Oracle®

Your First Cup: An Introduction to the Java EE Platform

Release 7 for Java Platform, Enterprise Edition

E39032-01

June 2013

ORACLE®

Your First Cup: An Introduction to the Java EE Platform, Volume 1, Release 7 for Java Platform, Enterprise Edition

E39032-01

Contents

3 Creating Your First Java EE Application

4 Creating Your Second Web Application

5 Next Steps

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.

Audience

This tutorial is intended for novice Java EE developers. You should be familiar with the Java programming language, particularly the features introduced in Java Platform, Standard Edition 7. While familiarity with enterprise development and Java EE technologies is helpful, this tutorial assumes you are new to developing Java EE applications.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

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 7 Tutorial
  (http://docs.oracle.com/javaee/7/tutorial/doc/home.htm)

- The Oracle GlassFish Server documentation set
- The NetBeans IDE documentation set

Conventions

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

Convention	Meaning	Example
Boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text.	From the **File** menu, select **New Project**. A **cache** is a copy that is stored locally.
`Monospace`	Monospace type indicates the names of files and directories, commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.	Edit your `.login` file. Use `ls a` to list all files. `machine_name% you have mail.`
Italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.	The command to remove a file is `rm` *filename*. Read Chapter 6 in the *User's Guide*. Do *not* save the file.

1

Introduction

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

1.1 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

1.2 Requirements for This Tutorial: A Checklist

To complete this tutorial, you need to:

- Get the Java EE 7 Software Development Kit.

- Get NetBeans IDE and all necessary plugins.

- Configure your environment.

- Get the latest updates to the tutorial bundle.

For up-to-the-minute information on which versions of the required software are compatible with this tutorial, see the First Cup compatibility page (`http://java.net/projects/firstcup/pages/FirstCupCompatibility`).

1.2.1 Getting the Java EE 7 SDK

To get the Java EE 7 SDK, go to `http://www.oracle.com/technetwork/java/javaee/downloads/`.

1.2.2 Getting NetBeans IDE

To get NetBeans IDE, go to `http://www.netbeans.org/downloads/`, and download the Java EE distribution.

1.2.3 Configuring Your Environment

Once you have all the necessary downloads, you must configure NetBeans IDE and get the latest tutorial updates.

1.2.3.1 Add GlassFish Server as a Server in NetBeans IDE

To run this tutorial in NetBeans IDE, you must register your GlassFish Server installation as a NetBeans server instance. Follow these instructions to register the GlassFish Server in NetBeans IDE.

1. From the **Tools** menu, select **Servers**.

2. In the Servers dialog, click **Add Server**.

3. Under **Choose Server**, select **GlassFish Server 4** and click **Next**.

4. Under **Server Location**, browse to or enter the location of your GlassFish Server installation.

5. Click **Next**.

6. Under **Domain Location**, do one of the following:

   - Use the list to select an existing domain.

   - Enter the path to the domain directly in the field.

   - Enter the name of a new domain to create.

   Normally, you will select the default domain, `domain1`.

7. Click **Finish**.

1.2.4 Getting the Latest Updates to the Tutorial

Check for any updates to this tutorial by using the Update Center included with the Java EE 7 SDK.

1.2.4.1 Update the Tutorial Through the Update Center

Open the Update Center and check for any updates to the tutorial.

1. In NetBeans IDE, select the **Services** tab and expand the **Servers** node.

2. Right-click the GlassFish Server instance and select **View Update Center** to display the Update Tool.

3. In the tree, select **Available Updates** to display a list of updated packages.

4. Look for updates to the First Cup for Java EE 7 (`javaee-firstcup-tutorial`) package.

5. If there is an updated version of First Cup, select First Cup 7.0 for Java EE 7 (`javaee-firstcup-tutorial`) and click **Install**.

2

Understanding Java Platform, Enterprise Edition

This chapter outlines the features of Java Platform, Enterprise Edition (Java EE), how it differs from Java Platform, Standard Edition (Java SE), Java Platform, Micro Edition (Java ME), and JavaFX, and the basic concepts behind enterprise application development.

2.1 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.

2.1.1 The Java Programming Language Platforms

There are four platforms of the Java programming language:

- Java Platform, Standard Edition (Java SE)

- Java Platform, Enterprise Edition (Java EE)

- Java Platform, Micro Edition (Java ME)

- JavaFX

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 technology 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.

2.1.1.1 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 technology applications.

2.1.1.2 Java EE

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

2.1.1.3 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 mobile 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 platform services.

2.1.1.4 JavaFX

JavaFX technology is a platform for creating rich internet applications using a lightweight user-interface API. JavaFX applications use hardware-accelerated graphics and media engines to take advantage of higher-performance clients and a modern look-and-feel as well as high-level APIs for connecting to networked data sources. JavaFX applications may be clients of Java EE platform services.

2.2 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.

2.2.1 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.

2.2.1.1 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 standalone application, or other servers, and they run on a different machine from the Java EE server.

2.2.1.2 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.

2.2.1.2.1 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 Java EE Technologies

Technology	Purpose
Servlets	Java programming language classes that dynamically process requests and construct responses, usually for 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 Faces Facelets technology	Facelets applications are a type of JavaServer Faces applications that use XHTML pages rather than JSP pages.
Expression Language	A set of standard tags used in JSP and Facelets pages to refer to Java EE components.
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 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

2.2.1.3 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.

2.2.1.3.1 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-RS RESTful web services
- JAX-WS web service endpoints
- Java Persistence API entities

2.2.1.4 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.

2.2.1.4.1 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 Java EE Connector Architecture
- The Java Transaction API (JTA)

2.3 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 like 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.

2.3.1 Java EE Containers

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

2.3.1.1 The Web Container

The web container is the interface between web components and the web server. A web component can be a servlet, a JavaServer Faces Facelets page, or a JSP 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.

2.3.1.2 The Application Client Container

The application client container is the interface between Java EE application clients, which are special Java SE 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.

2.3.1.3 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.

3

Creating Your First Java EE Application

This chapter gives an overview of the example applications and step-by-step instructions on coding and running the `dukes-age` web service example application.

3.1 Architecture of the Example Applications

The example applications consist of four main components: `DukesAgeResource`, a JAX-RS RESTful web service; `DukesBirthdayBean`, an enterprise bean; `FirstcupUser`, a Java Persistence API entity; and `firstcup-war`, a web application created with JavaServer Faces Facelets technology.

Figure 3–1   Architecture of the First Cup Example Applications
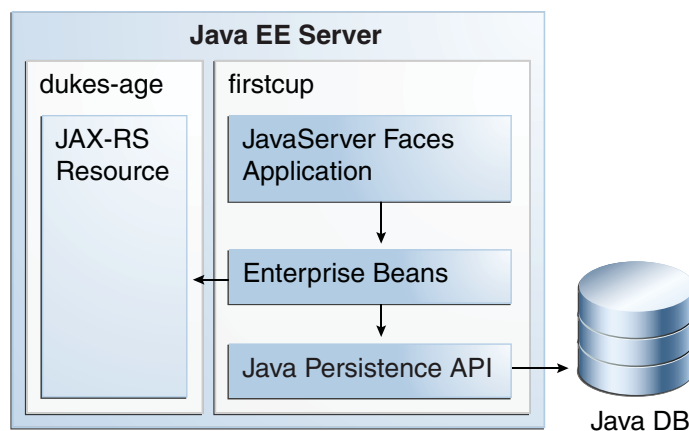
`DukesAgeResource` is a JAX-RS resource that calculates the age of Duke, the Java mascot. Duke was born May 23, 1995, when the first demo of Java technology was publicly released.

`DukesBirthdayBean` is a local, no-interface view stateless session bean that calculates the difference between the user's age and Duke's age, and stores the user-submitted data in a Java Persistence API entity.

`FirstcupUser` is a Java Persistence API entity that represents a particular user's birthday. It is stored in a Java DB database table, and managed by `DukesBirthdayBean`'s business methods.

The `firstcup-war` web application is a JavaServer Faces Facelets application that accesses `DukesAgeResource` 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 and the average age difference of all users.

The `firstcup-war` web application consists of the following:

- `greeting.xhtml`: An XHTML page that uses the JavaServer Faces Facelets tag libraries. Users can type their birth date in a field and submit it for comparison against Duke's birth date.

- `response.xhtml`: A Facelets-enabled XHTML 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.xhtml` page, and displays the average age difference of all users.

- `DukesBDay.java`: A CDI bean that defines properties to hold the user's birth date, uses the JAX-RS Client API to get Duke's current age from the `DukesAgeResource` web service, and calculate the age difference between the user and Duke from the enterprise bean.

- `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. It also specifies `greeting.xhtml` as the welcome file for the application.

- `WebMessages.properties` and `WebMessages_es.properties`: Java programming language properties files that contain the localized strings used in `greeting.xhtml` and `response.xhtml`. By default, the English language strings in `WebMessages.properties` are used, but Spanish language strings are also provided in `WebMessages_es.properties`.

- `DukesBirthdayBean.java`: as described above, the enterprise bean packaged within the `firstcup` application. `DukesBirthdayBean` calculates the difference between the user's birthday and Duke's birthday.

3.1.1 Tiers in the Example Applications

The example applications have one web tier component (the `firstcup` web client), three business tier components (the `DukesAgeResource` web service, the `FirstcupUser` entity, and the `DukesBirthdayBean` enterprise bean), and one enterprise information system (EIS) tier (the data in the Java DB database table). The user's web browser is the client tier component, as it accesses the rest of the application through the web tier.

3.1.2 Java EE Technologies Used in the Example Applications

The `DukesAgeResource` web service is a JAX-RS resource. The `DukesBirthdayBean` enterprise bean is a stateless session bean. The `FirstcupUser` entity is a Java Persistence API entity. The `DukesBDay` CDI managed bean uses the JAX-RS client API to access the `DukesAgeResource` web service. The `firstcup-war` web client is a JavaServer Faces application that runs in the web container of the Java EE server.

3.2 Coding the `dukes-age` Example Application

This section describes how to code the `dukes-age` example application, a web application containing a JAX-RS RESTful web service endpoint.

3.2.1 Getting Started

Before you start coding the example, you need to perform some configuration tasks:

1. Register the server with your NetBeans IDE as described in Configuring Your Environment.

2. Install the Maven archetypes used to create the example applications.

3.2.1.1 Install the Maven Archetypes

Maven archetypes are templates that create the structure of a particular application. There are two archetypes included in the example, `dukes-age-archetype` and `firstcup-war-archetype`. These archetypes create Java EE 7 web applications that you will then edit and deploy.

Before you can create applications based on the archetypes, you must first install the archetypes and supporting projects to your local Maven repository.

1. In NetBeans IDE select File, then Open Project, navigate to *tut-install*/, select `example`, and click Open Project.

2. Right-click the `firstcup` project in the Projects pane and select Build.

3.2.2 Creating the Web Service

The `DukesAgeResource` endpoint is a simple RESTful web service. REST stands for *representational state transfer*, and software architectures that conform to the principles of REST are referred to as *RESTful*. RESTful web services are web-based applications that use the HTTP protocol to access, modify, or delete information contained within a *resource*. A RESTful web service resource is a source of specific information identifiable by a uniform resource identifier (URI), for example `http://example.com/someResource`, and may be manipulated by calling the HTTP protocol's methods, for example `GET` or `POST`.

RESTful web services are often contrasted to SOAP web services (for example, web services created with the JAX-WS API that is part of Java EE 7). Compared to SOAP web services, RESTful web services are simpler, as they use HTTP directly rather than as a transport mechanism for an underlying XML document format, and typically offer better performance.

Web services are designed to be independent of their clients. Typically RESTful web services 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 HTTP-based requests and responses, and do not need to know each other's implementation details. For this reason, `dukes-age` will be developed in its own application module, and deployed separately from the `DukesBirthdayBean` enterprise bean and `firstcup-war` web client. `dukes-age` could be deployed on a completely different machine without affecting the functionality of the `firstcup-war` web client.

3.2.2.1 JAX-RS Resources

`DukesAgeResource` is a JAX-RS resource class that responds to HTTP `GET` requests and returns a `String` representing the age of Duke at the time of the request.

The basic `DukesAgeResource` resouce class is generated from the `dukes-age-archetype` Maven archetype. This class is annotated with the `javax.ws.rs.Path` annotation, which specifies the URL suffix to which the resource will respond. `DukesAgeResource` has a single method, getText, annotated with the `javax.ws.rs.GET` and `javax.ws.rs.Produces` annotations. @GET marks the method as a responder to HTTP `GET` requests, and @Produces specifies the MIME-type of the response sent back from getText to clients. In this case, the MIME-type is `text/plain`.

3.2.2.2 Creating the `dukes-age` Application Using the Maven Archetype

In NetBeans IDE, create a new web project using the `dukes-age-archetype` Maven archetype.

Create the Project in NetBeans

1. From the **File** menu, select **New Project**.

2. Under **Categories**, select **Maven**.

3. Under **Projects**, select **Project from Archetype**.

4. Click **Next**.

5. In the **Search** field, enter `dukes-age`.

6. In the **Known Archetypes** field, select `dukes-age-archetype`.

7. Click **Next**.

8. In the **Project Name** field, enter `dukes-age`.

9. In the **Package** field, enter `firstcup.dukesage.resource`.

10. Click **Finish**.

   You should now see the module you created in the **Projects** tab.

The `dukes-age-archetype` archetype creates the structure of the JAX-RS endpoint application, including:

- the `DukesAgeService` resource class.

- the `glassfish-web.xml` and `web.xml` deployment descriptors.

After you create the basic application structure with the archetype, you will configure how the application will run, implement the functionality of the resource class, and then deploy the application.

Configure the `dukes-age` Web Application

Set the default URL that is brought up in a web browser when you run `dukes-age`.

1. In the **Projects** tab, right-click the `dukes-age` project and select **Properties**.

2. Under **Categories**, click **Run**.

3. Under **Server** select the GlassFish Server instance you configured.

4. Under **Relative URL** enter `/webapi/dukesAge`.

5. Click **OK**.

Implement the `getText` Method

Add code to `DukesAgeResource.getText` that calculates Duke's age at the time of the request. To do this, use the `java.util.Calendar` and `java.util.GregorianCalendar` classes to create an object representing the date May 23, 1995, Duke's birthday. Then create another `Calendar` object representing today's date, and subtract today's year from Duke's birth year. If today's date falls before May 23, subtract a year from this result. Then return the result as a `String` representation.

1. Highlight the current code in `getText` and replace it with the following code:

```
// Create a new Calendar for Duke's birthday
Calendar dukesBirthday = new GregorianCalendar(1995, Calendar.MAY, 23);
// Create a new Calendar for today
Calendar now = GregorianCalendar.getInstance();
```

```
// Subtract today's year from Duke's birth year, 1995
int dukesAge = now.get(Calendar.YEAR) - dukesBirthday.get(Calendar.YEAR);
dukesBirthday.add(Calendar.YEAR, dukesAge);

// If today's date is before May 23, subtract a year from Duke's age
if (now.before(dukesBirthday)) {
    dukesAge--;
}
// Return a String representation of Duke's age
return "" + dukesAge;
```

2. In the editor window, right-click and select **Format**.

3. From the **File** menu, select **Save** to save the file.

3.2.2.3 Building and Deploying the Web Service

Build the JAX-RS web application and deploy it to your GlassFish Server instance.

Build and Deploy the Web Service Endpoint

Compile, package, and deploy `dukes-age.war` to GlassFish Server.

This task gives instructions on deploying `dukes-age.war` in NetBeans IDE.

In the **Projects** tab, right-click `dukes-age` and select **Run**.

After `dukes-age.war` deploys successfully to GlassFish Server a web browser will load the URL of the `DukesAgeResource` path, and you'll see the returned `String` representing Duke's age.

Example 3–1   Output of `DukesAgeResource`

Here's an example of the output of `DukesAgeResource` displayed in a web browser.

```
17
```

At this point, you've successfully created, deployed, and run your first Java EE application. Now you will create a web application that uses this web service data.

4

Creating Your Second Web Application

This chapter gives step-by-step instructions on coding and running the `firstcup-war` web application, which uses the `dukes-age` web service described in Chapter 3, "Creating Your First Java EE Application". The `firstcup-war` web application is a more complicated application, and uses several different Java EE APIs.

The `firstcup-war` example application retrieves, or "consumes," the data from the `dukes-age` web service using the JAX-RS client API. A JavaServer Faces web front-end asks users to enter their birthdays to find out who is older, the user or Duke. This data is stored in a Java DB database table using the Java Persistence API. The business logic, which provides the core functionality of the application, is handled by an enterprise bean.

All the tiers described in Tiered Applications are present in the `firstcup-war` web application. The web or client tier is the JavaServer Faces front-end. The enterprise information systems, or EIS, tier is the Java DB database. The business tier is the enterprise bean.

4.1 Creating the `firstcup-war` Project

The `firstcup-war` web application project consists of the Java Persistence API entity, the enterprise bean, and the JavaServer Faces web front-end.

4.1.1 Create the Web Application Project Using the Archetype

Follow these steps to create a new web application project using the `firstcup-war-archetype` in NetBeans IDE.

1. From the **File** menu, select **New Project**.

2. Under **Projects**, select **Project from Archetype**.

3. Click **Next**.

4. In the **Search** field, enter `firstcup`.

5. In the **Known Archetypes** field, select `firstcup-war-archetype`.

6. Click **Next**.

7. In the **Project Name** field, enter `firstcup-war`.

8. In the **Package** field, enter `firstcup`.

9. Click **Finish**.

   You should now see the module you created in the **Projects** tab.

The `firstcup-war-archetype` archetype creates the structure of the web application, including:

- the basic entity classes.

- the basic enterprise bean classes.

- the basic backing bean classes.

- the basic Facelets XHTML components and views.

- the `glassfish-web.xml`, `web.xml`, and `persistence.xml` deployment descriptors.

After you create the basic application structure with the archetype, you will configure how the application will run, implement the functionality of the classes, implement the Facelets views, and then deploy the application.

4.2 Modifying the Java Persistence API Entity

The Java Persistence API allows you to create and use Java programming language classes that represent data in a database table. A Java Persistence API *entity* is a lightweight, persistent Java programming language object that represents data in a data store. Entities can be created, modified, and removed from the data store by calling the operations of the Java Persistence API *entity manager*. Entities, or the data encapsulated by the persistent fields or properties of a entity, can be queried using the Java Persistence Query Language (JPQL), a language similar to SQL that operates on entities.

In `firstcup-war`, there is a single entity that defines one query.

4.2.1 Edit the Constructor of the `FirstcupUser` Entity

Add code to the constructor for `FirstcupUser`

1. Below the field definitions in the `FirstcupUser` class, add the following code in bold to the constructor:

```
public FirstcupUser(Date date, int difference) {
    Calendar cal = new GregorianCalendar();
    cal.setTime(date);
    birthday = cal;
    ageDifference = difference;
}
```

2. Right-click in the editor window and select **Format**.

4.2.2 Add a Named Query to the `FirstcupUser` Entity

Add a JPQL named query to the `FirstcupUser` entity that returns the average age difference of all `firstcup-war` users.

This query uses the `AVG` aggregate function to return the average of all the values of the `ageDifference` property of the `FirstcupUser` entities.

1. Directly before the class definition, copy and paste in the following code:

```
@NamedQuery(name="findAverageAgeDifferenceOfAllFirstcupUsers",
query="SELECT AVG(u.ageDifference) FROM FirstcupUser u")
```

The `@NamedQuery` annotation appears just before the class definition of the entity, and has two required attributes: `name`, with the unique name for this query; and `query`, the JPQL query definition.

2. Right-click in the editor window and select **Format**.

3. From the **File** menu, select **Save**.

4.3 Modifying the Enterprise Bean

`DukesBirthdayBean` is a *stateless session bean*. Stateless session beans are enterprise 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 an application requires conversational state, use *stateful session beans*.

`DukesBirthdayBean` is a *local enterprise bean* that uses a *no-interface* view, meaning two things. First, a local enterprise bean is only visible within the application in which it is deployed. Second, enterprise beans with a no-interface view do not need a separate business interface that the enterprise bean class implements. The enterprise bean class is the only coding artifact needed to create a local, no-interface enterprise bean.

`DukesBirthdayBean` will be packaged within the same WAR file as the Facelets web front-end.

4.3.1 Implement a Business Method to `DukesBirthdayBean` that Gets the Average Age Difference of `firstcup-war` Users

Add code to a business method to the `DukesBirthdayBean` session bean to call the `findAverageAgeDifferenceOfAllFirstcupUsers` named query in `FirstcupUser` that returns the average age difference of all users.

1. Find the business method called `getAverageAgeDifference` and add the following code in bold by copying and pasting:

```
public Double getAverageAgeDifference() {
    Double avgAgeDiff =
            (Double)
em.createNamedQuery("findAverageAgeDifferenceOfAllFirstcupUsers")
                            .getSingleResult();
    logger.log(Level.INFO, "Average age difference is: {0}",  avgAgeDiff);
    return avgAgeDiff;
}
```

The named query in `FirstcupUser` is called by using the `EntityManager`'s `createNamedQuery` method. Because this query returns a single number, the `getSingleResult` method is called on the returned `Query` object. The query returns a `Double`.

2. Right-click in the editor window and select **Format**.

4.3.2 Implement a Business Method for Calculating the Age Difference Between Duke and the User

Add code to a business method that calculates the difference in age in years between Duke and the user and creates a new `FirstcupUser` entity.

1. Find the `getAgeDifference` business method and add the following code in bold:

```
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.log(Level.INFO, "Raw ageDifference is: {0}",  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--;
                    }

                    // create and store the user's birthday in the database
                    FirstcupUser user = new FirstcupUser(date, ageDifference);
                    em.persist(user);

                    logger.log(Level.INFO, "Final ageDifference is: {0}",  ageDifference);

                    return ageDifference;
                }
```

This method creates the `Calendar` objects used to calculate the difference in age between the user and Duke and performs the actual calculation of the difference in age.

Similar to the `DukesAgeResource.getText` code, `getAgeDifference` subtracts Duke's birthday year from the user's birthday year to get a raw age difference. If Duke's birthday falls before the user's, and the raw difference is more than 0, it subtracts one year from the age difference.

A new `FirstcupUser` entity is created with the user's birthday and age difference, then stored in the Java DB database by calling the `EntityManager`'s `persist` method.

The final age difference is returned as an `int`.

2. Right-click in the editor window and select **Format**.

3. From the **File** menu, choose **Save**.

4.4 Modifying the Web Client

To add the correct functionality to the web client, you need to perform the following tasks:

- Modify the `DukesBDay` managed bean class.
- Modify the Facelets pages.

4.4.1 Modify the `DukesBDay` Managed Bean Class

The `DukesBDay` JavaBeans component is a CDI-enabled 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 modify the `DukesBDay` class.

4.4.1.1 Call the `dukes-age` Web Service to Retrieve Duke's Current Age

During this task, you wil modify the getAge method of DukesBDay to call the dukes-age web service using the JAX-RS Client API. This will retrieve Duke's current age, so it can be compared to the user's age.

1. Find the getAge method and implement its functionality by copying and pasting the following code in bold:

```
public int getAge() {
    try {
        Client client = ClientBuilder.newClient();
        WebTarget target =
client.target("http://localhost:8080/dukes-age/webapi/dukesAge");
        String response = target.request().get(String.class);
        age = Integer.parseInt(response);
    } catch (IllegalArgumentException | NullPointerException |
WebApplicationException ex) {
        logger.severe("processing of HTTP response failed");
    }
    return age;
}
```

2. In the editor window, right-click and select **Format**.

3. From the **File** menu, select **Save**.

4.4.1.2 Get the Age Difference from the `DukesBirthdayBean` Enterprise Bean

During this task, you will modify the processBirthday method to get the difference in age between the user's age and Duke's age from the DukesBirthdayBean EJB, set the absAgeDiff variable to the absolute value of the age difference, and set a result string that will forward the user to the display page.

1. Find the processBirthday method and implement the functionality by copying and pasting the following code in bold:

```
public String processBirthday() {
    this.setAgeDiff(dukesBirthdayBean.getAgeDifference(yourBD));
    logger.log(Level.INFO, "age diff from dukesbday {0}", ageDiff);
    this.setAbsAgeDiff(Math.abs(this.getAgeDiff()));
    logger.log(Level.INFO, "absAgeDiff {0}", absAgeDiff);

this.setAverageAgeDifference(dukesBirthdayBean.getAverageAgeDifference());
    logger.log(Level.INFO, "averageAgeDifference {0}",
averageAgeDifference);
    return "/response.xhtml";
}
```

This method calls the getAgeDifference method of DukesBirthdayBean to get the age difference and store it in the ageDiff property, sets the absolute age difference stored in the absAgeDiff property, and sets the average age difference stored in the averageAgeDifference property. It returns the relative URL of the response page to which the user will be forwarded.

2. In the editor window, right-click and select **Format**.

3. From the **File** menu, select **Save**.

4.4.2 Creating the Facelets Client

The Facelets client consists of a *resource library*, a *composite component*, and two XHTML files.

4.4.2.1 Resource Libraries in `firstcup-war`

A JavaServer Faces resource library is a collection of user-created components collected in a standard location in a web application. Resource libraries are identified according to a *resource identifier*, a string that represents a particular resource within a web application. Resources can be packaged either at the root of the web application or on the web application's classpath.

A resource packaged in the web application root must be in a subdirectory of a `resources` directory at the web application root.

```
resources/resource-identifier
```

A resource packaged in the web application classpath must be in a subdirectory of the `META-INF/resources` directory within a web application.

```
META-INF/resources/resource-identifier
```

Resource identifiers are unique strings that conform to the following format:

```
[locale-prefix/][library-name /][library-version/]resource-name
[/resource-version]
```

Elements of the resource identifier in brackets (`[]`) are optional. A resource name, identifying a particular resource (a file or a graphic, for example), is required. In `firstcup-war`, a resource library with the name `components` is packaged in the web application root, and this library contains one resource, a file called `inputDate.xhtml`. The resource identifier for this resource is therefore `components/inputDate.xhtml`, and it is located in the web application root at `resources/components/inputDate.xhtml`.

4.4.2.2 The `inputDate` Composite Component

A composite component is a set of user-defined JavaServerFaces and Facelets components located in a resource. In `firstcup-war`, the `inputDate.xhtml` resource, located in the `components` resource library, is a composite component that contains tags for reading in a date the user enters in a form. Composite components consist of an *interface* definition and an *implementation*.

The interface definition is specified with the `<cc:interface>` tag to define which attributes are exposed to pages that use the composite component. Attributes are identified with the `<cc:attribute>` tag.

The `inputDate.xhtml` interface definition is as follows. It defines a single attribute, `date`, that must be specified in pages that use the `inputDate` composite component.

```
<cc:interface>
    <cc:attribute name="date" />
</cc:interface>
```

The implementation of the composite component is specified with the `<cc:implementation>` tag. The tags within the `<cc:implementation>` are the actual component tags that will be added to pages that use the composite component. They can be any HTML Render Kit, JavaServer Faces, or Facelets tags. The `#{cc.attrs.attribute-name}` expression is used to get the value of the specified attribute from the page or component that is using the composite component.

The implementation of the `inputDate` composite component is as follows. An HTML input text component will store the entered text into the `date` attribute, accessed by the `#{cc.attrs.date}` expression. A JavaServer Faces `convertDateTime` component will convert the entered text to a date with the form of `MM/dd/yyyy` (04/13/2009, for example).

```
<cc:implementation>
    <h:inputText value="#{cc.attrs.date}">
        <f:convertDateTime pattern="MM/dd/yyyy" />
    </h:inputText>
</cc:implementation>
```

4.4.2.3 Implement the `inputDate` Composite Component

Modify the inputDate composite component in the `components` resource library.

1. In the **Projects** tab, select the `firstcup-war` project.

2. Expand Web Pages, then `resources`, then `components`, and open `inputDate.xhtml`.

3. Add the composite component interface definition between the opening and closing `<cc:interface>` tags in `inputDate.xhtml`:

   ```
   <cc:interface>
       <cc:attribute name="date" />
   </cc:interface>
   ```

4. Add the composite component implementation between the opening and closing `cc:implementation` tags:

   ```
   <cc:implementation>
       <h:inputText value="#{cc.attrs.date}">
           <f:convertDateTime pattern="MM/dd/yyyy" />
       </h:inputText>
   </cc:implementation>
   ```

5. In the editor window, right-click and select **Format**.

6. From the **File** menu, select **Save**.

4.4.2.4 The Facelets Web Interface

The `firstcup-war` web application interface has two XHTML files. The `greeting.xhtml` file displays Duke's current age and the form where the user can enter a birthday. The `response.xhtml` file displays the age difference between the user and Duke.

The `greeting.xhtml` file contains several pieces of the `firstcup-war` application detailed previously. It uses the localized strings contained in `WebMessages.properties` and `WebMessages_es.properties`. It uses the `DukesBDay` managed bean to call both the `DukesAgeResource` JAX-RS web service and the `DukesBirthdayBean` enterprise bean. It uses the `inputDate` composite component to create the input for the user to enter a birthday.

Here's the content of the `greeting.xhtml` file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:fc="http://java.sun.com/jsf/composite/components">
    <h:head>
        <title>Firstcup Greeting Page</title>
    </h:head>
    <h:body>
        <h:form>
            <h2>
```

```
                        <h:outputText value="#{bundle.Welcome}"/>
                </h2>
                <h:outputText value="#{bundle.DukeIs} "/>
                <h:outputText value="#{dukesBDay.age} #{bundle.YearsOldToday}"/>
                <p/>
                <h:outputText value="#{bundle.Instructions}"/>
                <p/>
                <h:outputText value="#{bundle.YourBD} "/>
                <fc:inputDate id="userBirthday" date="#{dukesBDay.yourBD}" />
                <h:outputText value=" #{bundle.Pattern}"/>
                <p/>
                <h:commandButton value="#{bundle.Submit}"
                                action="#{dukesBDay.processBirthday}"/>
                <p/>
                <h:message for="userBirthday" style="color:red"/>
        </h:form>

    </h:body>
</html>
```

The `greeting.xhtml` file uses the HTML Render Kit and the `components` resource library tag libraries. The `components` tag library has a prefix of `fc`, and is used to specify the `inputDate` composite component in the form below. The `<fc:inputDate id="userBirthday" date="#{dukesBDay.yourBD}" />` tag has the required `date` attribute, and it stores the value in the `yourBD` property in the `DukesBDay` managed bean by using the EL expression `#{dukesBDay.yourBD}`.

The localized strings are referred to by using the EL expressions `#{bundle.property-name}`. For example, the `<h:outputText value="#{bundle.Welcome}"/>` tag will display the following string in English locales:

```
Hi. I'm Duke. Let's find out who's older -- you or I.
```

The `<h:commandButton>` tag creates a Submit button and specifies that a successful submission should render the `response.xhtml` file by setting the `action` attribute to `#{dukesBDay.processBirthday}`. The `processBirthday` method returns the value `"/response.xhtml"`. The `action` attribute is used to define navigation rules for forms in Facelets pages.

If the form submission is unsuccessful, a warning message is displayed. This is done with the `<h:message for="userBirthday" style="color:red"/>` tag, which is connected to the `inputDate` composite component with the id `userBirthday`. That is, if there's an error with the input of the `inputDate` component, a warning message is displayed.

The `response.xhtml` file displays the age difference between the user and Duke and the average age difference of all users so far. Different strings are displayed based on whether the user is the same age, younger, or older than Duke. The text can be displayed or not based on the conditions specified by the `rendered` attribute of the `<h:outputText>` tag. The conditions used in the `rendered` attribute are EL language alternatives to the Java programming language conditional operators to allow XML parsing of the XHTML file.

Table 4–1   Conditional Operator EL Language Alternatives

Logical Condition	Java Programming Language Conditional Operator	EL Language Alternative
AND	&&	&&
EQUALS	==	==

Table 4–1   (Cont.) Conditional Operator EL Language Alternatives

Logical Condition	Java Programming Language Conditional Operator	EL Language Alternative
LESS THAN	<	lt
GREATER THAN	>	gt

Here's the content of the `response.xhtml` file.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Response Page</title>
    </h:head>
    <h:body>
        <h:form>
            <h:outputText value="#{bundle.YouAre} "/>
            <h:outputText value="#{bundle.SameAge}"
                        rendered="#{dukesBDay.ageDiff == 0}"/>
            <h:outputText value="#{dukesBDay.absAgeDiff}"
                        rendered="#{dukesBDay.ageDiff lt 0}"/>
            <h:outputText value=" #{bundle.Year} "
                        rendered="#{dukesBDay.ageDiff == -1}"/>
            <h:outputText value=" #{bundle.Years} "
                        rendered="#{dukesBDay.ageDiff lt -1}"/>
            <h:outputText value="#{bundle.Younger}"
                        rendered="#{dukesBDay.ageDiff lt 0}"/>
            <h:outputText value="#{dukesBDay.absAgeDiff}"
                        rendered="#{dukesBDay.ageDiff gt 0}"/>
            <h:outputText value=" #{bundle.Year} "
                        rendered="#{dukesBDay.ageDiff == 1}"/>
            <h:outputText value=" #{bundle.Years} "
                        rendered="#{dukesBDay.ageDiff gt 1}"/>
            <h:outputText value="#{bundle.Older}"
                        rendered="#{dukesBDay.ageDiff gt 0}"/>
            <p/>
            <h:outputText
                value="#{bundle.AverageAge} #{dukesBDay.averageAgeDifference}."/>
            <p/>
            <h:commandButton id="back" value="#{bundle.Back}" action="greeting"/>
        </h:form>
    </h:body>
</html>
```

For example, the `#{bundle.SameAge}` string is displayed if the user and Duke have the same birthday as specified by the condition `#{dukesBDay.ageDiff == 0}` in the `rendered` attribute. That is, display the following string if the `ageDiff` property of `DukesBDay` equals `0`:

```
You are the same age as Duke!
```

The form also contains a `<h:commandButton>` tag that creates a back button that will direct the user back to the `greeting.xhtml` page, as specified in the `action` attribute.

4.4.2.5 Add the Form to `greeting.xhtml`

Add the form that provides the user interface for displaying Duke's age and specifying the user's birthday.

1. In the **Projects** tab, double-click `greeting.xhtml` in the `firstcup-war` project and, in the editor window, replace the text between the `<h:form>` and `</h:form>` tags with the following:

```
<h2>
    <h:outputText value="#{bundle.Welcome}"/>
</h2>
<h:outputText value="#{bundle.DukeIs} "/>
<h:outputText value="#{dukesBDay.age} #{bundle.YearsOldToday}"/>
<p/>
<h:outputText value="#{bundle.Instructions}"/>
<p/>
<h:outputText value="#{bundle.YourBD} "/>
<fc:inputDate id="userBirthday" date="#{dukesBDay.yourBD}" />
<h:outputText value=" #{bundle.Pattern}"/>
<p/>
<h:commandButton value="#{bundle.Submit}"
                 action="#{dukesBDay.processBirthday}"/>
<p/>
<h:message for="userBirthday" style="color:red"/>
```

2. In the editor window, right-click and select **Format**.

3. From the **File** menu, select **Save**.

4.4.2.6 Add the Form to `response.html`

Add a form that displays the age difference between Duke and the user, displays the average age difference of all users, and allows the user to navigate back to `greeting.xhtml`.

1. In the **Projects** tab, double-click `response.xhtml` in the `firstcup-war` project and, in the editor window, replace the text between the `<h:form>` and `</h:form>` tags with the following:

```
<h:outputText value="#{bundle.YouAre} "/>
<h:outputText value="#{bundle.SameAge}"
              rendered="#{dukesBDay.ageDiff == 0}"/>
<h:outputText value="#{dukesBDay.absAgeDiff}"
              rendered="#{DukesBDay.ageDiff lt 0}"/>
<h:outputText value=" #{bundle.Year} "
              rendered="#{dukesBDay.ageDiff == -1}"/>
<h:outputText value=" #{bundle.Years} "
              rendered="#{dukesBDay.ageDiff lt -1}"/>
<h:outputText value="#{bundle.Younger}"
              rendered="#{dukesBDay.ageDiff lt 0}"/>
<h:outputText value="#{dukesBDay.absAgeDiff}"
              rendered="#{dukesBDay.ageDiff gt 0}"/>
<h:outputText value=" #{bundle.Year} "
              rendered="#{dukesBDay.ageDiff == 1}"/>
<h:outputText value=" #{bundle.Years} "
              rendered="#{dukesBDay.ageDiff gt 1}"/>
<h:outputText value="#{bundle.Older}"
              rendered="#{dukesBDay.ageDiff gt 0}"/>
<p/>
<h:outputText
    value="#{bundle.AverageAge} #{dukesBDay.averageAgeDifference}." />
```

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

2. In the editor window, right-click and select **Format**.

3. From the **File** menu, select **Save**.

4.5 Building, Packaging, Deploying, and Running the `firstcup-war` Web Application

In this section, you will build the `firstcup-war` web application, deploy it to the server, and run the application.

4.5.1 Build, Package, and Deploy the `firstcup-war` Web Application

While performing this task, you'll build and package the `DukesBirthdayBean` enterprise bean, the `FirstcupUser` entity, and the `firstcup-war` web client into a WAR file, `firstcup-war.war` then deploy it to the server.

1. In the **Projects** tab, select the `firstcup-war` project .

2. Right-click `firstcup-war` and select **Run**.

4.5.2 Run the `firstcup-war` Application

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

1. Launch a web browser.

2. Enter the following URL in the address field of the browser:

   `http://localhost:8080/firstcup-war/`

3. Enter your birth date in the **Your birthday** field. Make sure you use the date pattern specified on the page: MM/dd/yyyy.

4. Click **Submit**.

5. After the `response.xhtml` page is displayed, click **Back** to return to the `greeting.xhtml` page.

6. Enter a different birthday in the text field and click **Submit** again to see how the average age of First Cup users changes.

Example 4–1   A Successful Response Page for `firstcup-war`

```
You are 20 years older than Duke!

The average age difference of all First Cup users is 20.0.
```

5

Next Steps

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

5.1 The Java EE Tutorial

The Java EE Tutorial (http://docs.oracle.com/javaee/7/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.

5.2 More Information on the Java EE Platform

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

- The Java EE Platform site (http://www.oracle.com/technetwork/java/javaee/overview/)
- GlassFish (http://glassfish.java.net/)
- The Aquarium (http://blogs.oracle.com/theaquarium/), a blog about GlassFish and open-source Java EE projects

5.3 Java EE Servers

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

5.3.1 GlassFish Server Open Source Edition

GlassFish Server Open Source Edition is the reference implementation of the Java EE platform APIs.

5.3.1.1 The GlassFish Project

The GlassFish (http://glassfish.java.net/) project is the open-source basis for GlassFish Server.

5.3.2 Other Java EE Servers

There are several other popular Java EE servers used by developers and enterprises.

- Oracle WebLogic Suite
  (http://www.oracle.com/us/products/middleware/application-server/weblogic-suite/).

- JBoss from Red Hat (http://www.jboss.org/).

- Apache Geronimo (http://geronimo.apache.org/).

- WebSphere software from IBM
  (http://www.ibm.com/software/websphere/).