Beginning JBoss® Seam

From Novice to Professional

Joseph Faisal Nusairat

Apress®

Beginning JBoss® Seam: From Novice to Professional

ISBN-13 (pbk): 978-1-59059-792-7

ISBN-10 (pbk): 1-59059-792-3

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Steve Anglin
Technical Reviewer: Floyd Carver
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade
Project Manager: Denise Santoro Lincoln
Copy Edit Manager: Nicole Flores
Copy Editor: Sharon Wilkey
Assistant Production Director: Kari Brooks-Copony
Production Editor: Lori Bring
Compositor: Patrick Cunningham
Proofreader: Dan Shaw
Indexer: John Collin
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, email `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code/Download section as well as at `http://www.integrallis.com`.

To the memory of my grandparents, Kasim Nusair and Kurdeih Rashdan;
To my grandparents, Henry Albert Baker and Mary Baker;
To my parents, Janette Darr and AJ Nusairat;
And to all my friends and family who supported me throughout the years.

Contents at a Glance

Contents

About the Author

JOSEPH FAISAL NUSAIRAT is a software developer who has been working full-time in the Columbus, Ohio, area since 1998, primarily focused on Java development. His career has taken him into a variety of Fortune 500 industries, including military applications, data centers, banking, Internet security, pharmaceuticals, and insurance. Throughout this experience, he has worked on all varieties of application development—from design to architecture to development. Joseph, like most Java developers, is particularly fond of open source projects and tries to use as much open source software as possible when working with clients.

Joseph is a graduate of Ohio University with dual degrees in Computer Science and Microbiology and a minor in Chemistry. While at Ohio University, Joseph also dabbled in student politics and was a research assistant in the virology labs.

Currently, Joseph works as a senior partner at Integrallis Software (`http://www.integrallis.com`). In his off-hours he enjoys watching bodybuilding and Broadway musicals, specifically anything with Lauren Molina in them.

About the Technical Reviewer

FLOYD CARVER has been building software systems for 20 years. During this time, he has performed in many roles, from developer to architect and from student to instructor. He is currently providing consultant services as an applications architect. When not consulting, Floyd enjoys traveling, playing and coaching soccer, and coaching basketball.

Acknowledgments

As this is my first book, there are so many people to thank for helping me put this together. The order of my thanks in no way signifies importance; everyone here and even more helped in some way to get this off the ground. I would like to first thank the publisher, Apress, without whom there would be no book. Thank you to Steve Anglin for giving a starting author the opportunity to write. In addition, I would like to thank my copy editor, Sharon Wilkey, who helped me quite a bit with my writing. Denise Santoro Lincoln, my project manager, for continuously pushing me to try to meet my deadlines. Finally, my copy edit manager, Nicole Flores, and many other staff members of Apress for all the work they put into publishing the book.

And thank you to my technical reviewer Floyd Carver, who, when I asked if he would be my tech reviewer, said yes without thinking twice. I appreciate that, considering the amount of work he had in store—thanks for all your time spent. Also I would like to thank Brian Sam-Bodden, my business partner, for his mentoring and for pushing me to start writing. I would also like to thank Chris Judd for not only reviewing some of my chapters but also giving me good advice on writing early on (I actually followed some of it). Also Marie Wong, who not only helped keep me sane during this process, but also helped convert my drawings to meaningful diagrams.

Because this is a book on JBoss, I would be remiss not to thank Gavin King, JBoss, and the contributors to Seam for creating the Seam framework. Also I would like to thank all those who contributed to the Seam Forum on the JBoss site. I was a regular viewer, and I even tried to take note of items that people seemed to have trouble with in order to make this a better book.

Writing a book was one of the most pleasurable experiences I have had—well, pleasurable and stressful all in the same breath. Everyone along my school and career path helped me get where I am today. The Ohio University computer science department, and in particular Dr. Shawn Ostermann, helped inspire me not only to stick with computer science but to want to learn even more—something that I hope is with me today. In addition, Scott Carter of TSYS in Columbus, Georgia, was my first mentor as I was learning Java, and he definitely helped push me on the right path for clean, functional development practices.

Finally, my siblings, Sarah, Michael, Robert, Angela, Adam, and Ashley—you are all great, and I hope you all enjoy the book.

I am sure I left out someone, so just a general thanks to all those that helped.

And finally, you the reader, for picking this book to read out of all the Java books out there. I appreciate it and hope you come away with a better understanding of JBoss Seam.

Introduction

Agile, agile, agile, Ruby, Ruby, Ruby. It seems like every conference you go to these days talks about either agile or Ruby. Those are the big buzzwords in the industry. Everywhere you go, that's all you seem to hear. And as my friend Rob Stevenson says, that's all he wants to do. In fact, the only books he reads now are Ruby books. The real question is, why? Personally I think it's because he likes a limited selection of books. But the other reason is, Ruby is fun. It's fast, it's cool, it's new, and it makes development a pleasure. And computer-savvy developers seem to love anything new. I honestly get a bit tired of *everything* coming out calling itself agile. Its such a key word these days that I am just waiting for recruiters and sales managers of consulting companies to start telling their clients they need agile developers.

The real question has to be, what is meant by *agile*? What is needed to make something agile? Agile development keeps the ease of development while still making the code clean. And I think that's what every user is *really* looking for. It's why Ruby is popular, and it's the attraction to Trails. There is so much work going into plumbing these days that it's almost overwhelming. Every team seems to want to reinvent the wheel. Larger companies have extended frameworks such as Apache Struts and are using it for what they think are specific needs. Sometimes this is useful; other times all they have done is added a layer of confusion.

In today's business world, companies are trying to minimize cost and time while maximizing product. This often results in many shortcuts and can result in code that is even more difficult to maintain. This is where agile development comes into play. This is also where JBoss Seam comes into play. We as developers need to develop the business logic and the presentation tier as fast as possible. With agile development, this becomes possible.

I like refer to Seam as an enterprise agile framework, which to some people may seem like an oxymoron because *agile* precludes you to think something is small and easy, whereas *enterprise* often brings to mind bountiful amounts of code. However, I am hoping that is exactly what your experience will be while reading this book and using Seam.

Throughout this book, you will examine the concepts of web development, the parts of Seam, and various examples using Seam. By the end, you should have an appreciation that although Seam is complex behind the scenes, to the developer it can be fairly smooth. And although it may not have the kinks out of its armor yet, it is definitely proceeding down a path that is good for the Java community.

Items Covered in This Book

In this book, you will first learn some of the basics of web application design. You'll then learn about the JSF and EJB3 components. After that, the book will progressively move to more-advanced and interesting topics related to Seam. The following list outlines the contents of each chapter:

Chapter 1: What Is JBoss Seam?

This introductory chapter briefly explains Seam and provides an introduction to the Model View Controller (MVC) framework, Java 5, and JBoss 4. Both Java 5 and JBoss 4 are needed to run most of the applications in the book, and Java 5 is a must for Seam. If you know both of them, you can skip ahead.

Chapter 2: Web Applications

This chapter starts by covering the basics of web application design. We will step through basic design patterns when creating the presentation tier and compare and contrast them between Struts and Seam. The idea is to start the process of thinking how Seam will save you time as compared to traditional web application development. The end of the chapter presents the two basic samples we will use as the example applications throughout the book.

Chapter 3: JSF Fundamentals

Seam requires the use of JavaServer Faces (JSF) for its presentation tier component. Although you do not need the most advanced JSF knowledge to use Seam, you still need a basic understanding. This chapter provides the basic knowledge and architecture of JSF, while limiting discussion of certain topics, such as backing beans, because they do not have high reuse when using Seam.

Chapter 4: EJB3 Fundamentals

Seam requires Enterprise JavaBeans 3 (EJB3) for its business logic and persistence tiers. Although you could get away with having a limited or beginner's understanding of JSF to use Seam, an intermediate knowledge of EJB3 is more desirable. Because this is where the bulk of the coding takes place, this chapter introduces you to the three major facets of EJB3: the stateful session bean (SFSB), stateless session bean (SLSB), and entity bean (EB). I also go over the message-driven bean (MDB), but to a lesser extent. This chapter focuses more on the needs of the EB because those are radically different from the EJB 2.1 specification.

Chapter 5: Introduction to Seam

This is the first official chapter introducing you to Seam. The previous chapters presented background information required for beginners. In this chapter, you will learn how to write a basic Seam application. You will also learn the fundamentals of the Seam architecture. Near the end, you will learn about additional beginner components of Seam. By the end of this chapter, you will be able to write more-complex Seam applications.

Chapter 6: Seam Contexts

With basic Seam knowledge in hand, you will learn in this chapter more-advanced Seam topics, namely contexts. Contexts in Seam are essentially the same as they are in servlets. However, there are more of them and they have more functionality. This chapter discusses the Stateless, Event, Page, Conversation, Session, and Application contexts.

Chapter 7: Business Process in Seam

This chapter focuses on using JBoss Business Process Management (jBPM) with Seam. jBPM is JBoss's business process management system, which usually requires custom code to interact with. However, there is a Seam context specifically for Business Process components. This chapter covers the basics of jBPM and how to use it with Seam.

Chapter 8: Advanced Topics

By this point, all of the basics on using Seam and its various contexts have been covered. This chapter covers more-advanced topics, from internationalization and themes to Drools support. Although these topics may not be extremely difficult, they are necessary topics for users who want to make the most out of Seam.

Chapter 9: Advanced Configurations

Earlier I alluded to how you do not have to use EJB3 with Seam. This chapter starts by showing you how you can use EJB3 outside the application server. We will then go on to using Seam without EJB3 at all, just by using JavaBeans for our business logic and Hibernate for our persistence tier. This chapter will be especially helpful if your ability to deploy to a full application server is not quite there yet.

Chapter 10: Seam Tools

This chapter introduces you to free tools available to help create Seam applications. These are a mix of Seam-specific and non-Seam-specific tools that help make enterprise

development easier. This chapter also covers how to perform testing with Seam, specifically with TestNG.

Who This Book Is For

This book is a beginner's guide to Seam. However, the book also provides details on the components used by Seam such as JSF and EJB3. Although having a Java EE client/server developer background is not an absolute must, without it the benefit of using Seam may not be 100 percent clear, because most of its functionality deals with overcoming problems developers have had in the past. That being said, at the minimum, you should have the following:

- A beginner's understanding of Java (at least Java 1.2 and preferably Java 1.4)
- An understanding of basic web application development

Downloading and Running the Source Code

I have tried to include as much of the source code as I can in this book. The source code is also available from the Source Code/Download area of the Apress website (`http://www.apress.com`) and from my Integrallis website (`http://www.integrallis.com`). From the Integrallis site, click Publications and then select Beginning JBoss Seam. From either site, you can download a zip file that includes the following:

- Source code
- Dependent library JAR files
- Apache Ant build scripts
- Database build scripts (when applicable)

You can also find any notes or updates about the book on these websites.

Contacting the Author

If you have any questions or comments about this book, you can contact me via email at `jnusairat@integrallis.com`.

CHAPTER 1

What Is JBoss Seam?

seam *(sēm) n.*

A line of junction formed by sewing together two pieces of material along their margins.

A similar line, ridge, or groove made by fitting, joining, or lapping together two sections along their edges.

The preceding definition[1] of *seam* is usually used when discussing sewing. However, this definition also fits the latest in frameworks from JBoss—JBoss Seam. *JBoss Seam* is a framework that brings together existing Java Platform, Enterprise Edition (Java EE) standards to enable them to work as an integrated solution. At its core, the Seam framework ties the Enterprise JavaBeans 3 (EJB3) and JavaServer Faces (JSF) specifications. However, Seam does not just stop there—it will also join together other component models that you may be used to, such as jBPM, Drools, and more that we will get into as the book progresses.

When I mentioned *EJB*, you may have been taken aback with fear or may have started shaking your head profusely. Although EJB 2.1 had some negative connotations, especially regarding the way it relates to stateful session beans (SFSBs) and entity beans (EBs), any negative feelings about it should be reexamined today. With the new EJB3 specification, EJBs have become lightweight plain old Java objects (POJOs) that do not require as much of the "plumbing" as before. Hopefully those of you who may harbor negative feelings toward EJB3 will review and revise those feelings as you see how Seam enables you to not only cut development time but also more clearly separate your business logic and presentation tiers by cutting out the connecting code (for example, Struts actions) normally associated with web frameworks.

For years developers realized that the JavaServer Pages (JSP)/servlets paradigm was not enough to create enterprise-level web pages. That model provided the capability for a web tier that could pass objects from the client to the server, but essentially that was it. For most developers, this simple paradigm was not enough; more-complex operations were needed, and developers found themselves writing infrastructure code to deal with

1. http://www.thefreedictionary.com/seam

the shortcomings of the Servlet specification. Eventually, all the ideas learned from creating custom infrastructure code resulted in the web frameworks we know today, such as Apache's Struts and Tapestry, OpenSymphony's WebWork, and so forth. The Java community also got together and through the Java Community Process (JCP) created the JSF specification to tackle some of the issues raised and deal with the shortcomings of the Servlet specification.

Even though we now have web and business tiers with improved functionality, we have still been forced to create the plumbing code needed to connect them together. With Seam, however, these two areas can now focus more exclusively on what they do best—presentation and business logic.

What Does Seam Buy You?

When picking up this book, one of your first questions should be, "Why do I even need Seam if I have EJB3 and JSF already?" After all, they were designed to simplify the process of creating web applications. What is the benefit of using the Seam framework with these components if they have already taken care of much of the plumbing that was required before?

The answer lies in what you are trying to accomplish. To be useful, your application has to be a multitiered application that uses specific components for the presentation, business, and persistence tiers. Before, you may have accomplished this with a combination of Struts (presentation), Interface21's Spring (business), and JBoss's Hibernate (persistence) frameworks. Now, following the Java EE specification, you will be using JSF (presentation), EJB3-SB (business), and EJB3-EB (persistence). As before, each of these components requires lots of glue to link them together to talk to each other. Throughout this book I will show you how Seam has transformed this messy gluing you had to do before into a now seamless process.

Three-Tier Architecture

Any beginner may ask not only, "Why do we need all this?" but also, "Where does Seam fit into the equation?" Since about 1999, standard development practice in Java EE was to divide your application into three distinct tiers: presentation, business, and persistent tiers, as illustrated in Figure 1-1. (Java EE was then known as J2EE. It became Java EE after version 1.4 of the Enterprise Edition).

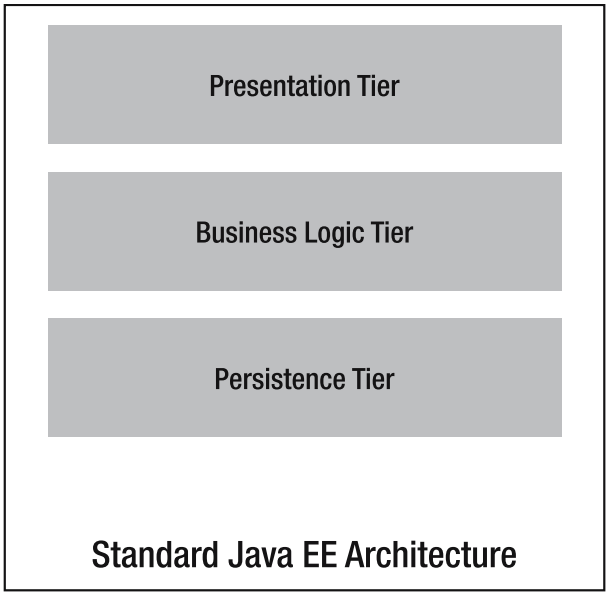

Figure 1-1. *The three-tier Java EE architecture*

These tiers are defined as follows:

Presentation tier: The presentation tier encompasses everything from the Hypertext Markup Language (HTML) page to objects controlling the request, such as Struts `Action` classes or JSF backing beans. Everything in this tier deals with communication to the client—be it a web page or BlackBerry.

Business logic tier: The business logic tier is where you make your business decisions; it contains the logic of the application. Also, this is where the business processing and (if needed) the database transactions occur.

Persistence tier: The persistence tier represents the interaction with the database. This is where you maintain data access objects (DAOs), Hibernate DAOs, or your entity beans. These classes can be either database specific or nonspecific, depending on what you need. This tier may also contain your database domain objects.

Three-Tier Architecture with Seam

Now you will take a look at the same architecture when using Seam. Figure 1-2 shows the injection points of Seam into the tiers; notice that it encompasses every tier.

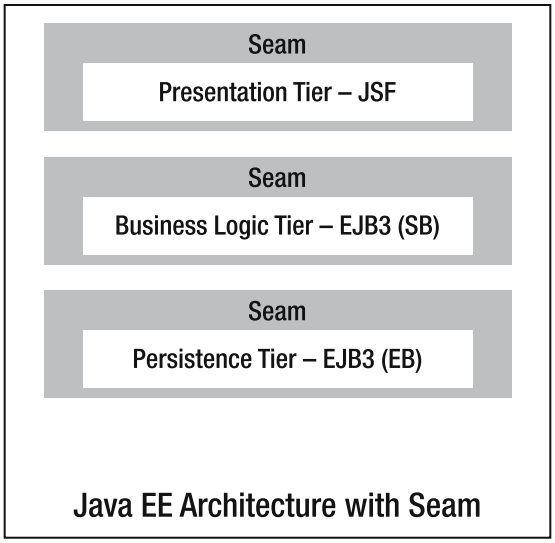

Figure 1-2. *The three-tier Java EE architecture with Seam*

In Figure 1-2, you can see that Seam wraps the components by using annotations to declare the POJOs to be wrapped. Then Seam identifies those annotated POJOs and binds the components across the tiers by using interceptors. This allows a smoother transition between tiers. Usually the binding and connecting between the tiers is much rougher in Java EE architecture without Seam. This allows you to then focus your development on the business, presentation, and persistence tiers without worrying about making them interact. Chapter 5 covers in more depth the interception of the Seam components as well as their life cycle.

Component Choices

As you know, and as I pointed out earlier, there are many different components that can work for the various tiers. So why should you use JSF + EJB3 + Seam? The simple answer is that the first two are the "new standard." However, there are additional, more-solid answers than that as well. In this section and in the following few chapters you will more closely examine the usefulness and clean implementations (due to being POJOs) of these components.

Why JSF?

JSF is now a Java standard from the JCP for presentation tier development. Now, just because something is a standard does not necessarily mean it is the best choice, but that you should factor it into the equation. JSF being part of the JCP will guarantee that there

will always be support from an army of developers with skill sets specific to its implementation. Along with this support, there are also a wealth of built-in tools from vendors and third-party programs for creating robust JSF pages. Plug-ins are being created, and even more are planned to help make JSF pages as easy as possible to create by having drag-and-drop functionality for the various integrated development environments (IDEs). There are also multiple JSF implementations to choose from. Apache's MyFaces is what JBoss uses for its JSF implementation and is what we will be using throughout the book.

Another positive with JSF is its ability to handle multiple presentation tier types. Yes, many other frameworks can be hacked to do that as well, but this is out-of-the-box support. This is achieved through JSF's presentation tier component-based architecture. Although the JSF pages still use tag libraries for the display, the component orientation of the JSF components makes them more than just simple user interface (UI) pieces. The libraries are naturally then more involved with each other.

Another advantage to JSF is that it simplifies the development of code that is called from the client page. In JSF, these listeners are referred to as *backing beans*. As I said earlier, JSF is partially independent from the servlet context, so the backing beans used can be regular POJOs with no knowledge of the Servlet specification. I say *partially* because they can have access to the Servlet specification if it is directly declared.

Finally, JSF continues to make inroads into the community and with Struts. Many of the former Struts developers used their knowledge of what they did right and wrong in the past to help create the JSF specification.

Why EJB3?

EJB3 provides just about everything you could want for business logic and presentation tiers. You get enterprise-level state-managed classes, Java Message Service (JMS) listeners, and database-level objects. Now, most of you may think, "Well, so did EJB 2.1; why do I need EJB3?" Well, this is all provided without the need for Extensible Markup Language (XML), and all your classes are still POJOs. You no longer have the overhead of multiple interfaces for creation and remotes, no more convoluted deployment descriptors, and no more stub creations. Also, like JSF, this is an industry standard, so you also get the regular cadre of developers working on it as well as different implementations.

Why Seam?

The simple answer to why we are using Seam is to create simplicity from complexity. Regardless of how easy it is to implement JSF and EJB3, they still require you to create backing beans to allow your presentation tier to talk to your business logic tier. With JSF, you have to write extensive XML files telling the presentation tier what kind of domain objects to expect. By using Seam, you can remove extra coding and focus more on the parts that matter—the presentation and business logic.

Seam does this not by creating an extra class such as a JSF action listener, but by using Java annotations. Seam allows you to create a seamless transition between application tiers by utilizing metadata in the form of Java annotations to define how the presentation and business logic tiers can talk to each other. In addition, Seam also allows you to define EB domain objects to be used on the presentation tier. The question then becomes, "How are we just removing classes?" or "Were these classes really needed before?" After working with many web frameworks on many projects, I have noticed that often what your action listeners are doing is just translating presentation tier data to some business logic tier object, validating that data, and then sending it over to the business logic tier for processing. If they are doing more than that, you are often mixing business logic into your presentation tier. Seam helps you skip this step.

Of course, there are times where the presentation tier needs to do a bit more, so, if you wish, you can also use regular JavaBeans as your annotated objects as well. If you do this, note that the only reason to do so is because you need to prep the data before sending it over to the business logic tier. Based on what I have said thus far, Seam is not adding any features but just taking a few steps away—which, by itself, still makes Seam a valuable tool. However, it also does add some valuable context-management abilities, which allows you to have features such as wizards on your page without having to add a bunch of extra plumbing code. We will get into the details of this in later chapters (Chapters 5, 6, and 7). Hopefully, however, you can already start to see the value of not only Seam, but the combination of Seam, EJB3, and JSF.

Seam Environment Requirements

Because Seam is dependent on JSF and EJB3, it has a few more needs than frameworks such as Spring or Struts. Specifically, Seam requires the following:

- You have to use Java 5 or greater on the front end, because Seam relies heavily on annotations provided by Java 5 or higher implementations of Java Specification Request (JSR) 250.

- You have to use a JSF implementation for the presentation tier. This means either using an application server that supports it or using an open source provider of JSF such as Apache's MyFaces.

- For the most part, you need to have an application server that uses EJB3. Chapter 9 discusses alternatives to using EJB3. However, it is best to assume that you are either going to be able to use EJB3 now or in the near future.

Hello World Example

So before you dive into the intricate details of Seam and the functionality around it, let's take a quick look at building a simple Seam application. As I have said, it provides the glue between the presentation tier and the business logic tier, specifically EJB3. However, in reality, your business logic tier does not have to be EJBs—it can just be plain JavaBeans. So let's look at a Hello World application done with Seam. In this example, a simple JSF page will call the JavaBean to get the text of `outputText` that has been initialized. We will start with the Seam-annotated JavaBean in Listing 1-1.

Listing 1-1. *Our JavaBean, HelloWorldAction.java, with Seam Annotations*

```
package com.petradesigns.helloworld;

import org.jboss.seam.annotations.Create;
import org.jboss.seam.annotations.Name;

@Name("helloWorld")
public class HelloWorldAction implements IHelloWorld {

    private String outputText;

    @Create
    public void init() {
        outputText = "Hello World";
    }

    public String getOutputText() {
        return outputText;
    }
}
```

As you can see, this is a fairly straightforward Hello World JavaBean. The first `@Name` annotation defines for us that this is a Seam component with the name of `helloWorld`. You will then notice that there are two methods. The `init()` method has an `@Create` annotation signaling that this method should be called upon instantiation of the JavaBean. The public method `getOutputText()` can be used to retrieve the output text.

Now that we have the JavaBean, we will create the JSF page needed to retrieve it. The page is defined in Listing 1-2.

Listing 1-2. *Our JSF Page, helloWorld.jsp, to Display "Hello World"*

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<f:view>
    <h:outputText value="#{helloWorld.outputText}"/>
</f:view>
```

This is a JSP page; more specifically, it is a JSF JSP page. The first part contains the standard tag library definitions, and the second part defines the `<f:view>` component. This is a standard component to encapsulate any JSF functionality and is required.

Finally, our `#{helloWorld.outputText}` references the method `outputText` on the Seam component `helloWorld` that we defined previously. When this page is run, it gives us the output shown in Figure 1-3.

Figure 1-3. *The Hello World screen shot*

This is all the code it takes to create a Hello World example by using JSF and Seam. This small application consists of a presentation and business tier, even though our business tier is not really doing much.

■ **Note** If we wanted to make the JavaBean an EJB3 object, all we would have to do is add the `@Stateless` annotation to the class description of the JavaBean.

One of the first things you should notice is our JavaBean representing the business logic. Not only can it be a POJO, but there is nothing in it making it interact exclusively with the presentation tier; nor is there anything on the presentation side telling it to interact exclusively with a Seam component. Seam is handling all that messy middle work usually needed to pipe between the tiers. The end result: you are able to put the bulk of your effort in building a solid business logic tier as opposed to making those objects available to the presentation tier.

I strongly advise you to use my build script included in the source to create this code because there are some intricacies on how to create JSF + EJB3 + Seam that may not be entirely obvious to you yet. These intricacies are explained more clearly in Chapter 5. For now, this can give you a bit to play with.

Introduction to MVC Architecture

If you are familiar with web application design in Java, the MVC pattern is probably fairly well known to you. However, if this is your first time writing a web application, you should follow along closely. This is essentially the pattern that has been followed for designing most web frameworks.

This section presents the MVC architecture and gives examples of various implementations of it, including JSF. For those who are new to web applications, this should help provide some background into how it works.

Basics of MVC Architecture

If this is your first attempt at web development, the concept of Model-View-Controller (MVC) may be new to you. It is the basis for most web application development today. The core principle is to separate your business data from your view components, and then have the ability to translate the data back and forth, as illustrated in Figure 1-4.

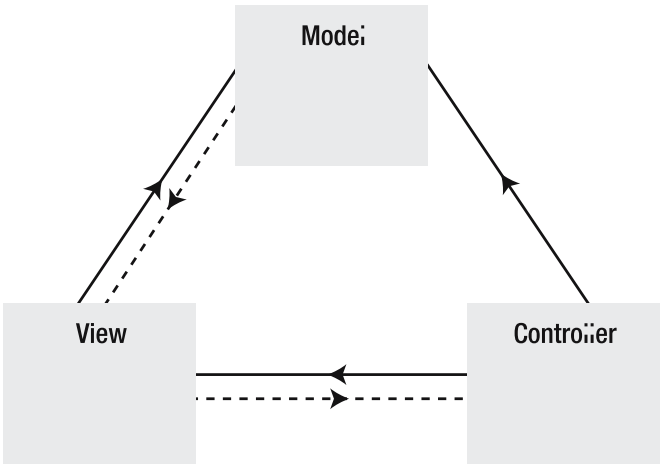

Figure 1-4. *A diagram of the MVC architecture*

The following are the definitions of the MVC parts:

Model: This represents data in the business logic tier and the business rules defining interactions with that data.

View: The view renders the contents of the model for the presentation tier.

Controller: The controller translates the model into the view. The translation will be performed by `POST/GET` requests from the Web.

Frameworks

A variety of web frameworks have come out over the last few years to implement MVC. For this book we will be using MyFaces, an open source implementation of the JSF specification. However, it would be handy to learn about a few of the others as well.

Fundamentally, all web frameworks work the same way. They all, in some way, extend the Servlet specification. How they usually differ is in how much you are aware that they are extending this specification. When using Struts 1.*x* and Spring MVC, you are fully aware that your classes are part of the Servlet specification. There are servlet-specific classes such as `HttpServletRequest` and `HttpServletResponse` given to you. When using frameworks such as Struts 2 and JSF, you are not as aware. In general, the servlet-specific objects are fairly well hidden and you have to explicitly ask for items such as `HttpServletRequest` and `HttpServletResponse`. Finally, with frameworks such as Tapestry, you are not really aware of the specification at all except through configuration files.

Next you will take a look at each of these frameworks. This small preview gives you an idea of what is out there and has been tried and will hopefully give you insight about why you do not have to worry about the nitty-gritty of these when using Seam.

Struts

Struts, one of the first web frameworks, has definitely proved to be the most popular one out there. It works by having a front controller servlet call action classes. The data from the screen is passed into the classes via an `ActionForm`. With Struts 1.*x* versions, the user was fully aware that they were working in a presentation tier—and the classes were concrete, not POJOs. Much of this has been redressed with Struts 2; however, Struts 2 does not have nearly the success and is not used as much as Struts 1.*x*.

Spring MVC

Spring MVC is a more recent entrant to the MVC game. Spring MVC is distributed with the regular Spring package and can perform business logic as well for a complete Java EE framework. However, Spring MVC is not as new as the rest of Spring. It does provide various ways of interacting with the presentation tier, and if you are looking for a low-key approach, this can work well. However, Spring MVC does not seem to have the robustness of the other web frameworks such as JSF, Struts, and Tapestry.

Tapestry

Tapestry is another web framework that has been around for a long time. The major recent upgrade of Tapestry 4 is powerful. One of its biggest advantages is that, contrary to JSF and most other web frameworks, your Tapestry classes have no knowledge that they are part of a web state because your controller classes (pages in Tapestry) have no knowledge of the

request or the session. However, unlike Seam, these pages are not POJOs but they implement the `IPage` interface. Another advantage of Seam is the use of Object-Graph Navigation Language (OGNL) for its presentation tier. This allows HTML and Java developers to create HTML pages that can be translated to dynamic code when run in the application server, but if viewed with a standard web browser outside the application server, the pages will still be viewable. This is not normal behavior for most web application servers.

A Seam-Tapestry combination would be great. Unfortunately, there is no plan to build one at this time.

JSF

Java Server Faces (JSF) is a JSR standard for presentation tier development. Like most other frameworks, it provides a separation between the presentation tier and the action listeners. Unlike most other web frameworks, the actual presentation tier can vary—it does not have to be a JSP-type page because JSF uses a component model to display the presentation tier. Your presentation tier can therefore switch more easily between a web page, XML page, or Java Micro Edition (ME) page. Also, because this is a standard, there are a variety of implementations of vendor support for it.

Java 5

Java 5, also known as Java 1.5.0, is the latest version (at the time of this writing) released from Sun Microsystems—which many thought was a long time coming. Java 5 adds items such as annotations and generics, which many developers had wanted for years. This latest release adds 15 JSRs and about 100 other major updates. This impressive level of enhancement and change reflected such a maturity that the external release was given a Java 5 designation. Table 1-1 lists the JSR enhancements for the release of Java 5; unfortunately, I cannot list all of them within the constraints of this book. However, a few of the enhancements are good to know, and others are important in order to understand how to use EJB3 and Seam. If you want to read a complete list of features, I suggest consulting the Sun site, `http://java.sun.com/j2se/1.5.0/docs/relnotes/features.html`.

Table 1-1. *JSRs for the Java 5 Release*

JSR Number	JSR Name	URL of Specification
003	Java Management Extensions (JMX) Specification	`http://jcp.org/en/jsr/detail?id=3`
013	Decimal Arithmetic Enhancement	`http://jcp.org/en/jsr/detail?id=13`
014	Add Generic Types to the Java Programming Language	`http://jcp.org/en/jsr/detail?id=14`

Continued

Table 1-1. *Continued*

JSR Number	JSR Name	URL of Specification
028	Java SASL Specification	http://jcp.org/en/jsr/detail?id=28
114	JDBC Rowset Implementations	http://jcp.org/en/jsr/detail?id=114
133	Java Memory Model and Thread Specification Revision	http://jcp.org/en/jsr/detail?id=133
160	Java Management Extensions (JMX) Remote API 1.0	http://jcp.org/en/jsr/detail?id=160
163	Java Platform Profiling Architecture	http://jcp.org/en/jsr/detail?id=163
166	Concurrency Utilities	http://jcp.org/en/jsr/detail?id=166
174	Monitoring and Management Specification for the Java Virtual Machine	http://jcp.org/en/jsr/detail?id=174
175	A Metadata Facility for the Java Programming Language	http://jcp.org/en/jsr/detail?id=175
200	Network Transfer Format for Java Archives	http://jcp.org/en/jsr/detail?id=200
201	Extending the Java Programming Language with Enumerations, Autoboxing, Enhanced `for` Loops and Static Import	http://jcp.org/en/jsr/detail?id=201
204	Unicode Supplementary Character Support	http://jcp.org/en/jsr/detail?id=204
206	Java API for XML Processing (JAXP) 1.3	http://jcp.org/en/jsr/detail?id=206
250	Common Annotations for the Java Platform	http://jcp.org/en/jsr/detail?id=250

Next I will go over where to retrieve Java 5 and will discuss language features that will help you understand and use Seam.

Downloading Java 5

One of the biggest requirements for using Seam is that you have to use Java 5. This is pretty obvious when you realize that Seam is entirely implemented in your code with annotations and that Java 5 introduced us to annotations. Of course, as indicated in Table 1-1, there is a lot more to Java 5 than annotations, and I want to share some of the language features with you. However, in order to do that, you need to download and install Java 5 first.

Many computers come preinstalled with Java 5—all Apple OS X operating systems, for example, contain Java 5 by default. With Sun's relatively new installation of Java, it runs as a service and will update it for you. You can check whether you have the correct