CHAPTER 1

Introduction

I've been watching the evolution of Java technology since the very early JDK 1.0 releases. At that point, much of the excitement about Java had to do with the applet technology, and Java was seen as a serious threat to Microsoft for the desktop. Since then, many other "perfect matches" for Java technology have emerged; many have just as quickly left the scene. But while Java no longer seems likely to displace Microsoft, it has increasingly become the technology of choice for developing server-side applications. The recent emergence of Web-Service technologies has only served to put Java in even better shape on the server side, as a majority of IT managers have indicated they see Java 2 Enterprise Edition (J2EE) technologies as more ready to develop and deploy with than .NET.

Web Services will definitely play a major role in creating future applications; but I believe that role is still secondary to fundamental technologies such as Java servlets that can support both traditional, browser-oriented interfaces as well as newer, XML- and SOAP-based Web Services. For that reason, this book spends a lot of time exploring the foundation technologies that are vital to supporting Web Services, but that are themselves separate from the Web Service.

Web-Service technologies are still immature. Much of the software used to develop the example applications in this book was prerelease, early access, or otherwise not production quality. In many cases, the specifications themselves are still evolving, so the software will change to reflect the final specification. For this reason, anyone starting a Web-Service project today should be budgeting time for rework down the road. In the early chapters of this book, much emphasis is placed on layering software and making good use of object-oriented principles; this will help isolate those portions of your application that will need to be changed to reflect any updates to the underlying technologies.

1.1 Why This Book Was Written

This book came about not because of any burning desire to teach other people J2EE and Web Services, but rather because of a desire to learn more about these topics myself. To learn the new technologies, I did a lot of reading; but the most important activity was creating the sample programs. The same will be true for you; by reading through the text and examples presented herein, you'll be able to get a feel for the technologies that make up J2EE. My goal for each technology covered is to help you understand when it could be used, what the alternative technologies might be, and how to decide which technology is the best fit for the task you're faced with. This level of understanding can be achieved through reading. But once you've decided that a particular technology is something you will use, reading itself will not be sufficient; you'll need to roll up your sleeves and do some programming. I provide exercises in each chapter that can certainly help get you started, but it will be even more important to create your own programs that deal with the kinds of problems you need to solve.

My goals in writing this book are the following:

- Introduce J2EE technologies, explaining each one well enough so that you can determine which ones deserve further study, and which ones aren't a good fit for the job you are trying to do.
- Explain Web Services and show how J2EE technologies can be used to implement them. I believe that Web Services are rarely standalone, but are instead part of larger Web applications. So technologies that are commonly used in Web Services receive extra emphasis, but other Web-application technologies are covered as well, even if they are not directly involved in the delivery of Web Services.
- Provide sample programs for each technology that can be used as a template for how to use the targeted technology.
- Provide only the level of technical detail needed to grasp the capabilities and limitations of each technology. Don't try to examine every single feature or obscure usage. Apply the 80-20 rule.
- Provide exercises that help cement an understanding of each technology.
- Provide references for further study, both printed and online, for those readers who need to gain a more detailed understanding of specific technologies.
- Have fun. Make the examples interesting, and try to avoid dry, boring text.

1.2 Who Needs This Book?

You do. In fact, you probably need several copies; one for home, one for work, maybe one for the car. They make great gifts as well. Tell the guy behind the counter that there aren't nearly enough copies of this book on display, and that they should be up front instead of that Tom Clancy book.

This book won't teach you Java programming (although it provides lots of opinions about good programming practice), so it's intended for readers who already know at least a little about

programming in general and Java in particular. It covers many topics, but not in great depth; so if you're looking for the definitive book on servlets, XML, Enterprise JavaBeans (EJB), or any other topic plucked from the table of contents, look for a book focusing on just that topic. This book is more of a sampler. Each of the technologies that make up Java 2 Enterprise Edition are covered in this book, so it can certainly be used as an introduction to J2EE technology. It also introduces the main concepts and technologies used in Web Services. If you're new to both Web Services and J2EE, then everything in the book should interest you. If you know one but not the other, then you'll want to pick and choose particular topics of interest.

Once I help you understand the set of technologies that you need to know, my goal is to provide enough information so that you can be productive with that technology using just the information presented here. If you need more than the basics, I'll try to point you in the direction of more detailed material on the Web and in other books.

1.3 What's This Book About?

There are several complementary things going on in this book. On the one hand, it's an introduction to J2EE technologies, and can serve as a primer for anyone interested in learning about J2EE. The book emphasizes Web Services, so if you are already somewhat familiar with J2EE, but looking to dig more deeply into those technologies that can be used in the development of Web Services, then you should find more information about those topics here than in J2EE books with a different focus. Finally, it's a book about application architecture and design. I'm a strong believer that good software evolves; in fact, Web Services are a perfect example of this. Many companies today are faced with adding Web-Service functionality to existing applications. The degree to which they'll be able to do so easily is directly related to how much flexibility was designed into their applications initially. By using the same application over and over again within this book, deploying it as a standalone application, a servlet, a Web Service, and in other configurations, you'll see how an application can be designed to handle whatever future changes might be required.

Throughout the course of the book, I'll take a simple set of Java classes and evolve them into a number of different architectures. The purpose is twofold. Primarily, you want to gain an understanding about the various architectures that are available. This includes how difficult they are to implement, how well they perform, how well application development tools support the architecture, and similar factors. The secondary goal is to learn what features vary widely between architectures, and what things stay largely the same. By understanding this, we can segregate the architecture-dependent portions of an application so that if we choose, we can change the application architecture at some time in the future in a straightforward manner.

When I speak of changing application architecture, it doesn't mean changing the functionality provided by the application; instead, it means changing the environment in which the application runs. Architectural changes might be changes in the user interface technology used, changes in the way data is stored by the application, or changes in the way the application is distributed across multiple computers and how those distributed components are tied together.

In this book, the initial application will go through lots of changes along the way. Some of those changes will be permanent, as we discover better ways to structure the application to make it more flexible in adapting to different environments. Other changes might be specific to just one architectural model, and not carried forward into later versions; but that doesn't necessarily mean that they represent a dead end.

1.4 Why J2EE and Not Java?

Originally, I set out to cover all possible Java technologies with an emphasis on application architecture and design—which would include JavaBeans, JFC, and Applets, for example, along with the topics presented here such as servlets and Enterprise JavaBeans. The topic was simply too broad, and I could not possibly do justice to the "interesting" architectures simply because of the number of permutations that were to be covered.

Once it became clear that a narrower focus was needed, the Java 2 Enterprise Edition set of technologies became the obvious area on which to focus. The topic was then further refined to include an emphasis on J2EE Web Services. Since the intended audience for this book is programmers who have at least a little prior Java experience, chances are you already have some familiarity with Java 2 Standard Edition (J2SE) features such as JavaBeans and Applets—probably to about the same level that I could have covered in a single chapter. So I decided not to cover those items explicitly. But because some example code had already been developed, you'll probably find some details about these J2SE technologies tucked away in various places.

Java 2 Enterprise Edition is interesting because that's where the action is. While the core Java APIs are relatively (an important qualification) stable, the J2EE APIs are still evolving at a rapid pace, and there is a much greater need for instructional material.

1.5 Architectural Principles

If at every turn we make just the changes required by the architecture at hand, we'll end up reworking the same areas of code repeatedly. To minimize rework and improve the chances of code reuse, it helps to have some guiding principles as we go. As with all object-oriented programming, one important principle to apply is to separate things that change from things that stay the same. We'll also try to introduce some "layering" of the code, and separate out code that deals with areas such as data access, business logic, presentation, and remote access. While this isn't a book on patterns, we'll apply them where appropriate and mention them wherever they're used.

Another guiding principle is the idea of "refactoring"—that is, the idea of making very small incremental changes to code, not for purposes of adding new functionality or fixing errors but merely to improve the design or readability of the code. Not all of the refactoring that took place in the sample application is visible in the finished code, because in most cases you're just looking at the finished product. However, many times there is code that changes just slightly from one version to the next; in some of those cases I'm taking advantage of an opportunity to refactor something in the code along with other changes I may be making. I'll try to point out these refactorings as they happen; I've found looking for such opportunities really improves the quality of the code I produce and I'd like to try to infect you with the habit.

1.6 Organization

Here's what you'll find in this book, chapter by chapter.

1.6.1 Part I: Foundations

- *Chapter 1: Introduction*. The usual introductory fluff. Contains the sentence you are now reading, plus those that immediately precede and follow it.
- *Chapter 2: The Example Application*. Introduces the application that will be used throughout the book. The actual code listings for the application are in the Appendix, to avoid making the chapter too dense. Includes coverage of object design, JavaBeans, and the Strategy and Singleton design patterns.
- *Chapter 3: The Layered Application.* Reworks the example application to make it more suitable for evolving into the various architectural models to be covered. Separates interface from implementation. Creates an Application Layer, a Presentation Layer, and a Persistence Layer within the application. Introduces Model-View-Controller architecture. Covers the Factory pattern and an idiom for handling enumerated types.

1.6.2 Part II: Web Applications

- *Chapter 4: Introduction to Presentation Architecture.* Covers the basic types of presentation typically used: textual interfaces, graphical interfaces, Web-browser interfaces, and Web Services. Provides a brief introduction to HTML, HTTP, and XML as background for the chapters that follow.
- *Chapter 5: Servlets.* Several example servlets are created. Basic servlet APIs are introduced. The example application is altered to behave more dynamically, allowing concurrent access by multiple users. New capabilities are added to the example application. Advanced servlet topics are briefly discussed.
- *Chapter 6*: *JavaServer Pages.* JSPs are explored as a replacement for servlets. Various methods of embedding Java code in a Web page are examined, including scriptlets, expressions, and declarations. The ugliest ways of writing JSP code are demonstrated for your amusement.
- *Chapter 7*: *Integrating JavaServer Pages with JavaBeans and Servlets*. Sanity is restored as large chunks of Java code are excised from JSPs and placed in more suitable locations, including JavaBeans and servlets. Making JSPs work well with other code is the focus. Custom Tag Libraries are covered, including the Standard Tag Library and an application-specific tag library that we'll create.
- *Chapter 8*: *Struts*. The Struts framework is part of the Jakarta project from the Apache Group. The Struts framework is introduced and used to create an example showing servlets, JavaBeans, JSPs, and custom tag libraries all used in a single application. This is the culmination of the discussion of servlets and JavaServer Pages.
- *Chapter 9*: *Web Presentation via XML and XSLT.* This is the first of several chapters to cover XML topics. The Document Object Model (DOM) and JDOM are used to create

an XML document on the fly. Extensible Stylesheet Language for Transformation (XSLT) is used to convert the XML document to HTML format. The JAXP API is covered. The Apache Xalan and Xerces XML packages are used to parse and transform XML documents.

- *Chapter 10: Using XML with Wireless Clients*. Introduces WML and XHTML, the XML-based languages used for marking up content for wireless devices. Shows how a servlet could detect the client device type and send the appropriately formatted content.

1.6.3 Part III: Distributed Objects and Web Services

- *Chapter 11: Introduction to Distributed Objects and Web Services*. An introduction to technologies used for distributed object applications (RMI and JMS) and Web Services (JAXM and JAX-RPC). Discusses the various models typically used to build distributed applications. Introduces the topic of Web Service security.
- *Chapter 12: The Java Message Service*. Introduces the concept of Message-Oriented Middleware (MOM), and specifically the Java Message Service (JMS). Features of JMS are described along with several deployment scenarios. A new service is created, which sends messages to our example application triggering data updates. The Java Naming and Directory Interface (JNDI) is also introduced, since it is the mechanism by which clients of the JMS locate the JMS server.
- *Chapter 13: XML Messaging: SOAP and JAXM*. The discussion of MOM continues with a look at the SOAP standard for XML-encoded messages. A Java implementation of SOAP, the Java API for XML Messaging (JAXM), is introduced. The JMS examples from the previous chapter are reworked to use SOAP messages via JAXM. These examples provide the best prototype yet of a true Web Service.
- *Chapter 14: Parsing and Manipulating XML*. With a true Web Service now serving XML content, focus turns to XML parsing. Parsing XML messages such as a Web Service returns is demonstrated using DOM, SAX, and JDOM. The new Java API for XML Binding (JAXB), which provides conversion capabilities between XML documents and Java classes, is shown.
- *Chapter 15: Remote Method Invocation (RMI)*. The use of Java RMI to build distributed object applications is demonstrated. New server and client classes are created to work in the remote environment. The ObjectFactory paradigm is extended for creating non-local objects.
- *Chapter 16: Building a Web Service with JAX-RPC*. Just as JMS was followed by XML-based messaging, RMI is followed by XML-based procedure calls. The Java API for XML Remote Procedure Calls (JAX-RPC) is introduced. SOAP-based RPC is shown as an alternative to Java RMI, and the example programs from the previous chapter are adapted accordingly.
- *Chapter 17: Describing, Publishing, and Finding Web Services*. An introduction to Web-Service registries. The Web Services Description Language (WSDL) is

introduced and examples based on our Web Service are shown. The functions of registries and repositories are covered, and then UDDI and ebXML are shown as specific examples. Utility software for dealing with WSDL and registries is demonstrated, including HP Service Composer, HP Registry Composer, and BEA's UDDI Explorer.

- *Chapter 18: Clients for JAX-RPC Web Services*. Illustrates the three primary types of clients for JAX-RPC–based Web Services: static clients that access the service via stubs, clients that access the service via dynamic proxies, and clients that use the Dynamic Invocation Interface.

1.6.4 Part IV: Enterprise JavaBeans

- *Chapter 19: Session Beans*. An introduction to Enterprise JavaBeans. The various interfaces supported by EJBs are introduced. Stateless and Stateful session beans are introduced. An example is built using a Stateless Session Bean that replaces the servlet-based implementations used in previous examples. Various options for building and deploying EJBs are covered, including EJB-JAR files and Enterprise ARchive (EAR) files.
- *Chapter 20: Message-Driven Beans*. The new (in EJB 2.0) Message-Driven Bean capability is introduced. An example bean is developed to consume JMS messages from the example server developed in Chapter 12. Using references in the deployment descriptor to assemble an application from multiple beans is demonstrated.
- *Chapter 21: Entity Beans*. Entity Beans are examined as a way of providing persistence for application data. Bean-Managed and Container-Managed Persistence are introduced, and an EJB using Container-Manager Persistence is created. The EJB Query Language (EJB-QL) is used to implement a finder method for the bean.
- *Chapter 22: A Look Back, a Look Ahead*. Includes a brief recap of some of the themes that have emerged through the course of the book. Then, a look at how the standards and APIs are evolving, and what is known and expected for upcoming releases of J2EE and JAX APIs.

1.7 Software Used

Java 2 Standard Edition version 1.3.1 was the basic JDK used to produce all of the example code in this book. On top of this, various portions of J2EE were used. Most of the examples do not require the use of a full-blown application server, so frequently a J2SE SDK and a servlet engine such as Apache Tomcat are all that is needed.

For the J2EE components, J2EE 1.3 was the basis for all examples. Primarily, the servlet 2.3 and JSP 1.2 specifications were used as the foundation for many of the examples in the book. Table 1.1 summarizes the various components included in the J2EE specification for the latest releases.

In the table, I've also included some early information about the forthcoming J2EE 1.4 release. The specification is currently scheduled for final release in the first quarter of 2003;

actual implementations will follow. The specification is not yet in public draft form, and I wasn't able to find any information about some of the components. However, support for Web Services is the major focus of J2EE 1.4, and as it becomes available it will provide the best foundation on which to develop and deploy Web Services. Details about some of the new components are found in various places throughout this book; refer to the index if you just can't wait.

Table 1.1 J2EE Versions and Component Technology Versions

Component	J2EE 1.2 Version	J2EE 1.3 Version	J2EE 1.4 Version
Enterprise JavaBeans	1.1	2.0	2.1
JavaServer Pages	1.1	1.2	1.3
Servlets	2.2	2.3	2.4
JDBC Standard Extension	2.0	2.0	no change?
Java Transaction API	1.0	1.0	no change?
JavaMail	1.1	1.2	no change?
Java Message Service	optional	required	no change?
JAXP	not included	1.1	1.2
J2EE Connector	not included	1.0	1.5
RMI/IIOP	included	Part of J2SE 1.3	Part of J2SE
JNDI	included	Part of J2SE 1.3	Part of J2SE
JAXR	not included	not included	1.0
JAX-RPC	not included	not included	1.0
JAXM	not included	not included	optional
JAXB	not included	not included	1.0?

Examples were tested on various platforms and operating systems, including HP 9000s (HP-UX 11.0 and later operating systems), HP e3000s (MPE/iX 6.0 and later operating systems), HP Vectras (Windows NT 4.0 and Windows 2000), and HP Pavilions (Windows ME).

1.8 Using This Book

In this section, I suggest how, in my opinion, you'll get the most out of this book.

1.8.1 Navigating the Material

I've tried to introduce topics in a logical order, but also to accommodate those readers who will want to skip directly to topics that are of interest. While I'd recommend taking everything in the order presented, you should be able to jump to any chapter, or any major topic within a chapter, and find what you need in that fashion. Within individual topics, you'll find that reading from top to bottom and left to right is absolutely essential for comprehension.

I've tried to make liberal use of tables and boxed material, especially near the beginning of each chapter to call out summary information about the material covered in each chapter.

1.8.2 Exercises

Most chapters have exercises provided at the end of the material. These exercises range from fairly simple to cruelly complicated. The exercises are intended not so much as a review of previous material, but rather as a jumping-off point to explore the material further. In many cases, you'll find that if you attempt the exercises, the material provided in the chapter won't be sufficient. The Further Reading links that are provided conveniently nearby should provide any additional information you need for completing the exercises.

1.8.3 Design Centers

One of the occasional features that will pop up in various chapters is a boxed item called a Design Center. In the Design Centers, I try to summarize the decision criteria that come into play in deciding between two or more technologies that are somewhat similar, where it might be unclear which one is better-suited for a particular task. Some of the Design Centers you'll find include when to use XML Attributes versus XML Elements; when to use DOM, SAX, and JDOM to parse XML documents; and choosing between SOAP Messaging and SOAP RPC models for designing Web Services.

1.8.4 API Reference Material

This book is intended to be instructional in nature. Many of the topics covered here in a single chapter could easily be expanded to full book length. Because of the amount of material covered, it's impossible to provide exhaustive reference material for every API and product that is touched upon.

I've tried to strike an appropriate balance between instructional content and reference material. For the most part, I favored the instructional material, but I also tried to pay close attention to how I do development work. If you're going to always be sitting in front of your computer, then it is simple enough to just open a window to the javadoc API guide for whatever packages you're using. But since I tend to write a lot of code sitting at the kitchen table or

sprawled on the family room floor, I like to have at least enough reference documentation close at hand to let me scope out the basic flow of each method I develop. I can always dig deeper into the exact details of an API when I'm actually entering the code back at the computer.

If this isn't your working style, then this is of little interest to you. But if you do something similar, I've tried to selectively include API documentation for those classes in each package that you're most likely to need. In servlets, for example, I include full API documentation for the `HttpServlet`, `HttpServletRequest`, and `HttpServletResponse` classes. I don't include the various superclasses they extend, or the details of the exception classes, or the stream classes that they utilize, since these are all pretty much what you'd expect them to be.

I've also placed the API documentation inline in the chapters where they are discussed, rather than collecting them separately into an appendix. I feel that this is where you're most likely to want them, and the intent is to optimize the experience you're having as you read the material sequentially. For coming back later to refer to something, you're either going to be more likely to grab another book that is more reference-oriented, or be willing to make the extra stop by the index on the way to your destination.

1.9 On the Web

Errata or clarifications for this book will be posted on the Web at my site, `http://www.theYawns.com`. There may also be supplemental material or drafts of material for future publication. If you have feedback, comments, or have found an error, please drop me a note at `j2eejax-feedback@theYawns.com`.