" value="other value"/>

Add a note hereAdd a note hereIn Listing 11.12, an XML fragment is defined because there is no single root XML element. Listing 11.13 is an XML document that includes the XML fragment defined in Listing 11.12.

Add a note hereAdd a note here**Listing 11.13**

Add a note hereAdd a note here<!DOCTYPE project [

<!ENTITY common SYSTEM "file:XmlFragment.xml">

]>

<project name="Hello World" default="build" basedir=".">

&common;

<target name="build">

<echo message="DTD Value is ${dtd.identifier}"/>

</target>

</project>

Add a note hereAdd a note hereIn Listing 11.13, the DTD is defined using the <! and ]> characters. Within the DTD declaration is an ENTITY declaration. The ENTITY declaration is called Image from book[common](http://cdcontent.books24x7.com/id_7265/common.zip) and references the file "file:XmlFragment.xml". This declaration means that whenever the identifier &Image from book [common;](http://cdcontent.books24x7.com/id_7265/common.zip) is used in the XML document, the contents of the file XmlFragment.xml are inserted. This means that in Listing 11.13, even though the &commons; identifier exists, when the XML document is processed, the XML fragment is inserted and made part of the parent XML document as if it had been typed directly into the XML document. No other XML technique available allows a developer to inject one piece of XML into another.

### Add a note hereAdd a note hereTask Definition: Compiling Java Files

Add a note hereAdd a note hereThus far in our examples, the Ant build script file has done nothing interesting other than output text to the console. Originally, Ant was used to compile Java class files, which is shown in Listing 11.14.

Larger View

Add a note hereAdd a note here**Listing 11.14**

Add a note hereAdd a note here <target name="default-compile" depends="init">

<javac classpath="${standard-jars}:${build}"

srcdir="${src}" destdir="${build}" debug="${debug}"

deprecation="on" >

</javac>

</target>

Add a note hereAdd a note hereListing 11.14 contains the XML element javac, which is the Ant task for the tool javac (the Java compiler). The javac task is an example of how a tool such as the Java compiler command line tool can be converted into an Ant task. A specific Ant task implements the Ant-required interfaces and bridges the XML attributes to the Java compiler. The Ant manual that comes with every distribution of Ant contains specifics of the javac target. What should be noticed is that almost every attribute is defined by some variable. This is typically the case because it allows a developer to custom-define the actual values.

Add a note hereAdd a note hereIn the Ant infrastructure are tasks to do basically whatever is necessary. The simplest is compiling the Java class files. However, there are tasks to create an archive, or tasks to generate a jar file. The types of task range from core to optional to user-added tasks. *Core tasks* are tasks that are distributed with the Ant distribution. *Optional tasks* are those that are distributed in a jar that can be download and installed where the Ant distribution resides. *User-added tasks* are tasks that are defined somewhere on the Java classpath and then defined as additional tasks in the Ant build file.

### Add a note hereAdd a note hereHandling a Set of Files

Add a note hereAdd a note hereWhen the javac task executes, the source files defined by the attribute srcdir in Listing 11.14 are compiled to the directory defined by the attribute destdir. In a traditional build environment, you compile the files by specifying them individually and compiling each file. Ant is more clever in that files are sets that operations are executed on. Compare this to how the programming languages Java and SQL handle data. Java references another object individually, even if there is a collection. SQL, on the other hand, references data as a set. Referencing an individual record is more complicated in SQL than referencing an entire set. The same could be said about Ant. In Ant, data is referenced using a set. Consider Listing 11.15, which copies an individual file to a specific directory.

Add a note hereAdd a note here**Listing 11.15**

Add a note hereAdd a note here<target name="deploy">

<copy file="afile.txt" todir="${axisDirectory}"/>

</target>

Add a note hereAdd a note hereIn Listing 11.15, the XML element copy has two attributes: file and todir. The attribute file selects a specific file. The attribute todir defines the directory to where the file is to be copied. To copy two files, another XML element copy would have to be added. In the Ant paradigm, the attribute file can be replaced with a fileset as defined in Listing 11.16.

Add a note hereAdd a note here**Listing 11.16**

Add a note hereAdd a note here<target name="deploy">

<copy todir="${axisDirectory}">

<fileset dir="${build}">

<exclude name="\*.class" />

</fileset>

</copy>

</target>

Add a note hereAdd a note hereIn Listing 11.16, the XML element fileset is not a task but an inherent operation that exists within Ant. The XML element fileset has an attribute dir, which specifies the directory. The directory, including any subdirectories that are encountered, is iterated. Whatever is found is added into the file set. A child XML element exclude exists, so the files defined by the attribute name are removed from the file set. Notice how the name attribute has a value of \*.class, which uses an asterisk wildcard character to match anything. When the name attribute is evaluated, all files that end with the extension class are removed from the file set. The remaining files are then passed to the copy command.

Add a note hereAdd a note hereThe file set defined in Listing 11.16 added all files that were encountered. You can specifically include files, as shown in Listing 11.17.

Add a note hereAdd a note here**Listing 11.17**

Add a note hereAdd a note here<fileset dir="${build}">

<include name="\*\*/\*.java"/>

<exclude name="\*\*/\*Main\*"/>

</fileset>

Add a note hereAdd a note hereIn Listing 11.17, the XML element include selects all files that have a Java extension and creates a file set. Then, the XML element exclude removes from the file set all of the files that contain the text Main. There are other types of sets, such as directory sets. However, for the vast majority of tasks, the file set is the most useful construct. A file set is used in three different contexts. The first context, which we have already covered, is the copying, moving, or deleting of files.

Add a note hereAdd a note hereAnother context is definition of the classpath when you're executing any task that requires a classpath. The task junit shown in Listing 11.18 is an example.

Larger View

Add a note hereAdd a note here**Listing 11.18**

Add a note hereAdd a note here <junit dir="./" printSummary="yes"

fork="true" haltonerror="true">

<sysproperty key="basedir" value="src/test"></sysproperty>

<formatter type="xml"></formatter>

<formatter usefile="true" type="plain"></formatter>

<classpath>

<fileset dir="lib">

<include name="\*.jar"></include>

</fileset>

<pathelement location="target/${final.name}.jar" />

<pathelement path="${testclassesdir}" />

</classpath>

<batchtest todir="${testreportdir}">

<fileset dir="src/test">

<include name="\*\*/\*Test\*.java" />

</fileset>

</batchtest>

</junit>

Add a note hereAdd a note hereListing 11.18 is an excellent example of using fileset to reference paths that cannot be defined in the attributes. The XML element junit is a task used to run *JUnit* tasks defined in *JUnit* section earlier in this chapter. In Listing 11.18, are two XML elements that use file sets: classpath and batchtest. The XML element classpath defines a Java classpath for the task junit to use. The classpath has been split into three parts. In command line terms, this would be the same as putting three pieces of the Java classpath into one overall path. The XML element fileset, which we have already explained, shows how to include a number of jar files that exist in a specific directory. The XML element pathelement is an individual reference to either a jar file or a path that contains a number of classes. For the XML element batchtest, a file set is used to reference the Java test files.

Add a note hereAdd a note hereAnother way of referencing a Java classpath is to use a reference ID, a file set identified somewhere in the build script that is referenced by a classpath or other task that requires a file set. Listing 11.19 shows how to reference file sets.

Add a note hereAdd a note here**Listing 11.19**

Add a note hereAdd a note here<project name="Hello World" default="build" basedir=".">

<path id="compile.classpath">

<fileset dir="lib">

<include name="\*.jar"></include>

</fileset>

<pathelement location="${build.home}/classes"/>

</path>

<target name="build">

<javac srcdir="${source.home}"

destdir="${build.home}/classes"

debug="${compile.debug}"

deprecation="${compile.deprecation}"

optimize="${compile.optimize}">

<classpath refid="compile.classpath"/>

</javac>

</target>

</project>

Add a note hereAdd a note hereIn Listing 11.19, the file set is defined by the XML element path, which has one attribute, id. The attribute id is the identifier that is used to uniquely identify the file set. Then, to use the file set, it is referenced using the attribute refid. In the case of Listing 11.19, the reference is part of the XML element classpath. The advantage of using file sets is that you can reference a file set multiple times without having to explicitly type out the file set. When the file set changes, so do all of the places where the reference is used.

Add a note hereAdd a note hereWe used the wildcard asterisk character to select whatever files are found. However, you can select the files using various techniques. [Table 11.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=392444298) shows the different way matches can be made.

**Add a note hereAdd a note hereTable 11.3:** Different ways of making a selection in a file set.

| **Add a note hereAdd a note herePattern** | **Add a note hereAdd a note hereFile Set That Is Selected** |
| --- | --- |
| Add a note hereAdd a note here\*.java | Add a note hereAdd a note hereSelect all the files from the current directory that have the extension java. Matches: something.java Fails: something.javanext |
| Add a note hereAdd a note here??.java | Add a note hereAdd a note hereSelect all the files from the current directory that have exactly two letters for a name and have an extension of Java. Matches: a[a.java](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) Fails: aa[a.java](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) |
| Add a note hereAdd a note here\*\*/\*.java | Add a note hereAdd a note hereSelect all files from the current directory and subdirectories that have the extension java. Matches: /something/dir/something.java Fails: /something/another.txt |
| Add a note hereAdd a note here\*\*/src/\*.java | Add a note hereAdd a note hereFrom the current directory find all of the directories that have src child directory. And from the src directory select all of the files that have an extension of java. Matches: /something/dir/src/something.java Fails: /something/dir/src/dir/something.java |
| Add a note hereAdd a note here\*\*/src/\*\*/\*.java | Add a note hereAdd a note hereFrom the current directory find all of the directories that have src child directory. From that directory select all of the child directories that contain java files Matches: /something/src/dir/something.java Fails: /something/dir/something.java |

### Add a note hereAdd a note hereDefining Custom Tasks and Paths

Add a note hereAdd a note hereIn the Axis toolkit are a number of helper routines to generate WSDL and to generate Java files from WSDL. The functionality is exposed from the *axis-*[*ant.jar*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) file. To use the additional Ant tasks, you have to add the jar file to the classpath and then reference it in the build script, as shown in Listing 11.20.

Larger View

Add a note hereAdd a note here**Listing 11.20**

Add a note hereAdd a note here<taskdef

name="wsdl2java"

classname="org.apache.axis.tools.ant.wsdl.Wsdl2javaAntTask"

classpath="${standard-jars}:${build}" />

<taskdef

name="java2wsdl"

classname="org.apache.axis.tools.ant.wsdl.Java2WsdlAntTask"

classpath="${standard-jars}:${build}" />

Add a note hereAdd a note hereIn Listing 11.20, there are two XML elements with the same name, taskdef. The task taskdef is used to load an Ant-compliant task and make it available as another task. In the case of Listing 11.20, the names of the other tasks are identified by the attribute name. The attribute classname identifies the class that implements the Ant interfaces for Ant task purposes. The attribute classpath identifies the Java classpath used when loading the value of the attribute classname.

Add a note hereAdd a note hereListing 11.21 shows how to use either of the tasks defined in Listing 11.20.

Larger View

Add a note hereAdd a note here**Listing 11.21**

Add a note hereAdd a note here <wsdl2java

output="${project-src}/${package}/gen-wsdl-2-java"

deployscope="application"

testcase="false"

verbose="true"

skeletondeploy="false"

serverside="true"

helpergen="true"

url="${project-src}/${package}/webservice.wsdl" >

</wsdl2java>

<java2wsdl

classname="${classname}"

output="${project-src}/${package}/webservice.wsdl"

location="${location}"

namespace="${namespace}"

>

<mapping

namespace="${namespace}"

package="${package}.def" />

</java2wsdl>

### Add a note hereAdd a note hereCalling Other Ant Files or Targets

Add a note hereAdd a note hereWhen setting up an Ant build file, you need to be able to execute multiple targets. The reason is that, for most build processes, there is an initialization, compilation, and packaging phase. Maybe all phases will be called; maybe not. Regardless, the idea is to be able to split up the build process and be able to call individual targets. There are three ways of calling another target: using dependencies, making a target method call, and calling a target into another file.

Add a note hereAdd a note hereThe easiest way of chaining targets together is to use the dependency attribute shown in Listing 11.22.

Larger View

Add a note hereAdd a note here**Listing 11.22**

Add a note hereAdd a note here<project name="Hello World" default="compile" basedir=".">

<target name="init">

<echo message="initialization" />

</target>

<target name="compile" depends="init">

<javac classpath="${standard-jars}:${build}"

srcdir="${src}" destdir="${build}" debug="${debug}"

deprecation="on" >

</javac>

</target>

</project>

Add a note hereAdd a note hereIn Listing 11.22, the project has a default target of compile. The Ant process will execute the compile target; however, there is the attribute depends with a value of init. The attribute depends tells the Ant process to first process the target init before processing the target compile. The attribute depends is a list of comma-separated targets that are executed first.

Add a note hereAdd a note hereThe second way to execute an Ant target is to call it like a method call, as shown in Listing 11.23.

Larger View

Add a note hereAdd a note here**Listing 11.23**

Add a note hereAdd a note here<target name="default-java-to-wsdl">

<antcall target="compile">

<param name="src"

value="${project-src}/${package}/def"/>

</antcall>

<java

classpath="${standard-jars}:${build}"

classname="org.apache.axis.wsdl.Java2WSDL"

fork="on" dir=".">

<arg

value="-o${project-src}/${package}/webservice.wsdl"/>

<arg value="-l${location}"/>

<arg value="-n${namespace}"/>

<arg value="-p${package}.def" />

<arg value="${namespace}" />

<arg value="-wAll"/>

<arg value="${classname}"/>

</java>

</target>

<target name="java2wsdl">

<antcall target="default-java-to-wsdl" >

<param name="package" value="${package-name}" />

<param name="namespace" value="${namespace-name}" />

<param name="classname" value="${gen-classname}" />

<param name="location" value="${endpoint-name}" />

</antcall>

</target>

Add a note hereAdd a note hereIn Listing 11.23, there are two targets: java2wsdl and default-java-to-wsdl. The idea behind this strategy is to define a generic target that compiles a number of Java class files, which are then used to generate a WSDL file. This is the strategy we defined in [Chapter 10](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=776#776). The target java2wsdl found at the bottom of Listing 11.23 can execute another target by using the XML element antcall, where the attribute target represents the target to execute. Within the XML element antcall are a number of XML param elements. The XML element param represents a parameter. A parameter is an incorrect term to use here, though, since Ant does not understand the concept of parameters. Instead, consider the XML element param as the definition of properties that are valid only for the scope of the Ant call. We could consider the variables defined in Listing 11.2 to be global variables, and the properties defined by the XML element param to be local variables.

Add a note hereAdd a note hereNotice how when the target default-java-to-wsdl is executed an Ant call is made to the compile target. We could have achieved this same effect by setting a dependency to the compile target for the target default-java-to-wsdl. There is a big difference between making an Ant call and using dependencies. In [Chapter 10](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=776#776), we used classes to define what the Web Service should look like. Those classes were compiled and a WSDL file generated. The actual implementation of the Web Service would be similar classes, but not the same classes. Therefore, the Web Service definition classes and Web Service implementation classes are located in two entirely different directories. When the Ant process started, some properties were set, including the main sources. The secondary sources, which are the Web Service definition files, would not be set. Therefore, the only solution is to call the compile target explicitly and redefine the location of the sources using an Ant call parameter.

Add a note hereAdd a note hereAlso shown in Listing 11.23 is the use of the java task. Notice how parameters can be passed to the Java command line using the XML element arg.

Add a note hereAdd a note hereThe last way of calling another target is to explicitly call a target in another file, as shown in Listing 11.24.

Add a note hereAdd a note here**Listing 11.24**

Add a note hereAdd a note here<target name="GenerateWSDLStub">

<ant antfile="build.xml" dir="design" target="deploy"/>

<ant antfile="build.xml" dir="server"

target="GenerateWSDLStub"/>

<ant antfile="build.xml" dir="client"

target="GenerateWSDLStub"/>

</target>

Add a note hereAdd a note hereIn Listing 11.24, the target GenerateWSDLStub has three child XML ant elements, which are used to execute three different Ant tasks in the same file. Listing 11.24 is shown for illustrational purposes. The XML element ant is a task that calls a target in another file. The ant task has three attributes: antfile, dir, and target. The attribute antfile defines the name of the Ant file to be loaded. The attribute dir defines the directory where the Ant file to be loaded is located. The attribute target defines the target that is called on the Ant file to be loaded. Once the target destination has been called, the target destination is subject to all target-defined dependencies and Ant calls.

### Add a note hereAdd a note hereSome Tips for When You Use Ant

Add a note hereAdd a note hereHere are some tips on building good Ant files:

* Add a note hereAdd a note here**Structure your code so that it is logical.** This means that sources should be stored in the source directory, tests in the test directory, and configuration information in the conf directory. You can see multiple good examples of how to do this right in the Jakarta Commons and Commons Sandbox projects.
* Add a note hereAdd a note here**Use property files or Ant properties to define the location of specific jar files.** In addition, when defining them, use an abstract syntax similar to that used in Listing 11.25. (Note that Listing 11.25 uses the properties file notation, which could also have been the Ant property task.)

Add a note hereAdd a note here**Listing 11.25**

Add a note hereAdd a note herexml-rpc.jar=${jars-dir}/xmlrpc-1.1.jar

Add a note hereAdd a note hereIn Listing 11.25, the variable xml-rpc.jar is defined and references the specific file *Image from book*[*xmlrpc-1.1.jar*](http://cdcontent.books24x7.com/id_7265/xmlrpc-1.1.jar). This is very useful because it allows a developer to abstractly define the jar files that are being used, without defining a specific version number. When a new version of the jar file arrives, only one location needs updating.

* Add a note hereAdd a note here**Define abstract targets that are very generic and that are then referenced using specific dependencies or Ant calls.** The abstract target should be defined in a file called *targ*[*et.xml*](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053) and included into main *Image from book*[*build.xml*](http://cdcontent.books24x7.com/id_7265/build.xml) file using the DTD technique described earlier.
* Add a note hereAdd a note here**Use obvious identifiers, even if the identifiers are verbose.** Using obvious identifiers make it simpler to debug a problem in an Ant script.
* Add a note hereAdd a note here**If there are problems when you are running the Ant program, run the program with the –verbose or –debug flag.** In those modes, a large amount of information that can help a developer figure out what is not working is generated. For example, finding dead classpath references is very simple using these modes.

## **Source Management Using CVS**

Add a note hereAdd a note hereWhen you manage the sources, you are managing the files associated with a software project. Traditionally, source management refers to the files used to build the application. Now, it refers to the entire application, including design documents and management files. There are many different source code management systems: CVS, BitKeeper, ClearCase, SubVersion, etc. Each of the source code management systems has a different way of interacting with the source code repository. The focus in this book is on CVS, which, while not command-identical, contains principles shared by the other source code management systems. In addition, CVS is very popular.

### Add a note hereAdd a note hereLocking Files or Everybody-Access Files

Add a note hereAdd a note hereThe objective of a source code management system is to allow you to control access to a set of files. Let's say that you have two programmers, and instead of using a real source code management system, you use a file server. At one point in time, both programmers modify the same file. If both programmers save their changes, the question is what happens to the individual changes? The answer is not predictable, but we can predict that it will be the wrong source code because a file system will not merge two different changes to a file. Therefore, one programmer's changes will be lost. A source code management system manages these sorts of problems.

Add a note hereAdd a note hereIf two programmers are manipulating the same file, a source code management system can take two strategies. The first strategy is to provide access to the file for the first programmer and force the second programmer to wait until the first programmer is done with the file. This strategy is called the *lock-out strategy*, because the file is locked until a programmer unlocks it. The problem with this strategy is that only one person can manipulate the file, and other people have to wait. At first consideration, this is a good strategy because it ensures that files are in a consistent state. However, the problem with this strategy is that only one person can get access to the file. Maybe another person wants to try something or add some code to the file. The other person has to wait until the person who locked the file is done and has released the file.

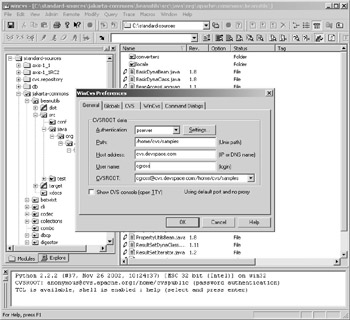
Add a note hereAdd a note hereThe second strategy is to allow everybody access to all the files, even if two programmers are manipulating the same file at the same time. However, we can quickly see that this brings up a problem. We'll have multiple versions of the same file that we will have to merge. The thought of manually merging the files is scary for any developer because it means sources could be lost. The reality is that source code management systems that use this strategy are extremely good at knowing how to merge source files. Therefore, the result is that these source code management systems are superior to lock-based file management systems. They are simpler to manage and integrate into an overall development process.

### Add a note hereAdd a note hereIdentifying Yourself

Add a note hereAdd a note hereWe will now discuss how to use the GUI tool *wincvs*. This tool exists on other platforms in essentially the same form (in UNIX, it's *gcvs;* in Macintosh, it's *MacCVS*). There are dozens of other tools to use, and you can use the command line as well. However, a GUI tool will ideally make your CVS experience that much simpler. The examples shown in this chapter can be used to download only the demos.

Add a note hereAdd a note hereEvery CVS repository is defined by a CVSROOT variable, which is an environment variable if the command line is used. [Figure 11.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851) shows how to define the CVSROOT variable

Larger View



Add a note hereAdd a note hereFigure 11.1: A dialog box that shows how to enter the CVSROOT parameters.

Add a note hereAdd a note hereIn [Figure 11.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851) the dialog box WinCvs Preferences allows a user to enter the user details without having to define the CVSROOT path. In the dialog box are a number of text boxes that need to be filled in. The Authentication textbox is a drop-down list of the different types of authentication mechanisms. For the most part, you will be using pserver, but local and ssh are also available. Local authentication is a very loose authentication because a read/write-accessible local directory is used. The ssh authentication is very useful for those with remote workers who need to retrieve sources from the Internet. The ssh will encrypt the requests, thus making it very difficult for anybody to decipher the usernames, passwords, and sources. The Path textbox references the path where the CVS repository is located on the server. The Host Address textbox is an address that contains the CVS sources. The User Name textbox specifies the username used to log in to the CVS repository. In [Figure 11.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the User Name is "cgross", which for downloading is normally "anonymous." The CVSROOT textbox is a generated string created by the values of the other textboxes. Clicking on the OK button will save these values.

Add a note hereAdd a note hereIf you want to download or upload a project, you must log in. You do this by using the WinCvs menu item Admin -> Login. If the login is a first-time login, then a password will be expected. Enter the password, or if the user "anonymous" is used, enter "anonymous", which is a default password. Listing 11.26 shows a successful login.

Larger View

Add a note hereAdd a note here**Listing 11.26**

Add a note hereAdd a note herecvs -d :pserver:cgross@cvs.devspace.com:/home/cvs/samples login

Logging in to :pserver:cgross@cvs.devspace.com:2401:/home/cvs/samples

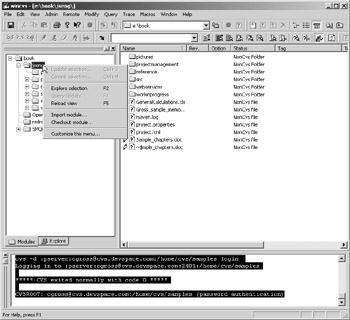
\*\*\*\*\* CVS exited normally with code 0 \*\*\*\*\*

CVSROOT: cgross@cvs.devspace.com:/home/cvs/samples (password authentication)

### Add a note hereAdd a note hereAdding a New Module

Add a note hereAdd a note hereAdding a new module to a repository means adding new sources. For example, the sources for this book have been made available using CVS. The advantage of this is that the developer can download the sources, and then get the latest updates when they arrive. However, since the author is the only one who can write to the database, the source tree does not go willy-nilly in all directions. The author can patch the sources with updates received by individual readers. (Hint: If you want other topics explained, or if you see problems or potential ways to improve the sources, then send in the patch files to the email address *jseng-patches@devspace.com*.) [Figure 11.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851) shows how the initial menu is used to upload a new module to the repository.

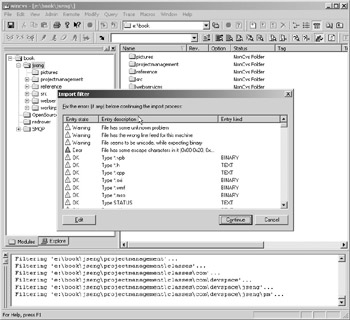
Larger View



Add a note hereAdd a note hereFigure 11.2: The context menu you use to upload a new module to the repository.

Add a note hereAdd a note hereFrom the left pane, which contains the various directories, select a specific directory with the cursor. Then, right-click and select the menu item Import module, which will cause WinCvs to go through the various directories and select a file set to be added to the repository. A small dialog box appears, which is the file and directory iterator. Once a file set has been defined, you can tweak how the files will be added, as shown by the dialog box in [Figure 11.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851).

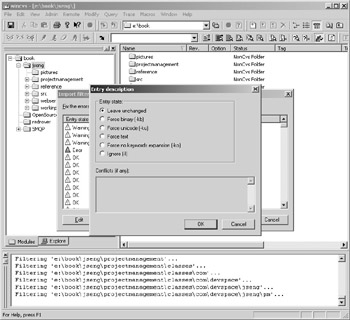
Larger View



Add a note hereAdd a note hereFigure 11.3: The dialog box that is used to tweak how files are imported.

Add a note hereAdd a note hereIn [Figure 11.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the dialog box Import filter contains a list of file extensions and types that it found. In this list box, you can edit what WinCvs should do with a file before importing to the repository. To edit a file type, select the item from the list box and then click on the Edit button. When you do so, a dialog box similar to [Figure 11.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851) appears.

Larger View



Add a note hereAdd a note hereFigure 11.4: Description details for a specific file type.

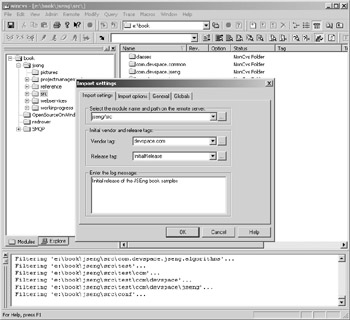
Add a note hereAdd a note hereIn [Figure 11.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the dialog box Entry description defines how to describe a file type. This is important because it defines how CVS handles merges and conflicts. The file type that we edited was for the extension cpp. The extension cpp represents a C++ source code file. Typically, this file will be treated as a text file. This means CVS will allow merges and other text operations.

Add a note hereAdd a note hereIf, instead, you want to treat the C++ source code file as a binary file, then you need to select the Force binary radio button in the dialog box. If a file is treated as binary, it means that no attempts will be made to translate the file, so merges and conflict resolution are not possible. Of course, for a truly binary file like an executable, this is a good thing. Instead of your attempting to merge a binary file, which does not work in any scenario, the different versions of the file replace each other. Binary is more like a file-server approach, except that the old versions are kept as history in the repository.

Add a note hereAdd a note hereIndicating that a file is Unicode-based means that the file is text-based but uses the Unicode character set, which is 16 bits wide in contrast to 8 bits. Selecting the Ignore option in this dialog box causes the *wincvs* tool not to include the specified file extension when you add new modules.

Add a note hereAdd a note hereWhen you let *wincvs* import the files using a search, all of the directories found will also be imported. In the case of the sources, this is not desired since the word processing documents used to create this book and the associated reference materials are also part of the Image from book[jseng](http://cdcontent.books24x7.com/id_7265/jseng.zip) module (including these documents will most likely not impress the publisher). These are private to the author and cannot be added. The *wincvs* tool does not have an option to prune directories. Therefore, to get around this, we will import the directories manually. To do this, we do the same thing as we did in [Figure 11.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851). This time, though, we will import the subdirectory src. If we click the Continue button in the dialog box Import filter instead of Edit, we see a dialog box similar to [Figure 11.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851).

Larger View

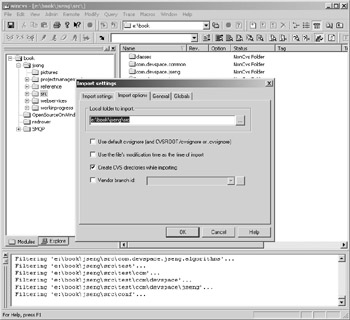


Add a note hereAdd a note hereFigure 11.5: Module repository details.

Add a note hereAdd a note hereIn [Figure 11.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the dialog box Import settings defines the details used to identify the module in the CVS repository. The textbox where the text "jseng/src" is located is the name of the remote repository location. The default that *wincvs* generated is "src", which corresponds to the name of the directory being imported. Generally this is acceptable, but not in this instance. This is because a directory within the module is being imported. Therefore, the text needs to be corrected to the value "jseng/src," where jseng is the actual module name. The Vendor tag textbox represents the name of the vendor that released the sources. The Release tag textbox represents the name of the initial release. More about tags will be discussed a bit later. The textbox for the log message is a description of the initial release. It is important that you always give the information on why something has been released, changed, or deleted. This makes it simpler to figure out which changes have occurred.

Add a note hereAdd a note hereWe now turn our attention to an important detail. When you're importing a set of sources using *wincvs*, the CVS directories used to manage the local sources are by default not generated. This means the imported directory will not be under source control. In the dialog box Import settings, it is important to select "Create CVS directories while importing" from Import options tab, as shown in [Figure 11.6](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851).

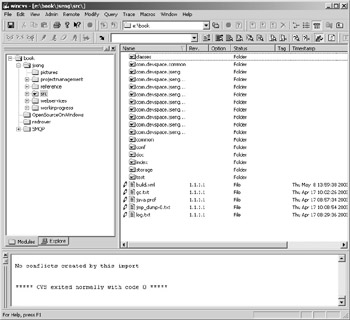
Larger View



Add a note hereAdd a note hereFigure 11.6: The Import Options tab of the dialog box Import Settings, with “Create CVS directories while importing” checked.

Add a note hereAdd a note hereOnce the source files have been imported, the source directory will change appearances in the *wincvs* window and will indicate that the sources are managed by CVS, as shown in [Figure 11.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851).

Larger View



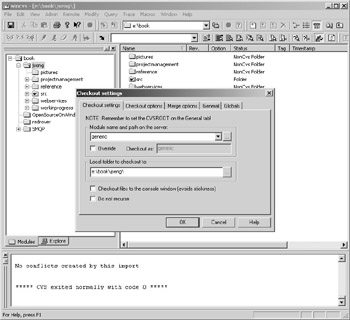
Add a note hereAdd a note hereFigure 11.7: The appearance of the CVS control sources.

Add a note hereAdd a note hereIn [Figure 11.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the CVS-managed directories appear as icons with a checkmark. Individual files that have a little pencil beside them mean that file is read/write. If the pencil is crossed out, then the file is read-only. A CVS-managed file has a normal paper icon as you can see in [Figure 11.7](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851). A non-managed CVS file has a small icon with a question mark.

### Add a note hereAdd a note hereChecking Out a Module

Add a note hereAdd a note hereIf instead of importing a module, you want to download one, you again call upon the context menu shown in [Figure 11.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851). This time, select Checkout module from the menu item; a dialog box similar to [Figure 11.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851) appears. Note that checking out the sources does not mean locking the sources, as would be the case in a locking-source code management system.

Larger View

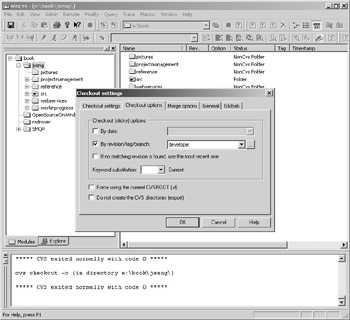


Add a note hereAdd a note hereFigure 11.8: A dialog box used to check out sources.

Add a note hereAdd a note hereIn [Figure 11.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the dialog box Checkout settings has only one textbox that needs filling in. The textbox where the value "generic" is typed in is called the module textbox. The module is the same thing that we imported in the [*Adding a New Module*](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851) section of the chapter. In the case of [Figure 11.8](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), we are attempting to download the module "generic." The module name can include subdirectories like the one shown in [Figure 11.5](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), where the module name "jseng/src" was used.

Add a note hereAdd a note hereAnother useful option is the ability to retrieve sources according to a date, tag, or branch, as shown in [Figure 11.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851).

Larger View



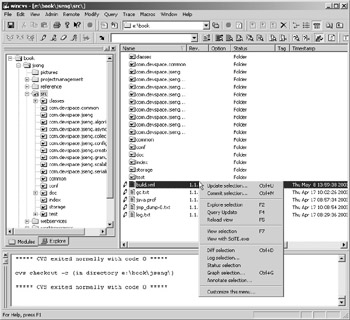
Add a note hereAdd a note hereFigure 11.9: Selecting a specific date, branch, or tag of the sources to check out.

Add a note hereAdd a note hereIn [Figure 11.9](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the sources that belong to the revision/tag/branch developer are being selected. (We will discuss the concept of sources or tags in more detail later in this chapter.) It is also possible to select sources according to a specific date. To do so, select the By date checkbox and then define a date for the sources. The idea of the Checkout options tab in the dialog box is allow you to download the sources in a particular state. Once all of the parameters have been set, you can click on the OK button, and the sources will be downloaded to the local machine.

### Add a note hereAdd a note hereUpdating, Making, and Undoing Changes

Add a note hereAdd a note hereIn the course of developing, you will make the application changes to the file. Sometimes, you will want to write the changes to the CVS server. When you're using *wincvs*, changes are highlighted using a red icon. In [Figure 11.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the file *Image from book*[*build.xml*](http://cdcontent.books24x7.com/id_7265/build.xml) is highlighted to indicate that is has been changed since the last update from the CVS server. To commit the changes, select the file and click on the context mouse button to make the context menu appear, as shown in [Figure 11.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851).

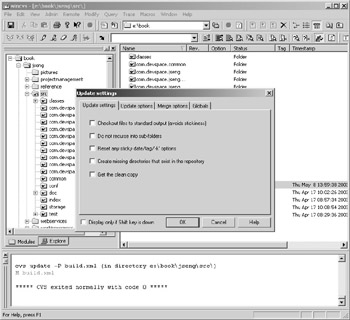
Larger View



Add a note hereAdd a note hereFigure 11.10: Selecting the context menu for an individual file item.

Add a note hereAdd a note hereThe context menu shown in [Figure 11.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851) has a number of available items. If you choose Update selection, the last release from the CVS server is pulled down. However, before that happens, a dialog box similar to [Figure 11.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851) appears. Note that the Update selection option can also be applied to directories, which is essentially saying, "get me the latest sources for a specific module."

Larger View



Add a note hereAdd a note hereFigure 11.11: The dialog box Update Settings.

Add a note hereAdd a note hereThe dialog box Update settings, shown in [Figure 11.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), allows you to exactly specify how the selection will be updated. If you only want to get an idea of the changes, then select the "Checkout files to standard output (avoids stickiness)" checkbox. This will dump the contents of the updated file(s) to the little console at the bottom of *wincvs*, as shown in [Figure 11.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851). The "Do not recurse into sub-folders" checkbox is used to stop updating all directories when a directory has been chosen. The checkbox "Reset any sticky date/tag/`-k' option" removes any tag or date information, essentially downloading the latest sources. The "Create missing directories that exist in the repository" checkbox is an extremely important checkbox, because when you're updating directories or modules, the default is not to get new directories. This means a developer could retrieve only part of the sources. If you're updating, you should check this option all the time. The "Get the clean copy" checkbox discards the old changes and downloads a fresh copy. No options are selected in [Figure 11.11](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), and Listing 11.27 is the result of updating a modified file.

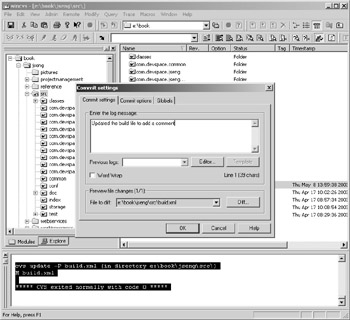
Add a note hereAdd a note here**Listing 11.27**

Add a note hereAdd a note herecvs update -P build.xml (in directory e:\book\jseng\src\)

M build.xml

\*\*\*\*\* CVS exited normally with code 0 \*\*\*\*\*

Larger View



Add a note hereAdd a note hereFigure 11.12: Committing changes for a specific file or directory set.

Add a note hereAdd a note hereIn Listing 11.27, CVS updated the file, but notice that things were changed. Instead of the old file being removed, a merge occurred. The merge is indicated by the letter M in front of the file *Image from book*[*build.xml*](http://cdcontent.books24x7.com/id_7265/build.xml). Although nobody but the author has updated the old file, in a team scenario, an update like this can cause a more complicated merge with conflicts. More about this a bit later in the chapter.

Add a note hereAdd a note hereLet's go back to [Figure 11.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), which showed the context menu. This time, we'll select the menu item Commit settings, and a dialog box similar to [Figure 11.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851) appears.

Add a note hereAdd a note hereIn [Figure 11.12](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the "Enter the log message" textbox is used to define a descriptive message for the log files. Clicking on the OK button will cause the file to be committed and updated. The log output of a successful commit is shown in Listing 11.28.

Larger View

Add a note hereAdd a note here**Listing 11.28**

Add a note hereAdd a note herecvs commit -m "Updated the build file to add a comment" build.xml (in directory e:\book\jseng\src\)

Checking in build.xml;

/home/cvs/samples/src/build.xml,v <— build.xml

new revision: 1.2; previous revision: 1.1

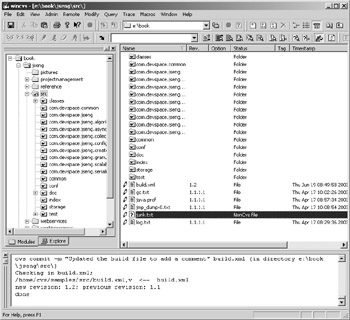
done

Add a note hereAdd a note hereIn Listing 11.28, notice how the version number jumps to 1.2 from 1.1, where it says new revision: 1.2.... In CVS, version numbers are not determined by the individual user, but by the number of commits that have occurred. Of course, when the sources have been branched, there are some numbering conventions where sub-number versions are generated, but this is generally the rule. Therefore, a number like 1.345 means that there have been 345 commits of the file.

### Add a note hereAdd a note hereAdding and Deleting a File

Add a note hereAdd a note hereAdding and removing files is not difficult with the *wincvs* tool. [Figure 11.13](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851) shows a file that has been added to the sources, but not the CVS repository. The file *junk.txt* has a little question mark next to it, which indicates that it has not been added to CVS repository.

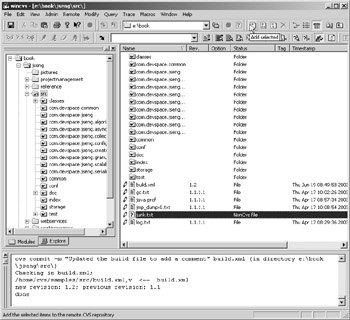
Larger View



Add a note hereAdd a note hereFigure 11.13: A highlighted file that is not part of the CVS repository.

Add a note hereAdd a note hereTo add the file to the repository, use the Add button icons, as shown in [Figure 11.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851).

Larger View



Add a note hereAdd a note hereFigure 11.14: Toolbar buttons to select when you’re adding a file.

Add a note hereAdd a note hereIn [Figure 11.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the mouse is hovering over an icon in the toolbar that has a little red cross. That little red cross toolbar button adds the file using the default add mode, which means let *wincvs* determine if the file is binary, text, or Unicode-based. If you need to force the file to be added in a specific mode, use the two buttons beside the mouse. The toolbar button with a little red cross and number 10 is used to add the file as a binary file. The toolbar button with a little red cross and letter U means to add the file as being Unicode-based. Once you've clicked on the correct Add toolbar button, the file is added to the repository. It is important to realize that the reference of the file has been added to repository. The file itself has not been committed. In *wincvs*, the file will appear with a read icon indicating changes. Therefore, to commit the changes, use the same sequence as defined in the [*Updating, Making, and Undoing Changes*](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851) section of this chapter.

Add a note hereAdd a note hereTo delete the file from the repository, select the file and then use the toolbar button that looks like an X, which is located beside the Add toolbar buttons in [Figure 11.14](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851). When you click on the button, the file is deleted from the repository. However, as with the Add, the delete has not been committed, and an additional commit has to be issued to fully delete the file from the repository.

### Add a note hereAdd a note hereTagging, Branching, and Merging a Source Tree

Add a note hereAdd a note hereOne of the powers of a source code management system like CVS is in its ability to tag, branch, and merge sources. In this section of the chapter, we will create a brand new module. The module will be called "test" and it includes a single file called *simple.txt*. The contents of simple.txt after the initial addition to the repository are shown in Listing 11.29.

Add a note hereAdd a note here**Listing 11.29**

Add a note hereAdd a note hereHere is a single line of text

Add a note hereAdd a note hereListing 11.29 is simple and shows a single line of text. Adding a second and third iteration, we create Listing 11.30.

Add a note hereAdd a note here**Listing 11.30**

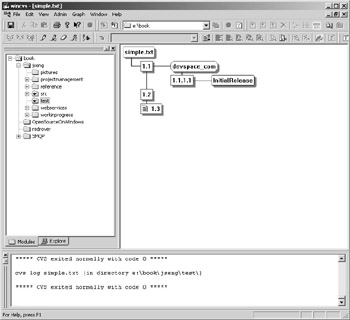
Add a note hereAdd a note hereHere is a single line of text

Line is added in second iteration

Line is added in third iteration

Add a note hereAdd a note hereTo track our changes, *wincvs* has an interesting ability to graphically show the various changes. The way to show the graphical diagram is to select the menu item Graph Selection from the context menu shown in [Figure 11.10](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851). The graphical change log is shown in [Figure 11.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851).

Larger View



Add a note hereAdd a note hereFigure 11.15: Log of changes graphically shown in wincvs.

Add a note hereAdd a note hereIn [Figure 11.15](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), there are four boxes that contain numbers. The box with the number 1.1.1.1 indicates an initial addition to the repository. The boxes with the numbers 1.1, 1.2, and 1.3 define individual changes. If one of the boxes is selected, the log message and change statistics are displayed in the output window. Listing 11.31 shows the code for the change 1.2.

Add a note hereAdd a note here**Listing 11.31**

Add a note hereAdd a note hereRevision: 1.2

Date : 2003/6/19 9:55:02

Author : 'cgross'

State : 'Exp'

Lines : +2 0

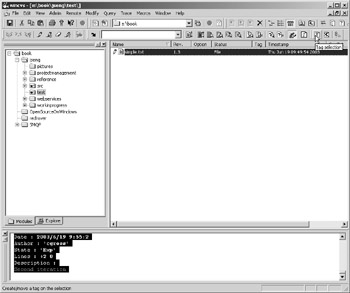
Description :

Second iteration

Add a note hereAdd a note hereIn Listing 11.31, six pieces of information are displayed about an individual change. Most of them are obvious, but the change "Lines" is not. "Lines" indicates the number of lines, added and deleted. The +2 indicates that two lines have been added, and the 0 indicates that no lines have been deleted. If any lines were deleted, then they would be indicated using a minus sign and associated number. The purpose of the "Lines" property is to display how often a module changes. Remember way back to the initial chapters of this book, where it was necessary to find buggy modules that cause the most problems. The "Lines" property could be an indicator.

Add a note hereAdd a note hereAfter all of these changes, you will now want to tag the current file as a released version. Tag the selection using the Tag Selection toolbar button, shown in [Figure 11.16](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851).

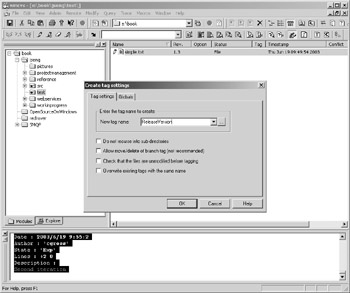
Larger View



Add a note hereAdd a note hereFigure 11.16: Tagging a specific file.

Add a note hereAdd a note hereOnce you've clicked on the Tag Selection toolbar button, a dialog box similar to [Figure 11.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851) appears.

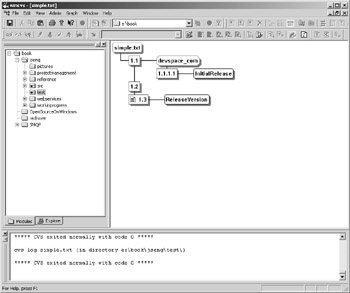
Larger View



Add a note hereAdd a note hereFigure 11.17: The dialog box Create Tag Settings.

Add a note hereAdd a note hereIn [Figure 11.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the dialog box Create tag settings has one text field and some checkboxes. The only required field is the textbox that defines the identifier of the tag. Note that the tag identifier cannot have spaces, which is a limitation of CVS. Once you've defined the tag identifier, click on OK. The various checkboxes are used to modify how the tag is generated and for which files. To see the results of our tagging, the change log file for the file *simple.txt* is generated again. It's shown in [Figure 11.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851).

Larger View



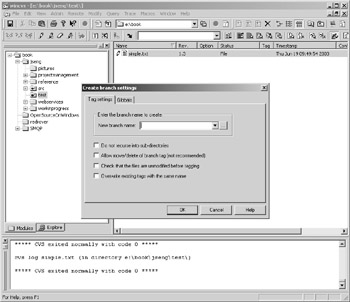
Add a note hereAdd a note hereFigure 11.18: Change log file showing ReleaseVersion.

Add a note hereAdd a note hereNotice now in [Figure 11.18](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851) that the box with the number 1.3 has a connected box with the text "ReleaseVersion." This tag now specifies that whenever anybody checks out the sources with the tag "ReleaseVersion," that person will get version 1.3 of the file *simple.txt*. Tagging is useful when you're trying to reproduce bugs or create stable versions from an active source tree.

Add a note hereAdd a note hereHaving tagged the source file, let's branch the current version into a stable bug-fix branch and a development branch. The reason we want to do this is so that we can continue bug fixes on a source code file, but not include development changes. However, when we create a final version, we need to include the bug fixes in the final version. One branch will be used to track bugs, and the other branch to track new developments. Once the new developments have been completed, it is necessary to merge the two branches again. At this point, the sources are synchronized, and called a version that most likely has a release tag.

Add a note hereAdd a note hereTo create a branch, click on the Branch Selected toolbar button, as shown in [Figure 11.19](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851).

Larger View



Add a note hereAdd a note hereFigure 11.19: Branching a specific file.

Add a note hereAdd a note hereAfter you click on the Branch Selected button, a dialog box will appear and in the title area will be the text "Create branch settings." This dialog box will appear similar to the dialog box used to tag a branch as shown in [Figure 11.17](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851). The name of the new branch is "developerbranch," which is entered into the New branch name textbox.

Add a note hereAdd a note hereAt this point, the current source tree downloaded on the hard disk is still the main trunk and not the branched developer version. Making a change to the file *simple.txt* will update the main branch. To get the developer branch of the sources, the source code tree would have to be checked out again, but in the checkout of the sources the branch to retrieve has to be defined.

Add a note hereAdd a note hereWhen you check out and commit the changes of a developer branch-based file, the version numbers will change. For example, the numbering could be 1.3.2.1. What this version number says is that the second branch with the first commit modifies the 1.3 version file from the main trunk. Putting it a bit simpler, the 1.3 in the branch indicates the version that the branch is based on from the main trunk. The 2 is the unique identifier for the developer branch, and the trailing 1 represents the revision number of the file.

Add a note hereAdd a note hereListing 11.32 represents the developer branch state of the file *simple.txt*.

Add a note hereAdd a note here**Listing 11.32**

Add a note hereAdd a note hereHere is a single line of text

Added in developer branch

Line is added in second iteration

Line is added in third iteration

Second add in developer branch

Add a note hereAdd a note hereListing 11.33 represents the main branch of the file *simple.txt*.

Add a note hereAdd a note here**Listing 11.33**

Add a note hereAdd a note hereHere is a single line of text

Line is added in second iteration

Line is added in third iteration

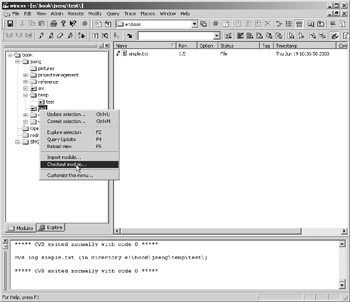
Added in release branch

Add a note hereAdd a note hereThe challenge is the merging of the two files, which is handled automatically when you use CVS. *wincvs* makes it easy to perform a merge by doing an update with a merge.

Add a note hereAdd a note hereTo do that, the directory has to be selected and then merged. In [Figure 11.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the directory to be merged is selected.

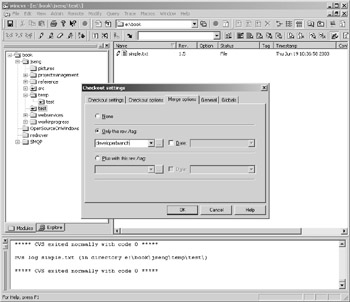
Add a note hereAdd a note hereIn [Figure 11.20](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the directory "test," which represents the source tree for the main trunk, is highlighted. If you right-click on it, the context menu appears. Select the menu item Update selection. You'll see a dialog box similar to [Figure 11.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851) when you select the Merge options tab in the dialog box Update settings.

Larger View



Add a note hereAdd a note hereFigure 11.20: Selecting a module to check out and update.

Larger View



Add a note hereAdd a note hereFigure 11.21: Selecting a branch to merge.

Add a note hereAdd a note hereIn [Figure 11.21](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the dialog box Update Settings with the tab Merge options selected exposes some radio buttons. The radio button "Only this rev./tag" is selected. This makes a drop-down list box become available; enter the text "developerbranch." What this dialog box is now saying is, "please check out the developerbranch of the test module on the current main trunk of the test module." Clicking OK will start the merge process. The results will then generate a *file.#simple.txt.1* and an updated *simple.txt* file. The file name *.#simple.txt.1* is the original file, so the old reference data is kept. Listing 11.34 shows the state of the file simple.txt after the merge.

Add a note hereAdd a note here**Listing 11.34**

Add a note hereAdd a note hereHere is a single line of text

Added in developer branch

Line is added in second iteration

Line is added in third iteration

<<<<<<< simple.txt

Added in release branch

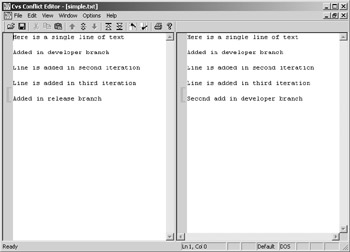
=======

Second add in developer branch

>>>> 1.3.2.2

Add a note hereAdd a note hereIn Listing 11.34, the second line of text that was added in the developer branch was merged into the main trunk without any problems. However, the last lines added in both the developer trunk and the main trunk were problematic. The merge processor has determined these files to be problematic and to need additional changes. At this point, it would be up to the developer to figure what the correct state is. An easy way to fix this up, is to an use a conflict editor, as shown in [Figure 11.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851).

Larger View



Add a note hereAdd a note hereFigure 11.22: Using a conflict editor to merge two files.

Add a note hereAdd a note hereIn [Figure 11.22](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=137660851), the conflict editor highlights the areas of conflict using square brackets in the margin. The developer can then select the conflict and merge it to the main program by copying one merge above the other or vice versa. Once the conflicts have been looked at and resolved, the file is committed back to the main trunk.

Add a note hereAdd a note hereThe process of managing merges is what the team leader does. In fact, this should not be a surprise to the team leader because the team leader should know what the sources are doing. Some people think that managing conflicts and merges like this is extra unnecessary work. The reality is that it makes the work easier because multiple people can work with multiple states of the source code tree. Imagine having to add a module that depends on another module. If the other module is constantly changing, then the added module will constantly have to adjust for the instability. This adds to the development time. It would be simpler to develop individual units and then perform an integration test when the merge has been carried out. However, this strategy does take some time for novices to get used to.

**Summary**

Add a note hereAdd a note hereThe programming language Java introduced not only an easier-to-use language but also a much better project management system. Ant and *JUnit* are written in Java, and these tools are the best of their kind. What is obvious is that Java makes people focus on other issues like writing good programs and less on making sure that the infrastructure used to write the programs works. Java programmers assume that the infrastructure they use works. And, in this case, the tools are project management tools. Not using Ant or *JUnit* is like saying, "I want to make my life more difficult." CVS is a bit older but works extremely well for the problems that it sets out to solve. Put the three tools together and a manageable software development process ensues.

Add a note hereAdd a note hereThe book has ended. I am happy you made it, and hope that your journey has been worth it. So, what should you have learned?

* Add a note hereAdd a note hereSoftware engineering can be decomposed into problems that solve specific tasks, such as object creation, or serialization. Convert those tasks into generic components and many software problems disappear.
* Add a note hereAdd a note hereThe Jakarta Commons is an incredible set of libraries that have already solved many of the tasks that everybody needs done.
* Add a note hereAdd a note hereThe Jakarta Commons is not a library based on somebody's idea of what would make a good library. The Jakarta Commons is a set of proven Open Source libraries that are used and have proven to be effective.

Add a note hereAdd a note hereE-mail the author at [*jseng@devspace.com*](mailto:jseng@devspace.com) if you have any questions.

Add a note hereAdd a note hereOtherwise, thanks for taking the time to read the book, and have a good day!

## **Questions**

Add a note hereAdd a note here**11.1:** Write a test script that tests the calculator and calculator Web Service implemented in previous questions.

Add a note hereAdd a note here**11.2:** Write an Ant script that compiles, deploys, tests, and packages everything that you have written. The objective of this example is to allow an administrator to download your sources from CVS, and then automatically configure, compile, and deploy the application and Web Service. Success is measured by how easy the entire process is.

Add a note hereAdd a note here**11.3:** Describe what you did throughout when solving the questions that relate to the calculator, and look at what went right and what went wrong. All of the questions in the book do create some degree of chaos. This was intended to replicate what happens in real-life software engineering. Reflect and look at what could have been done better or worse. It is important that you reflect and jot down the good and bad of solving the questions, as it gives you a way to figure out how to approach the next programming task.

Add a note hereAdd a note here**11.4:** Add the sources and documents to a CVS system (note: it's OK to use another version control system). Structure the sources so that a team could develop further enhancements. Create a stable branch and development branch.

# Appendix A: **Making Decisions Quantifiable**

Add a note hereThe material presented in this appendix is not part of the overall book, but it is something that can be used to help you make decisions. Too often in software engineering, people make decisions based on what are the "right" reasons. However, those reasons are in fact decisions made in meetings based on somebody's opinion. That, in general, is not a bad thing, but using only that sort of decision-making leads to problems in the development of the software application. The purpose of this appendix is to make the decision involved in selecting features and finding problems a process that can be expressed using a mathematical formula.

## **Add a note hereAdd a note here****Step 1: Defining the Action Set**

Add a note hereAdd a note hereThe first step is to define the action set that will be implemented. The action could be correlated to the feature set, but defines a set of actions that need to be implemented. Features can be actions, and actions can be features, and if it makes you happier, you can use the term "feature" instead of "action." The term "action" is used because "action" is more generic. For example, an action could be maintaining old code, which is definitely not a feature.

Add a note hereAdd a note hereActions could be represented in any format you desire, which could be a spreadsheet or word processor. Since XML is used throughout this book, the actions will be represented as an XML document. For example, Listing A.1 shows a sample action and proposed sub- actions.

Larger View

Add a note hereAdd a note hereListing A.1

Add a note hereAdd a note here<action name="proposed" category="design"

date="2002-01-16" timeToImpl="1" timeToImplRisk="1">

<title>Define Web Service Interface</title>

<description>

Do a high-level analysis of what the Web Service will expose as

methods

</description>

<action name="proposed" category="meeting"

date="2002-01-16" timeToImpl="1" timeToImplRisk="8">

<title>Talk to Business</title>

<description>

Get the business team involved and ask them what

they would like to see

</description>

</action>

</action>

Add a note hereAdd a note hereIn Listing A.1, an XML fragment is used, and multiple actions, with various attributes, are being defined. First, consider in this fragment that there are two action tags. The reason for this is to simplify action dependencies. For example, a higher task might be designing the Web Service. Here, one of your tasks will be to talk to the business team. The two actions are dependent on each other. This is important because it means that if for some reason the Web Service Definition action is not required, then neither is the discussion with the business team.

Add a note hereAdd a note hereIt's very important, therefore, that the granularity of the action be fine. Otherwise, you will not have control of your software engineering process. The key when you are defining a good process is being able to conceptually figure out what is important and what is not. In the action definition process, your objective is to define a set of variables that you think is correct and then, after the project, to correlate what was actually correct. This way, you can adjust your software process to your specific situation.

Add a note hereAdd a note hereEach action tag has the following attributes:

* Add a note hereAdd a note here**name**: This is the identifier of the action tag used to uniquely define the action.
* Add a note hereAdd a note here**category**: This is the category of the action, which can be whatever your internal process is. Examples include design, management, and other terms related to the software engineering process.
* Add a note hereAdd a note here**date**: This is the date at which this action item was created or last edited.
* Add a note hereAdd a note here**timeToImpl**: This is the proposed time that it will take to realize this action. This could be hours, minutes, days, months, or years. It's typical to use days.
* Add a note hereAdd a note here**timeToImplRisk**: This is the certainty of realizing the action as per the timeToImpl attribute. A lower number indicates no risk, whereas a higher number indicates risk. The idea with this attribute is to assign how certain anybody is of realizing a specific action. It is a good idea because it adds a certain boundary of tolerance when you are realizing actions. When the attribute timeToImpl is combined with the attribute timeToImplRisk using a formula, the true time to implement is calculated even though you might not believe it initially.

Add a note hereAdd a note hereBesides the child element action, there are two other child elements: title and description. These two fields provide components that make the action understandable to humans. The child elements used in this example are for illustration purposes only and can be whatever the software process specifically needs.

## **Step 2: Implementing the Correct Actions**

Add a note hereAdd a note hereAfter you've defined all the actions, you need to decide which actions to implement and in which order. This is an absolutely crucial question because no one in any team has unlimited resources. Therefore, you must establish a priority. In a corporate environment, those priorities are often based on who cries loudest. This is wrong and leads the software engineering process into a spiral of chasing an unreachable goal. Just as when a dog chases its own tail, this way of doing things is not efficient.

Add a note hereAdd a note hereThe best way of knowing which action needs to be implemented is to calculate the value. The trick is to convert a gut feeling into a quantitative value. The calculation does not remove the gut feeling; rather, it converts the feeling into a really strong or really weak one. A really strong or really weak feeling is a number value. When you combine this value with other factors, you create a valuation.

### Add a note hereAdd a note hereAdditional Factors

Add a note hereAdd a note hereWhen you were defining the action, you had to define the time to implement and its associated risk. However, some additional factors, which help determine the implementation viability, are required. For example, attributes that can be associated with the action XML tag could be as defined in the following list:

* Add a note hereAdd a note here**political**: If you implemented this feature, would it add to the project's political viability? A low value indicates no political advantage, whereas a high value indicates the opposite.
* Add a note hereAdd a note here**benefit**: If you implemented this feature, would the benefit of the project increase? A low value indicates no benefit to the project, whereas a high value indicates the opposite.
* Add a note hereAdd a note here**cost-of-implementation**: If you implemented this feature, would the cost of the project significantly increase? The increase in cost is based on costs other than labor, such as testing by an external institute or the purchase of additional hardware. A low value indicates a high cost, whereas a high value indicates a low cost. It would seem that a low value equals high cost is opposite to how the value should be defined. This is because a low cost of implementation is beneficial and hence results in a higher value. The end comparison value must be as high as possible to be considered a candidate for implementation.

### Add a note hereAdd a note hereCalculating the Viability of an Action

Add a note hereAdd a note hereCalculating the viability of an action takes three steps. The first step is to make a calculation based on the time to implement and the risk associated with the implementation. The second step is to make a calculation that combines all of the additional factors. The last step is to combine the first and second steps so that you come up with an end number.

Add a note hereAdd a note hereOnly two factors are involved in the first-step calculation are time to implement and risk to implement. When you're attempting to define a good situation, which has the highest number, you want the least risk. A good action would be both short and risk free. Equation A.1 represents a good calculation.

Add a note hereAdd a note here(A.1)

Add a note hereAdd a note here((([time to implement] / [max time to implement]) \*

[constant 1] / ([constant 1] + [constant 2])) +

(([risk of implementation] / [max risk] \* [constant 2]) \*

[constant 2] / ([constant 1] + [constant 2]))) \* 100

Add a note hereAdd a note hereFor the calculation to be effective, the max values have to actually be the maximum values of the project. If they're not, the mathematical calculations will be incorrect and meaningless, which is very important to avoid. The constant values are used to balance the scalings.

Add a note hereAdd a note hereThe idea in Equation A.1 is to normalize the time to implement and the risk of implementation factors. *Normalization* is the process of taking a number and making its value between a fixed band of ranges. Normalization is important because it allows you to combine other factors without having one factor dominate the result. The purpose of the constants is to weigh one factor over the other. For example, a higher value for constant 1 than constant 2 means that you desire a better time to implement. The last multiplication of 100 converts the scaling to a value between 0 and 100. Once the equation has been applied to various actions, a higher number means a more desirable action.

Add a note hereAdd a note hereThe second step of the calculation requires you to combine the additional factors and apply normalization, as we did in the first equation. Equation A.2 demonstrates this.

Larger View

Add a note hereAdd a note here(A.2)

Add a note hereAdd a note here(([factor 1] / [max factor 1] \* (constant 1 / (constant 1 + ... + constant n))

+ ... +

([factor n] / [max factor n] \* (constant n / (constant 1 + ... + constant n)))

\* 100

Add a note hereAdd a note hereIn this equation, each factor is normalized against the maximum value of the associated factor. Then we apply a weighting to factor in the importance of additional factors. In the [*Additional Factors*](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=581946087) section of this appendix, some weightings that could be used for the additional factors are: *political 10*, *benefit 30*, *cost of implementation 20*. We developed these weightings by looking at an action that represents the weakest weighting. The weakest additional factor is political, which is assigned a value of 10. Then, the next weakest additional factor is considered and compared to the weakest additional factor. The question that you need to answer is how much more important the next weakest additional factor is in comparison to the weakest additional factor. In this example, the cost of implementation factor is twice as important as the political factor, which results in a weighting value of 20. This process is continued for all factors. A final consideration is to compare the individual weightings to each other. For example, consider if the benefit weighting is actually 1.5 times better than the cost of implementation weighting. If this does not correspond to your gut feeling, adjust the value.

Add a note hereAdd a note hereWe use this approach to weightings because it is easier to say an additional factor is two, three, or four times as important than another factor. It is harder to say if an additional factor is a half, quarter, or eighth as important as the most important additional factor than another factor. The second-step calculation normalizes the weightings, which therefore does not require the user to figure out if the distribution of the individual additional factors is mathematically correct. As with the first calculation, the second calculation includes a multiplication of 100 to the end result. This puts the additional factor value into the range of 0 and 100.

Add a note hereAdd a note hereThe third calculation combines the results from the first and second calculations to generate an end value. In the simplest approach, the two values can be added together and the result can be used to compare each action and determine which is the most important. While this approach works, it is susceptible to the problem of not having weightings associated with the first and second calculation. The simplest third-step calculation implies that you do not care more about the simpler-to-implement feature than the additional factors.

Add a note hereAdd a note hereIt is possible to normalize the first and second calculation results using Equation A.3.

Larger View

Add a note hereAdd a note here(A.3)

Add a note hereAdd a note here([result calculation 1] / 2.0 \* (constant 1 / (constant 1 + constant 2)) +

([result calculation 2] / 2.0 \* (constant 2 / (constant 1 + constant 2))

Add a note hereAdd a note hereA higher value for constant 1 means that you consider the best time to implement and lowest risk action to be more important than the best additional factor action. However, the third calculation is purely optional, as either implementation of the calculation is acceptable.

Add a note hereAdd a note hereSome additional notes about the calculations:

* Add a note hereAdd a note here**The various constants 1, 2, etc., are only scoped to the individual calculation.** Therefore, when you implement the equations, do not assume that you can reuse constant 1 from the first calculation in the second or third calculation.
* Add a note hereAdd a note here**Math is wonderful because it allows you to get the value when you want.** Therefore, it is vitally important that if you modify Equations A.1, A.2, and A.3, they normalize and weight the data. Otherwise, the generated values will be meaningless. You can modify the calculations by using mathematical curves to highlight specific valuations of a factor. Math can be used to do anything you want, and therefore it is important to be objective when devising the formulas. If you fudge the formula or the data, you are only fudging your own decision.

Add a note hereAdd a note hereOnce you have applied the calculations, you will get a list of numeric values. The highest-value numbers indicate actions that should be accomplished (higher numbers indicate a higher priority). Based on these priorities, you can begin distributing actions among your team members.



**Step 3: Analyzing a Completed Action**

Add a note hereAdd a note hereNext you need to consider the total time required to implement and debug the action. The problem with tracking these times for an action is that a programmer or QA team member must track how long they work on this. This can seem to be like filling out a time card, and knowledge workers who are part of a team don't usually like time cards. Therefore, your team members may resist. A good solution to this problem is not to use a time card but to have bi-weekly update meetings. The team leader or auditor manages these meetings, and he goes through the action items to see what work needs to be completed. This approach puts an individual touch on things, and it does not require the programmer or QA team member to track his own time.

Add a note hereAdd a note hereThe total time taken to implement and debug are represented as attributes of the action XML tag: timeImpl and debugTime. Once you come up with these values, you compare them to the time to implement and risk in implementation. At this point, the risk is in the form of a numeric value that indicates how risky the action is. We'll convert it into a time using Equation A.4.

Add a note hereAdd a note here**(A.4)**

Add a note hereAdd a note here[time to implement] + (e([risk of time to implement]) /

e([risk of time to implement] / [constant]))

Add a note hereAdd a note hereThe calculation uses the logarithmic function to calculate the risk. A logarithmic function reflects how we think. For example, if the risk indicates that the actual time to implement should be an extra couple of time units, then it is easy to judge. However, a risk of 10, which for this book is considered maximum, could add months or years to the project. This sudden jump from days to months or years is consistent with how humans think. Humans can assess low risk well, but not large risks. Large risks involve too much time and other variables. The *e* logarithmic function exponentially increases its result as the input value increases. The only problem with this way of calculating is that if the value of *e* is 10, a value of 22,000 results; this is way too high. The problem is that the exponential curve swings too high, too fast. The solution is to divide the *e* result with another *e* result that is modified by a constant as shown in Equation A.4. Note the variable in the entire equation is the true value of the constant variable. This is a real number between one and two and is found by tracking your own development process.We now use Equation A.5 to do a post-mortem analysis of the action.

Larger View

Add a note hereAdd a note here**(A.5)**

Add a note hereAdd a note here[time calculated with risk] - ([time to implement] + [time to debug])

Add a note hereAdd a note hereThe end result of Equation A.5 is a time difference between the actual time and the predicted one. Typically there will be a difference. The goal is to make the difference as small as possible. The way to do that is to create a feedback loop that fixes up your assessments for the future.

Add a note hereAdd a note hereMathematical statistics help us determine which actions to focus on. Using statistics, we can calculate a variance, and based on that variance we can pinpoint specific actions as problem actions. The age-old rule that 80 percent of the problems are caused by 20 percent of the actions is absolutely correct. In statistics, the rule roughly approximates to a standard deviation. A standard deviation assumes that the data falls into a bell curve. The bell curve says that 66.7 percent of the data is within one standard deviation. Ninety-five percent of the data is contained in two standard deviations, and 99.7 percent of the data is contained in three standard deviations.

Add a note hereAdd a note hereTo find the problematic actions, find all actions where the calculated time to implement and actual time to implement is greater than one standard deviation away from the calculated average. Note that this proposed calculation is not saying, "find all actions that were not implemented in a reasonable amount of time, but find the actions that gave the most problems." Finding those problematic actions allows a developer to find out what went wrong. To calculate the standard deviation, we use Equation A.6, which calculates the median value of the differences.

Add a note hereAdd a note here**(A.6)**

Add a note hereAdd a note here1 / [n] \*([difference 1] + ... + [difference n])

Add a note hereAdd a note hereOnce you have determined the median value, you can use Equation A.7 to calculate the standard deviation.

Add a note hereAdd a note here**(A.7)**

Add a note hereAdd a note hereSQRT( 1 / (n – 1) \*

(POW([difference 1] – [median], 2) + .. +

POW([difference n] – [median], 2))

## **Step 4: What to Do with a Problematic Action**

Add a note hereAdd a note hereWhen you find a problematic action, you need to resolve it. It's not always easy to figure out why an action was problematic, however. The causes could be everything and anything, or nothing. The first step in determining a problematic action is to find the factors that influence the action.

### Add a note hereAdd a note hereSimple Problematic Actions

Add a note hereAdd a note hereYou can easily solve a *simple problematic action* because only one fault is evident. Perhaps this single fault is that an inexperienced developer carried out a specific action, resulting in a longer implementation time. The solution to this simple problematic action is to assign the developer to another action that better suits him.

Add a note hereAdd a note hereWhen you apply a solution to a simple problematic action, this particular problematic action does not surface again, at least not in the same way. If a problematic action keeps coming up, then the action is complex, not simple.

### Add a note hereAdd a note hereComplex Problematic Actions

Add a note hereAdd a note hereA *complex problematic action* is more complicated than a simple one because you have more than one problem to resolve. In addition, it is often difficult to figure out what the exact problems are. One way to figure them out is to trace past projects and set up an experiment to test a hypothesis. In statistics, this process is called *robust design*. This process is beyond the scope of this book. If you want more information, search for materials on "robust design" and "Taguchi Quality Methods." Here, we will use the matrix approach to decompose the problematic action and design an experiment. The matrix approach is where actions, attributes, and factors are weighed against each other to find out what the cause of the problematic action is.

Add a note hereAdd a note hereTo figure out the cause of a problematic action, you need to break the project actions down into various factors. In addition, you need to assign to each factor a series of variable values. In the next few sections, we discuss some factors that you can use in your own projects. These factors are just examples. In your situation, you may have more, fewer, or entirely different factors.

#### **Factor: Programming**

Add a note hereAdd a note hereThe programming factor indicates whether or not the code was well written. The programming factor does not include the design of the code, because that is another factor. This factor deals solely with what you intended to accomplish. An indication of a bad programming factor is if it takes longer to debug the implementation than it did to implement it. This book focuses on how to manage this aspect of the software engineering cycle.

#### **Factor: Design**

Add a note hereAdd a note hereThe design factor indicates whether or not the project was well designed. The scope of the design factor includes the design of the UML model, interfaces, and higher-level items. It does not include the documentation but is based on the content of the documentation. This book focuses on the design of the software engineering cycle, too.

#### **Factor: QA**

Add a note hereAdd a note hereThe QA design factor indicates how well the action was tested, if testing was necessary. QA is problematic when the tests were incorrectly set up, when the tests were not correctly started, and when the bugs are not caught and are present when the end-user uses the application.

#### **Factor: Programming Tools**

Add a note hereAdd a note hereThe programming tools factor indicates which programming tools (such as editors, compilers, and build tools) are giving problems. It becomes a problem when the programmer or QA team member achieves poor productivity. The productivity of the tools is a very touchy issue because some people swear they are productive using editors like emacs or vi. This statement may or may not be true since it depends on whether the user is productive and comfortable with the tool. So, the objective is to figure out if the lack of productivity relates to inexperience or if it is not the right tool for the person.

#### **Factor: Hardware Used**

Add a note hereAdd a note hereThe hardware factor is when a person in the team cannot perform his daily work because of hardware problems. Defective or slow hardware can cause hardware problems, but so can configuration issues. Maybe the security of the environment is restricted so much that a developer has to do many tricks to get his work done.

Add a note hereAdd a note hereAnother cause of a hardware problem is when people use non-mainstream configurations. This type of problem occurred when multi-processor machines first became available for PCs. Windows NT was not optimized for multi-processor machines, so such compu- ters would periodically lock for no apparent reason. Software changes were lost, and productivity slowed to a crawl.

#### **Factor: Communication**

Add a note hereAdd a note hereOne of the most important aspects of the software engineering process is communication. Communication includes documentation, e-mails, and meetings. The documentation of the software project is included in communication because the technique of documentation could cause people to become confused and issue bug reports when in fact the documentation was incorrect.

#### **Factor: Contracts**

Add a note hereAdd a note hereContracts are a factor in the action because they are often created by your team and other companies, departments, or teams. The problem with contracts is that they often have absolutely nothing to do with the implementation of the project. The contract is orthogonal to the software, but the contract can dictate conditions that can influence a project for better or worse.

Add a note hereAdd a note hereA contract that does not allocate enough time for project completion would be an example of a contract influencing the project for the worse. If you have too little time, all of the actions will be rushed, so the quality of the project will suffer. Another problematic contract might not allocate enough money, which strains the resources.

### Add a note hereAdd a note hereConstructing a Problematic Matrix

Add a note hereAdd a note hereAfter you have found the factors, you must assign values to them and then generate a problematic matrix. A *problematic matrix* is a simplification of a robust-design orthogonal matrix. In robust design, you use an orthogonal matrix to generate experiments, whereas you use a problematic matrix to figure out what went wrong and then try some new ideas to fix the problem. When the investigator of the matrix generates a problematic matrix, looking at past projects will help him immensely because he will be able to identify trends and patterns.

Add a note hereAdd a note hereFor each factor defined, you need to define a set of valid states, such as:

Add a note hereAdd a note here**Programming**: John, Michael, Lisa, Marie

Add a note hereAdd a note here**Tools**: editor1, editor2, editor3, editor 4, etc

Add a note hereAdd a note here**Communication**: Paper communications, Paper and phone communications, Paper, Phone, and e-mail communications

Add a note hereAdd a note hereNext, you add these factors and valid states to the problematic matrix, as shown in [Table A.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116725006)

**Add a note hereAdd a note hereTable A.1:** Example Problematic Matrix Version 1

**Larger View**

| **Add a note hereAdd a note hereProgramming** | **Add a note hereAdd a note hereTools** | **Add a note hereAdd a note hereCommunication** | **Add a note hereAdd a note hereAction** | **Add a note hereAdd a note hereDifference** |
| --- | --- | --- | --- | --- |
| Add a note hereAdd a note hereJohn | Add a note hereAdd a note hereeditor1 | Add a note hereAdd a note herePaper | Add a note hereAdd a note hereImplement module 1 | Add a note hereAdd a note here30 |
| Add a note hereAdd a note hereLisa | Add a note hereAdd a note hereeditor2 | Add a note hereAdd a note herePaper | Add a note hereAdd a note hereImplement module 2 | Add a note hereAdd a note here25 |

Add a note hereAdd a note hereIf you analyze this table, you'll see in the Action column that the implementation of module 1 produced a bigger difference than the implementation of module 2 as defined in the Difference column. It seems that the difference in implementation has to do with the tools that John and Lisa are using. It seems that because John used editor1, productivity is hampered. However, this is absolutely the wrong conclusion. It is true that the variance due to the difference is on the shoulders of John; editors could be to blame. However, we also need to consider that both John and Lisa use paper to communicate: notice that in all of the problematic actions, communication is via paper. As a result, you need to cross- reference this factor to successful actions to see if, in those situations also, communication was by paper only.

Add a note hereAdd a note hereWhen you analyze a problematic matrix, the first thing you need to ascertain is not what is different, but what is common. Consider the matrix as a huge algebraic equation where your task is to solve the values of each factor. Because the problematic matrix has more variables than equations, there is no one solution. If we convert [Table A.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116725006) in a set of equations, Listing A.2 results.

Add a note hereAdd a note hereListing A.2

Add a note hereAdd a note hereJ + e1 + p + m1 = 30

L + e2 + p + m2 = 25

Add a note hereAdd a note hereAssigning values to what is common between all equations, and then assigning numbers to the rest of variables, solves the algebraic equations in Listing A.2.

Add a note hereAdd a note here[Table A.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116725006) shows the same matrix as that in [Table A.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116725006), with some values assigned.

**Add a note hereAdd a note hereTable A.2:** Example Problematic Matrix Version 2

**Larger View**

| **Add a note hereAdd a note hereProgramming** | **Add a note hereAdd a note hereTools** | **Add a note hereAdd a note hereCommunication** | **Add a note hereAdd a note hereAction** | **Add a note hereAdd a note hereDifference** |
| --- | --- | --- | --- | --- |
| Add a note hereAdd a note hereJohn (8) | Add a note hereAdd a note hereeditor1 (9) | Add a note hereAdd a note herePaper (13) | Add a note hereAdd a note hereImplement module 1 | Add a note hereAdd a note here30 |
| Add a note hereAdd a note hereLisa (6) | Add a note hereAdd a note hereeditor2 (6) | Add a note hereAdd a note herePaper (13) | Add a note hereAdd a note hereImplement module 2 | Add a note hereAdd a note here25 |

Add a note hereAdd a note hereIn this version of the matrix, we are looking for the common pieces, and we assign generic values to them. The best way to know which value to assign to the common element is to find the lowest difference value from the Difference column among all common equations and then to assign to the variable half the value. In [Table A.2](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116725006), the lowest common difference is 25, which, when divided by two is 12.5 or 13, and that is then assigned to the variables in the Communication column.

Add a note hereAdd a note hereFor the rest of the values, assign the variables by taking half of the leftover value and assigning it. For the second row, 25 minus 13 results in 12; dividing this value evenly across all factors results in 6 and 6. This is not extremely scientific, but it provides a way of comparing different factors in a non-discriminatory way. For the first row, that means the values of 8 and 9 are calculated.

Add a note hereAdd a note hereIf you had another common value between row one and two, then you would assign three-quarters of the lowest difference to each common value. [Table A.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116725006) shows a sample matrix with two common attributes for Tools and Communication.

**Add a note hereAdd a note hereTable A.3:** Example Problematic Matrix Version 3

**Larger View**

| **Add a note hereAdd a note hereProgramming** | **Add a note hereAdd a note hereTools** | **Add a note hereAdd a note hereCommunication** | **Add a note hereAdd a note hereAction** | **Add a note hereAdd a note hereDifference** |
| --- | --- | --- | --- | --- |
| Add a note hereAdd a note hereJohn (12) | Add a note hereAdd a note hereeditor1 (9) | Add a note hereAdd a note herePaper (9) | Add a note hereAdd a note hereImplement module 1 | Add a note hereAdd a note here30 |
| Add a note hereAdd a note hereLisa (7) | Add a note hereAdd a note hereeditor1(9) | Add a note hereAdd a note herePaper (9) | Add a note hereAdd a note hereImplement module 2 | Add a note hereAdd a note here25 |

Add a note hereAdd a note hereOnce the numbers have been subbed in, the objective is to figure out what went wrong. The way to figure this out is to look at the largest numbers. In the case of [Table A.3](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116725006), John has an associated number of 12. When you look at all of the numbers, this means John is more problematic than the tool he is using to edit the source code file.

Add a note hereAdd a note hereWhen you're reading problematic matrices, it is essential to realize that the numbers are a potential indicator, not the indicator. A problematic matrix makes it simpler to decide what attributes and factors to look at when actions are problematic.

### Add a note hereAdd a note hereRelation of Debugging Time to Implementation Time

Add a note hereAdd a note hereWe now need to consider debugging time and implementation time. Architects, designers, and programmers make mistakes, which the QA generally catches. These mistakes have to be fixed until they are written off as fixed. It would therefore seem that the time of implementation and time to write off the action as fixed would be important. However, the truth is that they do not really matter, since fixing a bug might not be a good measure of good coding or bad coding. For example, a bug fix could be a design issue that was not considered when the software project was designed. Therefore, fixing that bug, which worked correctly as per the specification, might make the programmer look like a bad programmer when in fact an architectural issue was to blame.

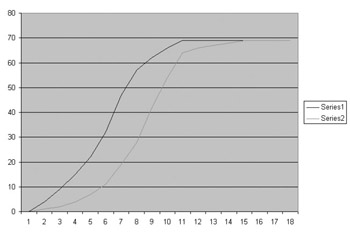
Add a note hereAdd a note hereA better measure of the overall software development process is to track the total bugs raised and total bugs fixed, as shown in [Table A.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116725006).

**Add a note hereAdd a note hereTable A.4:** Bugs raised and bugs fixed timeline

| **Add a note hereAdd a note hereTotal Bugs Raised** | **Add a note hereAdd a note hereTotal Bugs Fixed** |
| --- | --- |
| Add a note hereAdd a note here0 | Add a note hereAdd a note here0 |
| Add a note hereAdd a note here4 | Add a note hereAdd a note here1 |
| Add a note hereAdd a note here9 | Add a note hereAdd a note here2 |
| Add a note hereAdd a note here15 | Add a note hereAdd a note here4 |
| Add a note hereAdd a note here22 | Add a note hereAdd a note here7 |
| Add a note hereAdd a note here32 | Add a note hereAdd a note here11 |
| Add a note hereAdd a note here47 | Add a note hereAdd a note here19 |
| Add a note hereAdd a note here57 | Add a note hereAdd a note here28 |
| Add a note hereAdd a note here62 | Add a note hereAdd a note here42 |
| Add a note hereAdd a note here66 | Add a note hereAdd a note here54 |
| Add a note hereAdd a note here69 | Add a note hereAdd a note here64 |
| Add a note hereAdd a note here69 | Add a note hereAdd a note here66 |
| Add a note hereAdd a note here69 | Add a note hereAdd a note here67 |
| Add a note hereAdd a note here69 | Add a note hereAdd a note here68 |
| Add a note hereAdd a note here69 | Add a note hereAdd a note here69 |

Add a note hereAdd a note hereDo not put too much stock into the exact bug numbers since they are just numbers. What is important is the rows when you relate them to a time. Each row represents a unit in time and the number of bugs raised and bugs fixed. [Figure A.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116725006) shows the graphic representation of the data in [Table A.4](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116725006).

Larger View



Add a note hereAdd a note hereFigure A.1: A graphical representation of the total bugs raised and total bugs fixed statistics.

Add a note hereAdd a note hereWhat you should notice in [Figure A.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116725006) that both lines look like a slanted S shape. By default, all software processes follow this slanted S, which has three distinct phases. The first phase from time units 1 to 4 is the *ramping-up* *period*, where everybody is becoming used to the system and there are more bugs being generated than fixed. The phase from time units 4 to 9 is called *testing and fixing*. In the testing and fixing phase, bugs are generated and are being fixed. At this point, everybody is doing their job and the number of bugs fixed equals the number of bugs raised. In this phase, the total number of bugs raised still outnumber the total number of bugs fixed. This is OK and normal. The last phase, from time units 9 to 18, is called the *wind-down phase*. At this point, the number of bugs raised is not that significant, and the bugs are getting fixed. Notice also that in the testing and fixing phase, the individual graph lines have an inflection point where the number of bugs raised slows down and the number of bugs fixed increases.

Add a note hereAdd a note hereThe interesting part of the S-graph is that all software development projects follow the graph. Knowing that the S-graph works, you can predict when a piece of software is ready to ship, and you can generate a timeline in the testing phase. However, when presented with these graphs, you need to watch for certain conditions, since [Figure A.1](http://viewer.books24x7.com/assetviewer.aspx?bookid=7265&chunkid=116725006) is an idealized graphing. Following are some common problems you can detect using these graphs:

* Add a note hereAdd a note here**The inflection point of bugs raised has not been reached, or it has been reached and another inflection point of more bugs raised is started.** The problem in this case is that somebody changed something in the software that is causing the software to generate its own bugs. In other words, people are fixing the same thing or functionality over and over again.
* Add a note hereAdd a note here**In the testing and fixing phase, the distance between the bugs raised and bugs fixed decreases, but the bugs-raised graph does not reach its inflection point.** The problem in this case is that the testers are having problems properly testing the software.
* Add a note hereAdd a note here**In the testing and fixing phase, the distance between the bugs raised and bugs fixed increases, and the bugs-raised graph has passed its inflection point.** The problem in this case is a potential time bomb. What is happening is that the testers are doing their job, but the developers are slow to fix bugs. This could be because the programmers are slow or because they are struggling to implement the feature set. In either case, there could be an explosion of bugs. And, because the programmers are having problems at this point during the explosion of bugs, things might become entirely chaotic.
* Add a note hereAdd a note here**Neither the bugs fixed nor bugs raised are reaching their inflection point.** The problem in this case is that the project is running out of control. The list of bugs keeps increasing, and the programmers are fixing the bugs. But there is no sense of a light at the end of the tunnel. In this case, the problem is probably more related to the overall design. This is because the testers are seeing implementations that are probably correct but incorrectly defined.

## **Summary**

Add a note hereAdd a note hereThe purpose of this appendix is to provide a way to make the software engineering process quantifiable. The mathematical formulas make it simpler to figure out what should be implemented and what should not be implemented. It removes the gut feelings and should be used by software development leads or by the developers themselves. These formulas can be used to improve one's own estimation of what actions should be implemented, how long it takes, and how long it takes to figure out what went wrong. Knowing all of these things, the developer or software development lead can improve the quality of the next software application.

# Appendix B: **About the CD-ROM**

Add a note hereImage from book[CD Content](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=1053#1053)

## **Add a note hereAdd a note here****JSENG Source Code**

Add a note hereAdd a note hereThe software for the book *Applied Software Engineering Using Apache Jakarta Commons* is on this CD-ROM.

Add a note hereAdd a note hereEach subdirectory contains specific parts of the source code as follows:

* Add a note hereAdd a note here**axis**: Contains the Web Service Web application, including all registration information. The class files are ready to go.
* Add a note hereAdd a note here**Image from book**[**jars**](http://cdcontent.books24x7.com/id_7265/jars.zip): Contains all jar files necessary to compile and run the samples. The only missing piece is the J2EE jars, which can be downloaded from the Sun Web site [*www.javasoft.com*](http://www.javasoft.com).
* Add a note hereAdd a note hereImage from book[**jseng**](http://cdcontent.books24x7.com/id_7265/jseng.zip): Contains all of the source code shown in the book.
* Add a note hereAdd a note here**jseng/project**: Contains all the project files for the SlickEdit tool.
* Add a note hereAdd a note here**jseng/src**: Contains all of the sources for the book.
* Add a note hereAdd a note here**downloads**: Contains some downloaded applications that are used to build the samples.
* Add a note hereAdd a note here**src**: Contains the sources to the Jakarta Commons and Jakarta Commons Sandbox projects.

## **System Requirements**

* Add a note hereAdd a note hereAny operating system that supports the Java SDK environment.
* Add a note hereAdd a note hereJava SDK 1.4.x and J2EE SDK 1.3.x (necessary only for the Messenger examples in [chapter 6](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=392#392)) must be installed.
* Add a note hereAdd a note hereHard disk, CPU, and RAM are dependent on the operating system and depend on how resource-intensive Java is on the specific operating system. In addition to those requirements, about 200 MB of hard disk space is required for all of the sources.

# List of Figures

### Add a note here[Chapter 4:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=250" \l "250" \t "_parent) Scalability

* [Figure 4.1:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=274" \l "274" \t "_parent) An example of JMP windows showing method calls, memory usage, and object usage.
* [Figure 4.2:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=275#275) A saw tooth pattern that shows memory usage.
* [Figure 4.3:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=276#276) Object profiling.

### Add a note hereAdd a note here[Chapter 11:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=859" \l "859" \t "_parent) Project Management

* [Figure 11.1:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=909" \l "909" \t "_parent) A dialog box that shows how to enter the CVSROOT parameters.
* [Figure 11.2:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=912#912) The context menu you use to upload a new module to the repository.
* [Figure 11.3:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=913#913) The dialog box that is used to tweak how files are imported.
* [Figure 11.4:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=914#914) Description details for a specific file type.
* [Figure 11.5:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=915#915) Module repository details.
* [Figure 11.6:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=916#916) The Import Options tab of the dialog box Import Settings, with “Create CVS directories while importing” checked.
* [Figure 11.7:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=917#917) The appearance of the CVS control sources.
* [Figure 11.8:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=919#919) A dialog box used to check out sources.
* [Figure 11.9:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=920#920) Selecting a specific date, branch, or tag of the sources to check out.
* [Figure 11.10:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=922#922) Selecting the context menu for an individual file item.
* [Figure 11.11:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=923#923) The dialog box Update Settings.
* [Figure 11.12:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=925#925) Committing changes for a specific file or directory set.
* [Figure 11.13:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=928#928) A highlighted file that is not part of the CVS repository.
* [Figure 11.14:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=929#929) Toolbar buttons to select when you’re adding a file.
* [Figure 11.15:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=933#933) Log of changes graphically shown in wincvs.
* [Figure 11.16:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=935#935) Tagging a specific file.
* [Figure 11.17:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=936#936) The dialog box Create Tag Settings.
* [Figure 11.18:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=937#937) Change log file showing ReleaseVersion.
* [Figure 11.19:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=938#938) Branching a specific file.
* [Figure 11.20:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=941#941) Selecting a module to check out and update.
* [Figure 11.21:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=942#942) Selecting a branch to merge.
* [Figure 11.22:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=944#944) Using a conflict editor to merge two files.

### Add a note hereAdd a note here[Appendix A:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=947" \l "947" \t "_parent) Making Decisions Quantifiable

* [Figure A.1:](http://viewer.books24x7.com/assetviewer.aspx?bkid=7265&destid=978" \l "978" \t "_parent) A graphical representation of the total bugs raised and total bugs fixed statistics.

