Your First Cup: An Introduction to the Java EE Platform

Part No: 821–1745–10
August 2010 (originally published 2007)

ORACLE

100822@24661

Contents

Preface

This is *Your First Cup: An Introduction to Java Platform, Enterprise Edition*, a short tutorial for beginning Java EE programmers. This tutorial is designed to give you a hands-on lesson on developing an enterprise application from initial coding to deployment.

Who Should Use This Book

This tutorial is for novice Java EE developers. You should be familiar with the Java programming language, particularly the features introduced in Java Platform, Standard Edition 5. While familiarity with enterprise development and Java EE technologies is helpful, this tutorial assumes you are new to developing Java EE applications.

Before You Read This Book

Before you start this tutorial, you should:

- Be familiar with the Java programming language
- Be able to install software on your work machine
- Have a modern web browser installed on your work machine

Related Books and Projects

The following books and projects may be helpful to you in understanding this tutorial:

- *The Java EE 5 Tutorial*
- The Sun Java System Application Server documentation set
- The NetBeans IDE documentation set

Related Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

TABLE P–1 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell	`machine_name%`
C shell for superuser	`machine_name#`
Bourne shell and Korn shell	`$`
Bourne shell and Korn shell for superuser	`#`

◆ ◆ ◆ CHAPTER 1

Introduction

An introduction to this tutorial. This chapter outlines the goals and the prerequisites for completing this tutorial.

Goals of This Tutorial

At the completion of this tutorial, you will:

- Understand the basics of tiered applications
- Understand the basics of the Java EE platform
- Have created a multi-tiered Java EE application
- Have deployed and run your application on a Java EE server
- Know where to go next for more information on the Java EE platform

Requirements for This Tutorial

A Checklist

To complete this tutorial, you need to:

- Get the tutorial bundle
- Get the Java EE 5 Software Development Kit
- Get NetBeans IDE (optional)
- Configure your environment

Getting the Tutorial Bundle

To get the tutorial bundle, go to the project's java.net (`https://firstcup.dev.java.net/servlets/ProjectDocumentList`) site.

Getting the Java EE 5 SDK

To get the Java EE 5 SDK, go to `http://java.sun.com/javaee/downloads/previous_u6/index.jsp`.

Getting NetBeans 5.5

To get NetBeans 5.5 go to `http://www.netbeans.org/downloads/index.html`.

Configuring Your Environment

Once you have all the necessary downloads, you must configure the tutorial bundle to reflect your environment.

Building the Examples Using NetBeans IDE

To run the tutorial examples in NetBeans IDE, you must register your Application Server installation as a NetBeans Server Instance. Follow these instructions to register the Application Server in NetBeans IDE.

1. Select Tools→Server Manager to open the Server Manager dialog.
2. Click Add Server.
3. Under Server, select Sun Java System Application Server and click Next.
4. Under Platform Location, enter the location of your Application Server installation.
5. Select Register Local Default Domain and click Next.
6. Under Admin Username and Admin Password, enter the admin name and password created when you installed the Application Server.
7. Click Finish.

Building the Examples on the Command-Line Using Ant

Build properties common to all the examples are specified in the `build.properties` file in the *tut-install*`/example/bp-project/` directory. You must create this file before you can run the examples. Copy the file `build.properties.sample` to `build.properties` and edit it to reflect your environment. The tutorial examples use the Java BluePrints (`http://www.oracle.com/technetwork/java/index-jsp-136701.html`) build system and application layout structure.

To run the Ant scripts, you must set common build properties in the file *tut-install*/example/bp-project/build.properties as follows:

- Set the javaee.home property to the location of your Application Server installation. The build process uses the javaee.home property to include the libraries in *as-install*/lib/ in the classpath. All examples that run on the Application Server include the Java EE library archive, *as-install*/lib/javaee.jar. in the build classpath. Some examples use additional libraries in *as-install*/lib/; the required libraries are enumerated in the individual technology chapters.

  Note – On Windows, you must escape any backslashes in the javaee.home property with another backslash or use forward slashes as a path separator. So, if your Application Server installation is C:\Sun\AppServer, you must set javaee.home to javaee.home = C:\\Sun\\AppServer or javaee.home=C:/Sun/AppServer.

- Set the firstcup.tutorial.home property to the location of your tutorial. This property is used for Ant deployment and undeployment.

  For example, on UNIX:

  ```
  firstcup.tutorial.home=/home/username/firstcup
  ```

  On Windows:

  ```
  firstcup.tutorial.home=C:/firstcup
  ```

  Do not install the tutorial to a location with spaces in the path.
- If you did not accept the default values for the admin user and password, set the admin.user property to the value you specified when you installed the Application Server, and set the admin user's password in the admin-password.txt file in the *tut-install*/example/bp-project/ directory to the value you specified when you installed the Application Server.
- If you did not use port 8080, set the javaee.server.port property to the value specified when you installed the Application Server.

◆ ◆ ◆ CHAPTER 2

The Java Platform, Enterprise Edition

This chapter outlines the features of Java EE, how it differs from Java SE and Java ME, and the basic concepts behind enterprise application development.

Differences between Java EE and Java SE

Java technology is both a programming language and a platform. The Java programming language is a high-level object-oriented language that has a particular syntax and style. A Java platform is a particular environment in which Java programming language applications run.

There are several Java platforms. Many developers, even long-time Java programming language developers, do not understand how the different platforms relate to each other.

The Java Programming Language Platforms

There are three platforms of the Java programming language:

- Java Platform, Standard Edition (Java SE)
- Java Platform, Enterprise Edition (Java EE)
- Java Platform, Micro Edition (Java ME)

All Java platforms consist of a Java Virtual Machine (VM) and an application programming interface (API). The Java Virtual Machine is a program, for a particular hardware and software platform, that runs Java applications. An API is a collection of software components that you can use to create other software components or applications. Each Java platform provides a virtual machine and an API, and this allows applications written for that platform to run on any compatible system with all the advantages of the Java programming language: platform-independence, power, stability, ease-of-development, and security.

Java SE

When most people think of the Java programming language, they think of the Java SE API. Java SE's API provides the core functionality of the Java programming language. It defines everything from the basic types and objects of the Java programming language to high-level classes that are used for networking, security, database access, graphical user interface (GUI) development, and XML parsing.

In addition to the core API, the Java SE platform consists of a virtual machine, development tools, deployment technologies, and other class libraries and toolkits commonly used in Java applications.

Java EE

The Java EE platform is built on top of the Java SE platform. Java EE provides an API and runtime environment for developing and running large-scale, multi-tiered, scalable, reliable, and secure network applications.

Java ME

The Java ME platform provides an API and a small-footprint virtual machine for running Java programming language applications on small devices, like cellular phones. The API is a subset of the Java SE API, along with special class libraries useful for small device application development. Java ME applications are often clients of Java EE application services.

Overview of Enterprise Applications

This section describes enterprise applications and how they are designed and developed.

As stated above, the Java EE platform is designed to help developers create large-scale, multi-tiered, scalable, reliable, and secure network applications. A shorthand name for such applications is "enterprise applications," so called because these applications are designed to solve the problems encountered by large enterprises. Enterprise applications are not only useful for large corporations, agencies, and governments, however. The benefits of an enterprise application are helpful, even essential, for individual developers and small organizations in an increasingly networked world.

The features that make enterprise applications powerful, like security and reliability, often make these applications complex. The Java EE platform is designed to reduce the complexity of enterprise application development by providing a development model, API, and runtime environment that allows developers to concentrate on functionality.

Tiered Applications

In a multi-tiered application, the functionality of the application is separated into isolated functional areas, called tiers. Typically, multi-tiered applications have a client tier, a middle tier, and a data tier (often called the enterprise information systems tier). The client tier consists of a client program that makes requests to the middle tier. The middle tier's business functions handle client requests and process application data, storing it in a permanent datastore in the data tier.

Java EE application development concentrates on the middle tier to make enterprise application management easier, more robust, and more secure.

The Client Tier

The client tier consists of application clients that access a Java EE server and that are usually located on a different machine from the server. The clients make requests to the server. The server processes the requests and returns a response back to the client. Many different types of applications can be Java EE clients, and they are not always, or even often Java applications. Clients can be a web browser, a stand-alone application, or other servers, and they run on a different machine from the Java EE server.

The Web Tier

The web tier consists of components that handle the interaction between clients and the business tier. Its primary tasks are the following:

- Dynamically generate content in various formats for the client.
- Collect input from users of the client interface and return appropriate results from the components in the business tier.
- Control the flow of screens or pages on the client.
- Maintain the state of data for a user's session.
- Perform some basic logic and hold some data temporarily in JavaBeans components.

Java EE Technologies Used in the Web Tier

The following Java EE technologies are used in the web tier in Java EE applications.

TABLE 2–1 Web Tier Technologies

Technology	Purpose
Servlets	Java programming language classes that dynamically process requests and construct responses, usually for HTML pages

TABLE 2–1 Web Tier Technologies *(Continued)*

Technology	Purpose
JavaServer Pages (JSP)	Text-based documents that are compiled into servlets and define how dynamic content can be added to static pages, such as HTML pages.
JavaServer Faces technology	A user-interface component framework for web applications that allows you to include UI components (such as fields and buttons) on a page, convert and validate UI component data, save UI component data to server-side data stores, and maintain component state.
JavaServer Pages Standard Tag Library	A tag library that encapsulates core functionality common to JSP pages
JavaBeans Components	Objects that act as temporary data stores for the pages of an application

The Business Tier

The business tier consists of components that provide the business logic for an application. Business logic is code that provides functionality to a particular business domain, like the financial industry, or an e-commerce site. In a properly designed enterprise application, the core functionality exists in the business tier components.

Java EE Technologies Used in the Business Tier

The following Java EE technologies are used in the business tier in Java EE applications:

- Enterprise JavaBeans (enterprise bean) components
- JAX-WS web service endpoints
- Java Persistence API entities

The Enterprise Information Systems Tier

The enterprise information systems (EIS) tier consists of database servers, enterprise resource planning systems, and other legacy data sources, like mainframes. These resources typically are located on a separate machine than the Java EE server, and are accessed by components on the business tier.

Java EE Technologies Used in the EIS Tier

The following Java EE technologies are used to access the EIS tier in Java EE applications:

- The Java Database Connectivity API (JDBC)
- The Java Persistence API
- The J2EE Connector Architecture

- The Java Transaction API (JTA)

Java EE Servers

A Java EE server is a server application that the implements the Java EE platform APIs and provides the standard Java EE services. Java EE servers are sometimes called application servers, because they allow you to serve application data to clients, much as how web servers serve web pages to web browsers.

Java EE servers host several application component types that correspond to the tiers in a multi-tiered application. The Java EE server provides services to these components in the form of a *container*.

Java EE Containers

Java EE containers are the interface between the component and the lower-level functionality provided by the Java EE platform to support that component. The functionality of the container is defined by the Java EE platform, and is different for each component type. Nonetheless, the Java EE server allows the different component types to work together to provide functionality in an enterprise application.

The Web Container

The web container is the interface between web components and the web server. A web component can be a servlet, a JSP page, or a JavaServer Faces page. The container manages the component's lifecycle, dispatches requests to application components, and provides interfaces to context data, such as information about the current request.

The Application Client Container

The application client container is the interface between Java EE application clients, which are special Java applications that use Java EE server components, and the Java EE server. The application client container runs on the client machine, and is the gateway between the client application and the Java EE server components that the client uses.

The EJB Container

The EJB container is the interface between enterprise beans, which provide the business logic in a Java EE application, and the Java EE server. The EJB container runs on the Java EE server and manages the execution of an application's enterprise beans.

◆ ◆ ◆ CHAPTER 3

Creating Your First Java EE Application

This chapter gives an overview of the example application and step-by-step instructions on coding the example application.

Architecture of the Example Application

The example application consists of three main components: `DukesAgeService`, a web service endpoint; `DukesBirthdayBean`, an enterprise bean; and `firstcup`, a web application created with JavaServer Faces technology.

`DukesAgeService` is a JAX-WS endpoint that calculates the age of Duke, the Java mascot. Duke was born on May 23, 1995, when the first demo of Java technology was publicly released.

`DukesBirthdayBean` is a stateless session bean that calculates the difference between the user's age and Duke's age.

The `firstcup` web application is a JavaServer Faces application that accesses `DukesAgeService` to display Duke's age, reads in a date provided by the user, accesses`DukesBirthdayBean` to calculate who is older, and then displays the difference in years between the user and Duke.

The web application consists of the following:

- `greeting.jsp`: A JSP page with which a user can enter a date.
- `response.jsp`: A JSP page that tells the user whether he or she is older or younger than Duke, based on the date the user entered in the `greeting.jsp` page.
- `DukesBDay.java`: A JavaServer Faces managed bean that defines properties to hold the user's birth date, get Duke's current age from the web service, and get the age difference between the user and Duke from the enterprise bean.
- `faces-config.xml`: A file used to configure resources for the JavaServer Faces application. In the case of this application, the file configures a resource bundle containing messages, the managed bean, `DukesBDay`, and the page navigation rules.

- `web.xml`: The web application's deployment descriptor, which is used to configure certain aspects of a web application when it is installed. In this case, it is used to provide a mapping to the application's `FacesServlet` instance, which accepts incoming requests, passes them to the life cycle for processing, and initializes resources.

Tiers in the Example Application

The example application has one web tier component (the `firstcup`web client) and two business tier components (the `DukesAgeService`web service and the `DukesBirthdayBean`enterprise bean). The user's web browser is the client tier component, as it accesses the rest of the application through the web tier. The example application does not access the EIS tier.

Java EE Technologies Used in the Example Application

The `DukesAgeService` web service endpoint is a JAX-WS endpoint The `DukesBirthdayBean` enterprise bean is a stateless session bean. The `firstcup`web client is a JavaServer Faces application that runs in the web container of the Java EE server.

Coding the Example Application

This section describes how to code the example application.

Getting Started

Before you start coding the example, you need to perform some set-up tasks:

1. Register the server with your NetBeans IDE.
2. Create a directory for the example you will build.
3. Specify some settings.

▼ Register the Server with NetBeans IDE

1 Launch the NetBeans IDE.

2 From the menu, select Tools→Server Manager.

3 Click Add Server.

4 In the Add Server Instance dialog, select Sun Java System Application Server from the Server menu.

5. **(Optional) Enter a name for the server in the Name field.**
6. **Click Next.**
7. **Click Browse to find the location of the Application Server installation.**
8. **Click Next.**
9. **Enter the user name you chose when you installed the Application Server in the Admin Username field.**
10. **Enter the password for this username in the Admin Password field.**
11. **Click Finish.**
12. **Click Close.**

▼ Create a Directory for the Example

1. **Create another directory at the same level as the** `example` **directory and call it** `myexample`**. You'll put the First Cup application that you build while following this tutorial in this directory.**
2. **Copy the entire** `bp-project` **folder from the** `example` **directory to the** `myexample` **directory.**

▼ Specify Some Settings

1. **In the** `myexample/bp-project` **directory, copy the** `build.properties.sample` **file to** `build.properties`**.**
2. **Open the** `build.properties` **file in a text editor.**
3. **Set the** `javaee.home` **property to the path of your Application Server installation. Use forward slashes in the path, even if you are on the Windows platform. For example, you would enter** `C:/myServer` **instead of** `C:\myServer`**.**
4. **Set the** `firstcup.tutorial.home` **property to the location of your** `firstcup` **tutorial installation, such as** `C:/firstcup`**.**
5. **Change** `example` **to** `myexample` **in the path specified by the** `javaee.server.passwordfile` **property.**
6. **Open the** `admin-password.txt` **file from the** `myexample/bp-project` **directory in a text editor.**

7. **Set the `AS_ADMIN_PASSWORD` property to your Application Server password.**

8. **Save the `build.properties` and `admin-password.txt` files and close them.**

Creating the Web Service Endpoint

The `DukesAgeService` endpoint is a simple web service. Web services are web-based applications that use open, XML-based standards and transport protocols to exchange data with calling clients. Both the requests and responses are sent as XML documents, and are usually sent as HTTP packets. This makes interoperability between different systems and applications easy, as it is not necessary for the client to know the underlying architecture of the server and vice-versa to make a successful web service call.

Web services are designed to be independent of the client. Typically web service endpoints are publicly available to a wide variety of clients, and the clients are located throughout the internet. This is called "loose coupling," as the clients and servers are connected only by the standard XML-based requests and responses. For this reason, `DukesAgeService` will be developed in its own application module, and deployed separately from the `DukesBirthdayBean` enterprise bean and `firstcup` web client.

JAX-WS Endpoints

`DukesAgeService` is a JAX-WS endpoint implemented as a servlet. Servlets are web components that run in the web container.

You'll begin by creating a servlet class, then decorate the class with the `@WebService`annotation to make the class a web service endpoint, and finally you will add the `getDukesAge` method to calculate and return Duke's age.

Creating the Endpoint

In NetBeans IDE or another editor, create a Java class source file called `DukesAge.java` in the `com.sun.firstcup.webservice` package.

▼ Create the Project in NetBeans IDE

1. **Select File→New Project.**

2. **Select Web in the Categories pane.**

3. **Select Web Application in the Projects pane.**

4. **Click Next.**

5. **Set Project Name to `firstcup-dukes-age`.**

6. **Set the Project Location to `<INSTALL>/myexample`, in which INSTALL is the location of the `firstcup` tutorial installation.**
7. **Select your Application Server from the Server menu.**
8. **Select Java EE 5 from the Java EE Version menu.**
9. **Set Context Path to `/DukesAgeService`**
10. **Click Finish.**

    You should now see the module you created in the Projects pane.
11. **From the Projects pane, right-click on the `index.jsp` file and select Delete. Click Yes in the dialog.**

▼ Create the `DukesAge` Class

1. **Select File→New File.**
2. **Make sure `firstcup-dukes-age` is selected in the Project menu.**
3. **Select Java Classes in the Categories pane.**
4. **Select Java Class in the File Types pane.**
5. **Click Next.**
6. **Set Class Name to `DukesAge`.**
7. **Set Package to `com.sun.firstcup.webservice`.**
8. **Click Finish.**

   You should now see the `DukesAge.java` file inside the `com.sun.firstcup.webservice` package in the Projects pane. The `DukesAge.java` file should also be open in the editor pane.

▼ Annotate the `DukesAge` Class as a Web Service

- **Add a `@WebService` annotation to the class.**

```
@WebService
public class DukesAge {
...
}
```

▼ Remove the Default Constructor

- **Highlight the following default constructor and delete it, as web service endpoints do not require a default constructor.**

```
public DukesAge() {
}
```

▼ Add the `getDukesAge` Method

1. **Create a public `getDukesAge` method with a return type of `int`.**

```
public int getDukesAge() {
}
```

2. **Add a `@WebMethod` annotation to `getDukesAge`.**

```
@WebMethod
public int getDukesAge() {
}
```

3. **Add the following code to `getDukesAge`:**

```
Calendar dukesBirthday = new GregorianCalendar(1995, Calendar.MAY, 23);
Calendar now = Calendar.getInstance();

int dukesAge = now.get(Calendar.YEAR) - dukesBirthday.get(Calendar.YEAR);
dukesBirthday.add(Calendar.YEAR, dukesAge);

if (now.before(dukesBirthday)) {
    dukesAge--;
}
return dukesAge;
```

▼ Resolve the Import Statements

1. **Right-click in the Editor.**
2. **Select Fix Imports.**
3. **In the Fix Imports dialog, choose the `javax.jws.WebService` package for the `WebService` class.**
4. **Select File→Save from the menu to save the file.**

Building and Deploying the Web Service

▼ Building `DukesAgeService`

1. **Select `firstcup-dukes-age` in the Projects tab.**
2. **Right-click `firstcup-dukes-age` and select Build Project.**

▼ Deploying the Web Service Endpoint

The `DukesAgeService` endpoint was packaged in a WAR file, `firstcup-dukes-age.war`. Now you'll deploy `firstcup-dukes-age.war` to the Application Server. This task gives instructions on deploying `firstcup-dukes-age.war` in NetBeans IDE.

1. **Select `firstcup-dukes-age` in the Projects tab.**
2. **Right-click `firstcup-dukes-age` and select Deploy Project.**

Creating the Enterprise Bean

`DukesBirthdayBean` is a stateless session bean. Stateless session beans are beans that do not maintain a conversational state with a client. With stateless session beans the client makes isolated requests that do not depend on any previous state or requests. If you require conversational state, you use stateful session beans.

To create `DukesBirthdayBean` you need to create two Java source files: `DukesBirthdayBean`, the enterprise bean class; and `DukesBirthdayRemote`, the enterprise bean business interface. The enterprise bean class and the business interface are the only files you need to create an enterprise bean.

Creating `DukesBirthdayBean` in NetBeans IDE

This section has instructions for creating the enterprise application and `DukesBirthdayBean` enterprise bean.

▼ Creating the Enterprise Application

In this task, you will create an enterprise application archive (EAR) that will contain the `DukesBirthdayBean` enterprise bean and `firstup` web client.

1. **Select File→New Project.**
2. **Select Enterprise in the Categories pane.**

3. **Select Enterprise Application in the Projects pane.**
4. **Click Next.**
5. **Set Project Name to** `firstcup`**.**
6. **Set the Project Location to** `<INSTALL>/myexample`**.**
7. **By default the wizard creates an enterprise bean module** `firstcup-ejb` **and a web application module** `firstcup-war`**. Leave everything in the dialog as it is.**
8. **Click Finish.**

▼ Setting the Context Root

In this task, you will specify the context root to identify the web application in a J2EE server.

1. **Expand the** `firstcup` **module in the Projects pane.**
2. **In the Configuration Files node, double-click the** `application.xml` **file.**
3. **In** `application.xml`**, locate the** `context-root` **element.**
4. **Change the context root from** `/firstcup-war` **to** `/firstcup`**.**
5. **Select File → Save to save the file.**
6. **Right-click the** `firstcup-war` **module in the Projects pane.**
7. **Select Properties from the popup menu.**
8. **In the Categories tree, select Run.**
9. **Change the Context Path to** `/firstcup`**.**
10. **Click OK.**

▼ Creating the `DukesBirthdayBean` Enterprise Bean Class

Now you'll create the enterprise bean class and business interface source files in NetBeans IDE. The `DukesBirthdayRemote` business interface is a remote interface.

1. **Select** `firstcup-ejb` **project in the Projects tab.**
2. **Select File→New File.**

3. **Select Enterprise in the Categories pane.**
4. **Select Session Bean in the File Types pane.**
5. **Click Next.**
6. **Set EJB Name to** `DukesBirthdayBean`**.**
7. **Set the Package name to** `com.sun.firstcup.ejb`**.**
8. **Set the Session Type to Stateless.**
9. **Uncheck Local and check Remote under Create Interface.**
10. **Click Finish.**

▼ Modify `DukesBirthdayBean.java`

Now you'll add the code that calculates the difference in age in years between Duke and the user.

1. **In** `DukesBirthdayBean.java`**, delete the empty default constructor that NetBeans IDE generated.**
2. **Directly after the class declaration, paste in the following code:**

```
private static Logger logger =
        Logger.getLogger("com.sun.firstcup.ejb.DukesBirthdayBean");
public int getAgeDifference(Date date) {
    int ageDifference;

    Calendar theirBirthday = new GregorianCalendar();
    Calendar dukesBirthday = new GregorianCalendar(1995, Calendar.MAY, 23);

    // Set the Calendar object to the passed in Date
    theirBirthday.setTime(date);

    // Subtract the user's age from Duke's age
    ageDifference = dukesBirthday.get(Calendar.YEAR) -
    theirBirthday.get(Calendar.YEAR);
    logger.info("Raw ageDifference is: " + ageDifference);
    // Check to see if Duke's birthday occurs before the user's. If so,
    // subtract one from the age difference
    if (dukesBirthday.before(theirBirthday) && (ageDifference > 0)) {
        ageDifference--;
    }
    logger.info("Final ageDifference is: " + ageDifference);

    return ageDifference;
}
```

3. **Right-click in the editor window and select Fix Imports.**

4. **Choose the `java.util.logging.Logger` fully-qualified name for the `Logger` class.**
5. **Choose the `java.util.Date` fully-qualified name for the `Date` class.**
6. **Click OK.**
7. **Right-click within the new `getAgeDifference` method and select EJB Methods→Add to Remote Interface.**
8. **Select File→Save.**

▼ Modify `DukesBirthdayRemote.java`

1. **Expand the `firstcup-ejb` module in the Projects pane.**
2. **Double-click `DukesBirthdayRemote.java` under Source Packages→`com.sun.firstcup.ejb`.**
3. **Right-click in the editor window and select Fix Imports.**
4. **Select `java.util.Date` as the fully-qualified class name of the `Date` class and click OK.**
5. **Remove the `throws` clause from the method definition so that you are left with `int getAgeDifference(Date date);`.**
6. **Select File→Save.**

Creating the Web Client

To create the web client, you need to perform the following tasks:

- Set the `firstcup-war` module to support JavaServer Faces technology. This will create a `web.xml` file that has a mapping to `FacesServlet`.
- Create a web service client.
- Create a resource bundle to hold localized messages used by the JSP pages.
- Configure the resource bundle in the configuration file.
- Create the `DukesBDay` managed bean class.
- Configure `DukesBDay` in the configuration file.
- Create the `greeting.jsp` page.
- Configure the navigation rules in the configuration file.
- Create the `response.jsp` page.

Setting `firstcup-war` to Support JavaServer Faces Technology

All JavaServer Faces applications must include a mapping to the `FacesServlet` instance in their deployment descriptors. The `FacesServlet` instance accepts incoming requests, passes them to the life cycle for processing, and initializes resources.

You create the mapping to `FacesServlet` in the web application's deployment descriptor. Rather than adding the mapping to the `web.xml` file directly, NetBeans IDE will create the `web.xml` file and perform the mapping to `FacesServlet` for you when you specify that your web application supports JavaServer Faces technology. To do this, perform the following task.

▼ Setting `firstcup-war` to Support JavaServer Faces Technology

1. **Right-click the `firstcup-war` module of the `firstcup` enterprise application in the Projects pane.**
2. **Select Properties from the popup menu.**
3. **Select Frameworks from the Categories tree.**
4. **Click Add.**
5. **Select JavaServer Faces from the list of choices in the dialog.**
6. **Click OK.**
7. **Change the servlet URL mapping to `/firstcupWeb/*`.**

   This path will be the path to the `FacesServlet` instance. All requests must include this path in between the application's context path and the page in the URL. Users don't have to include this path in the URL because `firstcup` includes an `index.jsp` page that forwards users to the `greeting.jsp` page when they enter the following URL:

   ```
   http://localhost:8080/firstcup
   ```

8. **Deselect the Validate XML checkbox.**
9. **Click OK.**

   The remaining steps tell you how to perform the forward to `greeting.jsp` from the `index.jsp` page.

10. **Expand the `firstcup-war` module in the Projects pane and double-click Web Pages.**
11. **Right-click `welcomeJSF.jsp`, select Delete from the popup menu, and click Yes in the dialog.**

12 **Double-click** `index.jsp`**.**

NetBeans IDE generated this file when you created the `firstcup` project.

13 **Delete everything on the page.**

14 **Enter the following in the** `index.jsp` **page:**

```
<jsp:forward page="/firstcupWeb/greeting.jsp"/>
```

15 **Save the file by selecting File → Save from the menu bar.**

Creating a Web Service Client for the `firstcup-war` Web Module

The `firstcup-war` web module must consume the `firstcup-dukes-age` web service in order to get Duke's current age. For this to happen, you need to create a web service client for the `firstcup-war` web module.

▼ Creating a Web Service Client for the `firstcup-war` Web Module

1 **Select** `firstcup-war` **from the Project pane.**

2 **Select File → New File.**

3 **Select Web Services from the Categories pane.**

4 **Select Web Service client from the File Types pane.**

5 **Click Next.**

6 **Select WSDL URL.**

7 **Into the WSDL URL field, enter the following location of the WSDL file of the web service that the web service client will consume.**

```
http://localhost:8080/DukesAgeService/DukesAgeService?WSDL
```

8 **Into the Package field, enter the following package where the client files will be generated.**

```
com.sun.firstcup.webservice
```

9 **Click Finish.**

Creating a Resource Bundle

In this section, you'll create the resource bundle that contains the static text and error messages used by the JSP pages. The `firstcup` client supports both English and Spanish locales. Therefore you need to create two properties files, each of which will contain the messages for one of the locales.

▼ Creating a Resource Bundle

1. **Right-click `firstcup-war` in the Projects pane.**
2. **Select New → File/Folder from the popup menu.**
3. **Select the Other category, then Properties File from the New File dialog.**
4. **In the New Properties File dialog, enter `WebMessages` in the File Name field.**
5. **In the Folder field, enter `src/java/com/sun/firstcup/web` as the location of the file.**
6. **Click Finish.**
7. **After NetBeans IDE creates the properties file, enter the following messages or copy them from here to the file:**

```
Welcome=Hi. My name is Duke. Let us find out who is older -- You or me.
DukeIs=Duke is
YearsOldToday=years old today.
Instructions=Enter your birthday and click submit.
YourBD=Your birthday
Pattern=MM/dd/yyyy
DateError=Please enter the date in the form MM/dd/yyyy.
YouAre=You are
Year=year
Years=years
Older=older than Duke!
Younger=younger than Duke!
SameAge= the same age as Duke!
Submit=Submit
Back=Back
```

   These messages will be referenced from the JSP pages.

8. **Save the file by selecting File → Save from the menu.**
9. **To add the Spanish translations of the messages, copy the properties file `WebMessages_es.properties` from**

`<INSTALL>/firstcup/example/firstcup/firstcup-war/src/java/com/sun/firstcup/web` **to** `<INSTALL>/firstcup/myexample/firstcup/firstcup-war/src/java/com/sun/firstcup/web`**.**

You can create multiple properties files, each with a set of messages for a different locale. By storing localized static text and messages in resource bundles, you don't need to create a separate set of JSP pages for each locale.

10 **Refresh the files in the application by selecting File → Refresh All Files from the menu bar.**

Configuring the Resource Bundle in the Configuration File

To make the resource bundle available to the application, you need to configure it in the configuration file, by performing the following task.

▼ Configuring the Resource Bundle

1 **Expand the** `firstcup-war` **module in the Projects pane.**

2 **Open the folders Web Pages → WEB-INF.**

3 **Double-click** `faces-config.xml`**.**

4 **Before the end-tag of the** `faces-config` **element, add the following elements:**

```
<application>
    <resource-bundle>
        <base-name>com.sun.firstcup.web.WebMessages</base-name>
        <var>bundle</var>
    </resource-bundle>
    <locale-config>
        <default-locale>en</default-locale>
        <supported-locale>es</supported-locale>
    </locale-config>
</application>
```

The `base-name` element of the `resource-bundle` element identifies the fully-qualified class name of the resource bundle. The `var` element identifies the name by which the JSP pages will reference the resource bundle. The `locale-config` element identifies the locales supported by the resource bundle.

5 **Save the file by selecting File → Save from the menu bar.**

Adding a Dependency on the Enterprise Bean Module

The `firstcup-war` module depends on some classes in the `firstcup-ejb` module. You need to tell NetBeans IDE that this dependency exists by performing the following task.

▼ Adding a Dependency on the Enterprise Bean Module

1. **Right-click the `firstcup-war` module in the Projects pane.**
2. **Select Properties.**
3. **Select Libraries from the Categories pane.**
4. **Select Add Project.**
5. **Navigate to `<INSTALL>/myexample/firstcup`.**
6. **Select the `firstcup-ejb` project.**
7. **Click Add Project JAR Files.**
8. **Make sure Build Required Projects is selected.**
9. **Click OK.**

Creating the `DukesBDay` Managed Bean Class

The `DukesBDay` JavaBeans component is a backing bean. A backing bean is a JavaServer Faces managed bean that acts as a temporary data storage for the values of the components included on a particular JavaServer Faces page. A managed bean is a JavaBeans component that a JavaServer Faces application instantiates and stores in scope. The section following this one describes more about managed beans and how to configure them.

This section describes how to create the `DukesBDay` class. To create the class you need to do the following:

- Create an empty class.
- Decorate the bean with a web service reference and an enterprise bean reference.
- Add a property that accesses Duke's current age from the web service.
- Add a property that accesses the user's current birth date.
- Add a property that accesses the age difference from the `DukesBirthdayBean` enterprise bean.
- Add a property that acceses the absolute value of the age difference.

Creating an Empty Class

1. **Right-click the `firstcup-war` module in the Projects pane.**
2. **Select New → Java Class**
3. **Enter `DukesBDay` in the Class Name field.**
4. **Enter `com.sun.firstcup.web` in the Package field.**
5. **Click Finish.**

Adding an Enterprise Bean Reference

- **Directly after the class declaration, add a private variable to hold a reference to the enterprise bean using the `@EJB` annotation:**

```
@EJB
private DukesBirthdayRemote dukesBirthday;
```

Adding Properties to the Bean

During this task, you will add the following properties to the `DukesBDay` bean:

- `age` for getting Duke's age from the web service.
- `yourBD` to hold the user's birth date.
- `ageDiff` to get the age difference from the enterprise bean.
- `absAgeDiff` to hold the absolute value of the age difference.

1. **After the `dukesBirthday` variable declaration, add the following `private` variables:**

```
private int age;
private Date yourBD;
private int ageDiff;
private int absAgeDiff;
```

2. **Initialize the variables by adding the following inside the default constructor:**

```
age = -1;
yourBD = null;
ageDiff = -1;
absAgeDiff = -1;
```

Generating the Accessor Methods for the Properties

1. **Right-click in the editor window.**
2. **Select Refactor→Encapsulate Fields from the popup window.**

3. **In the Encapsulate Fields dialog, de-select the `getDukesBirthday` checkbox and the `setDukesBirthday` checkbox.**

4. **Click Next.**

5. **Click Do Refactoring in the Refactoring pane at the bottom of the IDE window.**

   You should now see two methods for each property, one to set the value and one to get the value of the variable for that property.

Getting Duke's Current Age

While performing this task, you will add some code to the `getAge` method to access Duke's current age from the web service.

1. **Expand the `firstcup-war` module.**

2. **Expand the Web Service References node in the `firstcup-war` module.**

3. **From within the Web Service References node, go to `DukesAgeService`→`DukesAgeService`→`DukesAgePort`→`getDukesAge`.**

4. **Drag the `getDukesAge` operation from inside the `DukesAgePort` node to the `getAge` method in `DukesBDay.java` in the editor, directly before the `return age;` statement so that your `getAge` method looks like this:**

```
public int getAge() {

    try { // Call Web Service Operation
        com.sun.firstcup.webservice.DukesAgeService service =
                    new com.sun.firstcup.webservice.DukesAgeService();
        com.sun.firstcup.webservice.DukesAge port =
                    service.getDukesAgePort();
        // TODO process result here
        int result = port.getDukesAge();
        System.out.println("Result = "+result);
    } catch (Exception ex) {
        // TODO handle custom exceptions here
    }
    return age;
}
```

5. **In the `getAge` method of `DukesBDay`, change the line `int result = port.getDukesAge();` so that the `age` variable rather than the `result` variable is set to the result of the call to `port.getDukesAge`:**

```
age = port.getDukesAge();
```

6. **Remove the following line from the `getAge` method.**

```
System.out.println(“Result = “+result);
```

▼ Getting the Age Difference from the `DukesBirthdayBean` Enterprise Bean

During this task, you will add code to the `getAgeDiff` method to get the difference in age between the user's age and Duke's age from the EJB and to set the `absAgeDiff` variable to the absolute value of the age difference.

- **Inside the `getAgeDiff` method, directly before the `return` statement, add the following code:**

```
ageDiff = dukesBirthday.getAgeDifference(yourBD);
setAbsAgeDiff(Math.abs(ageDiff));
```

▼ Adding Import Statements

1. **Right-click in the editor window.**
2. **Select Fix Imports from the popup menu.**
3. **Select `java.util.Date` as the fully qualified name of the `Date` class.**
4. **Click OK.**

▼ Saving `DukesBDay`

- **Select File → Save.**

Configuring the `DukesBDay` Bean in the Configuration File

JavaServer Faces technology allows you to use the configuration file to initialize, configure, and store managed beans in one of the following scopes:

- Request, which begins when the user submits a page and ends when the response is rendered.
- Session, which begins when a user first accesses a page and ends when the user's session ends, such as when the user's session times out.
- Application, which lasts until the server stops the application.

Once a bean is configured, a JavaServer Faces page can create and access the bean. In this section, you'll configure the `DukesBDay` managed bean.

▼ Configuring the `DukesBDay` Managed Bean

1. **Expand the `firstcup-war` module in the Projects pane.**
2. **Open the folders Web Pages → WEB-INF.**
3. **Double-click `faces-config.xml`.**
4. **Right-click in the editor window and select JavaServer Faces → Add Managed Bean.**
5. **In the Add Managed Bean dialog, enter `dukesBDay` in the Bean Name field.**
6. **Enter `com.sun.firstcup.web.DukesBDay` in the Bean Class field.**
7. **Select `session` from the scope menu.**
8. **Enter `DukesBDay backing bean` in the Bean Description field.**
9. **Click Add.**

   You should now see the following in the `faces-config.xml` file:

```
<managed-bean>
  <description>DukesBDay backing bean</description>
  <managed-bean-name>dukesBDay</managed-bean-name>
  <managed-bean-class>com.sun.firstcup.web.DukesBDay</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

   The `managed-bean-name` element is the name by which the JavaServer Faces pages will refer to the bean. The `managed-bean-class` element is the fully-qualified class name of the bean. The `managed-bean-scope` element is the scope in which the bean is saved. You specify session scope because you need an instance of the bean to be available for the entire session so that all pages can access the values held by the bean.

10. **Select File → Save to save the file.**

Creating the `greeting.jsp` Page

The `greeting.jsp` page includes the welcome message, displays Duke's current age, and accepts the user's birth date. To create the page, you need to perform the following tasks:

- Create an empty JSP page.
- Declare the JavaServer Faces tag libraries.
- Add an `f:view` and an `h:form` tag.
- Add the output label components to display localized messages and text.
- Add an input component to accept the birth date.

- Register a converter on the input component to convert the date to the proper type.
- Add a custom error message to display if conversion fails.
- Add a button component so that the user can submit the page.

Creating an Empty JSP Page

To create an empty JSP page, do the following:

1. Right-click the `firstcup-war` module in the Projects pane.
2. Select New → JSP from the popup menu.
3. Enter `greeting` in the JSP File Name field.
4. Click Finish.
5. In the `greeting.jsp` file, after the `html` tag, replace any `head` tag that is already in the page and add the following one, which defines the content type:

```
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <title>Firstcup Greeting Page</title>
</head>
```

6. Remove the beginning and ending `body` tags.
7. Remove `<h1>JSP Page</h1>` from the file.
8. Select File → Save to save the file.

Declaring the JavaServer Faces Tag Libraries

JavaServer Faces technology defines JSP custom tags for representing UI components, converters, validators, and event listeners on a JSP page. A custom tag provides a way to reference some Java logic from a JSP page, thereby freeing page authors from including the logic in the page and allowing them to reuse the logic by adding the same tag in any of their JSP pages. See the Custom Tags chapter of the Java EE tutorial (`http://download.oracle.com/javaee/5/tutorial/doc/`) for more information.

Custom tags are defined in tag libraries. JavaServer Faces technology defines two tag libraries. The HTML Render Kit library defines all of the standard UI component tags. Each instance of a component tag used on a page represents a corresponding stateful component object on the server. The core tag library defines tags for representing type converters, data validators, event listeners, and other objects and functions. Similarly to the component tags, each instance of a converter, validator or listener tag represents an object on the server.

To use the tags in a page, you need to declare the tag libraries.

To declare the tag libraries, do the following:

1. After the ending `head` tag in `greeting.jsp`, add a `taglib` declaration for the HTML render kit tag library:

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

When using a particular tag from this tag library, a page author must use the prefix `h:` to specify in which tag library the tag is defined. The `uri` resolves to the tag library definition.

2. After the `taglib` declaration from the previous step, add a `taglib` declaration for the core tag library:

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

Again, when the page author uses a tag from this library, he or she must use the `f:` prefix to specify that the tag is defined in the core tag library.

Adding the `f:view` and `f:form` Tags

A JavaServer Faces page is represented by a tree of UI components. When a JavaServer Faces implementation processes the page during the page's life cycle, it will traverse the tree of components and perform such tasks as converting the values of the components to the proper type, validate the components' values, and store the components' values in backing beans.

Every JSP page that uses JavaServer Faces technology must include a `view` tag, which is defined in the core tag library. The `view` tag represents the root of the component tree. All JavaServer Faces component tags must be included inside the `view` tag.

A typical JavaServer Faces page also includes a form in which the user enters some data in input fields and clicks a button to submit the form. If the page contains a form, it must also have a `form` tag. All components that are part of the form submission must be inside the `form` tag.

To add the view and form tags to the page:

1. After the tag library declarations, add beginning and ending `f:view` tags:

```
<f:view>
</f:view>
```

2. In between the beginning and ending `f:view` tags, add beginning and ending `h:form` tags:

```
<f:view>
   <h:form>
   </h:form>
</f:view>
```

Adding Output Labels to Display Read-only Content

One of the more commonly used component tags is the `outputText` tag, which represents a read-only component that only displays content. The `greeting.jsp` page contains `outputText` tags that display the welcome message, the instructions, the label for the input component, and Duke's current age.

The `outputText` tags that display the localized messages reference the messages from the resource bundle that you configured previously. The `outputText` tag that displays Duke's age references the age from the `age` property of `DukesBDay` bean.

To reference values for display, the `outputText` tags use the expression language syntax defined by the unified expression language. The expression language syntax allows you to use the `.` or `[]` notation to reference objects and their properties. The name by which an expression references an object is defined in the configuration file. For example, an expression must refer to the `WebMessages` resource bundle with the name `bundle`. For more information on the expression language, see the Expression Language section of the Introduction to JavaServer Pages chapter of the Java EE tutorial (`http://download.oracle.com/javaee/5/tutorial/doc/`).

While doing the following exercise, you might want to refer to the `WebMessages.properties` file, the `DukesBDay` bean, and the `faces-config.xml` file that you created earlier.

To reference the read-only values displayed on the greeting page using `outputText` tags and expressions, add the following tags:

1. In between the beginning and ending `h:form` tags, add an `outputText` tag that displays "Hi. I'm Duke. Let's see who's older - you or me." and add some `h2` tags around it.

   ```
   <h2><h:outputText value="#{bundle.Welcome}"/></h2>
   ```

   The `value` attribute of the `outputText` tag specifies the text to display. In this case, the `value` attribute uses an expression to reference the message stored in the resource bundle, called `bundle`, under the key, `Welcome`.
2. After the `outputText` tag you added in the previous step, add an `outputText` tag that displays "Duke is". Add an extra space between the closing curly brace and the double quote of the expression so that there will be a space between "Duke Is" and the text following it on the page:

   ```
   <h:outputText value="#{bundle.DukeIs} "/>
   ```
3. After the `outputText` tag you added in the previous step, add an `outputText` tag that displays Duke's age:

   ```
   <h:outputText value="#{dukesBDay.age}"/>
   ```

   The `dukesBDay` part of the expression refers to the `DukesBDay` bean. The `age` part of the expression refers to the `age` property of `DukesBDay` bean.
4. After the `outputText` tag you added in the previous step, add a paragraph tag and another `outputText` tag to display the instructions for filling out the form.

   ```
   <p><h:outputText value="#{bundle.Instructions}"/>
   ```
5. After the `outputText` tag you added in the previous step, add another paragraph tag and an `outputText` tag that displays the label for the input component. Add an extra space at the end of this expression too:

   ```
   <p><h:outputText value="#{bundle.YourBD} "/>
   ```

Adding an Input Component to Accept the User's Birth Date

Another commonly used component tag is the `inputText` tag. This tag represents a text field, which accepts input from the user. The `inputText` tag also uses expressions to reference values. However, the `inputText` tag can use the expressions to set values as well as get them. The `inputText` tag on the `greeting.jsp` page accepts the user's birth date and sets this value into the `yourBD` property of the `DukesBDay` bean.

To add the `inputText` tag, do the following:

1. Add a beginning and ending `inputText` tag, right after the previous `outputText` tag that you added and give it an ID of `userBirthday`:

   ```
   <h:inputText id="userBirthday">
   </h:inputText>
   ```

   Later, you'll register a converter on this component. Any error messages displayed as a result of conversion failing will reference this `id` attribute.

2. Add to the `inputText` tag a `value` attribute that references the `yourBD` property of `DukesBDay`:

   ```
   <h:inputText id="userBirthday"
      value="#{dukesBDay.yourBD}">
   </h:inputText>
   ```

Registering a Converter on the Input Component

JavaServer Faces technology includes a set of standard converters and validators that you can register on components in order to convert and validate their data. You register a converter or validator by nesting the tag representing the converter or validator inside the input component's tag.

To add a converter to the input component discussed in the previous section, do the following:

1. Add an `f:convertDateTime` tag inside the `inputText` tag and specify the acceptable format of the input:

   ```
   <h:inputText ...>
      <f:convertDateTime pattern="MM/dd/yyyy" />
   </h:inputText>
   ```

   If the user does not enter something that can be converted to a `Date` type or the date the user entered is not of the specified pattern then a conversion error occurs. The next section describes how to create an error message and display it.

2. After the ending `inputText` tag, add an `outputText` tag that tells the user what the acceptable pattern is. Add an extra space in between the double-quote and the pound sign of the expression so that there will be space between the text field and the text displayed by this `outputText` tag:

   ```
   <h:outputText value=" #{bundle.Pattern}"/>
   ```

Adding an Error Message to Display if Conversion Fails

To specify an error message to be displayed if conversion fails, do the following:

1. Add a `converterMessage` attribute to the `inputText` tag and use it to specify the error message:

   ```
   <h:inputText id="userBirthday"
      value="#{dukesBDay.yourBD}"
      converterMessage="#{bundle.DateError}">
   </h:inputText>
   ```

   If the conversion of the user's input fails, the following message, stored with the `DateError` key in the resource bundle will display on the page:

   ```
   Please enter the date in the format MM/dd/yyyy.
   ```

2. Add a paragraph tag followed by a `message` tag to indicate where the error message must appear on the page, right before the ending `form` tag:

   ```
   <p><h:message for="userBirthday" style="color:red"/>
   </h:form>
   ```

   The `style` attribute indicates the style of the error message. In this case, the error message text will be red. The `for` attribute references the ID of the component that generated the error.

Adding a Button for Submitting the Form

In order for the user to submit the form with his or her birth date to the server, he or she needs to click a button or a hyperlink, thereby causing a form submit.

To add a button to the page, do the following:

1. Right before the message tag you added in the previous section, add the following paragraph tag and `commandButton` tag:

   ```
   <p><h:commandButton value="#{bundle.Submit}" action="success"/>
   ```

2. Save the file by selecting File → Save from the menu.

The `value` attribute indicates the text that is displayed on the button. Again, you could use an expression to refer to this value. The `action` attribute indicates a logical outcome, which tells the page navigation system which page to display next. The following section explains the navigation system.

Defining Page Navigation

JavaServer Faces technology supports a powerful rule-based system for defining the flow of pages in an application. These rules are defined in the configuration file. To define the page navigation rules for the `firstcup` application, do the following:

▼ Defining Page Navigation Rules

1. **Expand the `firstcup-war` module in the Projects tree.**
2. **Open the folders Web Pages → WEB-INF.**
3. **Double-click `faces-config.xml`.**
4. **Right-click in the editor window and select JavaServer Faces → Add Navigation Rule.**
5. **In the Add Navigation Rule dialog, enter `/greeting.jsp` in the Rule from View field.**
6. **Click Add.**
7. **Repeat steps 4 through 6 except enter `/response.jsp` in the Rule from View field.**
8. **Right-click in the editor again and select JavaServer Faces → Add Navigation Case.**
9. **Select `/greeting.jsp` from the From View menu.**
10. **Enter success in the From Outcome field.**
11. **Select `/response.jsp` from the To View menu.**
12. **Click Add.**
13. **Repeat steps 8 through 12, except select `/response.jsp` from the From View menu and select `/greeting.jsp` from the To View menu.**

    You should now see two navigation rules, one that defines how to navigate from `greeting.jsp` to `response.jsp`, and another one that defines how to navigate from `response.jsp` to `greeting.jsp`. The `from-view-id` tag indicates what the current page is. The `to-view-id` tag indicates what page to go to next.

    Recall from the section on adding a button that the `commandButton` tag has an `action` attribute that indicates a logical outcome string. When the button is clicked, the navigation system matches the current page ID and the logical outcome to a navigation rule. When it finds a match, it will navigate to the page that the rule specifies with its `to-view-id` tag.
14. **Save the file by selecting File → Save from the menu bar.**

Creating the `response.jsp` Page

To create the `response.jsp` page, do the following:

1. Right-click the `firstcup-war` module.
2. Select New → JSP from the popup menu.
3. Enter `response` in the JSP File Name field.
4. Click Finish.
5. In the `response.jsp` file, after the `html` tag, replace any `head` tag in the file with the following `head` tag, which defines the content type:

```
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <title>Response Page</title>
</head>
```

6. Remove the beginning and ending body tags.
7. Remove `<h1>JSP Page</h2>` from the file.
8. Add the following `taglib` declarations right after the ending `head` tag.

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

9. After the `taglib` declarations, add the beginning and ending `f:view` tags.
10. In between the beginning and ending `f:view` tags, add the beginning and ending `h:form` tags.
11. In between the beginning and ending `h:form` tags, add the following `outputText` tags:

```
<h2><h:outputText value="#{bundle.YouAre} "/>
<h:outputText value="#{bundle.SameAge}" rendered="#{dukesBDay.ageDiff == 0}"/>

<h:outputText value="#{dukesBDay.absAgeDiff}" rendered="#{dukesBDay.ageDiff < 0}"/>
<h:outputText value=" #{bundle.Year} " rendered="#{dukesBDay.ageDiff == -1}"/>
<h:outputText value=" #{bundle.Years} " rendered="#{dukesBDay.ageDiff < -1}"/>
<h:outputText value="#{bundle.Younger}" rendered="#{dukesBDay.ageDiff < 0}"/>

<h:outputText value="#{dukesBDay.absAgeDiff}" rendered="#{dukesBDay.ageDiff > 0}"/>
<h:outputText value=" #{bundle.Year} " rendered="#{dukesBDay.ageDiff == 1}"/>
<h:outputText value=" #{bundle.Years} " rendered="#{dukesBDay.ageDiff > 1}"/>
<h:outputText value="#{bundle.Older}" rendered="#{dukesBDay.ageDiff > 0}"/>
```

The `outputText` tags on the `response.jsp` page use the `rendered` attribute to control which one of the tags will actually have their content rendered. Let's take a look at the set of `outputText` tags:

The first tag will yield the text “You are”.

The `rendered` attribute of the second `outputText` tag references an expression that tests if the `ageDiff` property of `DukesBDay` is equal to zero. If this expression returns `true` then the message referenced by the tag's `value` attribute will be rendered. This message is “the same age as Duke”.

The `value` attribute of the third tag references the `absAgeDiff` property of `DukesBDay`. This property holds the absolute value of the age difference. The expressions referenced by the `rendered` attributes of the fourth and fifth tags test whether or not the age difference is equal to -1 or less than –1 to determine whether the age difference is only one year or more than one year so that "year" or "years" is rendered appropriately. The sixth tag tests if `ageDiff` is less than zero. If this is true then the absolute value of the age difference is rendered along with the message "younger than Duke!". So an example message that can be rendered by this group of tags is:

```
You are 10 years younger than Duke!
```

The last group of tags work similarly to the previous set of tags, except that their `rendered` attributes test for age differences greater than zero, indicating that the user is older than Duke.

12. Before the ending `</h:form>` tag, add the following paragraph tag and `commandButton` tag to render a button used for returning to the `greeting.jsp` page.

```
<p><h:commandButton id="back" value="#{bundle.Back}" action="success"/>
```

13. Save the file by selecting File → Save.

Building, Packaging, Deploying, and Running the `firstcup` Enterprise Application

In this section, you will build the `DukesBirthdayBean` and the `firstcup` web client, package them into an EAR file, deploy the EAR file to the server, and run the application.

▼ Preparing the Deployment Descriptor

In this task, you will remove some extra parameters from the deployment descriptor of the `firstcup-war` module. NetBeans IDE generates these parameters, but you don't need them in this example.

1 **Expand the `firstcup-war` module in the Projects pane.**

2 **Expand the Configuration Files directory.**

3 **Double-click `web.xml`.**

4 **Click General at the top of the editor window.**

5 **Select the plus sign next to Context Parameters.**

6 **From the table of context parameters, select the first context parameter in the table and click Remove.**

7. **Repeat step 6 until all context parameters are removed.**
8. **Click XML at the top of the editor window.**
9. **Remove the entire `welcome-file-list` element and all its contents.**
10. **Save the file by selecting File→Save from the menu bar.**

Building and Packaging the `firstcup` Enterprise Application

While performing this task, you'll build and package the `DukesBirthdayBean` enterprise bean and the `firstcup` web client into an EAR file, `firstcup.ear`, in the `dist` directory.

1. **Select `firstcup` in the Projects tab.**
2. **Right-click `firstcup` and select Build Project.**

Deploying the `firstcup` Enterprise Application

While performing the previous task, you packaged the `DukesBirthdayBean` enterprise bean and `firstcup` web client into the `firstcup.ear` file. Now you'll deploy them to the Application Server.

1. **Select `firstcup` in the Projects tab.**
2. **Right-click `firstcup` and select Deploy Project.**

Running the `firstcup` Application

This section describes how to run the `firstcup` application.

1. **Launch an internet browser.**
2. **Enter the following URL in the address field of the browser:**

   ```
   http://localhost:8080/firstcup
   ```

3. **Enter your birth date in the Your birthday text field. Make sure you use the date pattern specified on the page: MM/dd/yyyy.**
4. **Click Submit.**
5. **After the `response.jsp` page is launched, click Back to return to the `greeting.jsp` page.**

◆ ◆ ◆ CHAPTER 4

Building, Packaging, and Deploying and Running the Example Application

This chapter details how to build, deploy, and run the pre-coded example application included in the `firstcup` download bundle using the Ant build tool. If you want to build, deploy, and run the example that you built in the previous chapter, please use NetBeans IDE rather than the Ant build tool by following the instructions in the previous chapter.

Building and Packaging the Example Application

This section describes how to build the example application.

Building and Packaging the Example Application Using Ant

▼ Building and Packaging `firstcup-dukes-age` Using Ant

This task builds `firstcup-dukes-age` and creates a WAR file, `firstcup-dukes-age.war`, in the `dist` directory.

1. **Verify that you have added the Ant build tool that comes with the Application Server to your path. The location is the `lib/ant/bin` directory of your Application Server installation.**
2. **In a terminal, go to `<INSTALL>/firstcup/example/firstcup-dukes-age`.**
3. **Enter the following command:**

   ```
   ant
   ```

   You should see `Build Successful` when the command finishes.

▼ Deploying `firstcup-dukes-age.war` Using Ant

This task gives instructions on deploying `firstcup-dukes-age.war` using Ant.

1. **Start the Application Server if you haven't already. Refer to your Application Server documentation if you are not sure how to start the server.**
2. **In a terminal go to** `<INSTALL>/firstcup/example/firstcup-dukes-age`**.**
3. **Enter the following command:**

   ```
   ant deploy
   ```

   You should see `Build Successful` when the command finishes.

▼ Building and Packaging `DukesBirthdayBean` and `firstcup` Using Ant

1. **In a terminal, go to** `<INSTALL>/firstcup/example/firstcup`**.**
2. **Enter the following command:**

   ```
   ant
   ```

   You should see `Build Successful` when the command finishes.

Deploying the Example Application

This section describes how to deploy the example application.

Deploying the Enterprise Bean and Web Client

The `DukesBirthdayBean` enterprise bean and `firstcup` web client were packaged in `firstcup.ear`. Now you'll deploy them to the Application Server.

▼ Deploying `firstcup.ear` Using Ant

This task gives instructions on deploying `firstcup.ear` using Ant.

1. **In a terminal go to** `<INSTALL>/firstcup/example/firstcup`**.**
2. **Enter the following command:**

   ```
   ant deploy
   ```

   You should see `Build Successful` when the command finishes.

Running the Web Client

This section describes how to run the `firstcup` application.

To run the application, do the following.

▼ Running the `firstcup` Application

1. **Launch a web browser.**
2. **Enter the following URL in the address field of the browser:**
   `http://localhost:8080/firstcup`
3. **Enter your birth date in the Your birthday text field. Make sure you use the date pattern specified on the page: MM/dd/yyyy.**
4. **Click Submit.**
5. **After the `response.jsp` page is launched, click Back to return to the `greeting.jsp` page.**

Undeploying the Application

▼ Undeploying with Ant

1. **Go to the `<INSTALL>/firstcup/example/firstcup` directory.**
2. **Run `ant undeploy`.**
3. **Go to the `<INSTALL>/firstcup/example/firstcup-dukes-age` directory.**
4. **Run `ant undeploy`.**

▼ Undeploying with `asadmin`

1. **Run `asadmin undeploy firstcup-dukes-age`.**
2. **Run `asadmin undeploy firstcup`.**

◆ ◆ ◆ CHAPTER 5

Next Steps

This chapter points the user at additional resources for learning more about enterprise application architecture, the Java EE platform, and the Sun Java System Application Server.

The Java EE Tutorial

The Java EE Tutorial (`http://download.oracle.com/javaee/5/tutorial/doc/`) documents the technologies that make up the Java EE platform. The Java EE Tutorial describes each piece of the platform in detail, and includes code examples that demonstrate how to use each piece of the platform.

More Information on the Java EE Platform

For more information on the Java EE platform, see these resources:

- The Java EE site (`http://www.oracle.com/technetwork/java/javaee/overview/index.html`)
- GlassFish (`https://glassfish.dev.java.net`)
- The Aquarium (`http://blogs.sun.com/theaquarium`)

Java EE Servers

Java EE servers are application servers that implement the Java EE platform technologies.

The Sun Java System Application Server

The Application Server is the reference implementation of the Java EE platform APIs.

GlassFish Project

The GlassFish (`https://glassfish.dev.java.net`) project is the open-source basis for the Application Server.