JBoss
a division of Red Hat

PRENTICE HALL

SEAM FRAMEWORK

EXPERIENCE THE EVOLUTION OF JAVA™ EE

MICHAEL JUNTAO YUAN
JACOB ORSHALICK
THOMAS HEUTE

SECOND EDITION

Library of Congress Cataloging-in-Publication Data

Yuan, Michael Juntao.
Seam framework : experience the evolution of Java EE / Michael Juntao Yuan, Jacob Orshalick, Thomas Heute.—2nd ed.
p. cm.
Includes index.
ISBN 978-0-13-712939-3 (pbk. : alk. paper)
1. JBoss. 2. Web servers—Management. 3. Java (Computer program language) I. Orshalick, Jacob. II. Heute, Thomas. III. Title.
TK5105.8885.J42Y832 2009
005.2'762—dc22

2008047478

ISBN-13: 978-0-13-712939-3
ISBN-10: 0-13-712939-4

Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.
First printing, February 2009

3

Recommended JSF Enhancements

The Hello World example in Chapter 2 demonstrates how to build a Seam application with standard EJB3 and JSF. Seam chooses JSF as its web framework for many reasons. JSF is a standard technology in Java EE 5.0 and has a large ecosystem of users and vendors. All Java application servers support it. JSF is fully component-based and has a vibrant vendor community for components. JSF also has a powerful and unified expression language (EL, using the #{...} notation) that can be used in web pages, workflow descriptions, and component configuration files throughout the application. JSF also enjoys great support by visual GUI tools in leading Java IDEs.

However, JSF also has its share of problems and awkwardness. JSF has been criticized for being too verbose and too component-centric (i.e., not transparent to HTTP requests). Being a standard framework, JSF innovates more slowly than grassroots open source projects such as Seam itself and is therefore less agile when it comes to correcting design issues and adding new features. For these reasons, Seam works with other open source projects to improve and enhance JSF. For Seam applications, we strongly recommend that you use the following JSF enhancements:

- Use the Facelets framework for web pages. Write your web pages as Facelets XHTML files instead of JSP files. Facelets provides many benefits over the standard JSP in JSF; see Section 3.1.1 for more details.
- Use the Seam JSF component library for special JSF tags that take advantage of Seam-specific UI features, as well as Seam's extended EL for JSF.
- Set up Seam filters to capture and manage JSF redirects, error messages, debugging information, and so on.

Throughout the rest of the book, we assume that you already have these three JSF enhancements installed and enabled (see Section 3.3 for instructions). In Section 8.1.1, we explain how Seam supports lazy loading in JSF page rendering and expands the use of JSF messages beyond simple error messages. In Part III, we will cover integration of data components directly into the JSF web pages. Such direct integration allows Seam to add important features to JSF, including end-to-end validators (Chapter 12), easy-to-use data tables (Chapter 13), bookmarkable URLs (Chapter 15), and custom error pages (Chapter 17). In Part IV, we will discuss how to incorporate third-party AJAX UI widgets in Seam applications. In Section 24.5, we discuss how to use the jBPM business process to manage pageflows in JSF/Seam applications. This allows you to use EL expressions in page navigation rules and to have navigation rules that are dependent on the application state.

JSF 2.0

Many of the third-party JSF enhancements discussed in this chapter have made their way into the upcoming JSF 2.0 specification, so this chapter will help you with JSF 2.0 migration. Using Seam and the frameworks mentioned here, you can experience the JSF 2.0 productivity today!

In this chapter, we will first explore how those additional frameworks improve your JSF development experience. You will see how to develop applications with Facelets and Seam UI libraries. Then, in Section 3.3, we will list the changes you need to make in the Hello World example to support the Facelets and Seam UI components. The new example is in the `betterjsf` project in the book's source code bundle. Feel free to use it as a starting point for your own applications.

3.1 An Introduction to Facelets

JavaServer Pages (JSP) is the de-facto "view" technology in JavaServer Faces (JSF). In a standard JSF application, the web pages containing JSF tags and visual components are typically authored as JSP files. However, JSP is not the only choice for authoring JSF web pages. An open source project called Facelets (https://facelets.dev.java.net) allows you to write JSF web pages as XHTML files with significantly improved page readability, developer productivity, and runtime performance compared to equivalent pages authored in JSP. Although Facelets is not yet a Java Community Process (JCP) standard, we highly recommend that you use it in your Seam applications whenever possible.

3.1.1 Why Facelets?

First, Facelets improves JSF performance by 30 to 50 percent by bypassing the JSP engine and using XHTML pages directly as the view technology. By avoiding JSP, Facelets also avoids potential conflicts between JSF 1.1 and JSP 2.4 specifications, which are the specifications supported in JBoss AS 4.x (see the accompanying sidebar for details).

The Potential Conflict between JSF and JSP

In our Hello World example, we used JSP files (e.g., the `hello.jsp` file) to create the web pages in the JSF application. The JSP container processes those files at the same time they are processed by the JSF engine. That raises some potential conflicts between the JSP 2.0 container and JSF 1.1 runtime in JBoss AS 4.x. For a detailed explanation of the problems and examples, refer to Hans Bergsten's excellent article "Improving JSF by Dumping JSP" (www.onjava.com/pub/a/onjava/2004/06/09/jsf.html).

Those conflicts are resolved in JBoss AS 5.x, which supports JSP 2.1+ and JSF 1.2+. However, if you need to use JBoss 4.x for now, the best solution is to avoid JSP altogether and use Facelets instead.

Second, you can use any XHTML tags in Facelets pages. It eliminates the need to enclose XHTML tags and free text in the `<f:verbatim>` tags. These `<f:verbatim>` tags make JSP-based JSF pages tedious to write and hard to read.

Third, Facelets provides debugging support from the browser. If an error occurs when Facelets renders a page, it gives you the exact location of that error in the source file and provides context information around the error (see Section 17.5). It is much nicer than digging into a stack trace when a JSP/JSF error occurs.

Last, and perhaps most important, Facelets provides a template framework for JSF. With Facelets, you can use a Seam-like dependency injection model to assemble pages instead of manually including page header, footer, and sidebar components in each page.

The Case for JSP

If Facelets is this good, why do we bother to use JSP with JSF at all? Well, JSP is a standard technology in the Java EE stack, whereas Facelets is not yet a standard. That means JSP is supported everywhere, while Facelets might have integration issues with third-party JSF components. In the meantime, the JSP spec committee is certainly learning its lessons from Facelets. The next-generation JSPs will work a lot better with JSF.

3.1.2 A Facelets Hello World

As we discussed, a basic Facelets XHTML page is not all that different from the equivalent JSP page. To illustrate this point, we ported the Hello World sample application (see Chapter 2) from JSP to Facelets. The new application is in the `betterjsf` project. Below is the JSP version of the `hello.jsp` page:

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

<html>
<body>
<f:view>

<f:verbatim>
<h2>Seam Hello World</h2>
</f:verbatim>

<h:form>
<f:verbatim>
Please enter your name:<br/>
</f:verbatim>

<h:inputText value="#{person.name}" size="15"/><br/>
<h:commandButton type="submit" value="Say Hello"
                 action="#{manager.sayHello}"/>
</h:form>

</f:view>
</body>
</html>
```

Compare that with the Facelets XHTML version of the `hello.xhtml` page:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
<body>

<h2>Seam Hello World</h2>

<h:form>
Please enter your name:<br/>
<h:inputText value="#{person.name}" size="15"/>
<br/>
<h:commandButton type="submit" value="Say Hello"
                 action="#{manager.sayHello}"/>
</h:form>

</body>
</html>
```

It is pretty obvious that the Facelets XHTML page is cleaner and easier to read than the JSP page since the XHTML page is not cluttered up with `<f:verbatim>` tags. The

namespace declarations in the Facelets XHTML page conform to the XHTML standard. Other than that, however, the two pages look similar. All the JSF component tags are identical.

3.1.3 Use Facelets as a Template Engine

For most developers, the ability to use XHTML templates is probably the most appealing feature of Facelets. Let's see how it works.

A typical web application consists of multiple web pages with a common layout. They usually have the same header, footer, and sidebar menu. Without a template engine, you must repeat all those elements for each page. That's a lot of duplicated code with complex HTML formatting tags. Worse, if you need to make a small change to any of the elements (e.g., change a word in the header), you have to edit all pages. From all we know about the software development process, this type of copy-and-paste editing is very inefficient and error-prone.

The solution, of course, is to abstract out the layout information into a single source and thus avoid the duplication of the same information on multiple pages. In Facelets, the template page is the single source of layout information. The `template.xhtml` file in the Seam Hotel Booking example (the `booking` project in source code) is a template page.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
<head>
  <title>JBoss Suites: Seam Framework</title>
  <link href="css/screen.css" rel="stylesheet" type="text/css" />
</head>
<body>

<div id="document">
  <div id="header">
    <div id="title">...</div>
    <div id="status">
      ... Settings and Log in/out ...
    </div>
  </div>
  <div id="container">
    <div id="sidebar">
      <ui:insert name="sidebar"/>
    </div>
    <div id="content">
      <ui:insert name="content"/>
    </div>
  </div>
  <div id="footer">...</div>
</div>
</body>
</html>
```

The `template.xhtml` file defines the layout of the page header, footer, sidebar, and main content area (Figure 3.1). Obviously, the sidebar and main content area have different content for each page, so we use the `<ui:insert>` tags as placeholders in the template. In each Facelets page, we tag UI elements accordingly to tell the engine how to fill the template placeholders with content.

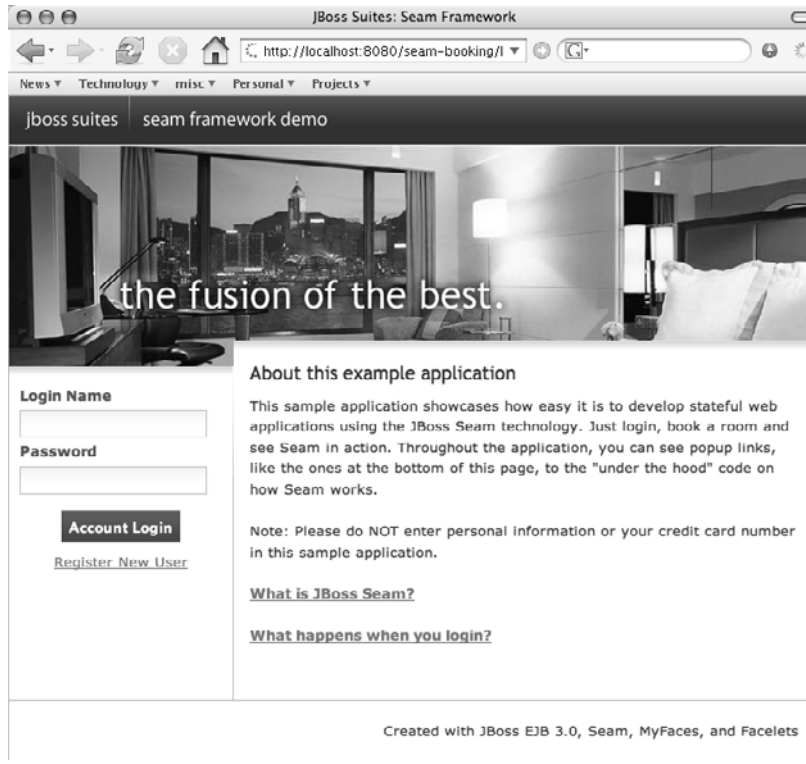

Figure 3.1 The template layout

Multiple Template Pages

Actually, we were not entirely accurate when we mentioned that the template is a "single" source for layout knowledge in an application. Facelets is flexible in managing template pages. In a Facelets application, you can have multiple template pages for alternative themes or for different sections of the web site. Yet, the basic idea of abstracting layout information to avoid duplicated code still applies.

Extensive Use of CSS

All pages in the Seam Hotel Booking example, including the `template.xhtml` page, are styled using CSS. We highly recommend using CSS in Seam/Facelet applications because it's concise and easy to understand. Even more importantly, CSS separates the styling from page content. With CSS, the web designer does not even need to understand the JSF/Seam symbols and tags in the page.

Of course, if you prefer to use XHTML tables to lay out your page, you can still do so in the `template.xhtml` file. Just make sure that you place the `<ui:insert>` tags in the right places within the nested tables.

Each Facelets page corresponds to a web page. It "injects" contents for the `<ui:insert>` placeholders into the template. Below is the `main.xhtml` page of the Seam Hotel Booking example application.

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                xmlns:ui="http://java.sun.com/jsf/facelets"
                xmlns:h="http://java.sun.com/jsf/html"
                xmlns:f="http://java.sun.com/jsf/core"
                template="template.xhtml">

  <ui:define name="content">
    <ui:include src="conversations.xhtml" />

    <div class="section">
      <h:form>
        <h1>Search Hotels</h1>
        ... ...
      </h:form>
    </div>

    <div class="section">
      <h:dataTable value="#{hotels}" ...>
        ... ...
      </h:dataTable>
    </div>

    <div class="section">
      <h1>Current Hotel Bookings</h1>
    </div>

    <div class="section">
      <h:dataTable value="#{bookings}" ...>
        ... ...
      </h:dataTable>
    </div>
  </ui:define>

  <ui:define name="sidebar">
    <h1>Stateful and contextual components</h1>
    <p>... ...</p>
  </ui:define>
</ui:composition>
```

At the beginning of the `main.xhtml` file, the code declares that the `template.xhtml` template is used to format the layout. The `<ui:define>` elements correspond to the `<ui:insert>` placeholders of the same names in the template. You can arrange those `<ui:define>` elements in any order, and at runtime, the Facelets engine renders the web pages according to the template.

3.1.4 Data List Component

One of the biggest omissions in the current JSF specification is that it lacks a standard component to iterate over a data list. The `<h:dataTable>` component displays a data list as an HTML table, but it is not a generic iteration component.

Facelets remedies this problem by providing a `<ui:repeat>` component to iterate over any data list. For instance, the following Facelets page snippet displays a list in a table-less format:

```
<ui:repeat value="#{fans} var="fan">
  <div class="faninfo">#{fan.name}</div>
</ui:repeat>
```

In Section 3.4.1 and Section 3.4.2, you will see that the Facelets `<ui:repeat>` component can be used in completely non-HTML environments.

In this section, we just scratched the surface of what Facelets can do. We encourage you to explore Facelets (https://facelets.dev.java.net/) and make the most out of this excellent framework.

3.2 Seam JSF Enhancements

Seam provides its own JSF enhancements that work with both Facelets XHTML and JSP pages. You can use Seam UI tags in your JSF view pages, use Seam's special extension to the JSF EL, and use the Seam filter to make Seam work better with the JSF URL redirecting and error handling mechanisms. Those Seam JSF components work with Seam framework features not yet discussed in the book. In this section, we will provide an overview of those enhancements but leave the details to later chapters of the book. Impatient readers can safely skip to Section 3.3 for instructions on how to install those Seam JSF components.

3.2.1 Seam UI Tags

The Seam UI tags give regular JSF UI components access to the Seam-managed runtime information. They help integrate Seam's business and data components more tightly

with the web UI components. Seam UI tags can be roughly divided into the following categories:

validation The Seam validation tags allow you to use Hibernate validator annotations on entity beans to validate JSF input fields. They also allow you to decorate an entire invalid (or valid) field when the validation fails. See Chapter 12 for more on using those components.

conversation management A key concept in Seam is the arbitrarily long web conversation (see Chapter 8). Normally, the web pages in a conversation are connected via hidden fields in HTTP `POST` operations. But what if you want to click on a regular hyperlink and still stay in the same conversation? Seam provides tags that can generate conversation-aware hyperlinks. See Sections 8.3.6 and 9.2.2 for more.

business process management Seam provides tags that can associate web page content with business processes in the background (see Chapter 24).

performance The `<s:cache>` tag encloses page content that should be cached on the server. When the page is rendered again, the cached region is retrieved from the cache instead of being dynamically rendered (see Chapter 30).

JSF replacement tags Some Seam tags are a direct replacement for JSF tags to fix certain deficiencies in JSF. Right now, the only such tag is `<s:convertDateTime>`, which fixes JSF's annoying default time zone problem.

alternative display output In addition to the standard HTML output, Seam provides JSF tags that render PDF and email outputs based on Facelets templates. It also provides tags to render Wikitext snippets into HTML elements. Refer to Section 3.4 for more details on those alternative display technologies supported by the Seam tag library.

Later chapters cover the use of these Seam UI tags when we discuss specific Seam features related to them. Here, we use the `<s:convertDateTime>` tag as an example to demonstrate how Seam UI tags are used. The `<s:convertDateTime>` tag replaces JSF's converter tag, `<f:convertDateTime>`, to convert the backend `Date` or `Time` objects to formatted output/input strings in the server's local time zone. The JSF tag is insufficient because it converts the time stamp to the UTC time zone by default. The sensible default time zone in the Seam tag makes life a lot easier for developers. To use the Seam UI tags in a web page, you need to declare the Seam taglib namespace as follows:

```
<html xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:s="http://jboss.com/products/seam/taglib">

  ... ...

  The old hello date is:<br/>
```

```
    <h:outputText value="#{manager.helloDate}">
    <s:convertDateTime/>
    </h:outputText>

    Please enter a new date:<br/>
    <h:inputText value="#{manager.helloDate}">
    <s:convertDateTime/>
    </h:inputText>

  </html>
```

3.2.2 Seam JSF EL Enhancement

Chapter 2 showed that the JSF `#{...}` EL notation is highly useful. However, in standard JSF EL, the "property" (value expression) and "method" (method expression) on the backend component are the same. As a result, the EL method expression cannot take any call arguments. For instance, the `name` property on the `person` component is expressed as follows:

```
<h:inputText value="#{person.name}" size="15"/>
```

The event handler method `sayHello()` on the `manager` component is written the same way, as shown below, and therefore cannot take any call arguments. All the objects the method operates on must be injected into the component before the method is called.

```
<h:commandButton type="submit"
                 value="Say Hello"
                 action="#{manager.sayHello}"/>
```

With the Seam EL extension, you can now call any component method with the `()` to improve readability:

```
#{component.method()}
```

The method can now take call arguments as well. So, with the following example, you no longer need to inject the `person` component into the `manager` component. That reduces the need for dependency injection and makes the application easier to read.

```
<h:commandButton type="submit"
                 value="Say Hello"
                 action="#{manager.sayHello(person)}"/>
```

Here is the new `ManagerAction` class with the new `sayHello()` method:

```
@Stateless
@Name("manager")
public class ManagerAction implements Manager {

  private Person person;
```

```
  @Out
  private List <Person> fans;

  @PersistenceContext
  private EntityManager em;

  public void sayHello (Person p) {
    em.persist (p);
    fans = em.createQuery("select p from Person p").getResultList();
  }
}
```

The enhanced EL allows multiple call arguments separated by commas. If the backend method takes a `String` argument, you can pass it directly in the EL as follows:

```
... action="#{component.method('literal string')}"/>
```

The new Seam JSF EL makes your code more readable and more elegant. Use it!

3.2.3 Use EL Everywhere

Seam not only expands the syntax of JSF EL but also makes the EL available beyond JSF web pages. In a Seam application, you can use JSF expressions to substitute static text in configuration files (Section 9.2.1), test cases (Chapters 26 and 27), JSF messages (Section 8.1.2), and jBPM processes (Chapter 24).

The expanded use of JSF EL greatly simplifies application development.

3.2.4 Seam Filter

Seam provides a very powerful servlet filter. The filter does additional processing before the web request is processed by JSF and after the web response is generated. It improves integration between Seam components and JSF.

- The filter preserves the conversation context during JSF URL redirects. That allows the Seam default conversation scope to span from the request page to the redirected response page (Chapter 8).
- It captures any uncaught runtime errors and redirects to custom error pages or the Seam debug page, if necessary (Chapter 17).
- It provides support for the file upload JSF component in Seam UI.
- It allows any non-JSF servlet or JSP page to access Seam components via the Seam `Component` class.

See Section 3.3 for how to install the Seam filter in your `web.xml`.

3.2.5 Stateful JSF

Perhaps the most important feature of Seam is that it is a stateful application framework. The stateful design has great implications for JSF. For instance, it enables much tighter integration between JSF and ORM solutions such as Hibernate (Section 6.1) and allows JSF messages to propagate across different pages (Section 8.1.2). Throughout the rest of this book, we will cover how Seam's stateful design improves web application development.

3.3 Add Facelets and Seam UI Support

To support the Facelets and Seam UI frameworks, you must first bundle the necessary library JAR files in the application. Three JAR files go into the `app.war` archive's `WEB-INF/lib` directory because they contain tag definitions. Facelets requires the `jsf-facelets.jar` file; Seam needs the `jboss-seam-ui.jar` and `jboss-seam-debug.jar` files. An additional JAR file, `jboss-el.jar`, goes into the EAR file `mywebapp.ear` to support the JSF Expression Language (EL) in both the web module (`app.war`) and the EJB3 module (`app.jar`).

```
mywebapp.ear
|+ app.war
   |+ web pages
   |+ WEB-INF
      |+ web.xml
      |+ faces-config.xml
      |+ other config files
      |+ lib
         |+ jsf-facelets.jar
         |+ jboss-seam-ui.jar
         |+ jboss-seam-debug.jar
|+ app.jar
|+ lib
   |+ jboss-el.jar
   |+ jboss-seam.jar
|+ META-INF
   |+ application.xml
   |+ jboss-app.xml
```

To use Facelets and Seam's enhancements to JSF EL, you need to load a special view handler in the `faces-config.xml` file, which is located in the `WEB-INF` directory in the `app.war` (or in the `resources/WEB-INF` directory in the project source). The view handler renders HTML web pages from Facelets template and pages. This is the relevant snippet from the `faces-config.xml` file:

```
<faces-config>
  ... ...
  <application>
    <view-handler>
      com.sun.facelets.FaceletViewHandler
    </view-handler>
  </application>
<faces-config>
```

In a Facelets application, we typically use the `.xhtml` filename suffix for web pages since they are now XHTML files, not JSP pages. We have to tell the JSF runtime about this change in the `web.xml` file (in the same directory as the `faces-config.xml` file):

```
<web-app>
  ... ...
  <context-param>
    <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
    <param-value>.xhtml</param-value>
  </context-param>
</web-app>
```

Finally, let's set up the Seam filter and resource servlet in the same `web.xml` file. The `SeamFilter` provides support for error pages, JSF redirects, and file upload. The Seam resource servlet provides access to images and CSS files in `jboss-seam-ui.jar`, which are required by Seam UI components. The resource servlet also enables direct JavaScript access to Seam components (Chapter 21).

```
<web-app>
  ... ...
  <servlet>
    <servlet-name>Seam Resource Servlet</servlet-name>
    <servlet-class>
      org.jboss.seam.servlet.ResourceServlet
    </servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Seam Resource Servlet</servlet-name>
    <url-pattern>/seam/resource/*</url-pattern>
  </servlet-mapping>

  <filter>
    <filter-name>Seam Filter</filter-name>
    <filter-class>
      org.jboss.seam.web.SeamFilter
    </filter-class>
  </filter>

  <filter-mapping>
    <filter-name>Seam Filter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

</web-app>
```

3.4 PDF, Email, and Rich Text

So far, we have discussed the JSF enhancements provided by Facelets and the `jboss-seam-ui.jar` library. Those are important usability and integration features required by almost all Seam web applications. In this section, we discuss several additional UI features Seam provides. To use those features, you need to bundle more library JAR files in your application and provide extra configuration as described below. You can choose and mix the UI feature sets you want in the application while keeping its footprint and configuration complexity to a minimum.

3.4.1 Generate PDF Reports

The Facelets XHTML files generate HTML web pages by default. However, a real-world web application sometimes needs to generate PDF output for printer-ready documents such as reports, legal documents, tickets, receipts, etc. The Seam PDF library leverages the open source iText toolkit to generate PDF documents. Here is a simple Facelets file, `hello.xhtml`, which renders a PDF document:

```
<p:document xmlns:p="http://jboss.com/products/seam/pdf"
            title="Hello">
  <p:chapter number="1">
    <p:title>
      <p:paragraph>Hello</p:paragraph>
    </p:title>
    <p:paragraph>Hello #{user.name}!</p:paragraph>

    <p:paragraph>The time now is

      <p:text value="#{manager.nowDate}">
        <f:convertDateTime style="date" format="short"/>
      </p:text>

    </p:paragraph>
  </p:chapter>

  <p:chapter number="2">
    <p:title>
      <p:paragraph>Goodbye</p:paragraph>
    </p:title>
    <p:paragraph>Goodbye #{user.name}.</p:paragraph>
  </p:chapter>
</p:document>
```

While the `hello.xhtml` file has the `xhtml` suffix, it is really an XML file with Seam PDF UI tags. When the user loads the `hello.seam` URL, Seam generates the PDF document and redirects the browser to `hello.pdf`. The browser then displays the `hello.pdf` file in its PDF reader plugin or prompts the user to save the PDF file. By passing the `pageSize` HTTP parameter to the URL, you can specify the page size of

the generated PDF document. For instance, the `hello.seam?pageSize=LETTER` URL produces a letter-sized `hello.pdf` document. Valid `pageSize` options also include `A4`, `LEGAL`, and others.

You can use any JSF EL expressions in the `xhtml` page; these EL expressions are resolved on the fly when the PDF document is rendered, just as are EL expressions on web pages. You can also use JSF converters to control text formatting, the `<f:facet>` tag to control table formatting, or the Facelets `<ui:repeat>` tag to render a list or table from dynamic data. See the Seam Reference Documentation (http://seamframework.org/Documentation) for more details on the tags.

To use the Seam PDF tags, you need to include the `jboss-seam-pdf.jar` and `itext.jar` files in the `WEB-INF/lib` directory of your WAR application archive.

```
mywebapp.ear
|+ app.war
   |+ web pages
   |+ WEB-INF
      |+ web.xml
      |+ faces-config.xml
      |+ other config files
      |+ lib
         |+ jsf-facelets.jar
         |+ jboss-seam-ui.jar
         |+ jboss-seam-debug.jar

         |+ jboss-seam-pdf.jar

         |+ itext.jar
|+ app.jar
|+ lib
   |+ jboss-el.jar
   |+ jboss-seam.jar
|+ META-INF
   |+ application.xml
   |+ jboss-app.xml
```

Then, you need to configure the PDF-related Seam component in the `components.xml` file. The `useExtensions` property indicates that the `hello.seam` URL should redirect to the `hello.pdf` URL. If the `useExtensions` property is set to `false`, the redirection would not happen and the web application would serve PDF data directly to the browser from a `.seam` URL, which could cause usability problems in some browsers.

```
<components xmlns:pdf="http://jboss.com/products/seam/pdf"
            xmlns:core="http://jboss.com/products/seam/core">

  <pdf:documentStore useExtensions="true"/>

  ... ...

</components>
```

Finally, you need to set up servlet filters for the `.pdf` files. Those filters are only needed when you have the `useExtensions` property set to `true` in the `components.xml` configuration we've just seen.

```
<web-app ...>

  ... ...

  <filter>
    <filter-name>Seam Servlet Filter</filter-name>
    <filter-class>
      org.jboss.seam.servlet.SeamServletFilter
    </filter-class>
  </filter>

  <filter-mapping>
    <filter-name>Seam Servlet Filter</filter-name>
    <url-pattern>*.pdf</url-pattern>
  </filter-mapping>

  <servlet>
    <servlet-name>
      Document Store Servlet
    </servlet-name>
    <servlet-class>
      org.jboss.seam.pdf.DocumentStoreServlet
    </servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>
      Document Store Servlet
    </servlet-name>
    <url-pattern>*.pdf</url-pattern>
  </servlet-mapping>
</web-app>
```

The Seam PDF library supports generating digitally signed PDF documents. The public key configuration, however, is beyond the scope of this book. See the Seam Reference Documentation and iText documentation for more details.

3.4.2 Template-Based Email

Sending email from your web application is not hard—but it can be a messy task. The standard JavaMail API requires developers to embed the email messages as literal strings inside Java code. That makes it very difficult to write rich email (i.e., HTML email with elaborate text formatting and embedded images), and makes it nearly impossible for non-developers to design and compose the email messages. The lack of design and branding in email messages is a major weakness in many web applications.

In Seam, we provide a template-based approach to handling email. A business person or a page designer writes the email as a web page. Here is an example email template page `hello.xhtml`:

```
<m:message xmlns="http://www.w3.org/1999/xhtml"
           xmlns:m="http://jboss.com/products/seam/mail"
           xmlns:h="http://java.sun.com/jsf/html">
  <m:from name="Michael Yuan" address="myuan@redhat.com"/>
  <m:to name="#{person.firstname} #{person.lastname}">
    #{person.address}
  </m:to>
  <m:subject>Try out Seam!</m:subject>
  <m:body>
  <p>Dear #{person.firstname},</p>
  <p>You can try out Seam by visiting
  <a href="http://labs.jboss.com/jbossseam">
    http://labs.jboss.com/jbossseam
    </a>.</p>
  <p>Regards,</p>
  <p>Michael</p>
  </m:body>
</m:message>
```

When a web user needs to send out the `hello.xhtml` message, he or she clicks on a button or a link to invoke a Seam backing bean method to render the `hello.xhtml` page. Below is an example method to send the `hello.xhtml` email. The message recipient is dynamically determined at runtime via the `#{person.address}` EL expression. Similarly, you can dynamically determine the sender address or any content in the message via EL expressions.

```
public class ManagerAction implements Manager {

  @In(create=true)
  private Renderer renderer;

  public void send() {
    try {
      renderer.render("/hello.xhtml");
      facesMessages.add("Email sent successfully");
    } catch (Exception e) {
      facesMessages.add("Email sending failed: " + e.getMessage());
    }
  }
}
```

If a message has multiple recipients, you can insert multiple `<m:to>` tags using the Facelets `<ui:repeat>` tag. You can also use the Facelets `<ui:insert>` tag to compose messages from a template.

To use the Seam email support tags, you need to bundle the `jboss-seam-mail.jar` file in the `WEB-INF/lib` directory of your WAR archive.

```
mywebapp.ear
|+ app.war
   |+ web pages
   |+ WEB-INF
      |+ web.xml
      |+ faces-config.xml
      |+ other config files
      |+ lib
         |+ jsf-facelets.jar
         |+ jboss-seam-ui.jar
         |+ jboss-seam-debug.jar

         |+ jboss-seam-mail.jar
|+ app.jar
|+ lib
   |+ jboss-el.jar
   |+ jboss-seam.jar
|+ META-INF
   |+ application.xml
   |+ jboss-app.xml
```

Then, you need to configure an SMTP server to actually send the email. That is done via the Seam `mailSession` component in `components.xml`. You can specify the host name, port number, and login credentials for the SMTP server. Here is an example SMTP configuration:

```
<components xmlns="http://jboss.com/products/seam/components"
            xmlns:core="http://jboss.com/products/seam/core"
            xmlns:mail="http://jboss.com/products/seam/mail">

  <mail:mailSession host="smtp.example.com"
                    port="25"
                    username="myuan"
                    password="mypass" />

  ... ...

</components>
```

3.4.3 Display Rich Text

A community-oriented web application often needs to display user-contributed content (e.g., forum posts, comments etc.). Here, a big issue is how to allow rich text formatting in user-contributed content. Allowing the web user to submit arbitrary HTML-formatted text is out of the question, as raw HTML is insecure and prone to various cross-site scripting attacks.

One solution is to use a WYSIWYG rich text editor widget to capture user input. The widget transforms its content to sanitized HTML when the form is submitted to the server. Refer to Section 21.3.2 for more on this subject.

Another solution, which we cover here, is to provide the web users with a small set of non-HTML markup tags they can use to format the content. When the application

displays the content, it automatically converts the markup to HTML tags. A popular non-HTML text markup language is Wikitext which is widely used on wiki community sites (e.g., the http://wikipedia.org site). The Seam `<s:formattedText>` UI component converts Wikitext to HTML formatted text. For instance, suppose that the `#{user.post}` Seam component contains the following text:

```
It's easy to make *bold text*, /italic text/,
|monospace|, -deleted text-, super^scripts^,
or _underlines_.
```

The UI element `<s:formattedText value="#{user.post}"/>` would produce the following HTML text on the web page:

```
<p>
It's easy to make <b>bold text</b>,
<i>italic text</i>, <tt>monospace</tt>
<del>deleted text</del>, super<sup>scripts</sup>,
or <u>underlines</u>.
</p>
```

Support for the `<s:formattedText>` tag is already included in the `jboss-seam-ui.jar` file. But it depends on the ANTLR (ANother Tool for Language Recognition, see www.antlr.org) parser to process the Wikitext grammar. In order to use the `<s:formattedText>` tag, you need to bundle the ANTLR JAR in your WAR archive:

```
mywebapp.ear
|+ app.war
   |+ web pages
   |+ WEB-INF
      |+ web.xml
      |+ faces-config.xml
      |+ other config files
      |+ lib
         |+ jsf-facelets.jar
         |+ jboss-seam-ui.jar
         |+ jboss-seam-debug.jar

         |+ antlr-x.y.z.jar
|+ app.jar
|+ lib
   |+ jboss-el.jar
   |+ jboss-seam.jar
|+ META-INF
   |+ application.xml
   |+ jboss-app.xml
```

With the ANTLR parser, Seam can potentially support other markup languages beyond the Wikitext. For instance, it might one day support sanitized HTML (i.e., HTML text with all potential security loopholes removed), BBCode (widely used in online forms), and others. Refer to Seam documentation for the latest updates on this subject.

3.5 Internationalization

JSF in general provides very good support for internationalization. To support the proper local encoding of web pages, you just need to select the default encoding for the XHTML pages. A safe choice would be to use UTF-8 encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
... ...
```

However, an issue in JSF is that it does not always submit the POST or GET data in the proper encoding format. To fix this, you can setup the following filter in `components.xml` to enforce UTF-8 encoding in HTTP requests.

```
<web:character-encoding-filter encoding="UTF-8"
                               override-client="true"
                               url-pattern="*.seam" />
```

Another important aspect of JSF is its ability to select different locales for localized strings in the UI. In Seam, you can define the locales supported by your application in `components.xml`.

```
<international:locale-config default-locale="en"
                             supported-locales="en fr de"/>
```

Then, we can offer the user to select the correct locale for the UI via standard JSF mechanisms.

```
<h:selectOneMenu value="#{localeSelector.localeString}">
  <f:selectItems value="#{localeSelector.supportedLocales}"/>
</h:selectOneMenu>
<h:commandButton action="#{localeSelector.select}"
                 value="#{messages['ChangeLanguage']}"/>
```

The localized strings are defined in message bundles in the `app.war/WEB-INF/classes` directory. For example, the `en` (English) locale strings are defined in the `messages_en.properties` file.

Index

Symbols

; (semicolon), in Groovy, 403
() (parentheses), in EL calls, 36
@ (at)
  in email addresses, 177
  in wildcards, 60
#{} notation, 13–14, 27, 36, 318, 352
+ operator, in Groovy, 404
<< operator, in Groovy, 402–404, 406

A

<a4j:commandButton>, 274
<a4j:commandLink>, 274
<a4j:form>, 276
<a4j:include>, 277
<a4j:log>, 277
<a4j:mediaOutput>, 277–278
<a4j:outputPanel>, 271
<a4j:page>, 276
<a4j:poll>, 276
<a4j:region>, 276
<a4j:status>, 277
<a4j:support>, 270–274
access control
  declarative, 237–243
  rule-based, 295–302
  to UI components, 239–243
  to web pages, 238–239
accrueRewards(), 198–199
ACID (Atomicity, Consistency, Isolation, Durability) transactions, 166
ACL (Access Control List), 296
@Action annotation, 420–421
<action>, 318
actors, 318–320
add(), 104
addRole(), 236, 249
@Admin annotation, 242
AJAX (Asynchronous JavaScript and XML), 7, 259–267, 279
  buttons in, 274–276
  calls in, 282–284
  debugging, 277
  JavaScript-less, 7
  routing calls, 281
AJAX Progress Bar example, 284–287
AJAX Validator example, 270–272, 280–284
Ajax4jsf, 260, 269–279
ajax4jsf.jar, 277
AJP (Apache JServ Protocol), 380
AMD64, 373
animations on web pages, 288
annotations, 4, 8
  and performance, 371
  custom, 191, 242
  identity, 244–250
  managing conversations with, 112–113
  overriding, 24
  starting nesting conversations with, 153
  type-safe, 241–243, 303–304
  validation, 35, 179–181
  wrapping objects with, 84
Ant, 54
  building applications in, 12, 61–62, 426–427, 429, 435, 441
  deploying applications in, 12, 61–62, 426–427
  running tests in, 18, 63, 341, 348, 351, 430
ANTLR (ANother Tool for Language Recognition), 45
antlr-*.jar, 439
Apache Commons, 51
Apache Open Source License, 267
app.jar, 21, 38, 50, 333, 453

app.war, 22, 38, 131, 182, 227, 277
application.xml, 21, 62, 367, 429, 448
@ApplicationException annotation, 164
applications
  building, 12, 61–62, 65, 426–427, 429, 435, 441–449
  compiling, 65
  database-driven, 213, 218–222
  deploying. *See* deployment
  developing, 61
  enterprise, 3, 5, 214
  for web. *See* web applications
  I/O versus CPU bound, 393
  lifecycle of, 371
  multitier, 3
  multiuser, 6
  restarting, 73
  root URLs for, 21
  scheduling jobs when starting up, 390–391
  scope of, 83, 417
  source code directories of, 13
@ApplicationScoped annotation, 417
assignableTickets.xhtml, 328
assignedTickets.xhtml, 329
<assignment>, 318
@Asynchronous annotation, 52, 386
<async:quartz-dispatcher>, 388
@Audited annotation, 422
authenticate(), 236
authentication, 234–237, 243, 295–296
authorization, 233–234, 237, 295, 297, 302, 412–413
  failed, 239, 241
AuthorizationException, 239, 241
autocommit mode, 173
autocompletion, 71, 98, 261–263
@AutoCreate annotation, 93
automatic payments, 386, 389
awards. *See* rewards

B

Back button
  across conversations, 129
  and HTTP sessions, 81
  during transitions, 332
  for abandoned conversations, 122
  for ended conversations, 122
  in nested conversations, 153
  within a conversation, 109, 226
backing beans, 4, 43, 104, 203, 213, 260, 262, 273, 289
bandwidth, 276, 278–279
BBCode, 45
BEA JRockit JVM, 373
begin(), 417
@Begin annotation, 112, 115, 153, 168, 207, 211
<begin-conversation>, 116
@BeginTask annotation, 325–327, 329
betterjsf example application, 28–46
  generated by seam-gen, 54–73
  with POJOs, 47–52
bijections. *See* dependency bijections
@BindingType annotation, 413
binding types, 412–415, 421–422
boilerplate code, 3–4, 9, 84–85, 89, 371
book.xhtml, 107–108, 136
bookHotel(), 117–118
BookingCaptcha class, 254
booking-ds.xml, 361
BookingObserver interface, 194
bookmarks, 4, 124
breadcrumbs, 156
browsers
  back-buttoning in. *See* **Back** button
  debugging in, 29
  dynamic graphics rendering in, 277
  executing JavaScript in, 259, 271
  interaction with servers, 261
  multiple windows/tabs of, 81–82, 101–102, 123, 133
  navigation in, 15–16, 81–82
  PDF in, 41
  right-clicking in, 189
build.properties, 12, 57, 426
build.xml, 60, 62, 312, 344, 428–438
  testing, 348–349
  third-party libraries in, 434
business charts, 267
business components, 19, 208
business processes, 83, 439
  and long-running conversations, 328
  binding data to, 323–324
  configurable, 315
  creating, 320–324
  defining, 316–318, 320–322, 335
  instances of, 322–324
  integrating in web applications, 335–338
  managing, 35, 315–334

business processes *(continued)*
starting, 336
state of, 333
triggering, 323
workflows of, 6, 9
business rules, 295–304
applying, 306–308
building, 312
dynamically updatable, 311
for access control, 295–296
for permissions, 296–297
integrating in web applications, 305–313, 335–338
invoking automatically, 338
navigation, 235, 337–338
per-instance, 299–302
security checks with, 309

C

`<cache:eh-cache-provider>`, 397
`<cache:jboss-cache-provider>`, 397
`CacheProvider` interface, 395–400
caching
and data refreshing, 399–400
multilayered, 393–400
of databases, 6, 121, 169, 376–378, 394
of entity beans, 376–378, 380
of states, 395
providers of, 394–400
second-level, 400
simplified, 398–400
calendars, 267
calling by value, 372
call stacks, 162, 165
`cancel()`, 122, 387
CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart), 252–255
`ChangePasswordAction` class, 240–241
`cheat.xhtml`, 332
`checkName()`, 280–283
`checkNameCallback()`, 283
`cid` HTTP parameter, 132–133, 147–148
class constructors, 92
class loaders, 22, 429
classpath, 22, 58, 61, 212, 296, 312, 344, 346, 348, 388, 407, 411, 442, 453
`ClusterConfig` class, 381
color pickers, 267
`Component` class, 37
components
accessing, 37, 286–287
built-in, 17–18
configuring, 97–99, 411
context of, 7
creating, 92–93, 199
decoupling, 91, 193–201
default behavior for, 8
destroying, 92
interaction among, 9
JavaScript access to, 39
lifecycle of, 92–94
loosely coupled, 166, 168
managing, 92–97
stateful, 87–99, 167
stateless, 83, 95
swapping, 94
`components.properties`, 60, 252
`components.xml`, 23
and incremental redeployment, 62
attribute swapping in, 60
authentication logic in, 237, 239, 246
business processes in, 334
business rules in, 311
caching in, 397
components in, 18, 97–98
DAOs in, 214–217
debugging in, 62, 230
filters in, 46
Hibernate in, 453
HTTP parameters in, 133
JNDI name pattern in, 429
locales in, 46
manual flushing in, 171
PDF support in, 41–42
persistence context in, 50, 168–169
ports in, 252
Quartz scheduler in, 388
queries in, 218–222
session beans in, 368
SMTP servers in, 44
starting a component from, 390–391
testing, 348, 429, 435
timeouts in, 122, 140
transactions in, 162
URL rewriting in, 140
working memory in, 309, 337
concurrency control, 171–172

configuration by exception, 8, 13
configuration files, 18, 20–25
  for deployment, 20
  for development, 59
  for out-of-the-container tests, 60
  for production, 60
  JSF expressions in, 37
  necessariness of, 97
confirm.xhtml, 109, 330, 332
confirm(), 117, 119, 121, 159–160, 165, 197, 201, 422
confirmation numbers, 160
confirmed.xhtml, 136
<Connector>, 375
context.xml, 363
convention over configuration, 8, 13
Conversation interface, 417–418
conversation.xhtml, 156
ConversationEntries class, 139
ConversationEntry class, 139, 156
conversation IDs, 104, 110–111, 132
  default name for, 134
  explicit synthetic, 135
  managing, 133–134
  natural, 134, 138
conversationList component, 130, 132, 328
conversations, 101–125
  abandoned, 122
  and memory leaks, 82
  atomic, 165–173
  background/foreground, 111, 141–142
  beginning, 124
  carrying across workspaces, 132–133
  concurrent, 123, 125, 129–130
  consecutive, 125
  context of, 117, 123, 395
  ending, 124, 141, 159, 171
  exiting, 124
  failing, 165
  hyperlinks in, 123–124
  interrupting, 129
  long-running. *See* long-running conversations
  multiple in a session, 83
  natural. *See* natural conversations
  navigating within, 122–124
  nested. *See* nested conversations
  opening new browser windows within, 133
  propagation of, 124
  scope of, 102–105, 109–110, 206, 210, 411, 415, 417
conversations *(continued)*
  stack of, 152–157
  states of, 417
  temporary, 102, 110, 115, 417
  timing out, 111–112, 122, 171
  tying components to, 83
conversations.xhtml, 131
@ConversationScoped annotation, 411, 417
conversationStack component, 149, 153–157
conversation switcher. *See* workspace switcher
conversation-timeout parameter, 112, 140–142, 156
Cookie field, 81
<core:entity-manager-factory>, 50
<core:hibernate-session-factory>, 453
<core:init>, 23, 50, 230
<core:managed-hibernate-session>, 454
<core:managed-persistence-context>, 50
<core:manager>, 171
CPU (central processing unit), 372, 393, 396
@Create annotation, 92, 94, 209–211, 390
createEntityManagerFactory(), 346
@CreateProcess annotation, 323
createRole(), 249
createUser(), 249
@CreditCard annotation, 413
CreditCardPaymentService class, 412–413
credit card processing, 412
  confirming, 121
  nonbookmarkable pages for, 204
  validating, 120, 185, 267
cron expressions, 388–389
crud example application, 213–222
CRUD (Create, Retrieve, Update, and Delete) applications, 10, 213–222
  failures in, 218
  generating from database, 72–73
CSS (Cascading Style Sheets), 33, 184
@Current annotation, 412–416, 418
customer service, 318–320

D

DAOs (Data Access Objects), 73, 213–214
  declarative, 215–218
  initializing, 217
  retrieving, 217
  using POJOs with, 214

databases
caching, 6, 121, 169, 376–378, 394
configuring, 58–61, 361–362
corrupted records in, 159
crashes of, 159
flushing, 80, 378
for identity management, 236, 243–244
GUI console for, 20
job persistence in, 388
loading, 77–79, 103, 105
mapping classes to, 13
mocking, 345–348
performance of, 79–80
production, 359–363, 376
Quartz jobs in, 388
querying, 218–222, 272
reverse-engineering, 53, 57, 72–73
roundtrips to, 7, 79, 165, 378–379, 394, 398
saving objects into, 93, 119, 159
selecting for deployment, 56
validating, 84
write operations in, 160–161
`DataBinder` interface, 191
data grids, in Otrix, 267
data lists, 34
data model
creating, 13
mapping to a web form, 13–14
rendering business charts from, 267
represented by entity objects, 11
`DataModel` class, 84, 132, 188
`@DataModel` annotation, 151, 188, 191–192, 354
`DataModelBinder` class, 191
`@DataModelSelection` annotation, 151, 189–192
`DataModelSelector` class, 191
`DataSelector` interface, 191
data tables, 187–192
displaying, 188–189
implementing, 188–191
in Tomahawk, 267
multipage, 221–222
refreshing, 274
scrollable, 264–265
date/time pickers, 289
`Date` class, 35
`debug.seam`, 230–231
debugging
displaying AJAX interactions for, 277
enabling, 62–63, 227
in Eclipse, 67
in Facelets, 29
information pages for, 229–231
in JBoss AS, 66
in JVM, 65–66
in NetBeans, 65
in Seam, 27, 37, 230–231
`<decision>`, 318
Delete button, 189
`delete()`, 191, 205
`@Delete` annotation, 304
dependency bijections, 8, 18, 115–116
and performance, 372
disabled for entity beans, 90
invoking components through, 197
overusing, 18–19
simulating, 344–345
with nested conversations, 152
dependency injections, 8, 18
and Seam EL, 36
in Facelets, 29
in Web Beans, 411–414
dependency outjections, 16, 18
`@Dependent` annotation, 419
deployment, 61–63
configuring, 20, 73
exploded, 62
in Ant, 12, 62, 426–427
in GlassFish, 367–369
in Java EE, 47, 50, 56, 365–369
in JBoss AS, 20, 51, 54, 61–62, 426
in NetBeans, 65
in Tomcat, 5, 47, 49–50, 52, 56, 312, 362–363, 365, 438
in WebLogic, 49, 51
outside JBoss AS, 51, 425, 427
redeployment, 62
types of, 410
`@DeploymentType` annotation, 410
Derby databases, 368
`destroy()`, 141
`@Destroy` annotation, 92–93
`*dev*` files, 59
`Dispatcher` class, 201
`<div>`, 184, 288–289

Dojo toolkit, 284, 287–292
drag-and-drops on web pages, 288
Drools, 5, 295–304
  deploying, 312
  library JARs for, 63, 308, 312, 438
DRY (Don't Repeat Yourself) principle, 195, 420
DTOs (Data Transfer Objects), 5, 79, 84

E

each operation, in Groovy, 402–403
eager loading, 78–79
EAR files, 21, 50, 56, 61–63, 425
  calling by value in, 372
  dependencies for, 448–449
  deployment directory for, 426
  deployment isolation in, 372
  Groovy JARs in, 407
  Hibernate JARs in, 369
  in Maven, 441–449
  Quartz JARs in, 388
  remoting JARs in, 280
  RESTEasy JARs in, 212
  RichFaces JARs in, 265
  structure of, 429
ear-deployer.xml, 372
Eclipse, 10, 316
  and seam-gen, 53
  debugging in, 67
  generating files for, 53, 58
  importing projects into, 66–67
  workspaces in, 56
e-commerce applications, 6, 106
  conversations in, 211
  URL generation in, 203
EditProjectAction class, 405–406
EHCache, 395–400
ehcache.jar, 396
EJB (Enterprise JavaBeans), 4
  and Web Beans, 411, 414
  default contexts of, 104
  dependencies for, 445–446
EJB3, 16, 425
  accessing, 23
  and manual flushing, 168
  component configuration in, 20–25
  converting to POJOs, 52
  generating an application from, 73
  implementing applications with, 52
EJB3 *(continued)*
  overriding annotations on, 24
  persistence context in, 165, 167–168
ejb3-clustered-sfsbcache-service.xml, 380
ejb3-entity-cache-service.xml, 377, 380
EJB3 Persistence. *See* JPA
ejb-jar.xml, 24, 50, 368
<ejb-local-ref>, 368
EL (JSF Expression Language), 11, 14, 27
  enhancements to, 36–37
  in data tables, 190–191
  in testing, 9, 352–354
  requesting components from, 93
el-api.jar, 366–367
el-ri.jar, 366–367
@Email annotation, 181, 270
email addresses, 177, 181, 270
email messages, 35, 42–44, 439
end(), 417
@End annotation, 110, 113, 119, 122, 153–155, 159, 169, 172–173
<end-conversation>, 116, 121–122
<end-state>, 318
@EndTask annotation, 325–327
entities
  accessing, 237–243
  detached, 166
  securing, 302–304
entity stereotype, 420
@Entity annotation, 13, 171, 403, 420
entity beans, 11, 25, 79
  caching, 376–378, 380
  debugging, 65
  generating from databases, 56
  JavaScript objects for, 283
  limitations of, 90
  managed by DAOs, 215
  reverse-engineering from databases, 53, 57, 72
  testability of, 90
  validation annotations on, 179–181
EntityListener class, 303
EntityManager class
  accessing databases via, 19–20
  and DAOs, 73, 213–215
  and persistence context, 48, 50, 247, 362
  database source for, 24
  database updating, 169

EntityManager *(continued)*
EJB3-managed, 172
enclosing operations in a transaction, 347
extended, 103
injecting, 92, 165, 168, 354
managing objects, 116, 451
mocking, 346
queries in, 219
saving invalid entity objects, 181
updating databases, 376, 379
version conflicts in, 172
EntityManagerFactory class, 168, 346
error messages, 27
customizing, 235
decorating, 35, 184–185
default, 180
displaying on web forms, 177–181, 183–185, 270, 273
EL expressions in, 104
for a failed validation, 120
initial hiding, 281, 284
internationalizing, 181
error pages, 39
custom, 37, 84, 160, 223, 225, 227, 229
redirecting, 227–229
sending JSF messages to, 229
standard, 160–161, 223–225
<error-page>, 223
event handlers, 12, 14–15
adding success messages to, 105
exceptions in, 224
in HTTP POST operations, 210
injecting objects into, 189–190
invoking, 353
triggering from JSF components, 276
validation in, 183
events, 193–201
asynchronous, 200–201
component-driven, 196–201
lifecycle, 237
notification, 201
processing, 199–201
raising, 196–198
recurring, 386–387
synchronous, 199–200
using names as constants, 198
Events API, 197–199–201
events.xml, 199
Exadel, 261, 269
exceptions, 223–231
annotated, 225–227
checked/unchecked, 164
filters for, 225, 230
handling, 155, 172
lazy loading, 79
propagating, 84
stack trace of, 227
system, 227–229
transaction-related, 223
uncaught, 223, 231
exception-type, 224
@Expiration annotation, 386, 389
exploded archives, 62

F

Facelets, 27–34, 288
as the view framework, 54
debugging, 29, 62, 229–230
error pages in, 225
page generating, 73
supporting, 38–39, 61, 64, 426, 439
using as template engine, 31–34
versus JSP, 429
faces-config.xml, 23, 38, 366–367
FacesContext class, 260
FacesMessages class, 17, 104, 120
FacesRequest class, 352, 355
@Factory annotation, 94–96, 136–138, 161–162, 208–211, 414
<factory>, 216
fading in/out on web pages, 288
fans.xhtml, 87, 91–92, 94
data table on, 187, 264–265
<f:facet>, 41, 184
file uploading, 37, 39
filters, 27, 37, 63, 104
for exceptions, 225, 230
for UTF-8 encoding, 46
setting up, 39, 42
@FinalExpiration annotation, 386, 389
findPerson(), 208–209
@FirstName annotation, 249
firstResult HTTP parameter, 221
flush(), 169, 171
forms
BBCode in, 45
displaying error messages on, 177–181, 183–185, 270, 273

forms *(continued)*
hidden fields in, 35, 143, 206, 211, 375
mapping beans to, 13–14
registration, 272, 280
submitting, 14, 220, 274–276, 289, 302, 352
frameworks
integration, 3
POJO-based, 341
rule-based security, 295–304, 438
scalability of, 80
stateful, 77–85
stateless, 80, 83–84, 143
unit-testing, 341
`@Future` annotation, 180
`<f:verbatim>`, 29–30
`<fwk:entity-home>`, 217–218
`<fwk:entity-query>`, 221–222

G

garbage collector, 82, 373
`generalError.xhtml`, 228
`@GeneratedValue` annotation, 13, 403
`GET` HTTP requests, 123
and web services, 211–212
creating workspaces for, 132
interrupting conversations via, 129
parameters in, 104, 135, 203–210, 217
starting new conversations via, 122, 128, 418
`@GET` annotation, 212
`getDelegate()`, 398
`getEmail()`, 270
`getField()`, 345
`getInstance()`, 282–283
`getNameHints()`, 262
`getProgress()`, 286–287
`getValue()`, 353
GlassFish, 365, 367–369
Google, 259, 261
`grant()`, 298
Groovy, 401–407
integrating, 406–407
JPA annotations in, 403
unit testing with, 404
using with seam-gen, 402, 406–407
`groovy-all.jar`, 407
`GroovyTestCase` class, 404
`GroovyTimesheet` class, 402–404
Groovy Time-Tracking example, 401–407

H

hash algorithms, 245
`hasRole()`, 239
`<h:commandButton>`, 123–124, 189, 274
`<h:commandLink>`, 123, 138, 189, 274
`<h:conversationStack>`, 156
`<h:dataTable>`, 34, 188–189, 262
`hello.jsp`, 30
`hello.xhtml`, 43, 87, 89
autocompletion on, 261
in Facelets, 30, 40
navigation rules for, 99
validation on, 177
with a text editor, 263, 289
`hellojpa` example application, 47–52
`hellojpa.war`, 50–51
`helloseamgen` example application, 58, 61
Hello World example, 11–25, 29–31
as a template, 429
stateful, 87–99, 177, 341, 344, 347
validation in, 270–272, 280–284
`hibernate` example application, 451–454
Hibernate, 5, 38, 77, 425
configuring, 57, 453–454
dialects of, 49, 51, 57
direct access to, 451–454
exceptions in, 79
in GlassFish, 367
integrating with providers, 394
library JARs for, 63, 367, 369
optimistic locking in, 172
persistence context in, 166
POJOs, 11
queries in, 218
testing, 430
using EJB containers with, 169
validation in, 35, 72, 84, 179, 181–183
`hibernate.cfg.xml`, 333, 434, 453–454
hidden fields, 143, 206, 211, 375
`hideCheckNameError()`, 281, 284
`<h:inputText>`, 271, 281
`<h:messages>`, 17, 104, 227
`<h:message>`, 183, 218, 254
`home.xhtml`, 235
`hotel.xhtml`, 108, 133, 136, 138, 142
`HotelBookingAction` class, 113, 116, 118, 136–138, 141, 147, 159, 166, 172, 194–198, 411–413, 418, 420–422

Hotel Booking examples. *See* Natural Hotel Booking, Nested Hotel Booking, Rewards Booking, Rules Booking, Seam Hotel Booking, Web Beans Hotel Booking
`HotelReviewAction` class, 299, 397, 399–400
`hotels.xhtml`, 135, 138
`HotelSearchCriteria` class, 419
`HotelSearchingAction` class, 110, 410, 414–415, 419–420
`<h:outputLink>`, 123, 133, 138
HSQL databases, 24, 359
  and production environments, 376
  for development, 56, 59
  for testing, 60
  GUI console for, 20
  in Tomcat, 362
  setting up, 56–57
`hsql.jar`, 362
`@HttpError` annotation, 227
HTTP protocol, 81
  404 error, 223
  500 error, 181, 223, 380
  enforcing encodings in, 46
  headers in, 81
  keepalive connections in, 374–375
  requests in, 251, 291
`HttpSession` class, 101, 260, 417
HTTP sessions
  and browser windows/tabs, 125
  and multiple browser windows, 81
  back-buttoning in, 81
  data replication in, 380
  IDs of, 81
  memory leaks in, 80, 82
  saving states in, 6, 83, 89–90, 101, 126, 375–376
  scope of, 85, 101, 235, 416
  sticky, 380–381
  storing, 372
  timing out, 83, 93, 111, 122, 235
  valid persistence context across, 79
hyperlinks, 35
  inside a conversation, 123–124
  starting nesting conversations with, 153

I

IBM JVM, 373
ICEfaces, 56, 266
`@Id` annotation, 13, 245–246, 403
`Identity` class, 198, 239
`IdentityManager` interface, 237, 243, 247, 249–250
`IdentityStore` interface, 243, 247, 251
`@IfInvalid` annotation, 183
`IllegalArgumentException`, 163
ILOG JViews, 267
`import.sql`, 59, 61, 360, 367
`import-dev.sql`, 59
`@In` annotation, 14, 17–18, 48, 93, 95, 168, 197, 201, 205, 344, 367, 372, 397
input fields
  autocompletion for, 261–263
  losing focus, 271, 273
  mapping beans to, 13–14
  matching to regular expressions, 180
  rich controls for, 263–264, 289–292
  validating, 35, 84, 177–179, 270–272
`INSERT` SQL statement, 360
`@Insert` annotation, 304
`@Install` annotation, 94, 410
install precedence, 94
Integration example, 177–185, 187–192, 214, 451
  as a template, 428–433
  bookmarkable URLs in, 203–212
  mavenized, 441
  POJO version of, 52
  testing, 352–355
integration tests, 342, 348, 351–355
  EL in, 9
  location of, 429, 435
  setting up, 349
  transactional data source for, 355
Intel EMT64, 373
`@Intercept` annotation, 324
`@Interceptor` annotation, 422
`@InterceptorBindingType` annotation, 421–422
interceptors, 411, 421–422
Internal Server Error, 223
internationalization, 46, 181
`<international:locale-config>`, 46
`@IntervalCron` annotation, 389
`@IntervalDuration` annotation, 386, 389
`InventoryException`, 164–165
`invokeApplication()`, 353–354
`invokeMethod()`, 353
IP addresses, 97
iText toolkit, 5, 40
`itext.jar`, 41, 439

J

JAAS (Java Authentication and Authorization Service), 234
JAR files, 21
  building in Maven, 441
  dependencies for, 442–445
  library, 22
  session beans in, 367
Java
  memory leaks in, 82
  server-side applications in, 82
  versions of, 4
JavaBeans, 104
JavaDB, 368
Java EE (Java Platform, Enterprise Edition), 401, 409
  exceptions in, 223–225
  required files for, 23
  security model of, 295
  standard technologies in, 29
  traditional applications for, 11
  version 1.4, 5, 47, 50, 56, 365
  version 5.0, 3–5, 10, 27, 47, 54, 77, 165, 365–369
  web component architecture in, 266
JavaMail API, 42
JavaScript
  accessing components from, 39, 286–287
  in AJAX, 276
  remoting library of. *See* Seam Remoting
  rich UI libraries in, 260
  third-party libraries in, 287
  triggering events on web pages, 281–282
Java SE (Java Platform, Standard Edition), 9, 165
Java SE 2D, 277
JAX-RS (Java API for RESTful Web Services), 211–212
`jaxrs-api.jar`, 212
`jboss-aop.jar`, 396
JBoss AOP (Aspect Oriented Programming), 396, 425
`jboss-app.xml`, 22, 51, 429, 448
JBoss AS (Application Server), 4, 47
  debugging in, 66
  deploying in, 48, 51, 54, 61–62
  installing, 12, 379, 426
  pool of threads in, 374–375
  starting, 426
  versions of, 12, 21, 29, 51, 54, 365–366, 426
JBoss Cache, 381, 395–400, 425
`jboss-cache.jar`, 396
JBoss Developer Studio, 67–71
`jboss-el.jar`, 22, 38
JBoss Microcontainer, 363
JBoss POJO Cache, 396
JBoss Rules. *See* Drools
`jboss-seam.jar`, 22, 426
`jboss-seam-debug.jar`, 38, 63, 230
`jboss-seam-mail.jar`, 43, 439
`jboss-seam-pdf.jar`, 41, 439
`jboss-seam-remoting.jar`, 280
`jboss-seam-resteasy.jar`, 212
`jboss-seam-ui.jar`, 22, 38–39, 45, 182–183, 426
`jboss-service.xml`, 380
JBoss Tools, 63, 66–71
JBoss Transaction Manager, 48, 425, 430
JBoss TreeCache, 377, 380
`jboss-web.xml`, 51, 434
jBPM (JBoss Business Process Manager), 5–6, 83, 315–334
  JSF expressions in, 37
  library JARs for, 63, 333–334, 439
  pageflow configuration in, 131, 330–333
  transaction manager disabling, 333
`jbpm.cfg.xml`, 333
`jbpm-x.y.z.jar`, 333
JCA (Java Cryptography Architecture), 245
JCaptcha, 254
JCP (Java Community Process), 28
JDBC (Java Database Connectivity API), 57, 361
JDK (Java Development Kit)
  version 5.0, 352, 425
  version 6.x, 426
JGroups project, 381
`jgroups.jar`, 396, 400
`JNDI` (Java Naming and Directory Interface), 11, 24, 49, 429
`JOIN` SQL command, 78–79
`jpa` example application, 51–52, 348
  as a template, 433–438
JPA (Java Persistence API), 11, 20, 72, 165, 167
  and Groovy, 403
  and manual flushing, 169
  directory for, 58
  features not supported in, 451
  generating an application from, 73
  in GlassFish, 367

JPA *(continued)*
optimistic locking in, 171
queries in, 218
`JpaIdentityManager` interface, 244
`JpaIdentityStore` class, 243–250
jPDL (Java Process Definition Language), 117
`*.jpdl.xml` files, 333–334
JSF (JavaServer Faces), 4, 132
and JSP, 29
component libraries of, 27, 185, 207, 211, 266–267
controlling text formatting, 41
criticisms of, 203
debugging, 29
enabling AJAX for, 7, 260, 269–278
enhancements to, 15, 27–46
facets in, 184
internal states of, 229
locales in, 46
messages in, 17, 37–38, 104–105
navigation links in, 123
problems of, 27
replacement tags for, 35
requests in, 83
servlets in, 23
stateful, 38
template framework for, 29
URL redirects in, 27, 37, 39
validation in, 35, 183, 185
version 2.0, 28
`jsf-facelets.jar`, 38, 426
JSP (JavaServer Pages), 28–31
and JSF, 29
debugging, 29
error pages in, 225
versus Facelets, 429
JTA (Java Transaction API), 48
JUnit framework, 9, 58, 342, 404
JVM (Java Virtual Machine) servers, 372–373
32-bit versus 64-bit, 373
amount of RAM for, 372
debugging in, 65–66
restrictions of, 345

K

keepalive connections, 374–375

L

`@LastName` annotation, 249
lazy associations, 161, 166
`LazyInitializationException`, 5, 166
lazy loading, 5, 77–79, 103, 105
LDAP (Lightweight Directory Access Protocol), 250
`LdapIdentityStore` class, 243, 250–251
`leftShift` operator, in Groovy, 402–404, 406
legacy schemas, 172
`@Length` annotation, 180
load balancers, 373, 380–381
locales, 46
`log4j.xml`, 373–374
logging, 373–374
login, 234
custom pages for, 229
failed, 235, 237
secure, 251
successful, 237
`login.xhtml`, 252, 319
logout, 236–237
long-running conversations, 80, 106–112
and business processes, 328
beginning, 115–117, 145, 207, 211, 418
concurrent, 418
data objects in, 190
destroying, 110
ending, 117, 119–122, 145, 225, 227, 418
lifecycle of, 109–111
managing, 35, 112–124
resuming, 417–418
loose coupling, 193, 409

M

`<mail:mailSession>`, 44
`main.xhtml`, 107
of Natural Hotel Booking example, 129, 138–139
of Rules Booking example, 299–300
of Seam Hotel Booking example, 33–34, 188
`Manager` interface, 16, 61
`ManagerAction` class, 14, 16, 19, 36, 103, 189–191, 205, 208–210, 280, 343–345
as a DAO, 213
configuration of, 97–98
decoupling in, 91
deploying, 61
refactored, 94
`ManagerPojo` class, 48, 451
manual flushing, 168–171, 378
`<mapping>`, 453
Maven, 441–449

maven-ear example application, 441–449
@Max annotation, 180
MD5 algorithm, 245
memory heap, 373
memory leaks, 80, 82, 101, 122
menus
  drop-down, 132
  in Otrix, 267
  pop-up, 189
merge(), 116, 451
message.properties, 235
MessageDigest class, 245
messages. *See* error messages, success messages
methods
  action, 73, 110, 207, 386, 389–390
  asynchronous, 390
  authenticator, 235
  backing bean, 43, 104, 203, 262
  called by value, 372
  event handler, 12, 14–15, 105, 183, 189–190, 210, 224, 276, 353
  factory, 94–96, 208–209, 414
  getter/setter, 18, 205, 283, 344
  lifecycle, 209–210, 303, 352–353
  nontransactional, 172–173
  producer, 414–416
  return types of, 115, 120, 153
  server-side, 277
  state-saving, 376
  wrapped in transactions, 355
@Min annotation, 180
mock services, 341
Model-View-Controller (MVC) component framework, 4
MPMs (Multi-Processing Modules), 380
MS SQL databases, 359
<m:to>, 43
MyFaces project, 366–367, 373
MySQL databases, 359
  configuring, 60
  initializing, 360
  installing, 359
  in Tomcat, 363
  setting up, 56–57
mywebapp.ear, 38

N

@Name annotation, 13–14, 17, 47, 90–91, 215, 371, 403
@Named annotation, 410, 412
NamedQuery interface, 218
@Namespace annotation, 98
natural conversations, 117, 134–140
  beginning via links, 135–137
  IDs of, 134, 138
  initiating, 123
  redirecting, 137–138
  resuming, 138–139
  unique keys for, 137
Natural Hotel Booking example, 106–110
  error handling in, 223–231
  long-running conversations in, 115–121
  natural conversations in, 134–140
  navigation approach in, 113–115, 131
  running, 61
  transactions in, 159
  workspaces in, 125–129, 140–142
navigation.xml, 61, 131, 428, 434
navigation rules
  conversation propagation in, 138
  defining, 15, 23, 116, 131, 151, 197, 330, 428, 434
  for restricted pages, 235, 238–239
  for timed out conversations, 129
  for unavailable pages, 133
  starting nesting conversations with, 153
  stateful, 99, 330, 428, 434
nested conversations, 145–157
  and dependency bijection, 152
  background/foreground, 156
  beginning, 152
  concurrent, 153
  continuing, 147–152
  ending, 152, 155
  parent, 149, 154
  root, 154–155
  states of, 147–149
  timing out, 156–157
Nested Hotel Booking example, 145–157
  breadcrumbs in, 156
  transactions in, 166–167
NetBeans, 10, 63
  and seam-gen, 53
  debugging, 65
  Facelets supports in, 64
  generating files for, 53, 58
new keyword, 9, 341
newInstance(), 283
notifyBookingEvent(), 195
NotLoggedInException, 229, 239, 241

@NotNull annotation, 180, 270
Number Guess example, 330–333, 335–338
numberGuess.xhtml, 330, 336–338
number spinner, 263–264

O

@Observer annotation, 97, 198–200
onblur event, 271, 273, 281
onclick event, 276
oncomplete event, 271, 276
OptimisticLockException, 172
optimistic locking, 171
Oracle databases, 359
Oracle Application Development Framework (ADF) Faces, 267
ORM (Object Relational Mapping), 5–6, 38, 77
  and DAOs, 213
  dynamic updates in, 171
  providers of, 166, 170, 172
  second-level caching in, 393–394
orm.xml, 303
oscache-xxx.jar, 277
Otrix, 267
@Out annotation, 16, 18, 95, 188, 205, 324, 344, 372

P

package-info.java, 98
packaging, 21–25, 50–51
PAGE scope, 110, 207
*.page.xml files, 73
pageflows, 113–115, 117, 153, 155
  stateful, 99, 439
pages, 11
  accessing, 229, 237–239, 302
  bookmarkable, 4, 124, 143, 203–212
  description of, 131–132
  dynamically generated, 203
  external JSF pages on, 277
  JavaScript on, 281–282
  layout of, 31
  parameters of, 204–207
  partial updating, 267, 269
  redirecting, 104, 239
  refreshing, 274
  rerendering components on, 271
  RESTful, 4
  templates for, 31–34
  timing out, 142
  tying components to, 83
pages.xml, 23, 61–62, 156
  conversations in, 112, 114, 118, 132
  error handlers in, 225, 227–229
  manual flushing in, 170
  natural conversations in, 135
  navigation rules in, 15, 99, 117, 129, 131, 133, 138, 151, 197, 238–239, 330, 428, 434
  nested conversations in, 153, 155
  page actions in, 428, 434
  page parameters in, 204–205, 207, 211, 217, 352, 428, 434
  server configuration in, 251
  timeouts in, 122
  triggering methods at page loading, 207
pageSize HTTP parameter, 40–41
<pages>, 129, 133, 238
paint(), 277
<param>, 136
passwords, 234
  annotating, 244–245, 247
  changing, 243
  encrypting, 245
@Past annotation, 180
@Path annotation, 212
@PathParam annotation, 212
@Pattern annotation, 180
patterns
  DAO, 213
  dependency bijection, 19
  factory method, 95–96
  manager components, 96–97
  observer, 193–196
  open session in view, 166
  publish-subscribe, 193
payment.xhtml, 310
PaymentService interface, 412, 415–416
PDFs
  digitally signed, 42
  page size of, 40–41
  rendering, 35, 40–42, 439
performance, 35, 371–381
  and annotations, 371
  and CPU, 393
  and database access, 376
  and dependency bijections, 372
  and I/O, 393
  on a single server, 372–379
  profiling, 394
PermissionCheck class, 298–299, 301
@PermissionCheck annotation, 304

PermissionResolver interface, 296
permissions, 296–297
  CRUD-based, 303
  resolving, 296
persist(), 20
persistence.xml, 48–50, 59
  caching in, 377, 400
  DAOs in, 214
  databases in, 24, 346, 362–363, 368
  for JBoss AS, 48
  for Tomcat, 49
  Hibernate in, 367
  persistence options in, 428, 434
  SMPC in, 168
  testing, 348, 355, 430
persistence context
  across HTTP sessions, 79
  extended, 167, 378, 451
  in EJB3, 165, 167–168
  in Hibernate, 166
  in POJOs, 52, 165
  managing, 165–167
  merging data objects into, 190
  naming, 346
  Seam-managed. *See* SMPC
PersistenceContext class, 394
@PersistenceContext annotation, 14, 20, 48, 116, 138, 169, 172
persistence-dev.xml, 59
<persistence:managed-persistence-context>, 168
@PersistenceProperty annotation, 169
PersistentPermissionResolver class, 296
Person class, 13, 179, 188
person.xhtml, 206, 208
pid HTTP parameter, 204–208, 217
POJOs (Plain Old Java Objects), 4, 7–8, 365
  annotated, 4, 11, 211
  converting EJB3 to, 52
  debugging, 65–66
  deploying, 48–50, 433–438
  enterprise, 341
  executing under transactions, 162
  instantiating, 341
  JavaScript objects for, 283
  persistence context in, 52, 165
  testing, 348, 435
  trade-offs of, 52
  transaction behavior of, 163
  using with CRUD DAOs, 214
polyglot programming, 401
pom.xml, 442–449
pooledTask component, 328–329
pooledTaskInstanceList component, 328
ports
  configuring, 97, 252
  port 443, 251
  port 8787, 65–66
POST HTTP requests, 123, 143
  and bookmarkable pages, 4
  dynamically generated pages from, 203
  event handler methods in, 210
  hidden fields in, 35
@PostLoad annotation, 303
@PrePersist annotation, 303
@PreRemove annotation, 303
@PreUpdate annotation, 303
processes. *See* business processes
prod files, 60
@ProduceMime annotation, 212
@Produces annotation, 414–416
@Production annotation, 410
profiles, 58–61, 379
progress bar, 277, 284–287
ProgressBarAction class, 285
progressCallback(), 286
projectEdit.xhtml, 405
projects
  automatically generated, 53. *See* seam-gen
  directories for, 21, 56, 58
  importing into Eclipse, 66–67

Q

Quartz example, 385–391
quartz.jar, 388
quartz.properties, 389
Quartz scheduler, 387–389
QuartzTriggerHandle class, 386–387
queries, 20, 79, 218–222
  dynamic, 219–221
  in Hibernate, 451
  in JPA, 247, 451
  results of, 84, 221–222
query(), 20
queryProgress(), 287

R

RAD (Rapid Application Development), 53–73, 401–407
raiseAsynchronousEvent(), 200

raiseEvent(), 199–200
@RaiseEvent annotation, 196, 199–200
raiseTimedEvent(), 200
raiseTransactionCompletionEvent(), 200
raiseTransactionSuccessEvent(), 200–201
@Range annotation, 180
@Read annotation, 304
reCAPTCHA, 255
Red Hat, 261
@Redirect annotation, 225, 229
reduceInventory(), 164–165
register.xhtml, 179, 248
RegisterAction class, 248–250
RegistrationAction class, 243
registration forms, 272, 280
remote.js, 282
@Remove annotation, 16, 93
renderResponse(), 353–354
Reply button, 325, 327
@RequestParameter annotation, 208, 211
@RequestScoped annotation, 416
RESTEasy project, 211
RESTful web services, 211–212
@Restrict annotation, 239, 296–298, 302–303
<restrict>, 239, 296–297, 299, 302
review.xhtml, 253, 300, 302
rewards, 194, 233–243, 248–249, 413
  rule-based, 305–308
Rewards Booking example, 194–199, 235, 305–313
RewardsManager class, 194–195, 197–199, 241–242
RFC 1321, 245
RI (Reference Implementation), 51, 366–367
<rich:datascroller>, 265
RichFaces, 56, 260–266, 405
  library JARs for, 63
richfaces-api.jar, 265
richfaces-impl.jar, 265
richfaces-ui.jar, 265
<rich:pickList>, 405
<rich:suggestionbox>, 261–262
rich text, 44–45
Role class, 246
@RoleCheck annotation, 242
@RoleName annotation, 244, 246
roles, 234–236, 249
  annotating, 244
  assigning in jBPM, 318–320
roles *(continued)*
  checking, 236, 239, 241–243
  CRUD operations for, 243
  granting, 243
  retrieving, 251
  revoking, 243
  storing, 250–251
RoomPreferenceAction class, 149–153
Ruby on Rails, 10, 53, 59, 72
RuleBasedPermissionResolver class, 296
rules. *See* business rules
Rules Booking example, 233–255, 297–302
  caching in, 395–396, 398–400
RunAsOperation interface, 249
RuntimeException, 160–161, 164, 181, 223, 227–229

S

sanitized HTML, 44–45
saveAndSchedule(), 386
sayHello(), 14, 36, 89, 92, 103, 210
<s:cache>, 35, 395, 398–399
scalability, 379, 393–400
schedulePayment(), 386–387
schedulers, 385–391
<s:conversationName>, 138–139
<s:conversationPropagation>, 124
<s:convertDateTime>, 35
@Scope annotation, 90–91, 102, 104, 127
scopes, 101
  application, 390, 417
  conversation, 91, 104, 110, 417
  default, 102–105, 206, 210
  event, 91, 104
  impedance of, 196
  of factory components, 95
  page, 110
  process, 320, 323–324
  pseudo, 416, 419
  request, 416
  session, 90, 101, 127, 416
<s:decorate>, 184–185, 271
seam script, 54
Seam, 3–10
  taglib namespace of, 35
  versions of, 237, 427
seam.bat, 54
seam.properties, 24
seam.quartz.properties, 388
Seam data-binding framework, 191–192

seamdemo.sql, 360
SeamELResolver class, 366–367
Seam Exception Filter, 225, 230
SeamFilter class, 39
seam-gen, 10, 53–73
  commands of, 73
  debugging, 62
  enabling DAO POJOs with, 214
  project generating in, 12, 20, 73, 351, 427
  setting up, 54–57, 60–61, 73
  testing, 348
  working with Groovy, 402, 406–407
  working with IDEs, 63–71
Seam Hotel Booking example, 31–34, 106, 115, 117, 119, 188
  annotation approach in, 112, 114
  conversations in, 130
  exceptions in, 164, 225–229
  MySQL database for, 359–363
  supporting workspaces in, 126
  timeouts in, 140
  URLs in, 132
  validation in, 179
Seam interceptor, 23–24
Seam-managed persistence context. *See* SMPC
Seam POJO example, 47–48
Seam Remoting, 7, 260, 279–292
SeamResourceServlet class, 253
Seam Test framework, 9, 58, 63
SeamTest class, 341–349, 351–354
Seam UI
  file uploading in, 37
  supporting, 38–39
  tags for, 34–36, 182
@Secure annotation, 421
security, 233–255, 295–304
  and restricted pages, 229
  defense-in-depth approach to, 237, 240, 302
security.drl, 296, 298, 305, 307–309
selectHotel(), 112, 115
semicolons, in Groovy, 403
server.xml, 375
servers
  clustering, 379–381
  continuation, 147
  fail tolerance of, 379
  polling, 276
  SMTP, 44
ServletException, 224
servlets, 23, 39
  and page parameters, 207
  configuring, 281
  exception filters in, 225
  for AJAX requests, 260
SESSION scope, 85, 127
Session class, 451–452
session beans, 12, 16, 52
  injecting, 367
  naming, 368
  replicating, 380
  stateful, 14, 90–94, 102, 203, 323, 325–326, 372, 411
  timing out, 92–93
  transactions in, 160, 162–163, 172
  validation in, 183
sessions. *See* HTTP sessions
@SessionScoped annotation, 416
setField(), 345–346
setName(), 273
setParameter(), 219
setPid(), 205
setTimeout(), 286
setValue(), 353
<s:formattedText>, 44–45
s:hasPermission(), 239, 297–299, 301, 306
s:hasRole(), 239
shopping carts, 7, 81–83, 102, 106
  checkout processing, 106
  saving into a database, 80
  starting conversations for, 211
showCheckNameError(), 281, 284
SHS (Secure Hash Standard), 245
@Size annotation, 181
slider.js, 286
<s:link>, 123–124, 133, 153, 189, 222, 239–240
<s:message>, 184–185
SMPC (Seam-managed persistence context), 116, 167–172, 247
  configuring, 168–169
  injecting, 168
  manual flushing, 169–171
  scoping, 170
software design
  and polyglot programming, 401
  convention over configuration principle, 8, 13
  domain-driven, 402
  DRY principle, 195, 420
  patterns of, 19, 95–96, 166, 193–196, 213, 420

spam, 253
, 184
  hiding and showing, 281, 284
Spring framework, 5, 78
startOver(), 343
<start-state>, 318
@Stateful annotation, 14, 91, 371, 411
stateful contexts, 83, 85, 90
stateful example application, 87–99, 177, 341, 344, 347
stateless stereotype, 420
@Stateless annotation, 420
states, 316–318
  caching, 395
  containers for, 148
  declarative managing, 101
  replicating, 380–381
  saving, 375–376
@Stereotype annotation, 420
stereotypes, 419–421
submit(), 299
submitComment(), 289–290, 292
success messages, 105
Sun Application Server, 433
Sun BluePrint Catalog, 267
Sun JVM, 373
<s:validate>, 84, 181–183, 270
<s:validateAll>, 182
Swing, 277
Sybase databases, 359

T

tabbed panels on web pages, 267, 288
taskId HTTP parameter, 327
taskInstanceList component, 329
taskInstanceListByType component, 329
<task-node>, 318
tasks, 316, 325–329
  assigning, 318, 329
  completing, 320, 325, 327
  IDs of, 327
  implementing business logic for, 325–326
  recurring, 385–391
  selecting in the UI, 328–329
  starting, 325, 329
  with multiple transitions, 326
TCP/IP NIO stack, 381
TDD (Test-Driven Development), 341–349
template.xhtml, 31–34
template files
  from example applications, 427–439
  generating, 12, 20, 53, 57–58, 73
  testing, 351
@Test annotation, 343
test files, 60
testing. *See also* integration tests, unit tests
  configuring, 344
  EL in, 9
  form submission, 352
  in POJO applications, 435
  JSF expressions in, 37
  mocking objects in, 94
  running test cases, 63
  simulating interactions for, 352
  test case location of, 70
testing.xml, 344
TestNG framework, 9, 58, 341–345, 352
  configuration files for, 63
testng.xml, 347, 429
<textarea>, 291
text editors
  inline, 263, 284, 289, 291–292
  rich, 44, 260, 289–291
text fields. *See* input fields
Throwable class, 224
Ticketing example, 315–334
ticketProcess.jpdl.xml, 320
tight coupling, 411
Time class, 35
timesheets
  initializing, 402
  updating, 405
Tomahawk, 185, 267
Tomcat, 5, 47, 49–50, 52, 56, 312, 362–363, 365, 438
Tomcat Connector, 380
tomcatjpa example application, 50, 52, 362, 438
TopLink, 367
@Transactional annotation, 163, 421
@TransactionAttribute annotation, 162–163, 172
<transaction:ejb-transaction>, 162
<transaction:no-transaction>, 162
TransactionPropagationType enumeration, 163

transactions, 6, 159–173
atomic, 121, 159–161, 165–173
attributes of, 162–163
beginning, 162
container-managed, 160
database, 378–379
default behavior of, 163
disabling, 162
exceptions in, 223
executing, 163
failing, 160
grouping database write operations into, 161
mocking, 345–347
propagation types of, 163
rolling back, 163–165
saving in a database, 159
Seam-managed, 116, 160–165
wrapping methods in, 355
<transition>, 318
treecache.xml, 396
trees
in Dojo, 288
in Otrix, 267
Trinidad project, 267
type safety, 199, 241–243, 303–304, 409

U

UI (user interface)
accessing, 237–243
dynamic, 259
rich web components of, 267
Seam-specific tags for, 34–36, 182, 426
selective displaying, 239–240
simulating interactions for testing, 352
visual effects for, 288–289
<ui:define>, 34
<ui:insert>, 32–34, 43
<ui:repeat>, 34, 41, 43
UML (Unified Modeling Language), 419
unit tests, 9, 18, 341–349
limitations of, 351
location of, 429, 435
of database-related functionality, 345–348
white box, 351
with Groovy, 404
unitTestSayHello(), 347
@Unwrap annotation, 96–97
@Update annotation, 304
updateModelValues(), 353–354
updateSettings(), 241–242
URLs (Uniform Resource Locators)
appending IDs to, 81
bookmarkable, 203–212
redirecting, 27, 37, 39, 104, 134, 252
RESTful, 73, 135, 203, 211, 217
root for an application, 21
user-friendly, 134, 139–140
use cases, 315
@UserName annotation, 247
usernames, 234
annotating, 244–245, 247
CRUD operations for, 243
enabling/disabling, 243
registering, 248
retrieving, 251
storing, 250–251
validating, 280–284
@UserPassword annotation, 244–245, 247
@UserPrincipal annotation, 244–245
@UserRoles annotation, 244, 246
UTF-8 encoding, 46

V

@Valid annotation, 181, 183
validation, 98, 177–185
failing, 120, 177–179
in Hibernate, 35, 72
in UI event handlers, 183
JSF components for, 185, 267
of email addresses, 177, 181, 270
server-side, 179
triggering, 181–183
<value>, 98
version attribute, 171
visual effects, 288–289
VMWare, 373
void return type, 115, 120, 153

W

WAR files, 22–24, 50, 56
AJAX JARs in, 277
building in Maven, 441
cached files in, 363
dependencies for, 446–448
Drools JARs in, 312
Groovy JARs in, 407
Quartz JARs in, 388
RESTEasy JARs in, 212
structure of, 433, 435
warning.xhtml, 87, 89, 99

web.xml, 39
  CAPTCHAs in, 253
  error handlers in, 223, 225
  Facelets in, 229
  MyFaces in, 366–367
  of Natural Hotel Booking example, 141
  servlets in, 23, 63, 281
  session beans in, 367
  skins in, 266
Web 2.0, 7, 259
web applications
  business layer of, 84, 305–313
  chatty, 80
  common layout of, 31
  contexts of, 125
  data access layer of, 78
  database-driven, 25, 159, 359–363
  EJB3-based, 428–433
  exceptions in, 166
  high-volume, 371
  multiworkspace, 143
  naming, 429, 434
  POJO-based, 433–438
  presentation layer of, 84
  request/response focused, 385
  rich UI in, 259–267
  stateful, 6, 25, 124
  stateless, 78
  transactional, 25
  URLs for, 426
Web Beans, 409–423
  and dependency injections, 411–414
  context model in, 416–419
  transient, 417
  versus EJB, 414
Web Beans Hotel Booking example, 409–423
web-beans.xml, 410–411
<web:character-encoding-filter>, 46
WebLogic, 49, 51, 433
WeblogicTransactionManagerLookup class, 49
@WebRemote annotation, 280–282, 286, 290
<web:rewrite-filter>, 140
@WebService annotation, 52
Wikitext, 35, 44, 439
wildcards, in components.properties , 60, 252
win.xhtml, 331
wizards, 80, 102, 106, 114, 277
Woodstock project, 267
working memory, 309–311, 338
WorkingMemory class, 298–299, 301–302
workspaces, 56, 82, 125–143
  carrying conversations across, 132–133
  managing, 129–134
  multiple, 128
  switching between, 18
  timing out, 140–142
workspace switcher, 122, 130–132, 139

X

XHTML files, 30, 39–40, 54, 288
XML (Extensible Markup Language)
  usefulness of, 9, 20
  XML hell, 9, 20