

- ❑ **AutowireCapableBeanFactory**—Helps in configuring an existing external object and facilitates its dependencies. This is done through the autowireBeanProperties() and applyBeanPropertyValues() methods. By using the specified autowire strategy, the autowire() method creates a new bean instance of the given class.
- ❑ **ConfigurableBeanFactory**—Provides additional configuration options on a bean factory that can be applied during the bean initialization stage.
- ❑ **ApplicationContext**—Provides extra functionality of Bean Factory. This interface provides support for message lookup, internationalization, and an event handling mechanism.

One of the most striking features of Spring is AOP, which will be discussed in the next section.

Exploring AOP with Spring

AOP is a complement of OOP, which is defined as a programming technique. The important unit of modularity in AOP is aspect; whereas in OOP, the units of modularity are classes and objects. Aspects provide the modularization of various concerns, such as transaction management, which crosscut multiple types and objects. In AOP, such concerns are termed as cross-cutting concerns. For example, security is a cross-cutting concern in an application, as EmployeeService, SalaryService, and other services of that application must implement the security functionality.

The Spring IoC containers don't depend on AOP; however, you can use these containers with AOP, depending on the requirements of the application. The following are the uses of AOP in Spring:

- ❑ Replaces the EJB declarative services with new declarative enterprise services such as declarative transaction management.
- ❑ Allows you to implement custom aspects by using OOP with AOP.
- ❑ Allows AOP to be combined with Acegi security framework to provide declarative security service. Acegi security framework is a kind of Spring security framework that provides security solutions for Java EE-based Web applications.

Describing the AOP Concepts

The following list defines and explains the various key terms of the AOP module:

- ❑ **Advice**—Defines both what and when of an aspect. We know that each aspect has a purpose in AOP; the purpose of an aspect is known as advice. In real world, an advice defines what the job is and when it can be performed. The advice can be applied either before or after method invocation by using the @Aspect annotation.
- ❑ **Joinpoint**—Refers to a point where an aspect can be plugged in. It is the single location in the code where an advice should be executed (i.e., method invocation). For specifying a joinpoint, you have to implement the org.springframework.aop.Pointcut interface, and the information about the joinpoint can be accessed by the MethodInvocation interface.
- ❑ **Pointcut**—Refers to many joinpoints that are grouped together to perform an advice, which makes a pointcut. As we know that in Spring, a joinpoint is always a method invocation; therefore, we can say that a pointcut is a set of methods invocation.
- ❑ **Aspect**—Combines both advice and pointcuts. Therefore, an aspect consists of information such as what is the role of the aspect, and where and when to apply the aspect.
- ❑ **Introduction**—Adds methods and fields to an existing object. For using this concept, Spring has introduced some interfaces that must be implemented by the advised class.
- ❑ **Target object**—Refers to an object that is being advised by one or more aspects.
- ❑ **AOP proxy**—Refers to an object created by the Spring AOP to implement the aspect. In Spring, there are two types of AOP proxies—JDK dynamic proxy and code generator library (CGLIB) proxy.
- ❑ **Weaving**—Refers to the process that links aspects with other application types or objects to create an advised object. This process can be performed at compile time, load time, and runtime.

Explaining Types of Advices

The different types of advices are as follows:

- ❑ **Around advice**— Specifies whether to proceed to the joinpoint or to make a cross-cutting concern by returning its own return value or throwing an exception. The around advice is used in context of a joinpoint such as a method invocation.
- ❑ **Before advice**— Fires an advice before the execution of a joinpoint.
- ❑ **After throws advice**— Fires an advice when a method throws an exception. To handle this exception, you have to write the corresponding try and catch block in a class.
- ❑ **After returning advice**— Fires an advice when a method invocation or joinpoint is completed. For example, if the return statement of a method is executed, it returns an appropriate value without throwing an exception.
- ❑ **After finally advice**— Fires an advice regardless of whether a joinpoint exits with a normal or exceptional return.

Most of the AOP frameworks provide only around advice. However, at other instances, you might need to use other types of advices. For example, if you want to update a cache with a method that returns a result, you need to implement an after returning advice instead of an around advice.

Spring AOP Capabilities and Its Goals

Since Spring AOP is a fully OOP-based concept, the compilation process of a Spring application is the same as that of the simple Java applications. As a result, AOP can be used successfully in Java EE compatible application server and Web container.

The classes that represent pointcuts and different advice types are provided by Spring. The term advisor is used for an object that represents an aspect. This aspect has both an advice and a pointcut, which targets a particular joinpoint.

In Spring, different types of method interceptors are used with advice. The org.springframework.aop package has defined all the advices. The org.aopalliance.aop.Advice tag interface must be implemented by every advice. Some of the common advice interfaces in Spring AOP are MethodInterceptor, ThrowsAdvice, BeforeAdvice, and AfterReturningAdvice.

Generally, the features of Spring AOP are used with a Spring IoC container. AOP's advice is defined by using simple bean definition syntax. Spring IoC is capable of managing both advice and pointcuts. There are some situations where Spring AOP is very difficult to use. For example, when an advice is very fine-grained, it is difficult to implement Spring AOP. In such a situation, AspectJ annotation should be used.

Java EE 5 application server provides secondary services, which are heavy weight. These secondary services are managing transactions, implementing security, and logging errors of an application. Therefore, heavy weight Java EE 5 application server should be run while accessing the business logic of an enterprise application. However, in case of Spring application, only AOP framework of the Spring framework (a lightweight container) should be used.

Managing Transactions

The Spring framework provides declarative and programmatic transaction management services that offer a consistent programming model in every environment. The transaction management services are implemented by using different transaction APIs such as JTA, JDBC, Hibernate, JPA, and JDO. Spring transaction management provides the following benefits:

- ❑ Supports declarative transaction management for handling transactions
- ❑ Provides simple API for programmatic transaction management as compared to complex transaction APIs such as JTA
- ❑ Provides easier integration with various data access abstractions

Need of Transaction Management Support

Transaction management supported by the Spring framework provides an easy solution for handling transactions such as declarative and programmatic transactions. Some application servers support JTA that can be easily used with Spring's declarative transaction management, which is more powerful than EJB Container Management Transaction (CMT). The only scenario in which we need an application server's JTA capability is when we need to handle transactions across multiple resources. This scenario is not common in many high-end applications as they use single, highly scalable database.

There are two types of transactions to be managed by the application server:

- Global transactions: Help work with multiple transactional resources
- Local transactions: Represent resource-specific transactions such as transactions associated with a JDBC connection

JTA is normally available in an application server environment; therefore, the reusability of application code is limited to global transactions.

Prior to the advent of the Spring framework, the first choice to use global transactions was by using EJB CMT. CMT provides declarative type of transaction management, which is always preferred over the programmatic version. While using EJB CMT, the code for transaction management and Java Naming and Directory Interface (JNDI) lookup are not required. EJB CMT needs JTA and a Deployment Descriptor, that is, ejb-jar.xml, for providing metadata information. Therefore, we can say that implementing business logic in EJB is more complex (requiring more configuration) as compared to its implementation in Spring.

Local transactions are easy to handle, but do not support transactions over multiple transactional resources. For example, the code to manage transaction through JDBC will not run in a global JTA environment.

Programmatic transaction management solves the problem of managing transactions over multiple transactional resources. Consequently, application developers can use consistent programming model, irrespective of the environment.

Spring Transaction Abstraction

In the Spring framework, the org.springframework.transaction.PlatformTransactionManager interface defines the manner in which the transaction should take place. The following code snippet lists the methods of the PlatformTransactionManager interface:

```
public interface PlatformTransactionManager {
    TransactionStatus getTransaction(TransactionDefinition definition)
        throws TransactionException;
    void commit(TransactionStatus status) throws TransactionException;
    void rollback(TransactionStatus status) throws TransactionException;
}
```

The PlatformTransactionManager interface is a Serial Peripheral Interface (SPI) that provides a serial exchange of data between two systems. PlatformTransactionManager implementations are defined similar to any other object or bean in the IoC container. All the methods of this interface throw an object of the TransactionException class, and this exception class extends java.lang.Exception. The developer can choose to catch and handle the TransactionException instance where the application code can be recovered, in case of transaction failure. One of the salient features of Spring is that the developer is not necessarily required to handle the exceptions.

The getTransaction() method of the PlatformTransactionManager interface returns a TransactionStatus object and also passes TransactionDefinition parameter. The properties of a transaction that are specified by the TransactionDefinition interface are as follows:

- Isolation**—Specifies that the degree of all transactions is isolated from each other. For example, the uncommitted updates of one transaction are kept isolated from the other transaction.
- Propagation**—Specifies that the transaction code is executed within a scope of transaction. The Spring framework allows all types of transaction propagation such as suspending the existing transaction and creating a new transaction if the transaction is not executed due to network failure.

- ❑ **Timeout**—Specifies the duration for which the transaction may run before being timed out. A timed out transaction is automatically rolled back by the underlying infrastructure.
- ❑ **Read-only status**—Specifies a transaction that does not modify any data.

The transaction code is required to manage the transaction execution and query transaction status. The following code snippet shows the execution of transaction by using the TransactionStatus interface:

```
public interface TransactionStatus {
    boolean isNewTransaction();
    void setRollbackOnly();
    boolean isRollbackOnly();
}
```

You can use declarative or programmatic transaction management in Spring. For this, you need to define the correct PlatformTransactionManager implementation in your application to handle transaction. This implementation is given by IoC. The following code snippet shows the configuration details of datasource (dataSource), such as username, password, and URL, to handle transactions:

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
    method="close">
    <property name="driverClassName" value="${jdbc.driverClassName}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
</bean>
```

The preceding code snippet defines the JDBC DataSource, dataSource.

The following code snippet is the related PlatformTransactionManager bean definition:

```
<bean id="txManager" <!-- JTA Transaction Manager -->
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

The preceding code snippet uses the Spring DataSourceTransactionManager class for transaction management and provides a reference of this class to the DataSource.

When using JTA in Java EE container, a DataSource instance is required, which is obtained through JNDI. In Spring, a JtaTransactionManager implementation is provided to manage transactions of a bean and to lookup a datasource through JNDI. In other words, the JtaTransactionManager implementation uses only container DataSource. The following code snippet shows the implementation of JtaTransactionManager:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jndi="http://www.springframework.org/schema/jndi"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee.xsd">
    <jee:jndi-lookup id="datasource" jndi-name="jdbc/sample"/>
    <bean id="txManager" <!-- JTA Transaction Manager -->
        class="org.springframework.transaction.jta.JtaTransactionManager" />
        <!-- other <bean/> definitions here -->
</beans>
```

The preceding code snippet uses the `<jndi-lookup/>` tag from the `<jee:jndi-lookup id="dataSource" jndi-name="jdbc/sample"/>` schema inside the definition of a dataSource bean.

We can also use Hibernate dialect in place of JDBC to configure a datasource in an application, as shown in the following code snippet:

```
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="mappingResources">
        <list>
            <value>com/kogent/hibernate/hospital.hbm.xml</value>
        </list>
    </property>
</bean>
```

```

</property>
    <property name="hibernateProperties">
        <value>
            hibernate.dialect=${hibernate.dialect}
        </value>
    </property>
</bean>
<bean id="txManager"
    class="org.springframework.orm.hibernate.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>

```

In the preceding code snippet, the Hibernate LocalSessionFactoryBean object can be identified by a unique ID, that is, sessionFactory. The sessionFactory bean is used to obtain Hibernate Session instances. The DataSource definition is the same as provided when using the JDBC transaction API. The txManager bean is of HibernateTransactionManager type. The HibernateTransactionManager class needs reference to the sessionFactory bean, and the DataSourceTransactionManager instance needs a reference to the DataSource instance.

Similar to JDBC, we can also use JtaTransactionManager implementation in Hibernate and JTA transactions. Replace the definition of the txManager bean, given in the preceding code snippet, with the following:

```
<bean id="txManager"
    class="org.springframework.transaction.jta.JtaTransactionManager"/>
```

The way in which transactions are managed can be changed by just changing the configuration; there is no need to change the application code even if you are changing the local transactions to global transactions.

Resource Synchronization with Transactions

Now, you know how different transaction managers are created and linked to related resources that need to be synchronized to transactions. For example, the DataSourceTransactionManager class is needed to create a JDBC DataSource. Now you need to consider how to ensure that the resources, such as JDBC DataSource, Hibernate SessionFactory, and JDO, are obtained and handled properly in an application. Various techniques, such as creation, reuse, and cleanup, are performed to handle these resources.

There are three approaches to synchronize the resources with transactions—high-level approach, low-level approach, and using the TransactionAwareDataSourceProxy class.

The High-Level Approach

High-level approach is the most preferred approach for handling resources using the Spring framework. The resource handling techniques, such as creation, reuse, and cleanup, are handled internally by this approach. It means that the user does not need to create, reuse, and cleanup the resources as these functions are performed by the Spring framework automatically. You should note that commonly the classes, such as JdbcTemplate, HibernateTemplate, and JdbcTemplate, are used in this approach to handle transactions.

The Low-Level Approach

In low-level approach, classes, such as DataSourceUtils for JDBC, SessionFactoryUtils for Hibernate, and PersistenceManagerFactoryUtils for JDO, are used to handle resources. A developer provides the application code to handle transaction directly with the resource type of the native persistence APIs. In this approach, these classes ensure that the Spring managed instances are obtained, transactions are synchronized, and the exceptions that occur in the process are mapped to a consistent API. For example, the org.springframework.jdbc.datasource.DataSourceUtils class is used to get an instance of connection, instead of calling the getConnection() method on the DataSource instance. The following code snippet shows how to use the DataSourceUtils class to establish connection with a datasource:

```
Connection conn = DataSourceutils.getConnection (dataSource);
```

If an existing transaction establishes a connection by using the getConnection() method, it returns an instance of this connection. Otherwise, a new connection is created and synchronized to any existing transaction. In the Spring framework, the SQLException instance is wrapped in a CannotGetJdbcConnectionException instance. Therefore, the CannotGetJdbcConnectionException instance raises more exceptional errors that cannot be easily obtained from the SQLException instance.

The TransactionAwareDataSourceProxy Class

The `TransactionAwareDataSourceProxy` class can be used as a proxy for a target `DataSource`. This class is used to handle resource synchronization with transactions at the lowest level. The target `DataSource` is wrapped into this proxy class. Therefore, it is similar to transactional JNDI `DataSource` as provided by Java EE technology.

Declarative Transaction Management

The Spring framework helps the developers to make few changes in application code to handle transaction through declarative transaction management. The Spring framework's declarative transaction management is implemented by using Spring AOP because transactional aspect code is executed with the help of `AspectJ` annotation of AOP. The following are the similarities and differences of EJB CMT and Spring declarative transaction management:

- ❑ The Spring framework's declarative transaction management is capable of working in any environment with JDBC, JDO, or Hibernate; while EJB CMT is bound to JTA only
- ❑ Spring's declarative transaction management can be applied to any class; while EJB CMT is for special classes such as an enterprise Java bean class
- ❑ Spring offers declarative rollback that can be used with both programmatic and declarative rollback; whereas these are not supported by EJB
- ❑ You can build customized transactions by using Spring's AOP; but you cannot build customized transactions using EJB

Rollback rules enable us to define which type of exception should cause a transaction rollback automatically. Rollback rules are declarative and provided in the configuration file, not in the Java code. This declarative approach is preferred over rolling back transactions programmatically by calling the `setRollbackOnly()` method. In declarative approach, the business logic is not dependent on the transaction infrastructure and you do not need to import any Spring APIs.

Programmatic Transaction Management

Programmatic transaction management is a type of transaction management defined by the developer in the source code using the Spring transaction API. This type of transaction management can be implemented in the Spring framework in the following two ways:

- ❑ Using the `TransactionTemplate` class
- ❑ Using a `PlatformTransactionManager` interface

Using the TransactionTemplate Class

The method of implementing the `TransactionTemplate` class in an application is similar to the implementation of the `JdbcTemplate` class. Programmatic transactions can be performed with a callback approach, and the application code is not required to explicitly acquire and release transactional resources such as JDBC datasource. The application code must be executed in a transactional context to explicitly use the `TransactionTemplate` class. For this, you need to implement a `TransactionCallback` interface that contains all the code you need to execute in the transaction context. Next, you have to pass an instance of `TransactionCallback` interface to the `execute()` method of the `TransactionTemplate` object. The following is an example of application code using the `TransactionTemplate` class:

```
public class SampleService implements Service {
    private final TransactionTemplate tTemplate;
    public SampleService(PlatformTransactionManager tManager) {
        Assert.notNull(tManager, "The 'transactionManager' argument must not be null.");
        this.tTemplate = new TransactionTemplate(tManager);
    }
    public Object sampleServiceMethod() {
        return tTemplate.execute(new TransactionCallback() {
            public Object doInTransaction(TransactionStatus status) {
                updateOperation1();
                return resultOfUpdateOperation2();
            }
        });
    }
}
```

```

        }
    });
}
}
}

```

The following code snippet shows the use of the TransactionCallbackWithoutResult class when the return type is void:

```

tt.execute(new TransactionCallbackWithoutResult() {
    protected void doInTransactionWithoutResult(TransactionStatus status) {
        updateOpt1();
        updateOpt2();
    }
});
```

Transactions can be rolled back by using the setRollbackOnly() method. This method is called on the TransactionStatus object, as shown in the following code snippet:

```

tTemplate.execute(new TransactionCallbackWithoutResult() {
    protected void doInTransactionWithoutResult(TransactionStatus status) {
        try {
            updateOperation1();
            updateOperation2();
        }
        catch (SomeBusinessException e) {
            status.setRollbackOnly();
        }
    }
});
```

Using the PlatformTransactionManager Interface

You can use the org.springframework.transaction.PlatformTransactionManager interface to manage your transaction. For using the PlatformTransactionManager interface, you need to pass the object reference of the PlatformTransactionManager interface to the bean. Transactions can be executed with the help of TransactionDefinition and TransactionStatus instances. The code snippet to manage a transaction through the PlatformTransactionManager interface is as follows:

```

public class SampleService implements Service {
    // single TransactionTemplate shared all methods
    private final TransactionTemplate tTemplate;
    // use constructor-injection to supply the PlatformTransactionManager
    public SampleService(PlatformTransactionManager tManager) {
        Assert.notNull(tManager, "The 'transactManager' argument must not be null.");
        this.tTemplate = new TransactionTemplate(tManager);
    }
    public Object sampleServiceMethod() {
        return tTemplate.execute(new TransactionCallback() {
            // the code in this method executes in a transactional context
            public Object doInTransaction(TransactionStatus status) {
                updateOperation1();
                return resultOfUpdateOperation2();
            }
        });
    }
}
```

Choosing between Programmatic and Declarative Transaction Managements

Choosing which type of transactional management, that is, programmatic or declarative, is dependent on the number of transactional operations required by an enterprise application. If you have small number of transactional operations, you can implement programmatic transactional management. In this scenario, the use of TransactionTemplate class is preferred since you do not need to set up transactional proxies using Spring.

In case of large number of transactional operations, we use declarative transaction management, since it separates business logic from the transaction management.

Application Server-Specific Integration

Spring's transaction management is dependent on the application server. The `JtaTransactionManager` class needs to perform JNDI lookup for the JTA `UserTransaction` and `TransactionManager` objects, which are auto detected. The `JTA UserTransaction` and `TransactionManager` objects vary in different application servers, such as IBM WebSphere and BEA WebLogic.

NOTE

IBM stands for International Business Machines.

BEA WebLogic

While using WebLogic 7.1 or higher as an application server, the `WebLogicJtaTransactionManager` class, which is a subclass of the `JtaTransactionManager` class, is used for handling transaction management. The `WebLogicJtaTransactionManager` class handles suspended transactions, which have been marked for rollback, and resumes all types of transactions. In WebLogic 8.1, the transaction manager can be configured, as shown in the following code snippet:

```
<!-- WebLogic 8.1 transaction manager -->
<bean id="tManager"
      class="org.springframework.transaction.jta.WebLogicJtaTransactionManager">
</bean>
```

IBM WebSphere

When using IBM WebSphere as an application server, the `WebSphereTransactionManagerFactoryBean` class may be used to retrieve the instances of JTA `TransactionManager` using WebSphere's static access methods. In IBM WebSphere, the transaction manager can be configured as follows:

```
<!-- WebSphere transaction manager -->
<bean id="tmanagerWebSphere"
      class="org.springframework.transaction.jta.WebSphereTransactionManagerFactoryBean"/>
  <bean id="tManager"
        class="org.springframework.transaction.jta.JtaTransactionManager">
    <property name="tnManager">
      <ref local="tmanagerWebSphere"/>
    </property>
  </bean>
```

The instance of JTA `TransactionManager` can be obtained by configuring Spring's `org.springframework.transaction.jta.JtaTransactionManager` class. In the preceding code snippet, the `WebSphereTransactionManagerFactoryBean` class is used for checking WebSphere version to handle the transactions.

Let's move ahead to learn about Spring tag library, which allows you to generate dynamic Web pages.

Exploring Spring Form Tag Library

Spring provides a set of custom tag libraries that allow you to generate dynamic Web pages and other Web page content. The tag library contains a set of tags for using JSP and Spring Web MVC. A set of attributes is attached with each tag. The Spring form tag library has simplified the task of creating JSPs.

Spring form tag library is bundled in the JAR file named `spring.jar`. A Tag Library Descriptor (TLD), namely, `spring-form.tld`, is used to provide tag-related information in an application. You can use taglib declaration for a JSP view implementation by adding the Spring form tag library at the beginning of a JSP page, as shown in the following code snippet:

```
%@taglib prefix="forms" uri="http://www.springframework.org/tags/forms" %
```

In the preceding directive, the form tag name is denoted by a prefix, which is used to access different tags, such as `form` and `input`, from the Spring form tag library.

Classifying the Types of Tags

The tags present in the Spring form tag library are as follows:

- form tag

- input tag
- checkbox tag
- checkboxes tag
- radiobutton tag
- radioButtons tag
- password tag
- select tag
- option tag
- options tag
- textarea tag
- hidden tag
- errors tag

The form Tag

The `form` tag is used to create an UI form. The client enters some data and submits the form, which is then sent to the server.

In the following code snippet, a domain object is used, that is, `Client`, which contains a JavaBeans with various properties such as `userName` and `age`. The following code snippet provides an example of the `forms.jsp` page:

```
<form:form>
  <table>
    <tr>
      <td>Your Name:</td>
      <td><form:input path="userName" /></td>
    </tr>
    <tr>
      <td>Age:</td>
      <td><form:input path="age" /></td>
    </tr>
    <tr>
      <td colspan="4">
        <input type="submit" value="Save" />
      </td>
    </tr>
  </table>
</form:form>
```

The `Client` object, which resides in the `PageContext` object, returns the values of `userName` and `age`. There are many inner tags, such as `<input type="text" ...>`, which can be used with the `form` tag. The following code snippet provides the code for the generated HyperText Markup Language (HTML) page:

```
<form method="POST">
  <table>
    <tr>
      <td>Your Name:</td>
      <td><input name="userName" type="text" value="San"/></td>
      <td></td>
    </tr>
    <tr>
      <td>Age:</td>
      <td><input name="age" type="text" value="24"/></td>
    </tr>
    <tr>
      <td colspan="4">
        <input type="submit" value="Save" />
      </td>
    </tr>
  </table>
</form>
```

In the preceding HTML page, it is assumed that the variable name of the `form` object is `Client`. You can bind the `form` into a `Client` object, as provided in the following code snippet:

```
<form:form commandName="Client">
    <table>
        <tr>
            <td>Name:</td>
            <td><form:input path="userName" /></td>
        </tr>
        <tr>
            <td>Age:</td>
            <td><form:input path="age" /></td>
        </tr>
        <tr>
            <td colspan="2" style="text-align: center;">
                <input type="submit" value="Save" />
            
        </tr>
    </table>
</form:form>
```

The input Tag

The input tag is used to accept input from a user. The type attribute of the input tag provides the type of the input that a user can enter. The preceding code snippet also shows the usage of the input tag.

The checkbox Tag

The checkbox tag is used to input multiple selections by a user. Let's assume that a user has to choose a newspaper and a list of hobbies. For this, a class, that is, Prefer, is described in the following example:

```
public class Prefer {
    private boolean letter;
    private String[] interest;
    private String favword;
    public boolean isletter() {
        return letter;
    }
    public void setletter(boolean letter) {
        this.letter = letter;
    }
    public String[] getInterest() {
        return interest;
    }
    public void setInterest(String[] interest) {
        this.interest = interest;
    }
    public String getFavword() {
        return favword;
    }
    public void setFavword(String favword) {
        this.favword = favword;
    }
}
```

The following code snippet provides the code of the forms.jsp with the checkbox tag:

```
<form:form>
    <table>
        <tr>
            <td>Find the letter :</td>
            <%--First Approach : This Property is of type java.lang.Boolean --%>
            <td><form:checkbox path="prefer.letter"/></td>
        </tr>
        <tr>
            <td> Your Interests:</td>
            <td>
                <%--Second Approach : This Property is of an array or of type
                java.util.Collection --%>
                Trekking: <form:checkbox path="prefer.interest" value="Trekking"/>
                Tennis: <form:checkbox path="prefer.interest" value="Tennis"/>
                Hypnotizing: <form:checkbox path="prefer.interest"
                value=" Hypnotizing"/>
            </td>
        </tr>
    </table>
</form:form>
```

```

</tr>
<tr>
<td>This is the favourite word:</td>
<td>Cosmopolitan: <form:checkbox path="prefer.favword" value="Cosmopolitan"/>
</td>
</tr>
</table>
</form:form>

```

The following are three approaches to input multiple selections by using the checkbox tag:

- ❑ **First approach** – Allows you to use the `java.lang.boolean` type as a bound value. The checkbox is marked as checked when the bound value is `TRUE`. The `value` attribute of `input` corresponds to the resolved value of the `setValue(Object)` method of the `value` property.
- ❑ **Second approach** – Allows you to use the `java.util.collection` type or array as a bound value. The checkbox is marked as checked when the `setvalue(Object)` method of the `value` property exists in the bound collection.
- ❑ **Third approach** – Refers to the approach in which the checkbox is defined as checked when the bound value is equal to the value returned by the `setValue(Object)` method, provided the bound value is of any other type. The following code snippet provides the code of an HTML page with some checkboxes:

```

<tr>
<td>Your Interests:</td>
<td>
Trekking: <input name="prefer.interest" type="checkbox"
value="Trekking"/>
<input type="hidden" value="1" name="_prefer.interest "/>
Tennis: <input name="prefer.interest" type="checkbox" value="Tennis"/>
<input type="hidden" value="1" name="_prefer.interest"/>
Hypnotizing: <input name="prefer.interest" type="checkbox"
value="Hypnotizing"/>
<input type="hidden" value="1" name="_prefer.interest"/>
</td>
</tr>

```

If a form is submitted and a checkbox is selected, the unchecked value is not sent to the server. An underscore (`_`) is used for each checkbox to include hidden parameters.

The checkboxes Tag

The `checkboxes` tag is used for rendering multiple input tags. For example, to list all the possible hobbies in a JSP page, you can either provide a list of the available options during runtime or provide the list as the value for the `items` attribute of the `checkboxes` tag. For doing this, you have to pass the list of options in the form of an array, a list, or a Map interface. The use of this tag is shown in the following code snippet:

```

<form:form>
<table>
<tr>
<td>Interests:</td>
<td>
<form:checkboxes path="prefer.interest"
items="${userInterestList}"/>
</td>
</tr>
</table>
</form:form>

```

In the preceding code snippet, `userInterestList` is a type of the `List` interface. The `userInterestList` interface is available as a model attribute and contains strings values of the options of the checkboxes to be selected from.

The radiobutton Tag

The `radiobutton` tag is used to select a single option at a time in a JSP page. It is used when there are many tag instances with different values in the same property. The following code snippet shows how to use the `radiobutton` tag:

```

<tr>
    <td> Your Initials:</td>
        <td>Mr.: <form:radiobutton path="initials" value="Mr"/> <br/>
            Mrs.: <form:radiobutton path="initials" value="Mrs"/> </td>
</tr>

```

The radiobuttons Tag

The radiobuttons tag is used to display a pair of radiobuttons on a JSP page. It is similar to the checkboxes tag, which is used to pass all possible options during runtime. You can pass the available options in the items property, in the form of array, or a List or Map interface. You can also use custom object, as discussed earlier in the checkboxes tag. You can use the radiobuttons tag as follows:

```

<tr>
    <td>Marital status:</td>
        <td><form:radiobuttons path="maritalStatus" items="${maritalOptions}"/></td>
</tr>

```

The password Tag

The password tag is used for displaying the password entered by a user. The password text is generally displayed as asterisks (*). The following code snippet uses the password tag:

```

<tr>
    <td>Enter Password:</td>
        <td> <form:password path="password" /> </td>
</tr>

```

The select Tag

The select tag is used for binding data to the selected option. The option and options tags are nested tags in the select tag. Suppose a user knows languages of many countries; the select tag can be used to select multiple options of languages simultaneously. The following code snippet shows an example of the select tag:

```

<tr>
    <td> Language known : </td>
        <td><form:select path="language" items="${language}" /> </td>
</tr>

```

For example, if the user knows French language, the HTML code would appear as follows:

```

<tr>
    <td>Language known:</td>
        <td><select name="language" multiple="true">
            <option value="Hindi">Hindi</option>
            <option value="French" selected="true">French</option>
            <option value="English">English</option>
        </select></td>
</tr>

```

The option Tag

The option tag is used to provide multiple options for a client. The following code snippet shows how to use the option tag:

```

<tr>
    <td>Place:</td>
    <td>
        <form:select path= "place">
            <form:option value="Canada"/>
            <form:option value="Switzerland"/>
            <form:option value="New Delhi"/>
            <form:option value="Singapore"/>
        </form:select>
    </td>
</tr>

```

Suppose a user's native place is Switzerland, then the HTML source would appear as follows:

```

<tr>
    <td>Place:</td>
    <td>
        <select name="place">

```

```

<option value="Switzerland" selected="selected">Switzerland</option>
<option value="Canada" selected="true">Canada</option>
<option value="New Delhi">New Delhi</option>
<option value="Singapore">Singapore</option>
</select> </td>
</tr>

```

The options Tag

The options tag is used to represent a list of option tags for a view page. The following code snippet shows how to use the options tag:

```

<tr>
    <td>Native Place :</td>
    <td>
        <form:select path="place">
            <form:option value="-" label="--Please Select--"/>
            <form:options items="${placeList}" itemValue="code" itemLabel="name"/>
        </form:select>
    </td>
</tr>

```

The preceding code snippet provides a list of places in the form of an Array. All options are populated using the options tag.

For example, if a user's native place is Germany, the HTML source code would appear as follows:

```

<tr>
    <td>Native Place:</td>
    <td>
        <td>Place:</td>
        <td>
            <select name="place">
                <option value="-" --Please Select--</option>
                <option value="France">France</option>
                <option value="US">United States</option>
                <option value="Germany" selected="selected">Germany</option>
            </select>
        </td>
    </td>
</tr>

```

The textarea Tag

The textarea tag is used to represent an HTML textarea. The following code snippet shows how to use the textarea tag:

```

<tr>
    <td>Important Tips:</td>
    <td><form:textarea path="tips" rows="5" cols="30" /></td>
    <td><form:errors path="tips" /></td>
</tr>

```

The hidden Tag

The hidden tag is used for submitting a hidden value, after the form submission. The HTML input tag with the hidden type is written as follows:

```
<form:hidden path="place" />
```

For submitting place value as a hidden field, the HTML code is as follows:

```
<input name="place" type="hidden" value="Switzerland"/>
```

The errors Tag

The errors tag is used to represent a field that may lead to an error. The errors tag is useful to generate errors created by a controller or by any validators associated with the controller.

For example, you can use the errors tag if you need to raise an error on the basis of the text entered in the name and age fields. The following code snippet shows the use of the ClientValidator class to show error messages:

```

public class Clientvalidator implements Validator {
    public boolean support(Class candidate) {
        return Client.class.isAssignableFrom(candidate);
    }
}

```

```

public void validate(Object obj, Errors errors) {
    ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", required, "Name field is required.");
    ValidationUtils.rejectIfEmptyOrWhitespace(errors, "age", "required", "Age field is required.");
}
}

```

The JSP page would appear as follows:

```

<form:form>
  <table>
    <tr>
      <td>Name:</td>
      <td><form:input path="name" /></td>
      <%-- It show the errors for name field --%>
      <td><form:errors path="name" /></td>
    </tr>
    <tr>
      <td>Age:</td>
      <td><form:input path="age" /></td>
      <%-- It show the errors for Age field --%>
      <td><form:errors path="age" /></td>
    </tr>
    <tr>
      <td colspan="4">
        <input type="submit" value="Save" />
      </td>
    </tr>
  </table>
</form:form>

```

If there are empty values in the name and age fields, the HTML form would appear as follows:

```

<form method="POST">
  <table>
    <tr>
      <td>Name:</td>
      <td><input name="name" type="text" value="" /></td>
      <%-- Errors associated to name field is displayed --%>
      <td><span name="name.errors">Name field is required.</span></td>
    </tr>
    <tr>
      <td>Age:</td>
      <td><input name="age" type="text" value="" /></td>
      <%-- Errors associated to name field is displayed --%>
      <td><span name="age.errors">Age field is required.</span></td>
    </tr>
    <tr>
      <td colspan="4">
        <input type="submit" value="Save" />
      </td>
    </tr>
  </table>
</form>

```

Some wildcard facilities support the errors tag. For displaying the entire list of errors for a given page, the single wildcard character (such as * and #) or a sequence of wildcard characters is used. For example, the wildcard for the path attribute is as follows:

- path="*"**— Displays all errors
- path="age"**— Displays all errors associated with the age field

The following code snippet displays a list of errors at the top of an HTML page with the field-specific errors:

```

<form:form>
  <form:errors path="*" cssClass="errorList" />
  <table>
    <tr>
      <td>Name:</td>
      <td><form:input path="name" /></td>
      <td><form:errors path="name" /></td>
    </tr>
    <tr>
      <td>Age:</td>
      <td><form:input path="age" /></td>
      <td><form:errors path="age" /></td>
    </tr>
  </table>
</form:form>

```

```

</tr>
<tr>
    <td colspan="4">
        <input type="submit" value="Save" />
    </td>
</tr>
</table>
</form:>

```

In the preceding code snippet the `cssClass` attribute of the `form` tag is used to specify the class that would be displayed on the occurrence of an error. The following code snippet provides the source of the HTML page:

```

<form method="POST">
    <span name="*.errors" class="errorList">Field is required.<br/>Field is
    required.</span>
    <table>
        <tr>
            <td>Name:</td>
            <td><input name="name" type="text" value="" /></td>
            <td><span name="name.errors">Name field is required.</span></td>
        </tr>
        <tr>
            <td>Age:</td>
            <td><input name="age" type="text" value="" /></td>
            <td><span name="age.errors">Age field is required.</span></td>
        </tr>
        <tr>
            <td colspan="4">
                <input type="submit" value="Save" />
            </td>
        </tr>
    </table>
</form>

```

Now, let's discuss about Spring's MVC framework.

Exploring Spring's Web MVC Framework

The Spring Web MVC framework is built on generic Servlet known as a `DispatcherServlet` class. The `DispatcherServlet` class is also known as Front Controller. This `DispatcherServlet` class sends the request to the handlers with configurable handler mappings, theme resolution, and locale and view resolution along with file uploading. The `handleRequest(request, response)` method is given by the default handler called the `Controller` interface. The application controller implementation classes of the `Controller` interface are as follows:

- ❑ `AbstractController`
- ❑ `AbstractCommandController`
- ❑ `SimpleFormController`

These are super classes of the application controllers, and you can choose an appropriate super class while developing MVC Web applications. Any object can be used as a command or form object—it is not required to implement an interface or inherit a base class. The data binding of Spring considers type mismatches as validation errors that can be evaluated by the application; not as system errors.

Key Features of Spring Web MVC

Spring's Web module provides a set of Web support features, which are as follows:

- ❑ **Powerful configuration of both framework and application classes**—Provides powerful configuration of both framework and application classes. It also contains the information about validators and business objects.
- ❑ **Separation of roles**—Allows each component of the MVC framework to perform a different role during request handling. Request is handled by components such as controller, validator, command object, form object, model object, view resolver, handler mapping, and so on. Request handling is dependent among these roles and provides clear separation of roles.
- ❑ **Flexibility in choosing subclasses**—Allows you to use subclasses of any controller in an MVC Spring application.

- **Model transfer flexibility** – Provides the flexibility in model transferring through name/value pair of Map. This model transfer is easily integrated by any view technology.
- **No need of the duplication of code** – Allows you to use the existing business code. Therefore, there is no redundancy of code and the existing objects can be used as a form or command objects.
- **Specific validation and binding** – Displays the errors to the client if any validation error occurs.
- **Specific local and theme resolution** – Provides the facility of tag library of Spring and Java Server Pages with or without support for JavaServer Pages Standard Tag Library (JSTL).
- **Facility of JSP form tag library** – Helps in writing forms in JSP pages, which was introduced in the Spring Web MVC framework.

After discussing the features of Spring Web MVC framework, let's now discuss the components of Spring Web MVC framework.

The DispatcherServlet Class

The DispatcherServlet class is an important part of the Spring Web MVC framework, which is used for dispatching the request to application controllers. The DispatcherServlet class is configured with the IoC container. It is a Servlet that is inherited from the HttpServlet base class. The DispatcherServlet class is configured in the web.xml file of a Web application. You have to map the request using Uniform Resource Locator (URL) mapping in the same web.xml file when you want any request to be handled by it. The following code snippet shows how to map DispatcherServlet class in the web.xml file:

```
<web-app>
  ...
  <servlet>
    <servlet-name>examples</servlet-name>
    <servlet-class> org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>examples</servlet-name>
    <url-pattern>*,.form</url-pattern>
  </servlet-mapping>
</web-app>
```

In the preceding code snippet, the DispatcherServlet class handles all the requests.

After initialization of DispatcherServlet class, the Spring framework creates the beans, which are defined in the Spring configuration file.

There are many special beans in Spring DispatcherServlet, which are used to process the requests and provide the appropriate views for displaying the resultant Web page. These beans can be configured in the WebApplicationContext context. Table 21.1 lists the special beans in the WebApplicationContext context:

Table 21.1: Special Beans in the WebApplicationContext Context

Expression	Explanation
Controller(s)	Handles the client's request.
handler mapping(s)	Manages the execution of controllers, provided they match the specified criteria. For instance, a specified URL matches with the controller.
handler exception resolver	Facilitates mapping of exceptions to views or providing code to handle complex exceptions.
locale resolver	Resolves the locale that a client is using to offer internationalized views
Multipart file resolver	Helps you to process file uploads
theme resolver	Resolves themes that an Web application can use
view resolver	Resolves view names to views used by the DispatcherServlet class

The DispatcherServlet class handles the client's request by:

- ❑ Using the WebApplicationContext reference to search and bind request so that the controller and other elements in an application can use it. It is attached by default under the DispatcherServlet.WEB_APPLICATION_CONTEXT_ATTRIBUTE key.
- ❑ Using the locale resolver to resolve the locale, such as client date, and language used in an application. It is a component capable of resolving the locale of a client, such as internationalized views. We use it when processing the request such as preparing the data and also rendering the view.

You can customize the Spring DispatcherServlet class by adding context parameters in the web.xml file or Servlet init parameters. Table 21.2 lists the various initialization parameters of the Spring Dispatcher class:

Table 21.2: Various Initialization Parameters

Parameter	Explanation
contextClass	Allows you to instantiate the context for the class that implements the WebApplicationContext interface. The context is used by the Servlet. If we don't provide the contextClass parameter, the XmlWebApplicationContext class is used.
contextConfigLocation	Allows you to locate the context. For this, a String is passed to the context instance. The corresponding String is divided into many Strings by using comma as a delimiter.
namespace	Allows you to provide the description of the namespace.

Let's now discuss about the controller classes that can be extended by controllers of a Spring application.

Controllers

In Spring MVC framework, a controller plays the key role of handling the client request. It is used for defining the attributes of an application. The user input is interpreted by the Controllers, which also transforms the user input into a model and then displays the view page by using this model. The org.springframework.web.servlet.mvc.Controller interface is very important for the Controller architecture.

The Controller interface is implemented as follows:

```
public interface Controller {
    // Process the request and return a ModelAndView object which the
    // DispatcherServlet will render.
    ModelAndView handleRequest( HttpServletRequest req, HttpServletResponse
        res) throws Exception; }
```

A single method of the Controller interface handles the request and returns an appropriate model and view. ModelAndView and Controller's references are the main components of the Spring MVC implementation. The controllers in the Spring framework are as follows:

- ❑ AbstractController class
- ❑ MultiActionController class
- ❑ Command Controllers (include classes such as AbstractCommandController and SimpleFormController)

The AbstractController Class

The AbstractController class is the basic controller in the Spring MVC framework. All Spring Controllers are inherited from the AbstractController class. This class also offers caching support. Table 21.3 lists the properties of this controller class:

Table 21.3: Properties of the AbstractController Class

Property	Explanation
cacheSeconds	Generates caching directives in the Hypertext Transfer Protocol (HTTP) response provided by the Controller. Its default value is -1.
requiresSession	Indicates whether or not the session is required for Controller.
supportedMethods	Allows you to find the methods that support the Controller. You can set this property to either GET or POST method.

Table 21.3: Properties of the AbstractController Class

Property	Explanation
synchronizeOnSession	Allows the Controller to synchronize a user session.
useCacheHeader	Allows the Controllers to provide the HTTP 1.1 compatible Cache-Control header. The default value of this property is true.
useExpiresHeader	Allows the Controllers to provide the HTTP 1.0 compatible "Expires" header. The default value of this property is true.

In the following code snippet, we override the `handleRequestInternal(HttpServletRequest, HttpServletResponse)` method to handle request. This method returns the `ModelAndView` object to forward a Web page. The following code snippet shows a Controller class that returns an object of the `ModelAndView` class:

```
package sample;
public class SampleController extends AbstractController {
    public ModelAndView handleRequestInternal(
        HttpServletRequest req, HttpServletResponse res)
    throws Exception {
        ModelAndView mv = new ModelAndView("fo");
        mv.addObject("message", "Have a Nice Day!");
        return mv;
    }
}
```

The preceding declared class allows you to set up a handler mapping for a Controller. A hard-coded view is also returned by this Controller.

The MultiActionController Class

The `MultiActionController` class is a special type of Controller used to group related functionality mapped to different request. It is defined in the following package:

`org.springframework.web.servlet.mvc.multiaction`

The `MultiActionController` class maps requests to method names and then invokes the right method name. Table 21.4 lists the main properties of the `MultiActionController` class:

Table 21.4: Properties of the MultiActionController Class

Property	Explanation
delegate	Defines a delegate object used to invoke the methods that are resolved by the Resolver. For this, you need to define the delegate using configuration parameter as a collaborator.
methodNameResolver	Helps to resolve the method specified in a subclass of the <code>MultiActionController</code> class. It has to invoke a method based on the incoming request. You can define a Resolver that uses the configuration parameter.

The Command Controllers

Command controllers, such as `AbstractCommandController` and `SimpleFormController`, are the essential parts of Spring MVC package. The Command controllers provide an easy way to handle requests by binding the request parameter to data objects. The role of the Command Controllers classes is similar to the `ActionForm` class of Struts.

The following are the types of Command Controller classes:

- ❑ **AbstractCommandController**—Helps to create a customize Command Controller. It is used for binding the request parameter to a data object. This class does not provide the form functionality, but provides the validation features.
- ❑ **AbstractFormController**—Models and populates the forms by using a command object retrieved in the Controller. This controller supports many features such as validation and normal Workflow.

- ❑ **SimpleFormController**—Creates a form with corresponding command objects. It specifies a command object, a viewname for the form, and a viewname for the page, which is displayed to the user after successful submission of the form.
- ❑ **AbstractWizardFormController**—Serves as an abstract class that should be extended by the WizardController class. The AbstractWizardFormController class consists of various methods such as validatePage(), processFinish(), and processCancel() methods. The validatePage() method is used for forwarding the next view. The processFinish() method is used for exiting from the wizard, and the processCancel() method is used to cancel the wizard before it performs the final action.

Handler Mappings

Handler mapping is used for mapping the incoming requests to the appropriate handlers. The SimpleUrlHandlerMapping class and the BeanNameUrlHandlerMapping class are the examples of handler mapping. It is used for delivering a HandlerExecutionChain object. The main task of the HandlerMapping's reference is to provide the HandlerExecutionChain object. The HandlerExecutionChain object must contain the handlers that should match with the incoming request and also contain a list of handler interceptors.

When the user sends the request, the DispatcherServlet instance passes the incoming request to the handlers to inspect the request and forward to the HandlerExecutionChain object. After that, the DispatcherServlet instance executes the handler and handler interceptors in the chain.

A subclass of HandlerMappings class chooses a handler, based on the URL of the incoming request, as well as a state of the session associated with the incoming request.

Views and View Resolvers

Each Spring MVC framework renders a way for resolving the views. In Spring, you do not need a specific view technology for providing models in a browser. This facility is provided by view resolvers. Spring supports JavaServer Pages, Velocity templates, and XSLT view technologies to display the model's data in a browser.

The Spring framework provides ViewResolver interface to handle views. The ViewResolver interface is used to map between view names and actual views, whereas the View interface is used for preparation of request and handling over the request to one of the view technologies.

The ViewResolver Interface

ViewResolver is an interface used for resolving views by name. It is used for providing a mapping between views and view names. The state of a view cannot be changed while running an application. In Spring, views are addressed by a view name and resolved by a view resolver. Table 21.5 lists the various ViewResolver classes:

Table 21.5: Various ViewResolver Classes

ViewResolver	Description
AbstractCachingViewResolver	Caches the views.
FreeMarkerViewResolver/ VelocityViewResolver	Supports FreeMarkerView and VelocityView, such as Velocity templates. It is a subclass of the UrlBasedViewResolver class.
InternalResourceViewResolver	Supports the InternalResourceView class, such as Servlets and JSPs. Apart from InternalResourceView, the InternalResourceViewResolver class also supports subclasses, such as JstlView as well as TilesView, and is a subclass of the UrlBasedViewResolver class.
ResourceBundleViewResolver	Implements the ViewResolver interface; the bean definitions in a ResourceBundle interface are used by this class.
UrlBasedViewResolver	Implements ViewResolver, and is used for direct resolution of symbolic view names to URLs, without an explicit mapping definition. There's no need of arbitrary mappings when your symbolic names match the names of your view resources.
XmlViewResolver	Implements ViewResolver, which accepts an XML file with the same Document Type Definition (DTD) as the bean factories of Spring.

The UrlBasedViewResolver class allows you to choose JSP for a view technology. The following code snippet defines the ViewResolver bean using the UrlBasedViewResolver class for viewing JSP page:

```
<bean id = "viewresolverex" class =
    "org.springframework.web.servlet.view.UrlBasedViewResolver">
<property name = "prefix" value ="/WEB-INF/view/" />
<property name = "suffix" value=".jsp"/>
</bean>
```

If the ViewResolver bean returns a viewname tests, the request will be passed on to the RequestDispatcher object. The RequestDispatcher object sends the request to /WEB-INF/view/tests.jsp. The RequestBundleViewResolver class is used for mixing of different view technologies. The following code snippet defines the ViewResolver bean using the RequestBundleViewResolver class:

```
<bean id="viewresolverex"
    class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
<property name="base" value="views"/>
<property name="defaultParent" value="parentViews"/>
</bean>
```

In the bean definition, the ResourceBundleViewResolver class examines the ResourceBundle, which is recognized by the property name base. It is resolved by using the property value [viewname.class], which acts as a view class, and the property value [viewname].url, which acts as a view URL. A parent view can be recognized by the value of the property parentViews, from which all views in the properties file can be sorted.

Chaining ViewResolvers

When there are more than one view resolvers for handling a view, the Spring framework supports a chain of view resolvers, such as displaying both JSP and Excel views. You can create a chain of the resolvers by adding more resolvers in application context and setting the order property of multiple view resolvers for forwarding Web pages in an application. The order property is used for specifying the order of execution of the resolvers when it is found that there is more than one resolver in the application context.

There are two resolvers—InternalResourceViewResolver class and XmlViewResolver class—in the chain of view resolvers. The InternalResourceViewResolver object is always the last resolver in the chain of view resolvers, whereas the XmlViewResolver object is used for displaying Excel view. Excel view is an Excel worksheet that is used for displaying as a Web page without using Microsoft Excel. The chaining view resolver is described in the following examples of bean definition:

```
<bean
    id="jspResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="viewClass"
    value="org.springframework.web.servlet.view.JstlView"/>
<property name="prefix" value="/WEB-INF/view/" />
<property name="suffix" value=".jsp"/>
</bean>
<bean id="excelResolver"
    class="org.springframework.web.servlet.view.XmlViewResolver">
<property name="order" value="1"/>
<property name="location" value="/WEB-INF/view.xml"/>
</bean>
<beans>
<bean name="reports" class="org.springframework.example.ReportExcelView"/>
</beans>
```

When a particular view resolver of Spring does not give a view, then Spring examines the application context to find the other configured view resolver. If Spring finds an additional view resolver, it inspects it; otherwise, it throws an Exception. If a view resolver does not find any view, then it returns null.

Implementing Spring Web MVC Framework

In this section, we develop a simple Spring MVC application (springapp) that separates the business logic, presentation logic, and allows a controller servlet to handle the requests of the clients. The controller is responsible for accepting the user's request and interacting with the business objects to fulfill the request. Based on the user's request, the controller decides which view is used to display the result. In the springapp application, we need to develop the following modules for creating the springapp Spring MVC application:

- ❑ The controller (DemoController.java)
- ❑ The View (hello.jsp)
- ❑ Spring configuration file
- ❑ Web configuration file (web.xml)

Creating the Controller

Create a controller called DemoController.java and place it in the src\com\kogent\spring directory of the application directory. The DemoController.java Controller handles the request and returns a ModelAndView named hello.jsp (/jsp/hello.jsp). Listing 21.1 shows the code of the DemoController.java file (you can find this file on the CD in the code\JavaEE\Chapter21\springapp\src\com\kogent\spring folder):

Listing 21.1: Showing the Code of the DemoController.java File

```
package com.kogent.spring;
import org.springframework.web.servlet.mvc.AbstractController;
import org.springframework.web.servlet.ModelAndView;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import java.io.IOException;
import java.util.Date;
public class DemoController implements Controller {
    protected final Log logger = LogFactory.getLog(getClass());
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse
        response)
        throws ServletException, IOException {
        String now = (new Date()).toString();
        logger.info("Returning hello view with " + now);
        return new ModelAndView("hello", "now", now);
    }
}
```

In Listing 21.1, the ViewResolver is used to return the view, hello.jsp and forward it to the URL that matches the name of the view specified in firstspring-servlet.xml file. It also sets the current date and time value to the now key identifier, and makes the model available to the hello.jsp view.

Creating the Views

In this subsection, we need to create two JSP pages that are used to display the view page based on the Spring MVC paradigm. The two JSP pages are as follows:

- ❑ index.jsp
- ❑ hello.jsp

The index.jsp Page

The index.jsp page is used to forward client requests to the hello.htm page, which is mapped with the <servlet-mapping> element in the web.xml file. Listing 21.2 shows the index.jsp file (you can find this file on the CD in the code\JavaEE\Chapter21\springapp folder):

Listing 21.2: Showing the Code of the index.jsp File

```
<html>
<head>
    Index.jsp page
</head>
<body>
    <jsp:forward page="/hello.html"></jsp:forward>
</body>
</html>
```

The hello.jsp Page

Now, you need to create the hello JSP page (/jsp/hello.jsp) that is used to forward the user request matched to the HelloController.java file. Add the JSTL capability in your JSP page to display current date and time by using the <c:out/> tag, which is retrieved from the model mentioned in DemoController.java. Listing 21.3 shows the hello.jsp file (you can find this file on the CD in the code\JavaEE\Chapter21\springapp\jsp folder):

Listing 21.3: Displaying the Code of the hello.jsp File

```
<%@taglib prefix="c" uri="http://java.sun.com/jstl/core_rt"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt_rt"%>
<html>
<head><title>Hello, welcome to the Spring Application</title></head>
<body>
<h1>Welcome to the Spring Application</h1>
<p>Greetings, it is now <:out value="${now}" /></p>
</body>
</html>
```

Creating the Spring Configuration File

Now, you need to create an XML file called firstspring-servlet.xml in the springapp/WEB-INF directory. This file contains a bean definition that is used by the DispatcherServlet object. The name of this file is determined by two identifiers: one is firstspring, specified in the <servlet-name /> element in the web.xml file, and the other is servlet, appended with firstspring. This is the standard naming convention followed by Spring's MVC framework. This file contains bean entry named /hello.html, which specifies the class named com.kogent.spring.DemoController. This class acts as a controller that forwards the request to hello.html. Listing 21.4 shows the firstspring-servlet.xml file (you can find this file on the CD in the code\JavaEE\Chapter21\springapp\WEB-INF folder):

Listing 21.4: Displaying the Code of the firstspring-servlet.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/spring-beans-3.0.xsd">
    <!-- the application context definition for the firstspring DispatcherServlet -->
    <bean name="/hello.html" class="com.kogent.spring.DemoController"/>
    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

Creating the Web Configuration File

The web.xml file contains the configuration of the Spring MVC application, that is, springapp. It contains a DispatcherServlet instance, which acts as a Front Controller. All client requests are routed through this Front Controller. It also contains the <servlet-mapping> element that maps to the URL pattern with .htm extension, which is to be routed by the DispatcherServlet instance. Listing 21.5 shows the code of the web.xml file (you can find this file on the CD in the code\JavaEE\Chapter21\springapp\WEB-INF folder):

Listing 21.5: Showing the Code for the web.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                           http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <servlet>
        <servlet-name>firstspring</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
```

```

<servlet-mapping>
    <servlet-name>firstspring</servlet-name>
    <url-pattern>*.html</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>

```

Exploring the Directory Structure

The `springapp` is the root directory of Spring MVC application created here. All the JSP, Controller, and Spring configuration files are placed in this directory. You need to place the following .jar files in the `springapp\WEB-INF\lib` directory, as `springapp` application uses the APIs of JSTL, Servlet, and Spring Web MVC module:

- antlr-runtime-3.0.jar
- asm-2.2.3.jar
- asm-commons-2.2.3.jar
- asm-util-2.2.3.jar
- commons-logging.jar
- org.springframework.web-3.0.0.M1.jar
- org.springframework.web.servlet-3.0.0.M1.jar
- org.springframework.core-3.0.0.M1.jar
- org.springframework.beans-3.0.0.M1.jar
- org.springframework.context.support-3.0.0.M1.jar
- org.springframework.context-3.0.0.M1.jar
- org.springframework.expression-3.0.0.M1.jar
- org.springframework.aop-3.0.0.M1.jar

These JAR files help in successful compilation and execution of the `springapp` application. The directory structure of the `springapp` application is shown in Figure 21.2:

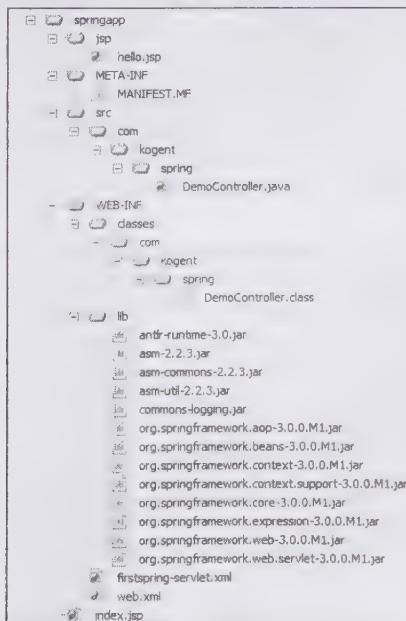


Figure 21.2: Displaying the Directory Structure of the `springapp` Application

Now, it's time to run the Spring MVC application.

Running the Application

Now, deploy the springapp application in your Glassfish server and run the application by typing the following URL on your browser:

• <http://localhost:8080/springapp>

After accessing the preceding URL, the index.jsp page is displayed, as shown in Figure 21.3:

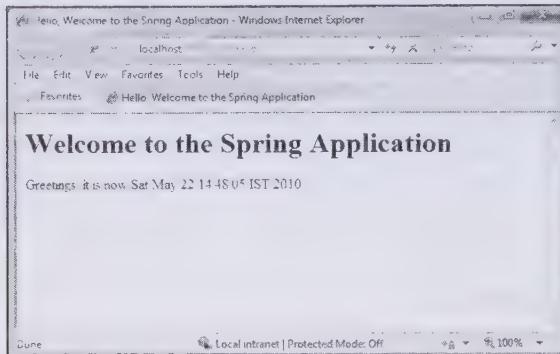


Figure 21.3: Displaying the Spring MVC Application

Testing Spring Applications

Testing is an integral part of enterprise software development. It is a process of finding bugs in the software under development to give reliable solution to the client. The software testing process is performed by testers to improve the quality of the application by checking the result and performance of the software application for various types of inputs. The process involves steps such as defining and executing test cases, and identifying problems and fixing them. There are many types of testing; however, this section focuses on unit and integration testing with Spring.

The Unit Testing

Unit testing is the basic type of testing in the development process. The smallest part of source code consisting of modules is called a unit. For procedural language, a unit can be a program, function, and so on; however, in object-oriented language, a unit is generally represented by a class. The test performed on these units is called unit test. The purpose of unit testing is to separate each piece of program and ensure that individual pieces of the code are correct.

While performing unit testing in the Spring framework, you do not need to set up the runtime infrastructure. If your application uses the Plain Old Java Object (POJO) concept, it can be tested in the Junit tests with objects whose instance is created by using the new operator.

Spring facilitates unit testing due to the DI feature of the Spring framework. For example, you can test the objects of the service layer by mocking the DAO interfaces, without accessing the persistent data while running unit tests.

The Integration Testing

The abbreviated name of integration testing is I&T. In I&T, individual modules are combined and tested as a group. I&T is normally performed after unit testing. I&T takes unit tested modules as input, groups them into larger units, and prepares the output for System testing. Test cases are designed to test that all the modules are functioning appropriately. The different types of I&T are big bang, backbone, top-down, and bottom-up.

The spring-mock.jar file is used for integration testing in the Spring framework. You can perform I&T to ensure the following in a Spring application:

- The correct wiring of Spring contexts

- Data access using JDBC or ORM tool

For integration testing, Spring provides various classes support bundled in the form of the spring-test JAR file. In this JAR file, you find the org.springframework.test package, which contains classes such as TestContext and TestContextManager, which are used for integration.

The following functionalities are provided by the Spring integration testing:

- Context management and caching
- DI of the test fixtures
- Transaction management
- Inherited instance variables

Let's discuss these in detail in the following subsection.

Context Management and Caching

There are two types of support provided by the org.springframework.test package: first for consistent loading of Spring contexts, and second for caching the loaded contexts. You should note that the caching of loaded contexts support is used if you are working on a large project. Spring container instantiates objects when you start the container, which consumes time and resource. To solve this problem, the AbstractDependencyInjectionSpringContextTests class provides an abstract protected method to provide the location of contexts:

```
protected abstract String[] getConfigLocations();
```

In the preceding code snippet, the signature of the getConfigLocations() method is provided. This method provides an array that contains the resource locations of XML configuration metadata, typically in the WEB-INF/classes folder of the application. The location of the XML configuration metadata is the same as the list of configuration locations specified in the web.xml file. The set of configuration is loaded once and reused later for each test case. This reusability of configuration ensures that subsequent tests are executed fast.

Dependency Injection of Test Fixtures

When the application context is loaded by the AbstractDependencyInjectionSpringContextTests class (and its subclasses), it provides an optional way to configure instances of the test classes by using Setter Injection. For this, you have to define the instance variable and the corresponding setters. The AbstractDependencyInjectionSpringContextTests class automatically finds the corresponding object in the set of configuration files passed as a parameter to the getConfigLocations() method.

Let's consider a scenario that you have a class, TestTitleDao, which implements the data access logic. Now, we want to write the integration tests to test the following aspects:

- The Spring configuration – Checks whether or not every information about TestTitleDao bean configuration is correct
- The Hibernate mapping file configuration – Verifies whether or not objects of the application are mapped correctly and the correct lazy-loading settings are specified in the Hibernate mapping file
- The logic of the TestTitleDao – Checks whether or not the configured instance performs its expected functionality successfully

An example of a test class is provided in the following code snippet:

```
package com.kogent.hibernate;
public class TestTitleDao extends
    AbstractDependencyInjectionSpringContextTests {
    // this instance will be (automatically) dependency injected
    private TestTitleDao titleDao;
    // a setter method to enable DI of the 'titleDao' instance variable
    public void setTitleDao(TestTitleDao tDao) {
        this.titleDao = tDao;
    }
    public void loadtestTitle() throws Exception {
        //provide code to load the title
    }
    // specifies the Spring configuration to load for this fixture
```

```

protected String[] getConfigLocations() {
    return new String[] { "classpath:com/kogent/conf.xml" };
}
}

```

In the preceding code snippet, the attendant file referenced by the `getConfigLocations()` method (`'classpath:com/kogent/conf.xml'`) is specified in the following code snippet:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/http://www.springframework.org/dtd/spring-
beans.dtd">
<beans>
<!-- this bean will be injected into the TestTitleDao class -->
<bean id="titleDao" class="com/kogent/ hibernate/TestTitleDao">
<property name="sessionFactory" ref="sessionFactory"/>
</bean>
<bean id="sessionFactory"
      class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
<!-- dependencies elided for clarity -->
</bean>
</beans>

```

You can use the `AbstractDependencyInjectionSpringContextTests` class to test the autowire relationship between the beans in a Spring application. However, if you have multiple beans of the same type, you cannot apply this autowire approach for these beans. In this case, you can use the `ApplicationContext` instance variable for explicit lookup and explicitly call the `ApplicationContext.getBean("titleDao")` bean.

If you don't need DI in your test case, you can create a customize class that is extended by the `AbstractSpringContextTests` class of the `org.springframework.test` package.

Transaction Management

One common issue in testing is that it requires a database to observe and test the state of the persistence store. If the state of a database is changed during testing, you need to ensure that the changes are rolled back so that the database can be used for testing in future.

The Spring integration testing supports frameworks to perform insert or update operations for multiple test cases on the same database. By default, the frameworks create and roll back a transaction for each test. Transactional support is provided to the test class through a `PlatformTransactionManager` bean defined in the test's application context. The benefit of transaction management in the Spring framework is that you can reuse the application contexts across various testing scenarios such as configuring Spring-managed object graphs, transactional proxies, and `DataSources`.

For example, if the test methods delete the contents of selected tables while running a transaction, the transaction will roll back to the default state, and the database will return to its prior state.

There are some useful methods to commit transactions, such as the following:

- ❑ `setComplete()`—Helps to commit a transaction. This method is inherited from the `AbstractTransactionalSpringContextTests` class.
- ❑ `endTransaction()`—Rollbacks a transaction, and commits it only if the `setComplete()` method has been called earlier.

Inherited Instance Variables

You can access the following protected instance variables by extending the `AbstractTransactionalDataSourceSpringContextTests` class:

- ❑ `applicationContext (ConfigurableApplicationContext)`—Helps to perform the bean lookup explicitly and also test the complete state of the context. This method is inherited from the `AbstractDependencyInjectionSpringContextTests` super class.
- ❑ `jdbcTemplate`—Inherits from the `AbstractTransactionalDataSourceSpringContextTests` class. It is used for querying a database to confirm its state.

The following example illustrates the use of the `AbstractTransactionalJUnit4SpringContextTests` class for performing integration testing:

```

@ContextConfiguration
public abstract class AbstractHospitalTests
extends AbstractTransactionalJUnit4SpringContextTests {
    protected Hospital hospital;
    public void setHospital(Hospital hospital) {
        this.hospital = hospital;
    }
    public void testGetVets() {
        //provide code to retrieve data from the datasource
    }
}

```

The following code snippet illustrates the JDBC implementation of the tests containing the getConfigLocations() method:

```

@ContextConfiguration
public class HibernateHospitalTests extends AbstractHospitalTests {
    protected String[] getConfigLocations() {
        return new String[] {
            //provide the location of the hibernate.xml file in the form of String};
    }
}

```

In the preceding code snippet, the Hibernate implementation of the HibernateHospitalTests class contains the @ContextConfiguration annotation, which is used to load the application context of a bean. Testing is performed by inheriting the AbstractHospitalTests class.

Integrating Spring with Hibernate

The integration of the Spring framework with Hibernate provides functionalities such as resource management, DAO implementation support, and transaction management. Although Hibernate supports ORM, the problem is that the Hibernate Web application uses the Hibernate APIs, such as Configuration, SessionFactory, and Session, to access a database through Hibernate DAO object. The Hibernate DAO object is scattered across the code throughout the Hibernate Web Application. In case of the Spring framework, the business objects are not scattered as they are configured with the help of IoC Container. Therefore, it is possible to use the Hibernate objects with Spring beans through the IoC Container. It is also possible to use all the functionalities, such as ORM and DAO implementation of Hibernate, with the Spring framework.

The following are some of the benefits of using the Spring framework to create Hibernate ORM DAOs:

- ❑ **Easy to test**—Facilitates the Spring's IoC container to use the implementations and configuration of Hibernate SessionFactory instances, JDBC DataSource instances, transaction managers, and ORM object implementations. This makes it easier to test each piece of persistence code by isolating the code in the Spring Web application.
- ❑ **Integrated transaction management**—Allows you to wrap the ORM code with either declarative transaction management or AOP style method interceptor. In either case, if any exception occurs during the execution of the transaction, the transaction can be easily rolledback. JDBC-related code provides full transactional support by using ORM.

Integrating Struts 2 with Spring

Struts 2.x, which was known as WebWork earlier, can be treated as a new Web framework. It has very few similarities with Struts 1.x. Struts 2.x is an integration of WebWork and Struts 1.x. You can learn about the Struts framework in *Chapter 20, Working with Struts 2.1*.

Struts 2 is a very simple way of implementing MVC pattern in our applications. It has a servlet filter, called FilterDispatcher, as the controller. All the requests pass through this FilterDispatcher. The FilterDispatcher takes the requests and passes it through a series of interceptors before it invokes an Action class. The Action class receives the requests, processes the requests, and returns a result that is sent to the user.

Struts 2 applications consist of application objects that help in controlling the application functionality. These application objects include service objects, along with actions and interceptors components. The Struts 2 framework creates and instantiates these objects. In Struts 2, an action class is created and mapped to

appropriate URL using XML or Java annotations. The Struts 2 framework uses the ObjectFactory component to automatically create the application objects, except the service objects, which are required for implementing actions in Struts 2 applications. Since the actions need to obtain the reference of service objects for implementing any functionality, this reference to a service object is provided manually using the new operator. Due to this, the service objects and actions may get tightly coupled with each other at some point of time. To overcome this, we need a framework that manages the creation of service object and solves the problem of tightly coupled objects. The solution to this problem is the Spring framework, which can be used for managing the creation of objects and reducing the coupling between objects.

Let's understand how to integrate Spring with Struts 2 next.

Configuring Spring in a Struts 2 Application

To integrate Spring and Struts 2 in an application, we need to perform the following broad-level steps:

- ❑ Add the Spring plug-in and explicitly define the objects in the source code of the application. Spring plug-in specifies how to integrate Spring and Struts 2 by manipulating the application and service objects.
- ❑ Download the Spring.jar file to create a Spring container in the Struts 2 application. Then, configure the Spring listener in the web.xml file.
- ❑ Set the mode of autowiring to inject dependencies in the created objects.

Let's explore these steps in detail.

Adding the Spring Plug-In

You need to download and add the Spring plug-in to the Struts 2 application. Struts 2 supports the Spring plug-in, which enhances the Struts 2 ObjectFactory component class for managing the created objects. This plug-in deals with objects that are explicitly specified in the source code of the application. You can download the spring plug-in file, that is, spring-plugin-2.0.9.jar, from the URL <http://cwiki.apache.org/S2PLUGINS/home.html>.

Downloading the Spring.jar File and Configuring the Spring Listener

You also need to download the Spring JAR, that is, spring.jar, from the URL <http://springframework.org>, for creating a Spring container to implement Spring functionality in the Struts 2 application. Next, the Spring listener is configured in the web.xml file of a Web application for using the Spring application Context listener, as shown in following code snippet:

```
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

To specify the objects explicitly, we need to declare the objects as Spring beans in the applicationcontext.xml file located in the WEB-INF folder. The ContextLoaderListener Spring listener searches for metadata information in the applicationcontext.xml file and then injects the object in a class, based on the metadata information.

Let's move further and set the autowiring mode required to inject dependencies in the objects.

Setting the Autowiring Mode

The Spring framework provides a feature to autowire dependencies of resources. Autowiring refers to the technique where you do not need to explicitly declare the object for injecting the dependencies; Spring automatically looks up for an object based on the object property. There are various modes of implementing autowiring, which can be set in the configuration files. The modes of autowiring are as follows:

- ❑ Autowiring by name
- ❑ Autowiring by type
- ❑ Autowiring by constructor
- ❑ Autowiring by auto

Now let's discuss each of them in detail.

Implementing the Name Mode

Autowiring by name is the default mode of autowiring. This mode uses the setter method to match the ID of a bean with the name of the property specified in the bean action.

The following code snippet shows how to implement the autowiring by name mode in the applicationContext.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi=
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
<bean id="strutsservice" class = "com.kogent.StrutsserviceJPA"/>
.....
</beans>
```

The preceding code snippet declares the ID for the bean to be injected in a resource. Spring automatically matches this ID with the setter method name for injecting the dependencies.

The following code snippet shows the setter method for receiving the dependencies:

```
StrutsserviceInterface strutsservice;
public void setStrutsservice(StrutsserviceInterface strutsservice)
this.strutsservice=strutsservice;
}
```

Implementing the Type Mode

Spring provides the type mode of autowiring, which helps search registered Spring beans that match the property specified in the defined action. For using this mode, you need to change the autowiring mode in the struts.properties file, as shown in the following code snippet:

```
Struts.objectFactory.spring.autowire=type
```

You can also specify the type mode in the struts.xml file, as shown in the following code snippet:

```
<constant name=" Struts.objectFactory.spring.autowire" value=type />
```

Implementing the Constructor Mode

The autowiring by constructor mode allows you to autowire the parameters specified in the bean constructor. This mode works with constructor objects, which take the dependencies as parameters. If more than one Spring bean has the same matching type, an exception is thrown instead of deciding which bean to be injected. The syntax to implement the autowiring by constructor mode is as follows:

```
Struts.objectFactory.spring.autowire=constructor
```

Implementing the Auto Mode

The autowiring by auto mode allows you to automatically detect the mode for injecting the dependencies. Usually, the autowiring by auto mode injects the dependencies by using the constructor mode and then the type mode. The syntax to implement the autowiring by auto mode is as follows:

```
Struts.objectFactory.spring.autowire=auto
```

Summary

The chapter has described Spring, a lightweight framework for developing Java EE applications. The seven modules of the Spring framework, namely Spring Core, Spring AOP, Spring Web MVC, Spring DAO, Spring ORM, Spring context, and Spring Web flow, have also been discussed. In addition, the chapter describes the key terms and advices of Spring AOP. The chapter has also helped you to create an application, that is, springapp, to demonstrate the implementation of Spring. Finally, the chapter has discussed the integration of Spring with Hibernate and Struts 2.

In the next chapter, you learn how to implement security in Java EE 6 applications.

Quick Revise

- Q1. is not a module in the Spring architecture.
- A. Spring AOP
 - B. Spring ORM
 - C. Spring APO
 - D. Spring DAO

Ans. C

Q2. The package provides the core JDBC framework.

- A. org.springframework.beans.factory
- B. org.springframework.aop
- C. org.springframework.jdbc.core
- D. org.springframework.beans

Ans. C

Q3. Name the package containing the super classes for integration testing.

- A. org.springframework.test
- B. org.springframework.jdbc.core
- C. org.springframework.beans.factory
- D. org.springframework.aop

Ans. A

Q4. What is Spring framework?

Ans. Spring framework is a collection of subframeworks, also called layers, such as Spring AOP, Spring ORM, Spring Web Flow, and Spring Web (Model View Controller) MVC. You can use any of these layers separately while creating a Web application. These modules may also be grouped together to provide better functionalities in a Web application.

Q5. Give the full form of AOP.

Ans. Full form of AOP is Aspect-Oriented Programming

Q6. What are the different types of advices in Spring AOP?

Ans. Different types of advices in Spring AOP are as follows:

- Around
- Before
- After throws
- After returning
- After finally

Q7. What is Dependency Injection?

Ans. Dependency Injection can be defined as a form of Inversion of Control (IoC). It is an object-oriented programming technique that is used to invert the control for object creation and linking.

Q8. What is the role of the DispatcherServlet class in the Spring framework?

Ans. The DispatcherServlet class plays the role of a central Servlet, which extends the HttpServlet class, and is fully configured with the IoC container. All the client requests to the controller are processed by this class.

Q9. What is a BeanFactory?

Ans. A BeanFactory is an implementation of the factory pattern that instantiates, configures, and manages Beans and Java objects.

Q10. Why do we need to implement the JDBC abstraction layer?

Ans. JDBC abstraction layer helps in reducing the amount of code and simplifying the process of error handling.

Q11. Which interface is responsible for handling the exceptions in the Spring framework?

Ans. The HandlerExceptionResolvers interface is responsible for exception handling. This interface is defined in the web.xml file.

Q12. What is the role of the JdbcTemplate class?

Ans. The JdbcTemplate class is the main class in the core package of the Spring framework that helps in avoiding common errors. This class is used to execute SQL queries, stored procedures, and operations on Resultsets in JDBC.

22

Securing Java EE 6 Applications

If you need an information on:

See page:

Introducing Security in Java EE 6	1042
Exploring Security Mechanisms	1045
Implementing Security on an Application Server	1046
Securing Enterprise Beans	1048
Securing Application Clients	1050
Implementing Security in Web Applications	1050
Implementing Security	1055

In today's scenario, every organization faces numerous security threats that can expose the confidential information of the organization to unauthorized access. These threats may occur due to system failure, lack of availability of resources, access to data by unauthorized users, and so on. The security threats to an organization can include disclosure of confidential information, modification or destruction of information, misappropriation of protected resources, and compromise on accountability. These security threats may appear depending upon the environment in which an enterprise application is executed. For example, sharing of confidential information stored in files over a network that is not protected may pose a security threat.

Therefore, organizations must take steps to identify these threats; and take the necessary corrective actions to eliminate them. Java EE platform provides various security mechanisms, such as authentication, authorization, encryption, and auditing, which allow you to expose data to authorized users.

In this chapter, you learn about the concept of security in Java EE 6. Next, you learn about different security mechanisms. You also learn how to implement security on an application server. Further, you learn how to secure enterprise beans and application clients. In addition, you learn how to implement security on Web applications. In the end, you learn how to implement security using different approaches.

Let's start by learning the concept of security in Java EE 6.

Introducing Security in Java EE 6

The Java EE 6 applications consist of Web components, such as servlet and JavaServer Pages (JSP). These components are deployed in different Web containers to build an enterprise application. After creating an enterprise application, you need to consider various security issues so that the data in the system can be protected from unauthorized access. In a Java EE 6 application, a security mechanism is configured to allow only authenticated users to access functions and data of the application. Java EE uses the Java EE container to supply low level platform specific functionality to the Web components. The Java EE container provides the following two types of security:

- ❑ **Declarative security**—Refers to the type of security provided to the Web components by using the Deployment Descriptor, which is an Extensible Markup Language (XML) file that explains the deployment of Web and enterprise applications. The Deployment Descriptor includes the mapping for application specific components, which provide security roles, users, and policies to protect the Web application and enterprise application components from unauthorized and unauthenticated access. Security roles, users, and policies are discussed later in this chapter.
- ❑ **Programmatic security**—Refers to the type of security that is embedded in an application and is used to make security decisions, such as determining whether or not a user is authorized to access a resource. Programmatic security defines the security model of the application.

Security deals with authentication and protection of Web resources for various clients. Let's discuss about authentication and protection domain next.

Authentication

Authentication is a security mechanism in which callers and service providers validate each other on behalf of specific users or systems in a distributed computing environment. While implementing authentication call, identities are created to validate whether or not the user is authenticated. The type of authentication in which the identity of caller is validated by a service provider and the identity of the service provider is validated by the caller is called mutual authentication. An entity is called unauthenticated if it participates in a call without setting up or proving its identity.

The caller identity refers to a user, who is running a client program and trying to access the server using the client end program.

When a client program, which is executed by the user, requests for an application component, the caller identity propagated to the requested component is of the user. In case request to an application component is made from another application component, which is between the user originating the request and the requested application component, caller identity propagated to the final application component may be of the intermediate application component or of the user.

Authentication is a mechanism that involves two phases. In the first phase, the service independent authentication is performed to establish an authentication context that encapsulates and verifies the user's identity. In the second phase, the authentication context authenticates the user's identity with entities on the server. Therefore, providing control over access to the authentication context, and authenticating the associated identity, becomes the base of authentication. To control access to an authentication context, you can perform the following tasks:

- ❑ Perform the initial authentication to allow a user to access the authenticated context
- ❑ Provide the authenticity of a component in any other trusted or related component of the same application
- ❑ Delegate the authenticated context from the caller to its called component when the component impersonates its caller

Protection Domain

A collection of entities that are expected or known to trust each other is called a protection domain. An entity in a protection domain is not required to prove its authenticity to others in the domain. In such cases, authentication is only needed when interactions span the protection domain boundary, as illustrated in Figure 22.1. When a component interacts with another component within the same protection domain, no constraint is enforced on the caller identity emanated from these components. Based on trust, instead of authentication, the caller, which is a component, can claim its identity in the following ways:

- ❑ Propagating an identity
- ❑ Selecting an identity on the basis of the authorization constraints implemented by the called components

If you implement the concept of protection domain, you should define the boundaries of the protection domain to restrict the unproven identities. This will also avoid the authentication of entities in the protection domain as only proved identities are allowed within the domain.

In the Java EE architecture, the authentication boundary is established by the container, which lies between external callers and the components hosted by the container. A container does not usually host different protection domain components; but exceptionally, it might host components from different protection domains. Figure 22.1 shows the details of protection domain:

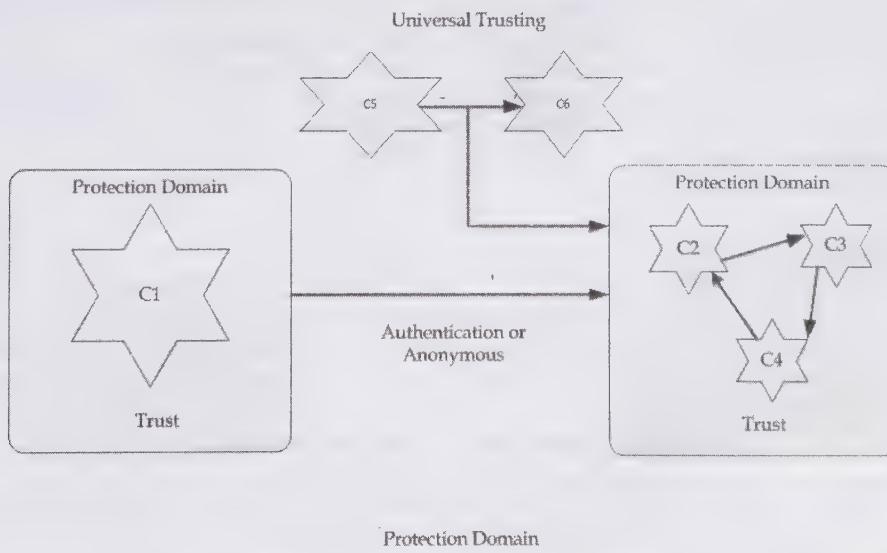


Figure 22.1: Showing Protection Domain

A caller identity is accessible to the components as credentials; for example a X.509 certificate or a Kerberos service ticket, which is mainly used for inbound calls. The bi-directional authentication functionality is provided by the container to impose protection domain boundaries of the deployed applications in the container.

The interacting containers need to decide if there is enough inter-container trust for accepting the container-provided depiction of component identities. In some environments, trust may simply be assumed, while in others, inter-container authentication need to be evaluated and the container identities may be compared with the trusted identities. In case the required information regarding the identity of the user is not provided and sufficient inter-container trust relationship does not exist, the container would reject the call. Figure 22.2 explains the authentication concepts using two scenarios, an authenticated user scenario and an unauthenticated user scenario:

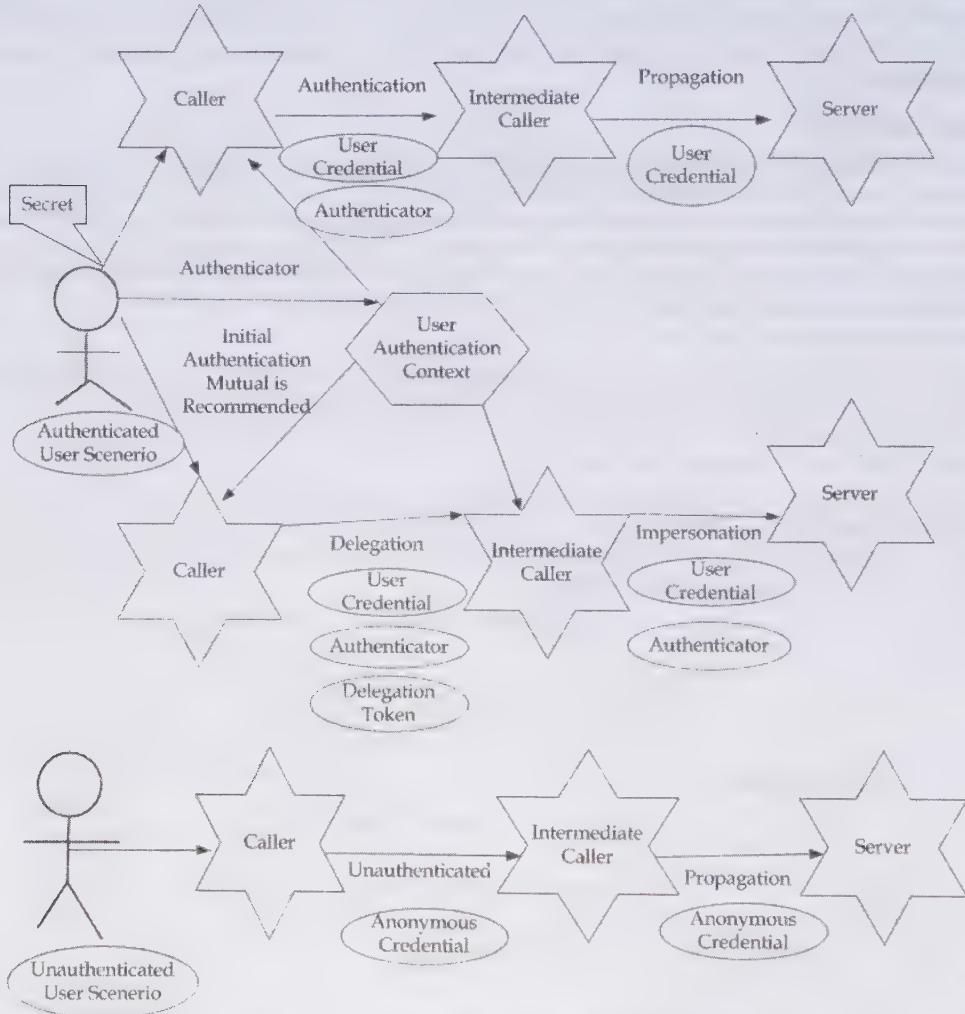


Figure 22.2: Showing the Authentication Scenarios

A user's authentication context may be used by the called component to prove its identity to the intermediate component. Caller's identity is propagated to a component if the called component makes a call to that component. The propagated identity is accepted only in the condition when the target trusts the caller, which implies that the called component and the target lies in the same protection domain.

In Unauthenticated User Scenario shown in Figure 22.2, caller's identity propagation is stopped from delegation and subsequent impersonation as it is found unauthenticated by the service provider. It is the responsibility of service providers to identify whether the propagated identities should be accepted as authentic during

propagation. The user provides the accessibility to a called component for its authentication context so that the called component can impersonate the user during calls in delegation. The only requirement of impersonation is that it is expected from the user to trust the impersonator, which acts on behalf of the user. Figure 22.2 also describes how the propagation of an unauthenticated user identity can be done in the form of an anonymous credential. An anonymous credential is a form of unproven identity that may be propagated even if there is no mutual trust between the caller and the service provider.

After having a brief idea about the implementation of security in the Java EE 6 application, let's explore various mechanisms used to implement security.

Exploring Security Mechanisms

The Java EE security mechanisms are easier to implement and configure. These mechanisms provide fine access control to data and functions available in Java EE 6 applications. In a Java EE 6 application, a user interacts with enterprise resources within the Web or Enterprise JavaBeans (EJB) tiers, through a client container. A user may access either the protected or unprotected resources. Protected resources are distinguished by using the authorization rules that restrict access to a subset of non-anonymous identities. Therefore, a protected resource may be accessed by the user provided that the user presents a non-anonymous credential. This non-anonymous credential helps in evaluating the identity of the user according to the resource authorization policy. When the client and the resource container do not trust each other, the credential must be associated with an authenticator that approves the credential's validity. In this section, you learn about various authentication mechanisms and their configurations that are expected by the Java EE platform. The Java EE security mechanisms provide a way to authenticate users and authorize them to access application functions and associated data at different layers. In this section, the security mechanism is discussed in context of the following three layers of security:

- Application layer security
- Transport layer security
- Message layer security

Now, let's discuss each layer of security in detail with its advantages and disadvantages.

Application Layer Security

You can provide an application layer security in Java EE 6 applications with the help of component containers. The application layer security provides security to a specific application type. In case of application layer security, application firewalls are employed in the application layer to enhance protection to a Web application by preventing unauthorized user access to the application stream, and all application resources. The security is provided to an application when the application is executed on different application servers. The security properties are non-transferable. The application layer security has certain advantages and disadvantages. The various advantages of application layer security are as follows:

- Applies security according to the requirement of the application.
- Provides the application specific security. Therefore, it is applied while executing the application.

The various disadvantages of application layer security are as follows:

- Makes the application dependent on the security attributes, which are non-transferable. This implies that these security attributes cannot be used in other application types.
- Does not support multiple protocols in an environment.

Transport Layer Security

The transport layer security is provided by transport mechanism associated with a specific type of application. The transport mechanism includes transferring information over the client-server network. Therefore, the transport layer security relies on secure Hypertext Transfer Protocol (HTTP) transport using Secure Socket layer (SSL). The transport layer security is a point-to-point security used for authorization, message integrity, and confidentiality. The server and client can authenticate one another during execution of an application over SSL-protected session. In addition, the server and the client agree on an encryption algorithm and cryptographic keys, which are source of information that determines the result of encryption algorithm, prior to transmission

and reception of data by the application protocol. In case of transport layer security, the message must be encrypted before it is sent.

Security mechanism through the transport layer is performed in three stages, which are as follows:

- ❑ The client and the server accede to utilize the same algorithm for encryption and decryption.
- ❑ A key is exchanged between the client and the server using the public key encryption and certificate-based authentication
- ❑ A symmetric cryptography chipper, which is an algorithm for encryption or decryption of data, is used while exchanging the information. It is symmetric, as it uses a key that is common during encryption and decryption of data. In the absence of a key, no result is produced during encryption or decryption of the data.

The transport layer security has various advantages and disadvantages. The advantages of transport layer security are as follows:

- ❑ Provides a standard approach for securing the Java EE 6 platform. Therefore, it is simple and easily understandable security, in comparison to the application layer security.
- ❑ Pertains to both the message body and the attachments.

The disadvantages of transport layer security are as follows:

- ❑ Remains tightly coupled with the transport layer protocol.
- ❑ Provides transient protection to messages during their transmission. The protection is automatically removed after the message leaves this layer.
- ❑ Provides point to point security; therefore, it is not an end solution for securing the systems.

Message Layer Security

When the message layer security mechanism is followed, the security information is transmitted to the appropriate recipient using a Simple Object Access Protocol (SOAP) message or SOAP message attachment. This SOAP attachment allows the message content to travel along the message or the attachment. In this case, a portion of the message may be signed by a particular sender and encryption may be provided for a specific receiver. The message is passed through intermediate nodes prior to its delivery to the receiver. The encrypted message remains opaque at intermediate nodes and can only be decrypted at the intended receiver. The message layer security is also known as the end-to-end security, as it completely secures the message content from the sender end to the receiver end, which was not possible in the transport layer security. This security mechanism also has certain advantages and disadvantages.

The advantages of the message layer security are as follows:

- ❑ Provides security that lasts until a message is received by the intended receiver
- ❑ Provides security to different portions of a message
- ❑ Provides interaction with intermediaries in case the message passes through multiple hops or devices in its way to the receiver over the network.
- ❑ Provides security to the message only, which is independent of the application environment or transport protocol

The disadvantages of message layer security are as follows:

- ❑ Is complex as compared to other security mechanisms
- ❑ Adds some overhead to processing in terms of operating cost

Now, after discussing each layer of security mechanism, let's discuss how to implement security on an application server.

Implementing Security on an Application Server

The deployment of a Web application on an application server offers highly secure, interoperable, and distributed component computing based on the Java EE security model. The security model is supported by the application server. The application server can be configured to perform the following tasks:

- ❑ Adding, deleting, and modifying authorized users. This can be done by adding realms, users, groups, and roles to the Web application.
- ❑ Configuring the HTTP and the Internet Inter-ORB Protocol (IIOP) listeners.
- ❑ Configuring the Java Management Extensions (JMX) connectors.
- ❑ Adding, modifying, and deleting existing or custom realms.
- ❑ Defining an interface using Java Authorization Contract for Containers (JACC) for the pluggable authorization providers. Security contracts between the application server and the authorization policy modules are provided with the help of JACC. These contracts specify the instructions for installing, configuring, and using authorization providers while making access decisions.
- ❑ Setting and changing an application's policy permissions.

To implement the security on the application server, it must be ensured that only authorized users are given access to the resources. Controlled access to protected resources is made possible with the help of authorization. Authorization is based on two concepts, which include identification and authentication. Identification is a technique using which a system can recognize an entity. Authentication is a process with the help of which it is possible to verify the user's identity, a device, or other entity in a computer system. You must perform the following steps to authenticate a user:

1. Write the code to prompt the users for their username and password
2. Provide information in the Deployment Descriptor to establish security for the application
3. Set authorized users and groups on the application server
4. Map the application's security roles to users, groups, and principals defined on the application server

Prior to creating an application that implements security, you need to understand how to work with realms, users, groups, and roles.

Realm

In terms of Java EE 6 application security, a realm is a database of users and groups that is used to identify valid users of a Web application. The authentication services available in Java EE 6 manage the realms available in an application server. In the application server, the admin-realm file and the certificate realm are configured. In the file realm, the server stores information of files in a file named keyfile. The file realm is used to check the authentication mechanism for entities. It is used for authenticating all clients, except the Web browser client containing the HTTP protocol and certificate.

In case of certificate realm, the server stores user's information in a certificate database. The application server uses the HTTP protocol and the certificates to authenticate clients. The Java EE authentication service verifies the identity of the user by verifying an X.509 certificate. The common field name of the X.509 certificate is utilized as the principal name to verify the identity of the user.

The admin-realm file can also be considered as a file realm, which is used to store user information in a local file named admin-keyfile. Users in an admin-realm are managed in the same way as they are managed in case of the file realm.

User

A user is an individual identity defined by the application server. It can have multiple roles corresponding with that identity. These multiple roles help the user to access all resources protected by the roles. The users can be related to a group. Note that a user in Java EE is same as a user in an operating system, but the security mechanisms involved in both the cases are different.

Group

A group is a collection of authenticated users, which have common traits specified in an application server. A Java EE user, who is also the user of the file realm, can belong to an application server group. An application server group is a category of user's type identified by the common traits, such as job title or customer profile. For example, it may be possible that majority of customers of an e-commerce application are associated with the CUSTOMER group, while the more profitable customers are associated with the PREFERRED group. In such

situations, to manage and control the access of large number of users, classifying users into groups is the best strategy.

Categorizing users into groups makes it easier to control the access of application components by the large numbers of users. An application server group possesses a wider scope of accessibility as compared to a role. A group is declared for all the applications that are deployed on the application server, while a role is related to a particular application deployed on the application server.

Role

To grant permission to access a particular collection of resources in an application, an abstract name called role is provided. A role may be considered as a key that is capable of opening a lock. Various users might possess a copy of the key, needed to open a lock; or the role, needed to access the application resources. An application's resources can be accessed by the original or the copy of the role.

After understanding implementation of security on the application server, let's understand security in the context of enterprise beans and application clients.

Securing Enterprise Beans

Enterprise beans are Java EE components that implement the EJB technology to create distributed, scalable, platform independent, and secure enterprise applications. They are executed inside the EJB container to provide security to the beans associated with the application. The EJB container provides application level security to enterprise beans associated with the application. These security services permit a user to rapidly build and deploy enterprise beans, forming the core of transactional Java EE applications. The declarative and programmatic security is used to protect enterprise beans from unauthorized and unauthenticated access. The resources present over the Internet that are protected from unauthorized and unauthenticated access include methods of EJBs, Web components and other enterprise beans. Figure 22.3 displays the basic configuration of a Java EE application:

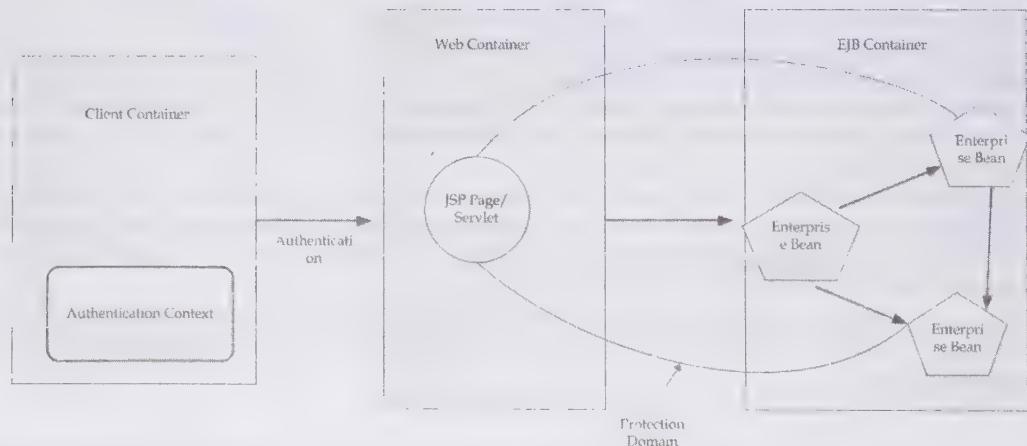


Figure 22.3: Showing the Configuration of a Typical Java EE Application

According to the Java EE 6 platform specification, there is no requirement of interoperable caller authentication at the EJB container. The direct connection of client containers and enterprise beans through Remote Method Invocation (RMI) is also avoided by the use of network firewall technology. The only way in which the EJB container can protect EJBs from unauthorized access is by entrusting the Web container to assure the user identity. This would restrict the accessibility of the enterprise beans through protected Web components only. Figure 22.3 shows how the protection domain boundaries are imposed for Web components, and enterprise beans that are called by the Web components.

Now, after learning about the basic configuration of a Java EE application, let's discuss about the role of programmatic approach and security identity propagation in implementing security in an application.

Using the Programmatic Security Approach

Enterprise beans can be secured programmatically by providing the code to implement security in the business methods of the bean. The programmatic security approach should be used when declarative security alone cannot provide full security to the enterprise bean. The programmatic security approach is particularly used in situations when security context information and caller's identity are to be retrieved by the enterprise bean business methods. The caller's identity is utilized to take security decisions. For example, you may need to grant access to the users of the enterprise application at a specific time of a day.

The following two methods of the javax.ejb.EJBContext interface are used to retrieve the security information of the caller of the enterprise bean:

- ❑ **javax.security.Principal getCallerPrincipal()**—Allows you to access the current caller principal's name. The `getCallerPrincipal()` method enables the enterprise bean's method to obtain the caller's principal name that can be used to identify the caller of the enterprise bean. The following code snippet shows the use of the `getCallerPrincipal()` method:

```
public void UseCallerPrincipal(...) {
    ...
    @Resource SessionContext myctx;
    @PersistenceContext EntityManager myem;

    // acquire the caller principal.
    mycallerPrincipal = myctx.getCallerPrincipal();

    // acquire the caller principal's name.
    studnameKey = mycallerPrincipal.getName();

    // use studnameKey as primary key to find StudentRecord
    StudentRecord myStudentRecord =
        myem.find(StudentRecord.class, studnameKey);

    // update address of Student
    myStudentRecord.setAddress(...);

    ...
}
```

In the preceding code snippet, the address of a student is updated. The `getCallerPrincipal()` method retrieves the callers' principal and stores it in the `mycallerPrincipal` variable, which is used to retrieve the caller's principal name with the help of the `getName()` method. The principal name of the caller is stored in the `studnameKey` variable. Then, the `EntityManager` instance uses this name to find the caller (student) record or the `StudentRecord` entity. In case student record is found, the address of the student is updated.

- ❑ **boolean isCallerInRole(String rName)**—Provides a mechanism to determine whether or not the current caller of the enterprise bean possesses the security role passed as a parameter to this method. Security roles are provided by the application assembler or enterprise bean developer. The security roles are assigned to principal groups of users. If the value of `rName` parameter matches with the security role specified in the Deployment Descriptor, the `isCallerInRole()` method returns true; otherwise, it returns false.

Using Security Identity

Security in enterprise beans can be implemented by propagating the security identity. To execute a specific method of an enterprise bean, security identity of a caller or the specific run-as identity of the enterprise bean is used. Consider a situation in which a Web client invokes an enterprise bean method in an EJB container. Consecutively, this enterprise bean's method invokes another enterprise bean method deployed in another container. In the call from the Web client to an enterprise bean method, the security identity propagated is the identity of the Web client. The security identity propagated to the target enterprise bean, in the call from an intermediate enterprise bean method to the target enterprise bean method, can be either of the following:

- ❑ **Web client**—Applies in situations when the trust exists between the target container and the intermediate container.

- ❑ **Specific run-as identity**—Applies in situations when the target container only supports specific run-as identity. The run-as identity represents the whole intermediate enterprise bean.

The run-as identity is applicable to the entire enterprise bean, which includes all the enterprise bean business methods, business interfaces, home interface, component interface, time-out callback method, and Web service endpoint interfaces. The run-as identity of a message-driven bean is applied to all its methods and message listener methods.

Securing Application Clients

The authentication mechanism for application clients is same as for the Java EE components or Java EE 6 application components. To access the protected Web resources, a user requires various authentication mechanisms, such as HTTP basic authentication, SSL client authentication, and HTTP login form authentication. These mechanisms are useful while accessing an enterprise Java bean in an application server.

Services offered by the application client containers, specifically used for authenticating the users, are utilized by the application client for authentication.

The users may be authenticated by the application container when an application server starts its operation. When the user accesses any protected resource in the application server, the container authenticates the user. An application client offers a class to hold the authentication data. To gather authenticated data, the application client must implement the `javax.security.auth.callback.CallbackHandler` interface. The class for gathering the data must be specified in the Deployment Descriptor. The application's callback handler must support callback objects defined in the `javax.security.auth.callback` package.

Now, let's discuss how to implement security in Web applications.

Implementing Security in Web Applications

You need to protect Web applications and their resources from malicious users who can harm the applications in numerous ways, such as disclosing confidential information, modifying or destroying original information, and using the resources inappropriately. Although it is not possible to completely secure a Web application, the threats to it can be reduced to an acceptable level by using security mechanisms, such as authentication and authorization. The Java platform provides various security technologies, which include a large set of Application Programming Interfaces (APIs), implementations for commonly used security mechanisms, and protocols. The Java security APIs provide various features including cryptography, public key infrastructure, secure communication, authentication, and access control, to secure the application from unauthorized and unauthenticated access.

This section describes various security mechanisms used in Web applications, such as authentication and authorization, and explains how these mechanisms are used to secure the applications. It also discusses various HTTP authentication mechanisms. In addition, it explains how to implement Web security using declarative security and programmatic security. However, before discussing these security mechanisms, let's first look at the Java Authentication and Authorization Service (JAAS) API, which is used to implement authentication in Web applications.

Using JAAS

JAAS API is used to implement authentication and authorization in Java applications. JAAS was introduced as an optional package in the Java 2 Platform, Standard Edition, version 1.3; and has been integrated into the Java 2 Platform, Standard Edition, version 1.4. The JAAS authentication is based on the Pluggable Authentication Module (PAM) framework or architecture, described later in the chapter. JAAS enables Java applications to be independent of the underlying technologies that are used for authentication purposes. The JAAS API is a collection of classes particularly used to make authentication services available and impose access controls on users. The JAAS API is used for three purposes:

- ❑ Finding out who is running the Java code, irrespective of whether the code is a standalone Java application, an applet, a bean, or a servlet

- ❑ Authorizing users to ensure that they possess the right permissions required to perform the respective actions
- ❑ Allowing implementation of data integrity

Let's now look at authentication, authorization, and data integrity in detail.

Implementing Authentication with JAAS

Authentication is the process of checking the validity of the security details submitted by a user. It is used by Web applications to relate a user identity to an HTTP session accessing a resource of the Web application. This resource may be protected; and authentication is used to validate the user identity and check whether the user is authorized to access the resource. To understand the concept of authentication better, let's consider the example of Internet banking or Web-based emails. In these cases, identity of a user is related to a real person and access depends on some proof of the identity based on authentication parameters.

An application that performs authentication is known as authenticator. An authenticator requires certain security details, which vary for different applications. Therefore, different authenticators (or different authentication technologies) are required to handle different security requirements. This, in turn, has brought in third party vendors to provide various authenticators. Apart from this, you can also develop your own authenticators using the JAAS API. The JAAS API provides a standard abstraction through which a Java application can communicate with any vendor-provided authenticators.

As already stated, JAAS authentication is based on the PAM framework. Figure 22.4 shows the authentication architecture of the PAM framework:

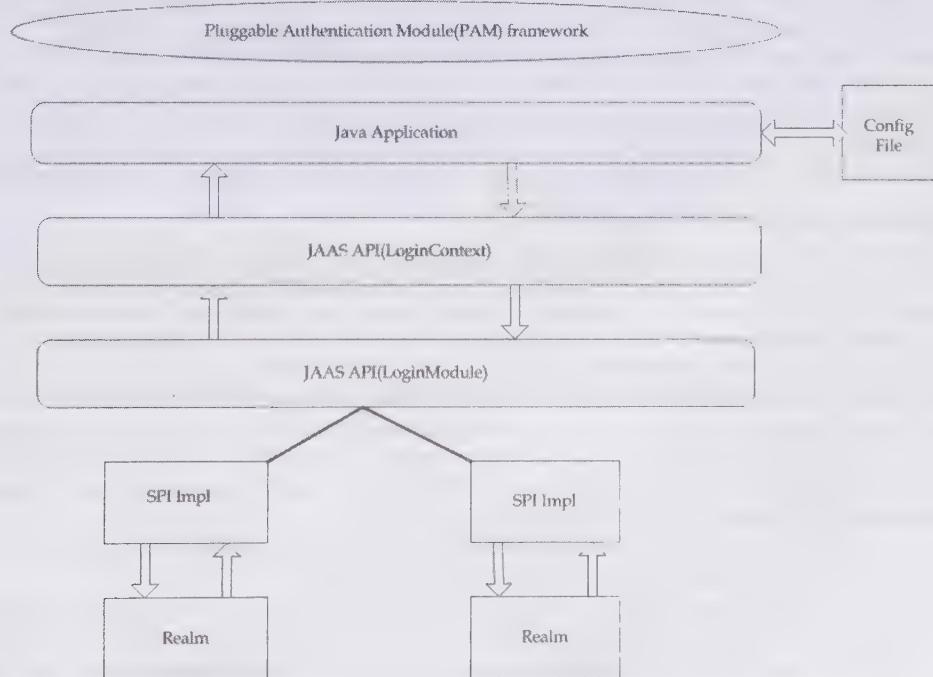


Figure 22.4: Showing the Authentication Architecture of the PAM Framework

As shown in Figure 22.4, when a Web application creates a `LoginContext` object and requests for login, the following operations are performed:

1. One or more `LoginModule` implementations are configured according to their names in the `Config file` associated with the `LoginContext` object. This is done during the creation of the `LoginContext` object.
2. When the `login` method is invoked on the `LoginContext` object, the new empty `javax.security.auth.Subject` object is created by the `LoginContext` object. The `LoginContext` object

- constructs the configured LoginModule object and initializes it with the Subject object and the given CallBackHandler object, which receives all JAAS events, and then returns the results back to JAAS.
3. The login method of the LoginContext object calls the methods in the LoginModule object to perform login and authentication. The LoginModule object utilizes the given CallBackHandler class to obtain the security details and check their validity.
 4. If authentication is successful, the LoginModule object creates a relation between the relevant principals (authenticated identities), credentials (authentication data such as cryptographic keys), and the Subject object.

The servlet specification provides support for the Web container to perform authentication through the <security-constraint> element in the Deployment Descriptor, web.xml. The servlet specification provides the following types of authentication:

- ❑ **HTTP basic authentication**—Refers to the authentication mechanism used to protect Web resources through specified username and password. This is the simplest authentication method supported by a Web application.
- ❑ **HTTP digest authentication**—Refers to the authentication mechanism in which the information is transmitted from the Web browser to the server in HTTP basic authentication containing a password. However, HTTP digest authentication is rarely used because only few Web browsers support this type of authentication.
- ❑ **Form-based authentication**—Provides a visual effect by using HTML in a Web application. It is implemented by using server-side session tracking, so the session can be invalidated when the user logs out.
- ❑ **HTTP client authentication**- Verifies identity of an end user using Public Key Certificate (PKC), which is an electronic document consisting of a digital signature to bind identity of the user with a public key. A public key is a value provided by some designated authority, such as Chartered Accountant (CA). PKC is used to verify that the public key belongs to an authenticated user, who is allowed access to the protected Web components.

You learn about HTTP authentication mechanisms later in this chapter. For now, let's discuss authorization.

Describing Authorization in Web Applications

Authorization is a process of ascertaining whether an authenticated user has permission to access a requested service. The application performing this operation is known as an authorizer. Authentication is different from authorization in that authorization determines whether the user has permission to access specific resources; whereas, authentication checks the identity of the user accessing the resources. The authorization process verifies whether the user has permission to access the resources by comparing the principal role submitted by the user during the logon process with the principal role of the same user existing in the authorization database.

In the servlet specification, the authorization model is role-based. This means that once a user is authenticated, he or she can access the resources based on his/her assigned role. When you develop a Web application, you should always keep in mind the kind of users who will access the application. For example, a Login module may be accessed by customers, administrators, and employees.

A security role is an abstract logical group of users provided by an assembler of the application. Users can be added or removed from a particular role to change their access rights.

The implementation of the authorization mechanism for the J2EE or JSP/servlet container is not defined by the servlet specification. Therefore, the implementation of authorization is specific to the Web container. The Glassfish application server supports a number of authorization realm implementations, such as an XML file, the relation database, and the Lightweight Directory Access Protocol (LDAP) connector. The authorization realm refers to a resource containing authorization details.

You can access Web resources with the help of the following authorization mechanisms:

- ❑ **Declarative authorization** – Allows you to implement access control rules enforced by a container in a Java EE application. The declarative authorization model is based on client-initiated requests.
- ❑ **Programmatic authorization** – Requires the implementation of authorization rules within the Java code of a Web application, but it uses the servlet security framework to authenticate the users.

These authorization models are discussed in detail later in this chapter. After discussing about authentication and authorization, let's now discuss data about integrity with reference to JAAS API.

Implementing Data Integrity with JAAS API

Data integrity is the property that restricts or prevents data from being modified during transmission. The servlet container is responsible for maintaining data integrity. Additionally, you can use the HTTP/SSL connection, which provides the data integrity feature, to provide security in the transport layer.

Now, after understanding the concept of security in the context of authorization, authentication, and data integrity, let's learn about HTTP authentication mechanisms.

Using Authentication Mechanisms

The authentication mechanism for a Web resource is configured in the web.xml file. This configured authentication mechanism is activated only when a user tries to access the protected Web resource. You can use the following authentication mechanisms to protect your resources:

- HTTP basic authentication
- Form-based authentication
- Client-certificate authentication or HTTPS client authentication
- HTTP digest authentication

NOTE

If you do not specify any of these mechanisms in the web.xml file, authentication will not be performed.

Let's now discuss these mechanisms in detail, one by one.

Describing HTTP Basic Authentication

The HTTP basic authentication mechanism is the simplest way of implementing authenticity in Java EE 6 applications. In this authentication mechanism, the resource requested by the user is accessible by providing the valid username and password. Figure 22.5 shows the process of HTTP basic authentication:

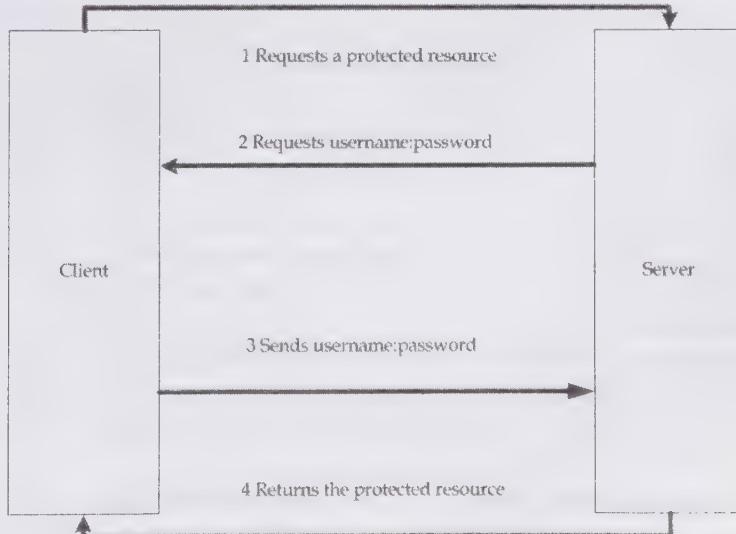


Figure 22.5: Showing HTTP Basic Authentication

As shown in Figure 22.5, the following events occur in HTTP basic authentication:

1. A client requests access to a protected resource

2. The Web server requests the client's name and password of the client, which are entered through a dialog box
3. The Web server validates the user credentials
4. If successful, the server returns the requested resource

There is a drawback of using the HTTP basic authentication. It is not secure because it transfers usernames and passwords over the Internet as Unix-to-Unix encoded (UU-encoded) encrypted text, which can be easily decoded.

Describing Form-Based Authentication

In form-based authentication, the username and password of a client are transferred in form of plain text, and no authentication is enforced on the target server. This authentication mechanism can disclose usernames and passwords, which can be easily decoded unless the connections are established over SSL. SSL is a technology that permits secure data transmission from the Web server to the Web browser. With the help of form-based authentication, you can alter the login and error pages of an application, which are displayed for the user. Figure 22.6 shows the working of form-based authentication:

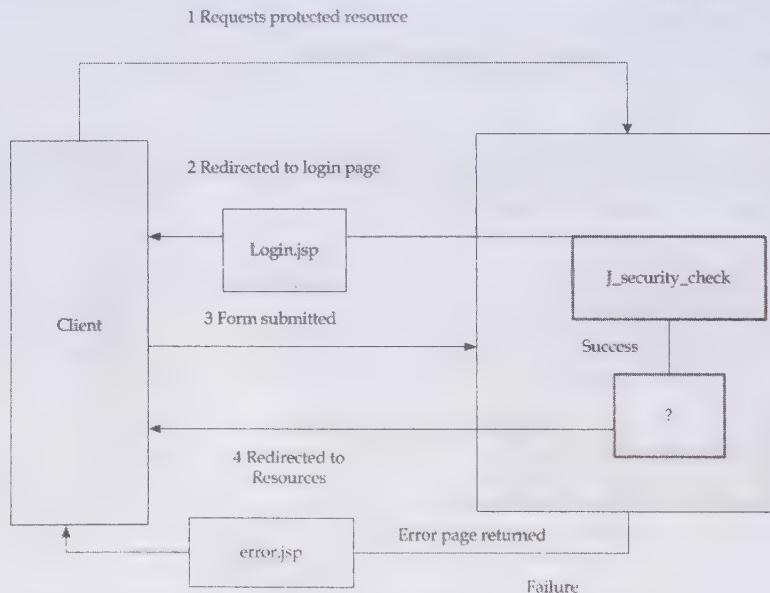


Figure 22.6: Showing Form-Based Authentication

As shown in Figure 22.6, the following events occur in the form-based authentication:

1. A client requests access to a protected resource.
2. If the client is authenticated, the server redirects the client to the login page.
3. The client submits the login form to the Web server.
4. If the login is successful, the server redirects the client to the requested resource. However, if the login fails, the client is redirected to an error page.

Describing Client-Certificate Authentication or HTTPS Client Authentication

The client-certificate authentication or HTTPS client authentication mechanism is the advanced authentication method which is more secure than the basic and form-based authentication mechanisms. This is because it uses HTTP over SSL (HTTPS).

In the client-certificate authentication, the server and, sometimes, the client, authenticate each other by using public key certificates. A public key certificate is a digitally signed document used to validate a username and other user credentials. It is provided by a trusted organization called certificate authority (CA), and gives

identification for the bearer. HTTPS offers data encryption, server authentication, and message integrity. HTTPS may also offer client authentication for a Transmission Control Protocol/Internet Protocol (TCP/IP) connection.

If you specify client-certificate authentication, the Web server authenticates the client by using the client's X.509 certificate, which is a public key certificate that conforms to a standard defined by X.509 Public Key Infrastructure (PKI). Before running an application that uses SSL (HTTPS), you must configure SSL (HTTPS) support on the Web server and set up the public key certificate.

Describing HTTP Digest Authentication

Similar to HTTP basic authentication, HTTP digest authentication authenticates users based on their username and password. In the HTTP digest authentication, the password is transmitted as encrypted data and in basic authentication; password is transmitted as simple base64 encoded data. You should note that transferring encrypted data is more protected than transferring base64 encoded data. Therefore, the HTTP digest authentication mechanism is more secure than the HTTP basic authentication mechanism. Digest authentication is currently not widely used since it is a new HTTP 1.1 feature and not supported by all browsers. If a non-compliant browser makes a request on a server that requires digest authentication, the server rejects the request and sends an error message.

After learning about various authentication mechanisms, let's learn about Web security, and see how to implement it in Web applications by using the different authentication mechanisms.

Implementing Security

A Web application consists of various resources, which can be used by multiple users. These resources often move through unprotected and open networks, such as the Internet. Consequently, a Web application needs to implement security. Web containers provide support to implement security for Web applications through the JAAS API. As already learned, Web security can be implemented using the following two ways:

- Declarative security
- Programmatic security

Let's discuss these two mechanisms in detail.

Describing Declarative Security

In declarative security, application's security structure can be demonstrated by a Web application provider. The Web application may include roles, access control, and authentication requirements, in a form that is external to the application. To prevent unauthorized access, the Web application's Deployment Descriptor (web.xml) is used to declare the Uniform Resource Locators (URLs) that need protection. You can also declare an authentication method, which the server uses to identify the users requesting the resources of the Web application. The server prompts the users for username and password when they access restricted resources, validates the user credentials against a pre-defined set of usernames and passwords, and keeps track of previously authenticated users. These processes are performed in a transparent manner in servlets and JSP pages.

Declarative security is server-based in most cases. This means that the server's configuration is used to provide protection to a resource or a set of resources. To secure a Web application's resources against unauthorized access, a Web container provides the authentication and authorization schemes. The authentication and authorization scheme is applied to the static content of a Web application and to the dynamic code (servlets, filters and JSP pages) requested by a client within the application.

Now, let's create an application that implements the declarative security.

Using Declarative Security

In this subsection, we create a simple application, simpledeclarativeex, to demonstrate the implementation of declarative security through the basic authentication mechanism. In this application, the username and password dialog box appears when a user submits an HTML form. The validation of the credentials is done by the server and on successful validation; the protected resource requested by the user is displayed. Otherwise, an

error message is displayed. You can find this application on the CD in the code\JavaEE\chapter22\simpleservletex folder.

Now, perform the following broad-level steps to create the simpleservletex application:

1. Create a Web client for HTTP basic authentication
2. Configure the application
3. Create the Web resource servlet
4. Create roles and users
5. Explore the directory structure of the application
6. Run the application

Creating a Web Client for HTTP Basic Authentication

In this subsection, we create a Web client, Hello.html, which initiates the request for a protected resource. Listing 22.1 provides the code for the Hello.html file. The Hello.html file creates two HTML submit type buttons. When a user submits a form by clicking either of these buttons, the control is forwarded to the URL path testser. The mapping between the URL path tester and TestServlet servlet is provided in Listing 22.2. Listing 22.1 shows the code for the Hello.html file (you can also find this file on the CD in the code\JavaEE\Chapter 22\simpleservletex folder):

Listing 22.1: Showing the Code for the Hello Page

```
<html>
  <body>
    <form method=post action="testser">
      <input type="submit" value="Make a POST Request to access TestServlet"/>
      </form><br><br>
      <form method=get action="testser">
        <input type="submit" value="Make a GET Request to access TestServlet"/>
      </form>
    </body>
  </html>
```

In Listing 22.1, Hello.html is the Web client for HTTP basic authentication. Let's now learn how to configure the simpleservletex application and implement the HTTP basic method of authentication.

Configuring the Application

To configure this application, you need to use the web.xml file. This file maps the action URL path testser forwarded by Home.html with the TestServlet class, and defines the security constraints. Listing 22.2 provides the code for the web.xml file (you can also find this file on the CD in the code\JavaEE\Chapter22\simpleservletex\WEB-INF folder):

Listing 22.2: Showing the Code for the web.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <servlet>
    <servlet-name>ts</servlet-name>
    <servlet-class>com.kogent.servlets.TestServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ts</servlet-name>
    <url-pattern>/testser</url-pattern>
  </servlet-mapping>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>myres</web-resource-name>
      <url-pattern>/testser</url-pattern>
      <!--<http-method>POST</http-method>-->
      <!--Uncomment the above line to perform security
          check only for HTTP POST method request-->
```

```

</web-resource-collection>
<auth-constraint>
    <role-name>myrole1</role-name>
</auth-constraint>
</security-constraint>
<login-config>
    <auth-method>BASIC</auth-method>
</login-config>
<security-role>
    <role-name>myrole1</role-name>
</security-role>

<welcome-file-list>
    <welcome-file>Hello.html</welcome-file>
</welcome-file-list>

</web-app>

```

In Listing 22.2, the `<security-constraint>` and `</security-constraint>` tags define the Web resource being protected. In Deployment Descriptor provided in Listing 22.2, the `<auth-constraint>` element is used to protect the Web resources, having the `testser` url-pattern defined using the `<url-pattern>` element. This protection is specified for the users that are assigned the `myrole1` role. The users are assigned `myrole1` role using `<role-name>` sub element of `<security-role>` element of the Deployment Descriptor. In the `<auth-constraint>` element, role name is specified using the `<role-name>` element.

The `<security-role>` element defines the logical grouping of the users defined by the application developer or assembler. Note that the authentication method specified in Listing 22.2 is `BASIC`.

The `com.kogent.servlets.TestServlet` class is mapped to the URL path `testser`. Therefore, the `TestServlet` class is a protected Web resource for the users assigned the `myrole1` role group. The following code snippet shows how to map the group role in the `sun-web.xml` file:

```

<security-role-mapping>
    <role-name>myrole1</role-name>
    <group-name>myrole1</group-name>
</security-role-mapping>

```

In the preceding code snippet, the `myrole1` group is mapped to the `myrole1` role.

Creating the Web Resource Servlet

As discussed, `TestServlet` is the Web resource accessed by users defined under the `myrole1` role group. The `TestServlet.java` file is created in the `com.kogent.servlets` package. Listing 22.3 provides the code for the `TestServlet.java` file (you can also find this file on the CD in the `code\JavaEE\Chapter22\simpleservletiveex\src\com\kogent\ servlets` folder):

Listing 22.3: Showing the `TestServlet.java` File

```

package com.kogent.servlets;

import javax.servlet.*;
import javax.servlet.http.*;
import java.security.*;
import java.io.*;

public class TestServlet extends HttpServlet
{
    public void service(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        Principal p=req.getUserPrincipal();
        String principal_name="UnAuthenticated";
        if (p!=null)
            principal_name=p.getName();
        PrintWriter out=res.getWriter();
        out.println("User <b>"+principal_name+"</b> has requested for TestServlet");
    }// end service
}// end class

```

In Listing 22.3, TestServlet is the HttpServlet class, which calls the getUserPrincipal () method on the request object. The getUserPrincipal () method returns a java.security.Principal object, which holds the current authenticated user's name.

Compile the TestServlet.java file and save it under the appropriate folder, as shown in Figure 22.7.

After compiling TestServlet.java, start the Glassfish application server to create roles and users for the simpledeclarativeex application.

Creating Roles and Users

To create roles and users for the simpledeclarativeex application, perform the following steps:

1. Navigate to the `http://localhost:4848` URL. This URL displays the Glassfish server administrator's Console page. Now, use either the administrator's username and password or the default username `admin` and password `adminadmin` to log on to the Admin Console.
2. Expand the Configuration node in the Admin Console and further expand the Security node and then the Realms node, as shown in Figure 22.7:

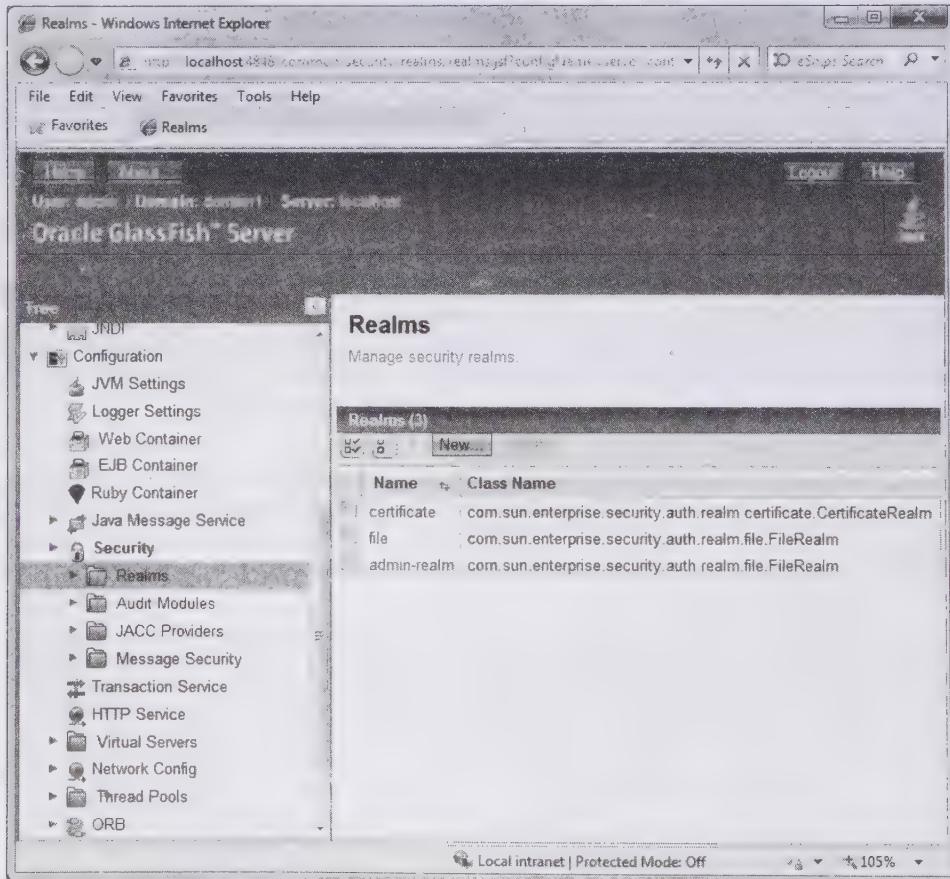


Figure 22.7: Displaying the Realms Pane

3. Select the file realm displayed in the Realms pane. The Edit Realm pane appears, as shown in Figure 22.8.

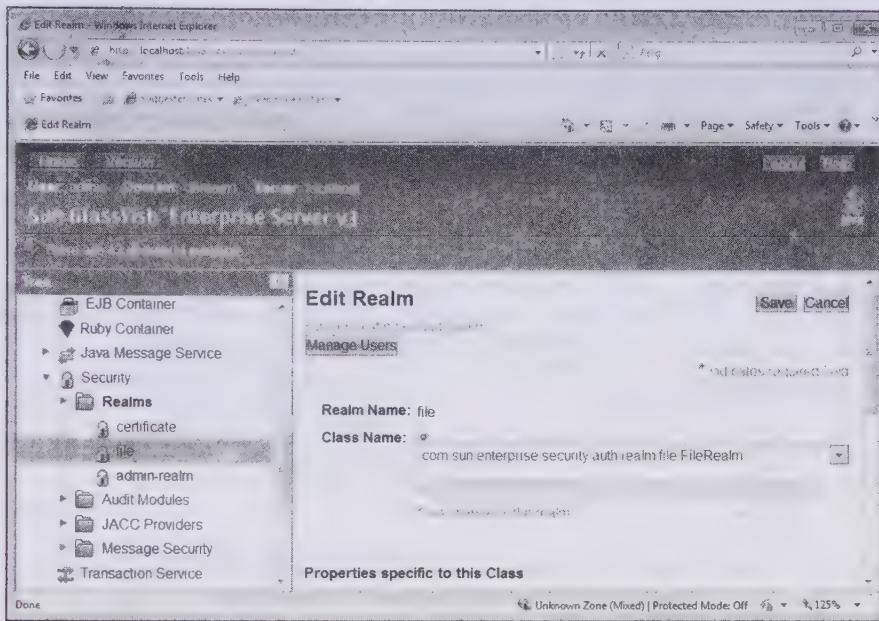


Figure 22.8: Displaying the Edit Realm Pane

- Click the Manage Users button (Figure 22.8), as a result the File Users pane appears as shown in Figure 22.9:

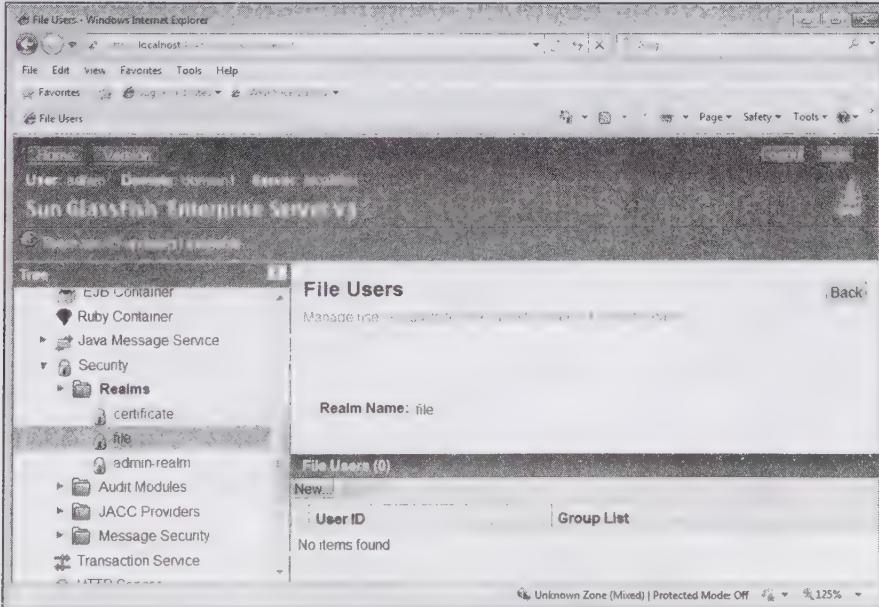


Figure 22.9: Displaying the File Users Pane

- Click the New button to create a new user, the New File Realm User pane appears (Figure 22.10).
- Enter the User Id, Password, and Group List in the New File Realm User pane. In our case, we have entered santosh as User ID, myrole1 as New Password, and myrole1 as Group List.
- Click the Ok button to add the user to the users list in the realm, as shown in Figure 22.10;

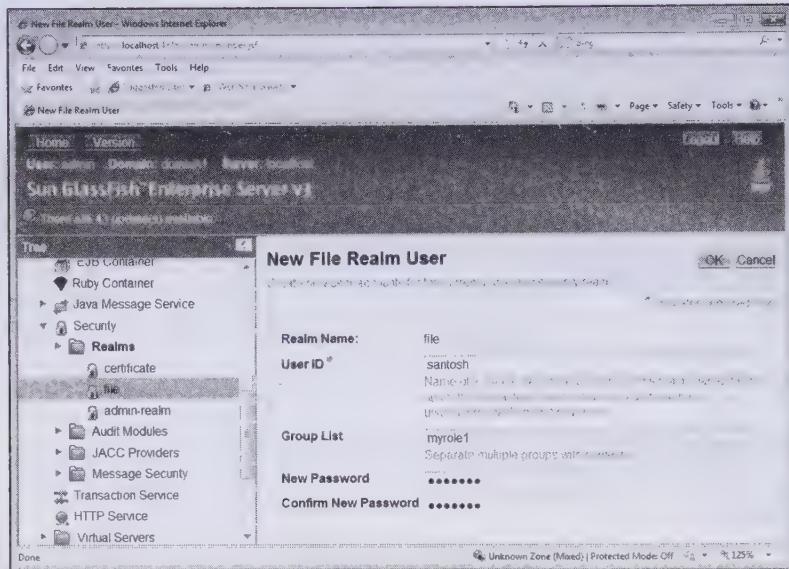


Figure 22.10: Displaying the New File Realm User

There are three types of realms under the realms pane, which are as follows:

- ❑ **File** – Allows adding users who have the permission to access applications running in this realm
- ❑ **Admin-realm** – Allows adding users who have the permission to become system administrators of the application server
- ❑ **Certificate** – Allows adding certificates to the certificate realm

Now it's the time to run the application, but before that you must learn about the directory structure of the simpledeclarativeex application which is discussed next.

Exploring the Directory Structure of the Application

The directory structure of the simpledeclarativeex application is shown in Figure 22.11:

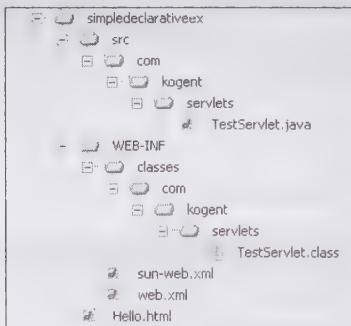


Figure 22.11: Displaying the Directory Structure

To create this structure, you need to create a folder for your application say, simpledeclarativeex (Figure 22.11). Now, save the different types of files at their proper locations in the directory structure in the following manner:

- ❑ Save all packages containing class files in the WEB-INF\classes folder
- ❑ Save the web.xml file in the WEB-INF folder

You can also create a src\com\kogent\servlets folder containing the source files (.java files) for all class files under the simpledeclarativeex folder. However, this folder is optional in the simpledeclarativeex application and you can save the source files of this folder at any other location.

The simpledeclarativeex folder is the root folder containing the src folder, the WEB-INF folder, and the Hello.html file (Figure 22.11). The WEB-INF folder has the classes folder, and two XML files, which are web.xml and sun-web.xml. As discussed previously, the WEB-INF\classes folder contains the TestServlet class file, with fully specified package. The WEB-INF\src\com\kogent folder is an optional folder containing the source file (TestServlet.java). The web.xml file designates the authentication method, which is used by the server to identify users.

Let's now run the application.

Running the Application

To run the simpledeclarativeex application, type the `http://localhost:8080/simpledeclarativeex/Hello.html` URL in your browser. As a result, the hello.html page appears, as shown in Figure 22.12:

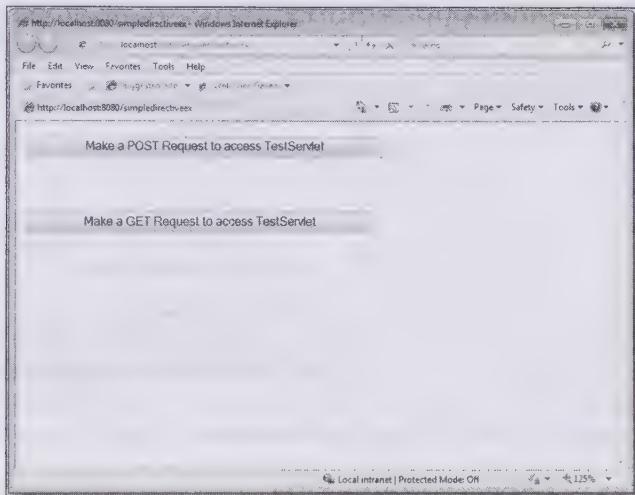


Figure 22.12: Displaying the Hello.html Page

The buttons displayed in Figure 22.12 are used to make requests through the GET and POST methods. Clicking either of these buttons displays the associated dialog box with blank username and password fields, as shown in Figure 22.13:

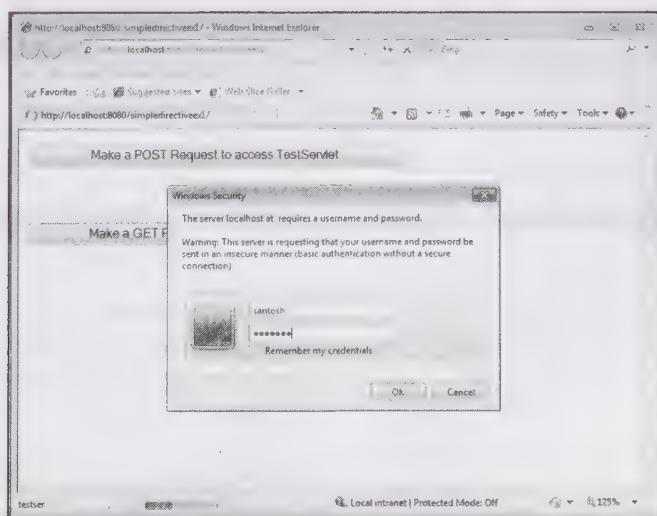


Figure 22.13: Displaying the Windows Security Dialog Box

In this dialog box, enter the username as santosh and password as myrole1. If the given username and password are valid, the user will be able to access TestServlet, as shown in Figure 22.14:

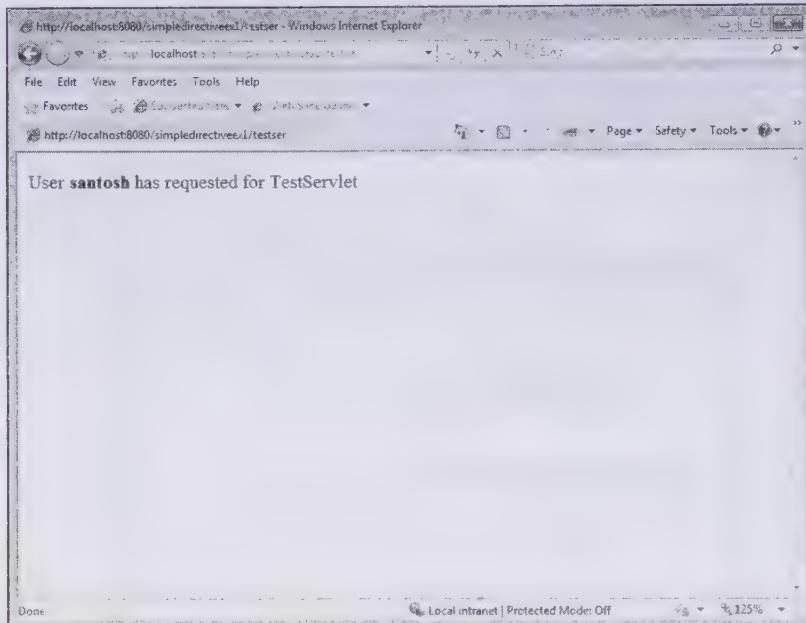


Figure 22.14: Displaying the Response after Accessing TestServlet

Figure 22.14 shows the result of entering a valid username and password. However, if this information is incorrect, the server displays a 401 error, as shown in Figure 22.15:

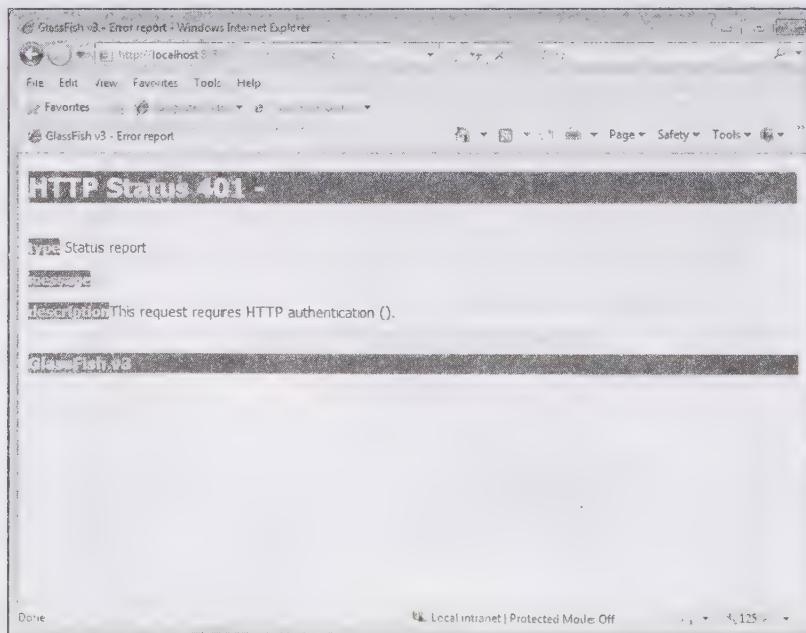


Figure 22.15: Displaying the Error Message for Incorrect User Details

Alternatively, if the username and password are correct, but the user is not associated with the myrole1 role (for example, if we type the username as admin and leave the password field blank), the server displays the Windows Security dialog box, as shown in Figure 22.16:

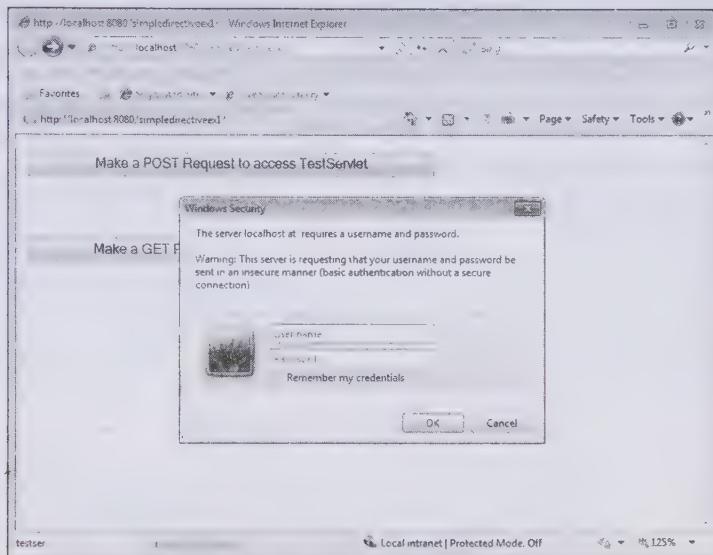


Figure 22.16: Displaying the Windows Security Dialog Box

NOTE

In this case, the user is authenticated but is not authorized to access the request resource.

Now, if you add <http-method>POST</http-method> in the web.xml file, as shown in Listing 22.2, and run the application, a security check is performed only in the case of the HTTP POST method and not in the default HTTP GET method request. A request made by using the HTTP GET method displays the response, as shown in Figure 22.17:

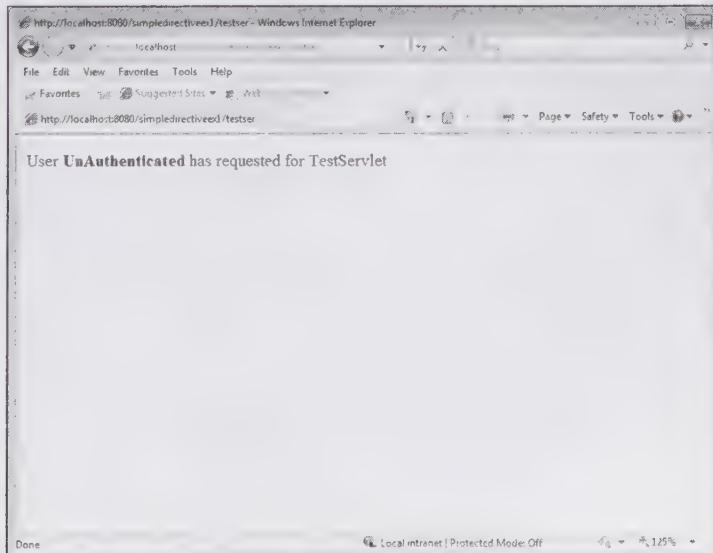


Figure 22.17: Displaying the Response for the HTTP GET Method

In this application, we used the `<auth-method>BASIC</auth-method>` element to implement basic authentication. However, if you want to display your own login page instead of a dialog box, you can use the `<auth-method>FORM</auth-method>` element.

Let's now learn how to implement the form-based authentication.

Using the Form-based Authentication for Declarative Security

Next, let's convert the simpledeclarativeex application from HTTP basic authentication to form-based authentication. For this, you need to perform the following broad-level steps:

1. Change the web.xml file for form-based authentication
2. Create the Login.jsp and MyError.jsp files
3. Change the directory structure
4. Run the application

Changing the web.xml File

To implement form-based authentication in the simpledeclarativeex application, the `<login-config></login-config>` element defined in Listing 22.2 must be changed to provide form-based authentication. Listing 22.4 provides the updated code for the web.xml file:

Listing 22.4: Showing the Code for the web.xml File in Form-based Authentication

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>/Login.jsp</form-login-page>
      <form-error-page>/MyError.jsp</form-error-page>
    </form-login-config>
  </login-config>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>myres</web-resource-name>
      <url-pattern>/testser/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>myrole1</role-name>
    </auth-constraint>
  </security-constraint>
  <servlet>
    <servlet-name>ts</servlet-name>
    <servlet-class>com.kogent.servlets.TestServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ts</servlet-name>
    <url-pattern>/testser/*</url-pattern>
  </servlet-mapping>
  <security-role>
    <role-name>myrole1</role-name>
  </security-role>
  <welcome-file-list>
    <welcome-file>Home.html</welcome-file>
  </welcome-file-list>
</web-app>
```

In Listing 22.4, the authentication method has been changed from `BASIC` to `FORM`. `Login.jsp` is declared as the login page by using the `<form-login-page>` element of the `<form-login-config>` tag; and the `MyError.jsp` page is declared as the error page by using the `<form-error-page>` element of the `<form-login-config>` tag.

Let's now create the login form and error pages for the application.

Creating the Login.jsp and MyError.jsp Files

While discussing form-based authentication, we have seen that a login page is displayed when a user requests for a protected resource. When the user submits the login page, the server verifies the credentials. If the login is successful, the requested protected resource is displayed; otherwise, an error page appears. Therefore, to implement form-based authentication in the simpledeclarativeex application, you must create the login and error pages. Listing 22.5 provides the code for the Login page (you can also find the Login.jsp file on the CD in the code\JavaEE\Chapter22\simpledirectiveex folder):

Listing 22.5: Showing the Code for the Login.jsp File

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    </head>
  <body>
    <form action="j_security_check" method="POST">
      Username:<input type="text" name="j_username"><br>
      Password:<input type="password" name="j_password">
      <input type="submit" value="Login">
    </form><br>
  </body>
</html>
```

Listing 22.5 shows a login form that consists of two textboxes, to let the user enter the username and password, respectively. As seen in Listing 22.5, the form action is j_security_check; and the field names, j_username and j_password, which take the username and password, respectively.

Next, we create an error page to display an error message for unauthenticated usernames and passwords. Listing 22.6 provides the code for the MyError.jsp file (you can also find this file on the CD in the code\JavaEE\Chapter22\simpledirectiveex folder):

Listing 22.6: Displaying the Code for the MyError Page

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Login Test: Err. logging in</title>
  </head>
  <body>
    <h1>User Name Password entered are not valid</h1>
    <br/>
  </body>
</html>
```

In Listing 22.6, a JSP page is created to display an error message for unauthenticated users.

The directory structure of the simpledeclarativeex application is changed to implement form-based authentication.

Changing the Directory Structure

The directory structure for the simpledeclarativeex application, shown in Figure 22.11, has to be changed; since we have added two JSP pages to the application. Figure 22.18 shows the modified directory structure:

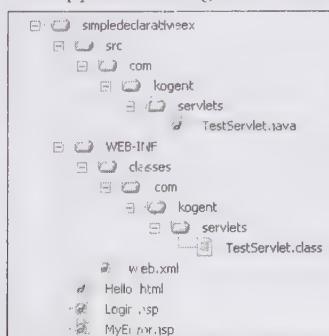


Figure 22.18: Displaying the Changed Directory Structure

In Figure 22.18, the two JSP pages, Login.jsp and MyError.jsp, have been added under the simpledeclarative folder.

Now, let's run the simpledeclarativeex application to see its output.

Running the Application

To run the simpledeclarativeex application, open the Web browser and type the <http://localhost:8080/simpledeclarativeex>Hello.html> URL.

The Hello.html page is displayed (Figure 22.12). After clicking any of the two buttons of the Hello.html page, the Login page is displayed, as shown in Figure 22.19:

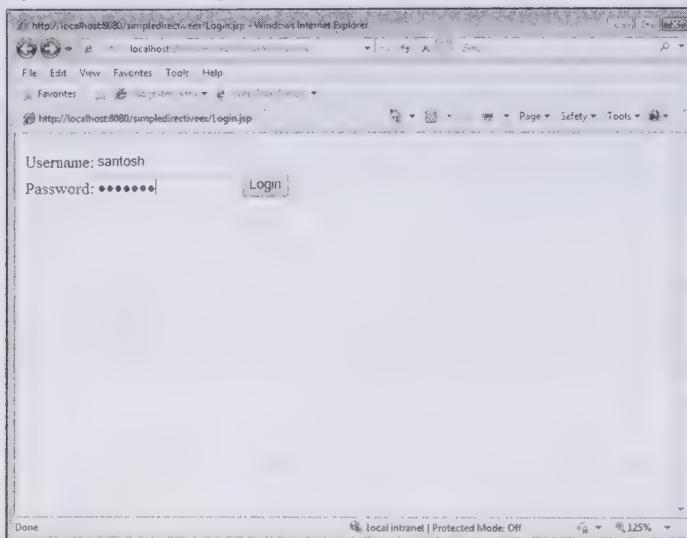


Figure 22.19: Displaying the Login Form for Form-Based Authentication

Figure 22.20 is displayed when a user enters santosh in the username and myrole1 in the password fields, respectively:

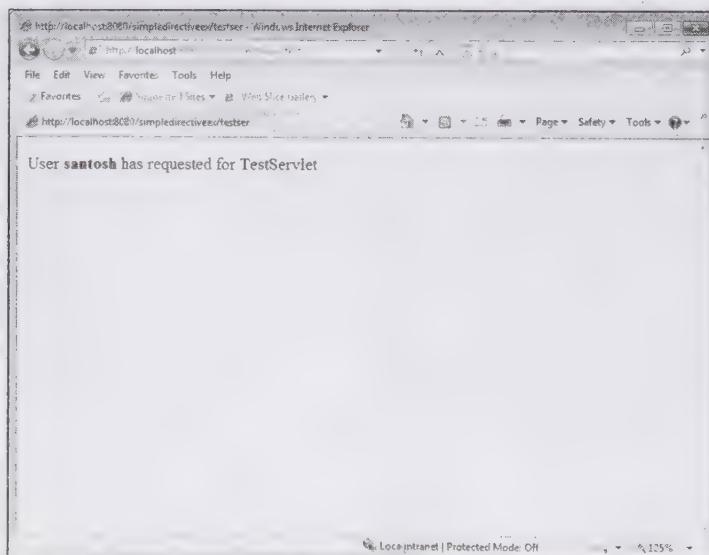


Figure 22.20: Displaying the Form-Based Authentication Response

We have learned that declarative security only verifies the user credentials. To implement security for accessing resources, we should implement programmatic security, which is discussed next.

Implementing Programmatic Security

In case of programmatic security, a Web application is responsible for authorizing a client; that is, the Web application verifies whether or not the client has permissions to access the resource(s). This mechanism is called programmatic security because the security implementation is coded into the Web application programmatically. The servlet specifications include some methods in the `HttpServletRequest` interface to implement programmatic security. These methods allow servlets to take business logic decisions based on the user's information. Some methods of the `HttpServletRequest` interface used in programmatic security are:

- ❑ **String getRemoteUser()**—Returns the username provided by a client for authentication. If the user is not authenticated, this method returns `null`.
- ❑ **Principal getUserPrincipal()**—Returns a `java.security.Principal` object. The `Principal` object is used to get the `Principal` name of the user. If the user is not authenticated, this method returns `null`.
- ❑ **boolean isUserInRole(String)**—Checks if a user is included in the specified security role. If the user is not authenticated, this method returns `false`. The `isUserInRole()` method takes a string, which has the same value as that of the role-name parameter.

The Deployment Descriptor contains the `<security-role-ref>` element with the `<role-name>` sub-element, which must be declared for a particular role. The `<role-name>` sub-element denotes the role name that is passed as an argument to the `isUserInRole (String rolename)` method. The `<security-role-ref>` element must contain the `<role-link>` sub-element, signifying the security role name, with which the user is mapped to. The Web container utilizes the mapping provided within the `<security-role-ref>` tag to determine the return value of the call to the `HttpServletRequest.isUserInRole (String rolename)` method. The following code snippet shows how to define the `myrole1ref` role using the `<role-name>` element and map the role with the `myrole1` role link using the `<role-link>` element:

```
<security-role-ref>
<role-name>myrole1ref</role-name>
<role-link> myrole1</role-link>
</security-role-ref>
```

According to this mapping, only a user belonging to the `myrole1` security role can call the specified servlet. To check the role of the user, you can use the `isUserInRole ("myrole1ref")` method, which returns the Boolean value, `true`.

If the `security-role-ref` elements are not declared, the Web container uses default role references to verify the role-names of the `security-role` elements for a Web application. The `isUserInRole()` method references the security roles defined in the `web.xml` file to determine whether or not the caller is mapped to a security role.

To enforce more control to access a Web application and its resources, a user-defined security model is implemented by a developer. In spite of the fact that a Web application developer configures the programmatic security, programmatic security usually communicates with the same systems, similar to the declarative security.

More fine-grained security is provided by the programmatic security, as compared to the declarative security; however, programmatic security can reduce a Web application component's reusability. Integrating various components to form a Web application, in which each application component uses the programmatic security, is difficult in case the programmed security model, which provides programmatic security, is inconsistent between the application components. Another drawback of using programmatic security is that whenever there is change in the security policy, each protected component must be revised and modified with new security authorization information.

Imagine a real-life scenario of a company with many users performing multiple tasks, such as adding and modifying employee records, and maintaining transaction records. Users can perform these tasks according to the roles assigned to them. For example, a clerk working in the organization does not have the right to make additions to the employee records. This can be done only by the manager of the organization. Therefore, the manager, as a user, is assigned the `admin` role, authorizing him or her to make changes in the employee records.

In this subsection, we create an application named `programmaticex`, to authenticate users based on their username and password. An invalid username or password displays an error message. A successful login allows users to perform the tasks based on the roles assigned to them. For example, users who are assigned the administrator role can modify or add the records of the employees, but if their assigned role is that of a clerk's role, they will be prevented from making such changes. You can find the code files of this application on the CD in the `code\JavaEE\Chapter22\programmaticex` folder.

Perform the following broad-level steps to create the `programmaticex` application:

1. Create the JSP pages for programmatic security
2. Create the Login.java and User.java files
3. Configure the application
4. Explore the directory structure of the application
5. Run the application

Creating JSP Pages

In the `programmaticex` application, various JSP pages are used to validate user details, add employee details, view employee details, and display the welcome page to authenticated users. In the application, we need six JSP pages to perform different tasks, which are as follows:

- ❑ **index.jsp** – Displays the login screen for users to enter their username and password.
- ❑ **validateuser.jsp** – Redirects users to the validateuser page. The validateuser page uses the Login class to verify the user.
- ❑ **retry.jsp** – Displays an error message if an invalid username or password is entered. The error message prompts the user to enter a valid or correct username and password.
- ❑ **welcome.jsp** – Displays a welcome message and provides the reference for the addemployee and viewemployee pages if the login is successful.
- ❑ **addemployee.jsp** – Redirects a user to the addemployee page. If the entered username has been assigned an administrator role, the form to add employee records is displayed; otherwise, a message of non-authorization is displayed.
- ❑ **viewemployee.jsp** – Displays a table with the name of the employees and their designation.

Let's now create these pages.

Creating the index.jsp File

The `index.jsp` file displays the login screen for entering the username and password. Listing 22.7 provides the code for the `index.jsp` file (you can also find this file on the CD in the `code\JavaEE\Chapter22\programmaticex` folder):

Listing 22.7: Displaying the Code for the `index.jsp` File

```
<%@ page language="java" session="true" pageEncoding="ISO-8859-1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>Programmatic Security</title>
    </head>
    <body>
        <form action="validateuser.jsp" method="POST" name="theForm">
            <table align="center">
                <tr>
                    <td><font size=2> Enter the User Name </font> </td>
                    <td><input type="text" name="userName"> </td>
                </tr>
                <tr>
                    <td><font size=2> Enter the Password </font></td>
                    <td><input type="password" name="password"></td>
                </tr>
                <tr>
                    <td>
```

```

        <br/>
        <input type="submit" value="Submit"/>
        <input type="reset" value="Reset"/>
    </td>
</tr>
</table>
</form>
</body>
</html>

```

Creating the validateuser.jsp File

As shown in Listing 22.7, two textboxes, `userName` and `passWord`, are created to accept the username and password from the user, respectively. When the user enters these details and submits the form, the `validateuser.jsp` file is displayed. Listing 22.8 provides the code for the `validateuser.jsp` file (you can also find this file on the CD in the `code\JavaEE\Chapter22\programmaticex` folder):

Listing 22.8: Displaying the Code for the `validateuser.jsp` File

```

<%@ page language="java" session="true" pageEncoding="ISO-8859-1"%>
<jsp:useBean id="idHandler" class="com.kogent.login.Login" scope="request">
<jsp:setProperty name="idHandler" property="*"/></jsp:useBean>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>Programmatic Security</title>
    </head>
    <body>
        <%
            String userName = request.getParameter("userName");
            ServletContext context=getServletContext();
            context.setAttribute("name",userName);
            String password = request.getParameter("password");
            if(idHandler.authenticate(userName, password))
            {
                response.sendRedirect("welcome.jsp");
            }
            else
            {
                response.sendRedirect("retry.jsp");
            }
        %>
    </body>
</html>

```

The `validateuser.jsp` file implements authentication by verifying the username and password entered by the user, with the `username` and `password` defined in the `Login` class.

In Listing 22.8, `idHandler` is created to handle the `Login` class, defined in the `com.kogent.login` package. The `username` and `password` entered by the user in the `index.jsp` file is retrieved in the `validateuser.jsp` file, and the `authenticate()` method of the `Login` class is then invoked. The `authenticate()` method of the `Login` class is discussed later in the chapter.

Creating the `retry.jsp` File

The `authenticate()` method in Listing 22.8 returns a boolean value; if this value is true, the `welcome.jsp` file is called; otherwise, the `retry.jsp` file is invoked. The `retry.jsp` file displays an error message for invalid username and password, and prompts the user to reenter the details. Listing 22.9 shows the code for the `retry.jsp` file (you can also find this file on the CD in the `code\JavaEE\Chapter22\programmaticex` folder):

Listing 22.9: Displaying the Code for the `retry.jsp` File

```

<%@ page language="java" session="true" pageEncoding="ISO-8859-1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>

```

```

<h1>
    Invalid User name and Password.
    Please Try Again!!!!
</h1>
<jsp:include page="index.jsp"/>
</body>
</html>

```

In Listing 22.9, the retry.jsp file displays an error message and includes the output of the index.jsp file.

Creating the welcome.jsp File

The welcome.jsp file is displayed if the username and password are correct and the login is successful. Listing 22.10 provides the code of the welcome.jsp file (you can also find this file on the CD in the code\JavaEE\Chapter22\programmaticex folder):

Listing 22.10: Displaying the Code for the welcome.jsp File

```

<%@ page language="java" session="true" pageEncoding="ISO-8859-1" %>
<html>
    <head>
        <title>Programmatic Security</title>
    </head>
    <body>
        <br/>
        <h1>welcome!!!! </h1>
        <h2>You have successfully Logged In</h2>
        <a href="adddemployee.jsp">Add Employee</a><br/>
        <a href="viewemployee.jsp">View Employee</a>
    </body>
</html>

```

Creating the adddemployee.jsp File

The welcome.jsp file in Listing 22.10 displays a welcome message and the reference link for the Add Employee and View Employee pages. Clicking the Add Employee link displays the adddemployee.jsp file. This page determines the role assigned to the user based on the username entered by him or her. In other words, the adddemployee.jsp file implements the authorization. If the username entered by the authenticated user has been assigned the administrator role, the employee form is displayed. However, if the user has been assigned the role of a clerk, he or she is unauthorized to modify the employee records. Listing 22.11 provides the code for the adddemployee.jsp file (you can also find this file on the CD in the code\JavaEE\Chapter22\programmaticex folder):

Listing 22.11: Displaying the Code for the adddemployee.jsp File

```

<%@ page language="java" session="true" pageEncoding="ISO-8859-1" %>
<jsp:useBean id="idHandler" class="com.kogent.login.Login" scope="request">
<jsp:setProperty name="idHandler" property="*"/></jsp:useBean>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>Programmatic Security</title>
    </head>
    <body>
        <br/><br/><br/>
        <%
            ServletContext context=getServletContext();
            String userName = context.getAttribute("name").toString();
            if(idHandler.authorize(userName))
            {
            %
        <br/>
        <h1 align=center>Enter the Employee Details</h1>
        <table border=4 cellpadding=1 align=center>
            <tr>
                <td>Enter the Employee Name</td>
                <td><input type="text" name="empname"/></td>

```

```

        </tr>
        <tr>
            <td>Enter the Designation</td>
            <td><input type="text" name="empdesg"/></td>
        </tr>
        <tr>
            <td>
                <input type="submit" value="Add Employee"/>
            </td>
        </tr>
    </table>
    <%
    }
    else {
        out.println("<h2>" + "you are Unauthorized to add records" + "</h2>");
        out.println("<a href=viewemployee.jsp>" + "View Employee" + "</a>");
    }
%>
</body>
</html>

```

In Listing 22.11, the addemployee.jsp file uses a Login class, and invokes the authorize() method of the Login class to check if the authenticated username entered by the user has been assigned the administrator role. If the user has been assigned the administrator role, the authorize() method of the Login class returns true; otherwise, it returns false.

If the value is true, the employee form is displayed to enter employee details. However, if the value returned is false, an unauthorized message is displayed.

Creating the viewemployee.jsp File

If the user clicks the View Employee reference link in the welcome.jsp file, the viewemployee.jsp file is displayed. Listing 22.12 provides the code for the viewemployee.jsp file (you can also find this file on the CD in the code\JavaEE\Chapter22\programmaticex folder):

Listing 22.12: Displaying the Code for the viewemployee.jsp File

```

<%@ page language="java" session="true" pageEncoding="ISO-8859-1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>Programmatic Security</title>
    </head>
    <body>
        <h1 align=center>Employee Records</h1>
        <table border=4 cellpadding=1 align=center>
            <tr>
                <td>Ambika Sharma</td>
                <td>Project Manager</td>
            </tr>
            <tr>
                <td>Sam Malik</td>
                <td>Technical Consultant</td>
            </tr>
            <tr>
                <td>John Smith</td>
                <td>Technical Writer</td>
            </tr>
        </table><br> </body>
</html>

```

The viewemployee.jsp file displays the employee names and their designation, as shown in Listing 22.12. This page can be accessed by an authenticated user defined in the Login class. However, only the authenticated user with the administrator role can make additions in employee records.

The `authenticate()` and `authorize()` methods of the `Login` class checks the authenticity and authorization of the user, respectively.

Creating the Login.java and User.java Files

In this section, we create the `Login.java` and `User.java` files used in the `programmaticex` application. However, before creating these files, let's understand what we are going to implement in the `Login.java` file and why it is required. In declarative security, a Web server manages users and their roles, as was done in the `simpledeclarativeex` application. However, in programmatic security, users and their roles are not managed by the server; instead, these are defined by the developer. Therefore, the `Login` class defines the users and associates the roles to these users. The `Login` class also implements the authorization and authentication modes of security based on these roles. Listing 22.13 provides the code for the `Login` class (you can also find this file on the CD in the `code\JavaEE\Chapter 22\programmaticex\src\com\kogent\login` folder):

Listing 22.13: Displaying the Code for the `Login` Class

```
package com.kogent.login;
import java.io.*;
import java.util.HashMap;
import java.util.Map;
public class Login
{
    private Map users;
    private static final String ADMIN_ROLE = "administrator";
    private static final String CLERK_ROLE = "clerk";
    public Login()
    {
        users = new HashMap();
        users.put("manager", new User("manager", "manager", new String[] {ADMIN_ROLE, CLERK_ROLE}));
        users.put("clerk", new User("clerk", "clerk", new String[] {CLERK_ROLE}));
    }
    public boolean authenticate(String username, String password) throws IOException
    {
        User user = (User) users.get(username);
        boolean passwordIsValid = user.passwordMatch(password);
        if(user != null && !passwordIsValid)
        {
            return false;
        }
        else
        {
            return true;
        }
    }
    public boolean authorize(String username) throws IOException
    {
        User user = (User) users.get(username);
        boolean role = user.isAdmin();
        if (!role)
        {
            return false;
        }
        else
        {
            return true;
        }
    }
}
```

In Listing 22.13, we have created a `users` object of type `HashMap`, which contains two `User` objects, namely `manager` and `clerk`. The `manager` user has been assigned the `administrator` role and the `clerk` user has been assigned the `clerk` role. This indicates that the `manager` and `clerk` are authenticated usernames; however, only the `manager` user is authorized to add employee records.

The manager and clerk users are defined with the help of the User class by passing three parameters (username, password, and role assigned) in the User class constructor. In Listing 22.13, the authenticate() method checks the password assigned in the User class for the username based on the parameters passed by the validateuser.jsp file. If the password is correct, the true Boolean value is returned; otherwise, false is returned to the validateuser.jsp page.

Similarly, in Listing 22.13, the authorize() method invokes the isAdministrator() method of the User class. The isAdministrator() method returns true if the user has the administrator role assigned to him or her; otherwise, it returns false. Based on this boolean value, the authorize() method also returns the boolean value to the addemployee.jsp file. Listing 22.14 provides the code for the User.java file (you can also find this file on the CD in the code\JavaEE\Chapter22\programmaticex\src\com\kogent\login folder):

Listing 22.14: Displaying the Code for the User.java File

```
package com.kogent.login;
import java.io.Serializable;
public class User implements Serializable {
    private String username;
    private String password;
    private String[] roles;
    public User() {
    }
    public User(String name, String pwd, String[] assignedRoles) {
        username = name;
        password = pwd;
        roles=assignedRoles;
    }
    public String getUsername() {
        return username;
    }
    boolean passwordMatch(String pwd) {
        return password.equals(pwd);
    }
    public boolean hasRole(String role) {
        if (roles.length > 0) {
            for (int i=0; i<roles.length; i++) {
                if (role.equals(roles[i]))
                    return true;
            }
        }
        return false;
    }
    public boolean isAdministrator() {
        return hasRole("administrator");
    }
}
```

In Listing 22.14, the User class is defined in the com.kogent.login package. The username, password, and role are assigned through the Login class. In the User class, the getUsername() method returns the username and the passwordMatch() method returns the boolean value after verifying whether the password entered by the user is same as provided in the User class. The isAdministrator() method verifies whether the user has been assigned the administrator role; if yes, it returns true; otherwise, it returns false.

After compiling the Login.java and User.java files, the resultant package is saved in the WEB-INF\classes folder. Let's now configure the programmaticex application.

Configuring the Application

The web.xml file, saved in the WEB-INF folder (Figure 22.21), is used to configure the programmaticex application. Listing 22.15 provides the code for web.xml file (you can also find this file on the CD in the code\JavaEE\Chapter22\programmaticex\WEB-INF folder):

Listing 22.15: Displaying the Code for the web.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
```

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>

```

In Listing 22.15, the `<welcome-file>` element contains the `index.jsp` file, which is the home page of the `programmaticex` application.

Let's now explore the directory structure of the application.

Exploring the Directory Structure

The directory structure of the `programmaticex` application is shown in Figure 22.21:

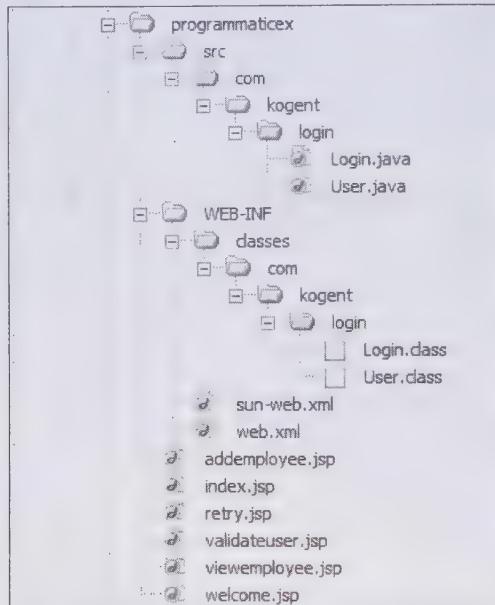


Figure 22.21: Displaying the Directory Structure of `programmaticex`

To create this structure, create a folder for the application as shown in Figure 22.21, which in our case is named `programmaticex`. Now, store the different types of files at their proper locations in the directory structure in the following manner:

- ❑ Store all packages containing class files in the `WEB-INF/classes` folder
- ❑ Store the Deployment Descriptor (`web.xml`) in the `WEB-INF` folder
- ❑ Store all JSP pages in the root directory, `programmaticex`

You can also create the `src` folder containing the source files (.java files) for all class files. This folder is optional in the `programmaticex` application; you can save your source files at any other location as well.

Now that we have created the root directory structure of `programmaticex`, let's

Let's now package, deploy, and run the `programmaticex` application to see its output.

Running the Application

To run the `programmaticex` application, start the Glassfish server and navigate to the [URL](http://localhost:8080/programmaticex/). Figure 22.22 displays the index page:

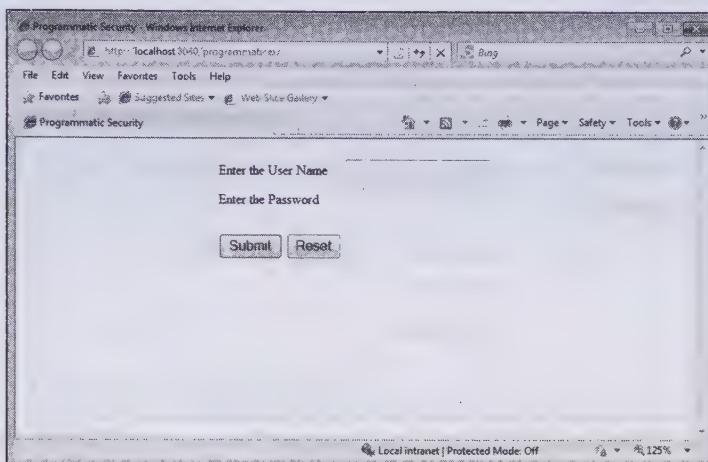


Figure 22.22: Displaying the Login Screen

Now, enter the username and password and click the Submit button, as shown in Figure 22.23:

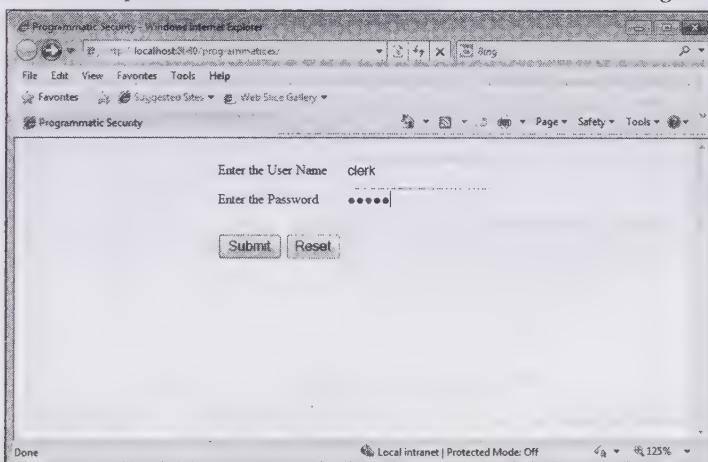


Figure 22.23: Entering a Username and Password

If the username and password are authentic, the welcome page is displayed, as shown in Figure 22.24:

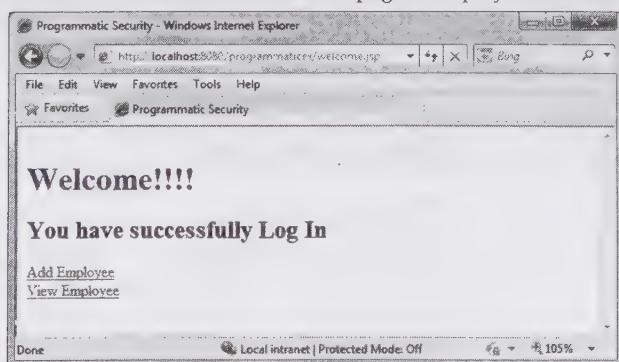


Figure 22.24: Displaying the welcome Page

If the information entered by the user is incorrect, the retry page is displayed, as shown in Figure 22.25:

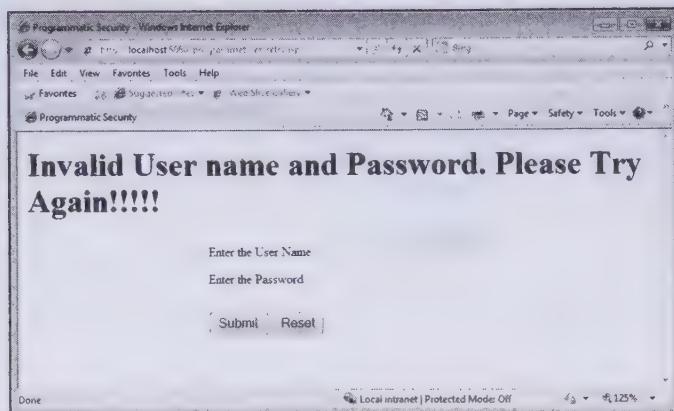


Figure 22.25: Displaying the Invalid Username and Password Error Message Page

Figure 22.25 displays the invalid username and password error message. The user is prompted to enter the required information again. If a user who entered the username and password as a clerk clicks the Add Employee link (Figure 22.24), he or she gets the message, as shown in Figure 22.26:

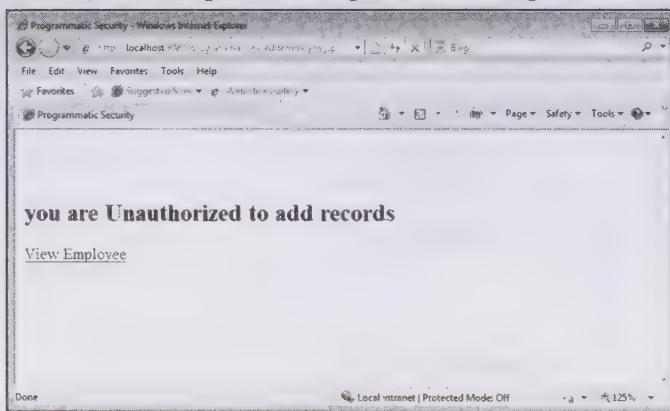


Figure 22.26: Displaying the Error Message Page for Unauthorized Users

However, if the clerk user clicks the View Employee link (Figure 22.26), the viewemployee page is displayed, as shown in Figure 22.27:

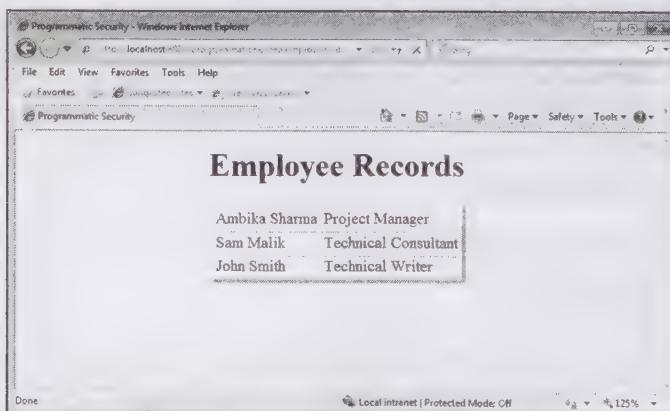


Figure 22.27: Displaying the viewemployee Page

If the user uses manager as a username and password to login (Figure 22.22), the user is authorized to add employee records. After logging with the manager username and password, the welcome.jsp page is displayed as shown in Figure 22.28:

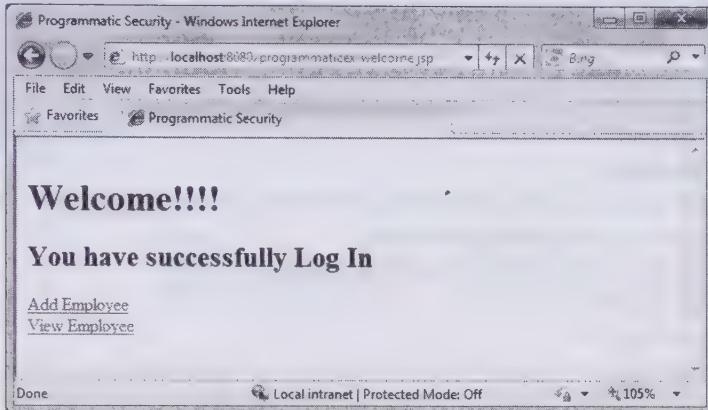


Figure 22.28: Displaying the Welcome Page

In the welcome page, when the user clicks the Add Employees link (Figure 22.28) the addemployee page is displayed.

Figure 22.29 shows the addemployee page:

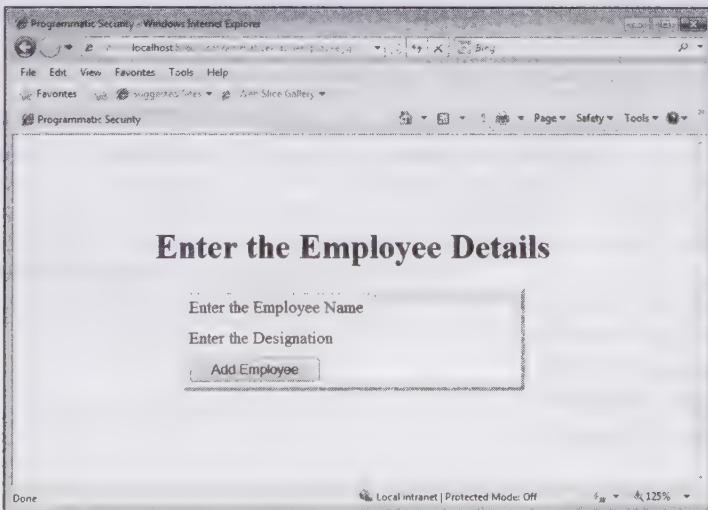


Figure 22.29: Displaying the addemployee Page for Authorized Users

With this, we come to the end of the chapter. Let's now summarize the main points of the chapter.

Summary

This chapter has discussed security related issues with respect to Web applications. It has introduced JAAS, which is used to implement authentication and authorization in Web applications, as well as the PAM framework on which the JAAS authentication is based. The chapter has also described various types of authentication mechanisms, such as HTTP basic authentication, form-based authentication, HTTP client authentication, and HTTP digest authentication. It has also explained the declarative and programmatic authorization mechanisms. Finally, we create two Web applications, simpledeclarativeex and programmaticex, to demonstrate the implementation of declarative and programmatic security, respectively.

Quick Revise

Q1. What is authentication?

Ans. Authentication is the process of checking the validity of the security details submitted by a user.

Q2. What is authorization?

Ans. Authorization is a process to determine whether the user has the appropriate access control rights to perform the requested action. The authorization mechanism restricts the access to resources according to the identity property.

Q3. What is data integrity?

Ans. Data integrity is the property that restricts or prevents data from being modified during transmission.

Q4. List the two ways to implement Web security.

Ans. The two ways of implementing Web security are as follows:

- Declarative security
- Programmatic security

Q5. List the HTTP authentication mechanisms that can be used to protect Web resources.

Ans. The following four types of authentication mechanisms are used to protect Web resources:

- HTTP basic authentication
- Form-based authentication
- Client-certificate authentication or HTTPS client authentication
- HTTP digest authentication

Q6. What is the use of JAAS?

Ans. Java Authentication and Authorization Service (JAAS), as the name suggests, is used to authenticate and authorize a user.

Q7. What is the difference between HTTP basic authentication and HTTP form-based authentication?

Ans. In the HTTP basic authentication mechanism, the caller is authenticated by comparing the username and password provided by the client in a dialog box with the authorized users database in the specified realm. In the form-based authentication, we can customize the user interface presented by an HTTP browser. Both mechanisms are simple but not secure as the content is sent as plain text.

Q8. The authentication mechanism allows developers to customize the authentication user interface presented by an HTTP browser.

- A. Form-based authentication
- B. HTTP basic authentication
- C. HTTP digest authentication
- D. HTTPS client authentication

Ans. A

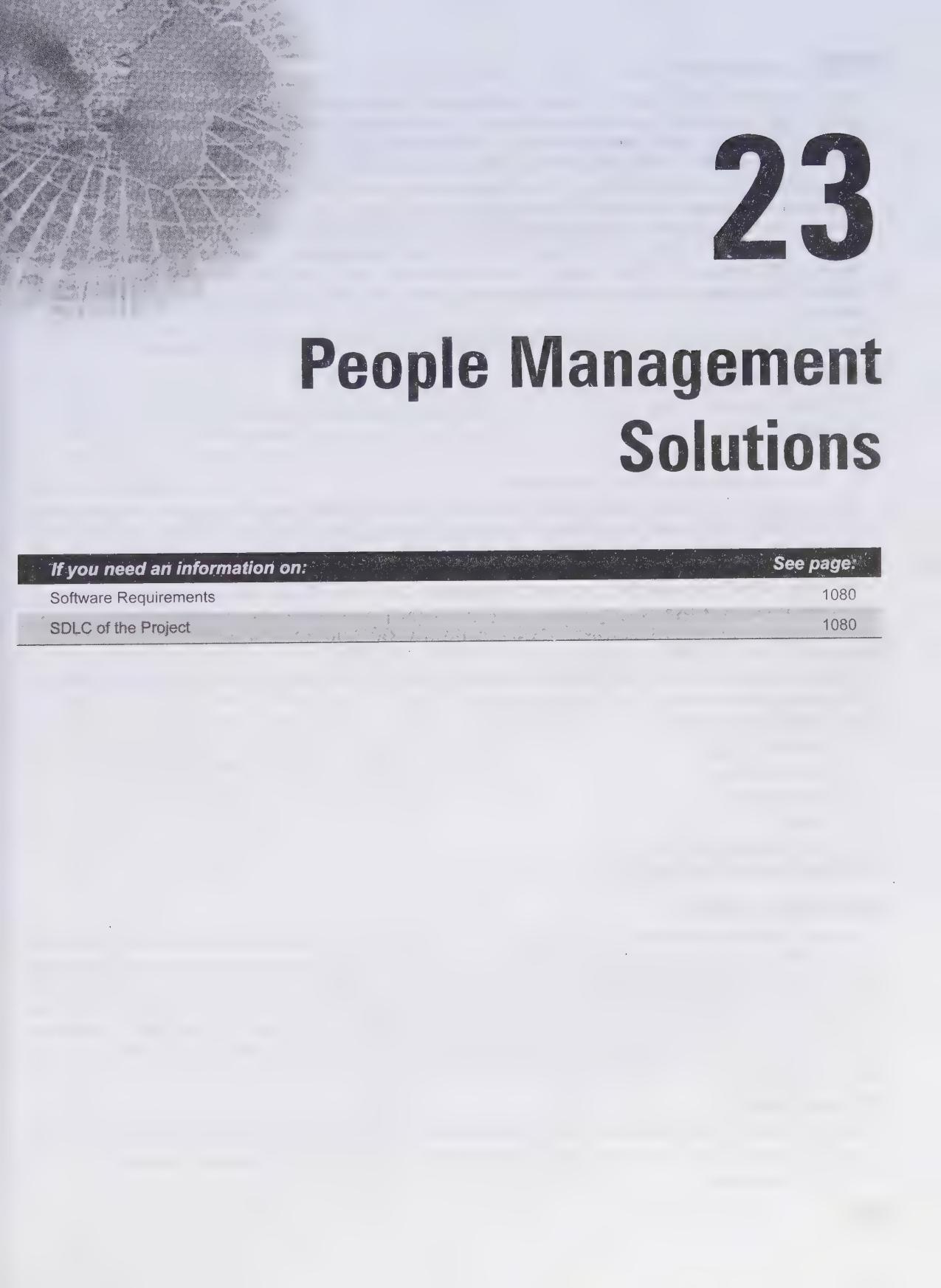
Q9. What is protection domain?

Ans. A protection domain is a collection of entities that are expected to trust one another. There is no need of authentication of entities in such a domain.

Q10. List the three layers in context of security mechanism.

Ans. Security mechanism is discussed in context of the following three layers of security:

- Application layer security
- Transport layer security
- Message layer security



23

People Management Solutions

If you need an information on:

See page:

Software Requirements

1080

SDLC of the Project

1080

Every organization has to perform certain functions, such as recruitment, maintenance of attendance records, and payroll management, to appoint and manage its human resources. In large organizations, managing the records of thousands of employees manually is a cumbersome task. As a result, a system is required that can help the human resource (HR) management to perform all these tasks in an automated and efficient manner.

In this project, you learn to develop a system, People Management Solutions (PeopleMgmt) that facilitates an organization to manage the details of each employee, such as salary and leaves status. In addition, this system helps the HR management to maintain the records of the candidates that are being interviewed. This system is basically designed to reduce the workload of the HR and administrative departments, as it allows you to automatically manage the data related to various processes, such as recruitment, leave management, and payroll. The software requirement for this project should be identified before the project development process begins.

The next section discusses about the software requirements of the proposed system.

Software Requirements

The following are the software requirements for developing the PeopleMgmt project:

- JDK 1.5 or JDK 1.6
- Sun Application Server (Glassfish)
- Oracle 10g Server as well as Client Editions

The latest versions of Servlet 3.0, and JSP 2.1 technologies are used in this project. In addition, for backend data handling, Oracle 10g is used as the database and Java DataBase Connectivity (JDBC) 4.0 provides Application Programming Interfaces (APIs) for connectivity of the database to the application program. The Glassfish V3 application server is used to deploy and execute this project.

The next section discusses the Software Development Life Cycle (SDLC) that identifies the stages of development of the project.

SDLC of the Project

Software Development Life Cycle (SDLC) entails the different stages of the project development. The SDLC of this project is discussed under the following heads:

- Requirement analysis
- Software design
- Database design
- Development
- Testing
- Implementation and maintenance

Let's now discuss the requirement analysis phase of SDLC.

Requirement Analysis

Any new software application is created to fulfill certain requirements that have not been met by the existing system. Therefore, the development process of any system starts with identifying the requirements. This process of analyzing the existing system to define the objectives of the new application is known as requirement analysis. The requirement analysis can be done by collecting feedbacks using questionnaires and surveys. In this case, automating the existing system to reduce manual labor and ensure greater accuracy in maintaining attendance, leave, and salary records of employees are some of the primary objectives for developing the proposed system.

Let's now describe the software design phase of the PeopleMgmt project

Software Design

The PeopleMgmt project consists of various modules that are required to fulfill the requirements analyzed in the requirement analysis phase of the SDLC. The project can be divided into the following modules:

- The Login module

- The Profile Management module
- The Recruitment module
- The Attendance Management module
- The Leave Management module
- The Payroll module

Let's begin the next section by discussing the Login module of the project.

The Login Module

Every system designed to perform administrative functions requires to be fully secured in a distributed environment. The most commonly-used security mechanism is to provide a login based authentication for users. The Login module of this project implements the login authentication mode in the system. This module ensures that only an authenticated and authorized user can access the links and navigation paths to perform operations, such as adding or editing employees, adding or updating applicants for recruitment, and updating daily attendance of the employees. A user name and password is provided to the users who perform these people management and administration related tasks. When the user enters the specified user name and password in the login page, the system verifies the user name and password. If the user is authenticated and authorized, the home page of the system is displayed.

The Profile Management Module

The Profile Management module manages the profiles of the employees of an organization. This module provides the interface to enter and update the information of the employees. You can insert, edit, update, and delete the profile of a person in this section. The information regarding a person, such as name, date of birth, address, joining date, and contact number are stored in the PEOPLE_EMPLOYEE table in the database.

The Recruitment Module

The Recruitment module keeps track of the recruitment process, which consists of different stages, such as written round, technical round, and HR round. When an applicant applies for an interview, the registration details of the applicant are maintained with the help of this module. After entering the registration details of the applicant, the automated system displays the applicant's name for different levels of tests to be conducted during the interviewing process. The Recruitment module also helps in scheduling tests and uploading the results of each applicant. This module also displays a list of finally selected candidates who have cleared all the tests according to the organization's expectations.

The next section discusses the Attendance Management module.

The Attendance Management Module

The Attendance Management module manages the attendance details of the employees. The interface of this module provides fields to enter the incoming and outgoing time of each employee for each day. The administrator is authorized to view the daily attendance report for the employees.

The next section discusses the Leave Management module.

The Leave Management Module

The Leave Management module manages leave-related functions, such as providing a leave form, submission of leave request to the management, and approval of a leave. In any organization, there is a limited amount of leave allocated to every employee. When an employee applies for leaves, this module provides a leave request form to be filled up by the employee. After filling and submitting the leave form, the information provided in the form is stored in the database. This module also displays whether or not the leave has been approved by the manager. All this information is considered while calculating the working days, and consequently, the salary of an employee at the end of each month.

The next section discusses about the Payroll module.

The Payroll Module

To calculate the salary of an employee, different aspects need to be considered, such as number of working days, tax induced, or incentives and other allowances. All this makes the task of calculating salaries of each and every employee manually a cumbersome task. The Payroll module is designed to reduce the workload of calculating salaries from the HR management. This module retrieves the details of the number of leaves taken by an employee from the Leave Management module to evaluate the salary.

The next section discusses about the database design.

Database Design

The PeopleMgmt project adds as well as updates the details of both employees and applicants by interacting with the database. Now, let's develop the database for this project. Table 23.1, 23.2, 23.3, 23.4, 23.5, 23.6, 23.7, and 23.8 provide the structure of different tables of the database, with their column names and data types.

Table 23.1 shows the structure of the PEOPLE_USER_LOGIN table:

Table 23.1: Showing the Structure of the PEOPLE_USER_LOGIN Table

Field name	Data type	Size	Primary key	Null
USER_ID	varchar2	10	Y	N
USER_NAME	varchar2	30		N
OLD_PSWD	varchar2	10		
NEW_PSWD	varchar2	10		
PSWD_EFF_DATE	date			
PSWD_EXP_DATE	date			

Table 23.2 shows the structure of the PEOPLE_EMPLOYEE table:

Table 23.2: Showing the Structure of the PEOPLE_EMPLOYEE Table

Field name	Data type	Size	Primary key	Null
EMP_ID	varchar2	10	Y	N
EMP_F_NAME	varchar2	20		N
EMP_M_NAME	varchar2	20		
EMP_L_NAME	varchar2	20		
ORG_ID	varchar2	10		N
LEVEL_ID	varchar2	10		N
DEPT_ID	varchar2	10		N
DOB	date			
DOJOIN	date			
ADDRESS_1	varchar2	50		
ADDRESS_2	varchar2	50		
CITY	varchar2	15		
STATE	varchar2	20		
NATIONALITY	varchar2	15		

Table 23.3 shows the structure of the PEOPLE_APPLICANT table:

Table 23.3: Showing the Structure of the PEOPLE_APPLICANT Table

Field name	Data type	Size	Primary key	Null
APPLICANT_ID	varchar2	10	Y	N
APPLICANT_NAME	varchar2	50		N

Table 23.3: Showing the Structure of the PEOPLE_APPLICANT Table

Field name	Data type	Size	Primary key	Null
ADDRESS_1	varchar2	100		
ADDRESS_2	varchar2	100		
EMAIL	varchar2	50		
PHONE	number	15		
MOBILE	number	11		
DOB	date			
GENDER	varchar2	10		
NATIONALITY	varchar2	30		
WORK_EXP	number	10		
SKILL	varchar2	100		
INDUSTRY	varchar2	30		
CATEGORY	varchar2	30		
ROLES	varchar2	30		
CURRENT_EMPLOYER	varchar2	100		
CURRENT_SAL	number	7,2		
HIGHEST_DEGREE	varchar2	100		
SECOND_HIGHEST_DEGREE	varchar2	100		
DOMAIN	varchar2	50		
CURRENT_LOCATION	varchar2	30		

Table 23.4 shows the structure of the APPLICANT_TEST_DETAIL table:

Table 23.4: Showing the Structure of the APPLICANT_TEST_DETAIL Table

Field name	Data type	Size	Primary key	Null
TEST_ID	varchar2	10	Y	N
TEST_NAME	varchar2	30	Y	N
APPLICANT_ID	varchar2	10	Y	N
APPLICANT_NAME	varchar2	50		
TEST_DATE	date			
TEST_TIME	date			
PRESENT_STATUS	varchar2	15		
TOTAL_MARKS	number	3		
MARKS_GAINED	number	3		
TEST_STATUS	varchar2	10		
PASS_FAIL	varchar2	10		
NEXT_ROUND	varchar2	10		

Table 23.5 shows the structure of the EMPLOYEE_DAILY_ATTENDANCE table:

Table 23.5: Showing the Structure of the EMPLOYEE_DAILY_ATTENDANCE Table

Field name	Data type	Size	Primary key	Null
EMP_ID	varchar2	10	Y	N
EMP_NAME	varchar2	30		
TODAY_DATE	varchar2	10	Y	N
MONTH	varchar2	10		
DAY	varchar2	10		
YEAR	number	4		
IN_TIME	date			
OUT_TIME	date			
REMARK	varchar2	50		

Table 23.6 shows the structure of the LEAVE_REQUEST table:

Table 23.6: Showing the Structure of the LEAVE_REQUEST Table

Field name	Data type	Size	Primary key	Null
REQ_ID	varchar2	10	Y	N
EMP_ID	varchar2	10		N
EMP_NAME	varchar2	30		
TODAY_DATE	date			
LEVEL_ID	varchar2	10		
DEPT_ID	varchar2	10		
FROM_DATE	date			
TO_DATE	date			
DAYS	number	2		
REASON	varchar2	100		
LEAVE_TYPE	varchar2	2		
ACTIVITY_1	varchar2	50		
ACTIVITY_2	varchar2	50		
ACTIVITY_3	varchar2	50		
PERSON_1	varchar2	50		
PERSON_2	varchar2	50		
PERSON_3	varchar2	50		
DETAIL_1	varchar2	100		
DETAIL_2	varchar2	100		
DETAIL_3	varchar2	100		
ADDRESS	varchar2	100		
REMARK	varchar2	100		
LEAVE_STATUS	varchar2	5		

Table 23.7 shows the structure of the EMPLOYEE AGREEMENT table:

Table 23.7: Showing the Structure of the EMPLOYEE AGREEMENT Table

Field name	Data type	Size	Primary key	Null
EMP_ID	varchar2	10	Y	N
EMP_NAME	varchar2	30		N

Table 23.7: Showing the Structure of the EMPLOYEE AGREEMENT Table

Field name	Data type	Size	Primary key	Null
LEVEL_ID	varchar2	10		
ALLOWANCE_TYPE	varchar2	10		N
ALLOWANCE_NAME	varchar2	20	Y	N
AMT	number	7,2		N
TAXABLE	varchar2	5		
PERCENTAGE	number	3,2		
AGREEMENT_DATE	date			

Table 23.8 shows the structure of the EMP_SAL table:

Table 23.8: Showing the Structure of the EMP_SAL Table

Field name	Data type	Size	Primary key	Null
EMP_ID	varchar2	10	Y	N
YEAR	number	4	Y	N
MONTH	number	2	Y	N
ALLOWANCE_TYPE	varchar2	10		N
ALLOWANCE_NAME	varchar2	20	Y	N
AMT	number	7,2		N
TAXABLE	varchar2	5		
PERCENTAGE	number	3,2		

Development

The PeopleMgmt project is developed using the Model View Controller (MVC) architecture, which allows a developer to concentrate on building business logic in the Controller servlet and other helper classes, while a designer can design the JSP pages. In the MVC architecture, a central Controller servlet handles all the business logic with the help of some helper classes. The Model part of the MVC architecture can be defined by JavaBeans that are used as data transfer objects to hold client data. They may be initialized automatically or manually using the ResultSet object. Finally, the View part of the architecture is implemented through the JSP pages. When the development process of a project is separated, the development becomes complex; however, it increases the reusability of the components.

To implement the MVC architecture, the designing of the PeopleMgmt project is divided into three layers—presentation, business logic, and data, which are developed separately. The presentation layer is designed using JSP to create user interface, the business logic is embedded in the Controller servlet, and the JavaBean classes hold the data to be stored in the database. In the PeopleMgmt project, every module has its own Controller servlet to handle client requests. Each Controller servlet has a helper class that defines all the required methods used by the servlet to implement a functionality. The JavaBean classes are populated manually using the methods of the helper classes of the Controller servlet. The packaging, deployment, and execution of the PeopleMgmt project are performed on the Glassfish application server.

Testing

You need to test the project to ensure proper navigation and validity of data being entered into the User Interface (UI) forms before providing the PeopleMgmt project to clients for their use. As a best practice, a project is considered ready to be implemented and used by the clients in the real environment only after performing a thorough testing process.

Implementation and Maintenance

The last stage in the SDLC is implementation and maintenance, in which you need to deploy the project in the real environment. Implementation includes proper deployment of the project on the application server. The maintenance of a project means updating the project for the required changes in the implemented system. While running the project in the real environment, many loopholes and areas of improvement may be discovered. The user may also suggest some changes or report some problems in the system, based on which the project is updated from time-to-time.

Summary

In this chapter, you learn about the phases of SDLC and their implementations in the PeopleMgmt project. The chapter also describes about the requirements of the project and approaches to develop the project. Next, you learned about the designing process, role, and functionality of various modules. In addition, you have also learned how to design database that includes number of tables and the column data types used in the project to store the data.

The next sections discuss about the development of different modules by using Java EE 6 and its related technologies. The PeopleMgmt project is divided into six different modules, which are described in the following sections:

- ❑ **Section A**—Developing the Login Module
- ❑ **Section B**—Developing the Profile Management Module
- ❑ **Section C**—Developing the Recruitment Module
- ❑ **Section D**—Developing the Attendance Management Module
- ❑ **Section E**—Developing the Leave Management Module
- ❑ **Section F**—Developing the Payroll Module

All these modules are created using Java EE 6 technologies and deployed on the Glassfish server, which is the standard Sun application server.

Section A

Developing the Login Module

If you need an information on:

See page:

Designing Login User Interface	1088
Directory Structure of the Project	1098
Login and Navigating to Home Page	1099
Changing Password	1101

The PeopleMgmt system is used to maintain and update the employee-related data, such as recruitment details, employee salary, and employee attendance. These records are of great importance for any organization and should always be protected from unauthorized access. The Login Module of the PeopleMgmt project is developed to verify the authentication of users. If the user is authentic, various links and interfaces are displayed to the user to add, update, and search important data.

The entire project is being developed on the basis of the MVC architecture. All the modules consist of JSP pages, servlets, and some other Java classes. These Java classes are used to handle the basic business logic for holding the data similar to JavaBeans.

The development of this module includes writing code for the following files:

- ❑ UserLoginDBObj.java
- ❑ UserLoginDBMethods.java
- ❑ people_user_login.java
- ❑ people_user_login.jsp
- ❑ people_user_login_pswd_change.jsp
- ❑ people_default.jsp
- ❑ people_footer.jsp
- ❑ people_header.jsp
- ❑ people_main_menu.jsp

The code for all files has been discussed in detail in the following sections. You can use notepad or any Integrated Development Environment (IDE), such as NetBeans or Eclipse, to develop these code files. All code files are also available in the CD provided with the book.

Designing Login User Interface

Any user interface through which a user can interact with the system is called a *JSP* page (View). In the Login module, the *people_user_login* JSP page serves as a login interface for the users. This page contains three text boxes to enter user id, user name, and password along with the Submit button. After entering the details in these text boxes, the user can submit the *people_user_login* JSP page by clicking the Submit button.

The code of the *people_user_login.jsp* file is shown in Listing 23.1 (you can find this file in the *PeopleMgmt\people-mgmt\jsp* folder on CD):

Listing 23.1: Showing the Code of the *people_user_login.jsp* File

```
<html>
<head><title>www.peoplemanagementsolutions.com/User Login</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<form name = "form1" method="post">
<table width="900" align=center>
<tr>
<td colspan="2"><%@ include file= 
"\jsp\people_header.jsp" %></td>
</tr>
<tr >
<td colspan="2">
<table border = "0"  align="center" >
<tr><td>User Id</td>
<td align="center" ><input type =
"text" name="user_id" id="user_id" value="" 
/></td></tr>
<tr><td>User Name</td>
<td align="center" ><input type ="text" name=
```

```

        : "user_name" id="user_name" value="" /></td></tr>
<tr><td>Password</td>
<td align="center" ><input type =
"password" name="user_pswd" id="user_pswd"
value="" /></td></tr>
<tr><td colspan="2" align="center" >
<input type ="submit" name="submit"
id="submit" value="Submit" />
<input type="hidden" name="action_submit" id="action_submit"
value="people_user_login_submit"/>
</td></tr>
<tr><td colspan="2" align="center">
<input type ="submit" name="submit" id=
"submit" value="Change Password" />
<input type="hidden" name="action_chngpswd" id="action_chngpswd"
value="people_change_pswd_submit"/>
</td></tr>
<%
String msg = (String)session.getAttribute("lErrorMsg");
if ( msg != null && msg.length() > 0 ){
%>
<tr>
<td colspan="2" align="center">
<% out.println("<div class=boldred>" +msg+ "</div>"); %>
</td>
</tr>
<%
}
</table>
</td></tr>
<tr>
<td colspan="2" align="center" ><%@include file="..../jsp/people_footer.jsp"%></td>
</tr>
</table>
</form>
</body>
</html>

```

In Listing 23.1, the `people_user_login.jsp` file includes two JSP files, `people_header.jsp` and `people_footer.jsp` using the `<%@ include %>` directive. These files can be copied from the `PeopleMgmt\people-mgmt\jsp` folder (present in CD) to your application directory.

Creating the `people_user_login` Servlet

The `people_user_login.java` file is a servlet that contains the business logic to authenticate a user. After a user is successfully authenticated, this servlet displays a default page from where the user can access details of employees or applicants. The `people_default` JSP page serves as the default page and shows various navigation menus, such as Employee, Recruitment, Time Management, and Payroll. In the `people_user_login.java` file, annotations are used to specify servlet-mapping instead of Deployment Descriptor (`web.xml` file).

Note

Java Servlet 3.0 provides the support for annotations instead of Deployment Descriptors to provide configuration or mapping details of a resource.

The code of the `people_user_login.java` file is shown in Listing 23.2 (you can find this file in the `PeopleMgmt\people-mgmt\WEB-INF\src` folder on CD):

Listing 23.2: Showing the Code of the `people_user_login.java` File

```

import javax.servlet.*;
import javax.servlet.http.*;

```

```

import java.util.ArrayList;
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.annotation.*;
import com.UserLogin.UserLoginDBMethods;
import com.UserLogin.UserLoginDBObj;
@WebServlet(name="people_user_login", urlPatterns="/servlet/people_user_login")
public class people_user_login extends HttpServlet
{
    String lDBUser = "";
    String lDBPswd = "";
    String lDBUrl = "";
    /**Initialize global variables*/
    @Override
    public void init(ServletConfig config) throws ServletException
    {
        System.out.println("initializing controller servlet.");
        ServletContext context = config.getServletContext();
        lDBUser = "scott";
        lDBPswd = "tiger";
        lDBUrl = "jdbc:oracle:thin:@localhost:1521:"+"orcl";
        super.init(config);
    }
    /**Process the HTTP Get request*/
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        doPost(request, response);
    }
    /**Process the HTTP Post request*/
    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        HttpSession session = request.getSession();
        session.setAttribute("lErrorMsg",null);
        String target = "/jsp/people_user_login.jsp";
        String action = request.getParameter("action");
        String action_submit = request.getParameter("action_submit");
        String action_chngpswd = request.getParameter("action_chngpswd");
        System.out.println("action_submit=="+action_submit);
        if( action_submit != null || action_chngpswd != null )
        {
            if( request.getParameter("submit").equals("Submit") )
            {
                System.out.println("in the Submit");
                if( action_submit.equals("people_user_login_submit") )
                {
                    System.out.println("in the people_user_login_submit");
                    action = "people_user_login_submit";
                }
                else
                if( action_submit.equals("login_pswd_change_submit") )
                {
                    action = "login_pswd_change_submit";
                }
            }
        }
    }
}

```

```

        }
    }
    else
    if (request.getParameter("submit").equals("ChangePassword") )
    {
        if (action_chngpswd.equals("people_change_pswd_submit") )
            action = "people_change_pswd_submit";
    }
}
if (action!=null)
{
    System.out.println("in the "+action);
    if (action.equals("people_user_login_submit"))
    {
        String luserId = "";
        String luserName = "";
        String luserPswd = "";
        luserId = (String)request.getParameter("user_id");
        luserName = (String)request.getParameter("user_name");
        luserPswd = (String)request.getParameter("user_pswd");
        UserLoginDBObj userLoginDBObj = new UserLoginDBObj();
        userLoginDBMethods = new
        UserLoginDBMethods(lDBUser,lDBPswd,lDBUrl);
        userLoginDBObj=(UserLoginDBObj).getRecordByPrimaryKey
        (luserId,luserName,luserPswd);
        if ( userLoginDBObj != null && (userLoginDBObj.user_id !=null &&
        (userLoginDBObj.user_id).length() > 0) )
        {
            target = "/jsp/people_default.jsp";
        }
        else
        {
            String lErrorMsg = "User Does Not Exist!!";
            session.setAttribute("lErrorMsg",lErrorMsg);
            target = "/jsp/people_user_login.jsp";
        }
    }
    else
    if (action.equals("people_change_pswd_submit"))
    {
        target = "/jsp/people_user_login_pswd_change.jsp";
    }
    else
    if (action.equals("login_pswd_change_submit"))
    {
        UserLoginDBObj popUserLoginDBObj = new
        UserLoginDBObj();
        UserLoginDBMethods userLoginDBMethods = new
        UserLoginDBMethods(lDBUser,lDBPswd,lDBUrl);
        String lUserId = "";
        String lUserName = "";
        String lCurPswd = "";
        String lNewPswd = "";
        String lRetypePswd = "";
        popUserLoginDBObj =(UserLoginDBObj)
        userLoginDBMethods.populateUser.LoginDBObjFromReq(request);
        lRetypePswd =
        (String)request.getParameter("retype_pswd");
        if ((popUserLoginDBObj.new_pswd).equals(lRetypePswd))
        {
    }
}

```

```

UserLoginDBObj userLoginDBObj = new
    UserLoginDBObj();
userLoginDBObj =
    (UserLoginDBObj)userLoginDBMethods.
        getRecordByPrimaryKey(popUserLoginDBObj,
            user_id, popUserLoginDBObj.user_name,
            popUserLoginDBObj.old_pswd);
if ( userLoginDBObj != null &&
    userLoginDBObj.user_id != null &&
    (userLoginDBObj.user_id).length() > 0 )
{
    int rval =
        userLoginDBMethods.updateUserLogin
        ByPrimaryKey(popUserLoginDBObj);
    if ( rval > 0 )
    {
        target =
            "/jsp/people_user_login.jsp";
    }
    else
    {
        target =
            "/jsp/people_user_login_
            pswd_change.jsp";
    }
}
else
{
    String lErrorMsg = "User Does Not Exist!!";
    session.setAttribute("lErrorMsg", lErrorMsg);
    target = "/jsp/people_user_login_pswd_
        change.jsp";
}
}
else
{
    String lErrorMsg = "Retype Correct Password!!";
    session.setAttribute("lErrorMsg", lErrorMsg);
    target = "/jsp/people_user_login_pswd_change.jsp";
}
}
}
// (action== null )
/* forwarding the request/response to the targeted view */
RequestDispatcher requestDispatcher =
    getServletContext().getRequestDispatcher(target);
requestDispatcher.forward(request, response);
}
// doPost closed
/* write the methods that are used in class */
}
// class closed

```

Creating the UserLoginDBObj Class

The UserLoginDBObj Java class provides various variables, such as user_id, user_name, old_pswd, and new_pswd to handle a user's login details. The object of this class represents a record in the PEOPLE_USER_LOGIN table of the Oracle 10g database.

The code of the UserLoginDBObj.java file is shown in Listing 23.3 (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src\com\UserLogin folder on CD):

Listing 23.3: Showing the Code of the UserLoginDBObj.java File

```

package com.UserLogin;
public class UserLoginDBObj
{
    public String user_id ;
    public String user_name;
    public String old_pswd ;
    public String new_pswd ;
    public String pswd_eff_date;
    public String pswd_exp_date;
}

```

Creating the UserLoginDBMethods Class

The UserLoginDBMethods class provides various methods to update and retrieve the login information of a user. This class also contains the code to establish a connection with the database to retrieve or update records in the PEOPLE_USER_LOGIN table. In the UserLoginDBObj class, the parameterized constructor is defined to assign the values to the DBUser, DBPswd, and DBUrl variables on the basis of the String arguments passed to the constructor. The values of these variables are used to establish the database connection.

The code of the UserLoginDBMethods.java file is shown in Listing 23.4 (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src\com\UserLogin folder on CD):

Listing 23.4: Showing the Code of the UserLoginDBMethods.java File

```

package com.UserLogin;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.util.ArrayList;
import com.UserLogin.UserLoginDBObj;
public class UserLoginDBMethods
{
    public String DBUser;
    public String DBPswd;
    public String DBUrl ;
    public UserLoginDBMethods()
    {
    }
    public UserLoginDBMethods(String inDBUser, String inDBPswd, String inDBUrl )
    {
        DBUser = inDBUser ;
        DBPswd = inDBPswd;
        DBUrl  = inDBUrl;
    }
    public void initializeUserLoginDBObj(UserLoginDBObj inUserLoginDBObj )
    {
        inUserLoginDBObj.user_id = "";
        inUserLoginDBObj.user_name = "";
        inUserLoginDBObj.old_pswd = "";
        inUserLoginDBObj.new_pswd = "";
        inUserLoginDBObj.pswd_eff_date = "";
        inUserLoginDBObj.pswd_exp_date = "";
    }
    public UserLoginDBObj getRecordByPrimaryKey(String inuserId, String inUserName, String
.inUserPswd)
    {
        UserLoginDBObj userLoginDBObj = new UserLoginDBObj();
        try

```

```

{
    System.out.println("DBUser=="+DBUser+",DBPswd=="+DBPswd+","
    DBUrl=="+DBUrl);
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
    Statement stmt = conn.createStatement();
    String lSqlString = "select * from PEOPLE_USER_LOGIN ";
    lSqlString = lSqlString + "where user_id='"+inUserId+"' ";
    lSqlString = lSqlString + "and user_name='"+inUserName+"' ";
    lSqlString = lSqlString + "and new_pswd='"+inUserPswd+"' ";
    ResultSet rs = null;
    rs = stmt.executeQuery(lSqlString);
    System.out.println("lSqlString====trtrt==within
        getRecordByPrimaryKey== "+lSqlString);
    if( rs.next())
    {
        System.out.println("fffff=="+rs.getString("user_id"));
        userLoginDBObj.user_id = rs.getString("user_id");
        userLoginDBObj.user_name = rs.getString("user_name");
        userLoginDBObj.old_pswd = rs.getString("old_pswd");
        userLoginDBObj.new_pswd = rs.getString("new_pswd");
        userLoginDBObj.pswd_eff_date = rs.getString("pswd_eff_date");
        userLoginDBObj.pswd_exp_date = rs.getString("pswd_exp_date");
        System.out.println("fffff=="+rs.getString("user_id"));
    }
    else {
        initializeUserLoginDBObj(userLoginDBObj);
    }
    System.out.println("fffff===="+userLoginDBObj.user_id);
}
catch(SQLException ex)
{
    ex.printStackTrace();
}
return userLoginDBObj;
}

public int updateUserLoginByPrimaryKey(UserLoginDBObj inUserLoginDBObj)
{
    int recupd = 0;
    String lQuery = "";
    lQuery = lQuery +"update PEOPLE_USER_LOGIN set
    old_pswd='"+inUserLoginDBObj.old_pswd+"' ";
    lQuery = lQuery +", new_pswd='"+inUserLoginDBObj.new_pswd+"' ";
    lQuery = lQuery + "where user_id='"+inUserLoginDBObj.user_id+"' ";
    lQuery = lQuery + "and user_name='"+inUserLoginDBObj.user_name+"' ";
    lQuery = lQuery + "and new_pswd='"+inUserLoginDBObj.old_pswd+"' ";
    System.out.println("lSqlString==:"+lQuery);
    try
    {
        DriverManager.registerDriver(new
        oracle.jdbc.driver.OracleDriver());
        Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
        Statement stmt = conn.createStatement();
        recupd = stmt.executeUpdate(lQuery);
    }
    catch(SQLException ex) {
        ex.printStackTrace();
    }
    return recupd;
}

```

```

}
public UserLoginDBObj populateUserLoginDBObjFromReq(HttpServletRequest inReq)
{
    UserLoginDBObj userLoginDBObj = new UserLoginDBObj();
    userLoginDBObj.user_id = (String)inReq.getParameter("user_id");
    userLoginDBObj.user_name = (String)inReq.getParameter("user_name");
    userLoginDBObj.old_pswd = (String)inReq.getParameter("old_pswd");
    userLoginDBObj.new_pswd = (String)inReq.getParameter("new_pswd");
    userLoginDBObj.pswd_eff_date = (String)inReq.getParameter("pswd_eff_date");
    userLoginDBObj.pswd_exp_date =
    (String)inReq.getParameter("pswd_exp_date");
    return userLoginDBObj;
}
}

```

In Listing 23.4, the following methods are defined in the UserLoginDBObj class:

- ❑ **The initializeUserLoginDBObj (UserLoginDBObj) method**—Initializes the passed object with empty String values
- ❑ **The getRecordByPrimaryKey () method**—Returns the UserLoginDBObj object with all member variables that are set with the retrieved login information of a user
- ❑ **The updateUserLoginByPrimaryKey (UserLoginDBObj) method**—Updates the login information of the user
- ❑ **The populateUserLoginDBObjFromReq (HttpServletRequest) method**—Accepts the HttpServletRequest object as an argument and returns the UserLoginDBObj object after populating it with the information stored in the request scope

Creating the people_default.jsp File

The people_default JSP page serves as the home page of the PeopleMgmt project. This home page contains the menus used to access other pages and modules of the project.

The code of the people_default.jsp file is shown in Listing 23.5 (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.5: Showing the Code for the people_default.jsp File

```

<%@ page contentType="text/html; charset=iso-8859-1" language="java" errorPage="" %>
<!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">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>www.peoplemanagementsolutions.com-welcome</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2"><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td width="900">
<%@ include file="../jsp/people_default_menu.jsp" %>
</td>
</tr>
<tr>
<td width="750" align="center" valign="top">
<%@include file="../jsp/design.jsp"%>
</td>
</tr>

```

```

<%
String msg = (String)session.getAttribute("ErrorMsg");
if (msg != null && msg.length() > 0){
%>
<tr><td align='center'>
<% out.println("<div class=boldred>" + msg + "</div>"); %>
<br/>
</td></tr>
<%}>
<tr>
<td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr>
</table>
</body>
</html>

```

The people_default JSP page is created by using the following four reusable JSP pages:

- ❑ **people_header**—Provides a header to a page. The people_header JSP page contains a logo of the company and the information that needs to be displayed on all JSP pages.
- ❑ **people_default_menu**—Provides the menus to be added to all user interfaces of the PeopleMgmt project.
- ❑ **people_footer**—Contains the footer information, such as copyright as well as company's basic information that needs to be added in all JSP pages of the project.
- ❑ **design**—Contains the design of the default (home) page.

The code for the preceding JSP pages has been provided in the PeopleMgmt\people-mgmt\jsp folder (present in the CD) and can be used within that specified location.

Let's now discuss the code of the people_default_menu.jsp file.

The people_default_menu JSP page shows the menus and submenus that need to be added to all user interfaces of the PeopleMgmt project. These menus and submenus are used to perform various functions, such as creating or updating the profile of an employee, registering a new applicant, and scheduling a written test, updating test results for each applicant, and selecting the applicants for the next round.

The code of the people_default_menu.jsp file is shown in Listing 23.6 (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.6: Showing the Code for the people_default_menu.jsp File

```

<%@ page contentType="text/html; charset=iso-8859-1" language="java" errorPage="" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<HTML>
<HEAD>
<link rel="stylesheet" href="../css/CascadeMenu.css" type="text/css"></link>
<script language="JavaScript" type="text/JavaScript"
src="../css/CascadeMenu.js"></script>
</HEAD>
<BODY OnLoad="InitMenu()" OnClick="HideMenu(menuBar)" ID="Bdy">
<DIV Id="menuBar" class="menuBar" >
<DIV Id="Bar5" class="Bar" menu="menu9" title="About Us/Log Out">
</DIV>
<DIV Id="Bar1" class="Bar" menu="menu1">Employee</DIV>
<DIV Id="Bar2" class="Bar" menu="menu2">Recruitment</DIV>
<DIV Id="Bar3" class="Bar" menu="menu3">Time Management</DIV>
<DIV Id="Bar4" class="Bar" menu="menu4">PayRoll</DIV>
</DIV><div align="right"><a href="/people-mgmt/jsp/people_default.jsp">Home Page</a></div>
<div Id="menu9" class="menu" >

```

```

<div Id="menuItem9_1" class="menuItem" title="About Us" cmd="/people-
mgmt/jsp/people_aboutus.jsp" class="whitelink">About Us </div>
<div Id="menuItem9_2" class="menuItem" title="Log Out" cmd="/people-
mgmt/servlet/people_user_login" class="whitelink">Log Out</div>
</div>
<div Id="menu1" class="menu" ><div Id="menu1_1" class="menuItem" title="Add Employees" cmd="/people-
mgmt/servlet/people_employee?dbopr=create" class="whitelink">Add New
Employee</div>
<div Id="menuItem1_2" class="menuItem" title="Edit Employee" cmd="/people-
mgmt/servlet/people_employee?dbopr=edit" class="whitelink">Edit Employee</div>
<div Id="menuItem1_3" class="menuItem" title="Employee Profile" cmd="/people-
mgmt/servlet/people_employee?dbopr=show" class="whitelink">Employee Profile</div>
</div>
<div Id="menu2" class="menu">
<div Id="menuItem2_1" class="menuItem" title="Register Applicants"
cmd="/people-mgmt/servlet/people_applicant?dbopr=register" class="whitelink">New
Applicant</div>
<div Id="menuItem2_2" class="menuItem" title="Update Applicants" cmd="/people-
mgmt/servlet/people_applicant?dbopr=select" class="whitelink">Update
Applicant</div>
<div Id="menuItem2_3" class="menuItem" title="Shortlisted Candidate"
menu="menu5">Written Round</div>
<div Id="menuItem2_4" class="menuItem" menu="menu6" title="Technical Round" >
Technical Round</div>
<div Id="menuItem2_5" class="menuItem" menu="menu7" title="HR Round" > HR
Round</div>
</div>
<div Id="menu3" class="menu">
<div Id="menuItem3_1" class="menuItem" title="Enter/Update Attendance"
cmd="/people-mgmt/servlet/time_management?dbopr=daily_attendance_entry"
class="whitelink">
Update Attendance</div>
<div Id="menuItem3_2" class="menuItem" title="Attendance Summary" cmd="/people-
mgmt/servlet/time_management?dbopr=daily_attendance_summary" class="whitelink">
Attendance Summary</div>
<div Id="menuItem3_3" class="menuItem" menu="menu8">Leave Management</div>
</div>
<div Id="menu4" class="menu">
<div Id="menuItem4_1" class="menuItem" title="Employee Agreement" cmd="/people-
mgmt/servlet/people_payroll?dbopr=employee_agreement" class="whitelink"><br />
Salary BreakUp</div>
<div Id="menuItem4_2" class="menuItem" title="Employee Salary" cmd="/people-
mgmt/servlet/people_payroll?dbopr=calc_employee_salary" class="whitelink">
Employee Salary</div>
</div>
<div id="menu5" class="menu">
<div Id="menuItem5_1" class="menuItem" title="Call for Written" cmd="/people-
mgmt/servlet/applicant_test_dtl?dbopr=call_for_written" class="whitelink">Call for
Written</div>
<div Id="menuItem5_2" class="menuItem" title="Update Results" cmd="/people-
mgmt/servlet/applicant_test_dtl?dbopr=upd_wrtn_performance"
class="whitelink">Results</div>
</div>
<div id="menu6" class="menu">
<div Id="menuItem6_1" class="menuItem" title="Shortlisted For Tech.Round"
cmd="/people-mgmt/servlet/applicant_test_dtl?dbopr=shortlist_after_wrtn"
class="whitelink">_
Shortlist For Tech. Round</div>

```

```

<div Id="menuItem6_2" class="menuItem" title="Update Results" cmd="/people-
mgmt/servlet/applicant_test_dtl?dbopr=upd_tech_performance" class="whitelink">
Update Result</div>
</div>
<div id="menu7" class="menu">
<div Id="menuItem7_1" class="menuItem" title="Shortlisted for HR Rounds"
cmd="/people-mgmt/servlet/applicant_test_dtl?dbopr=shortlist_after_tech"
class="whitelink">
Shortlist for HR Round </div>
<div Id="menuItem7_2" class="menuItem" title="Update Results" cmd="/people-
mgmt/servlet/applicant_test_dtl?dbopr=upd_hr_performance" class="whitelink">
Update Result</div>
<div Id="menuItem7_3" class="menuItem" title="Shortlisted for Selection"
cmd="/people-mgmt/servlet/applicant_test_dtl?dbopr=shortlist_after_hr"
class="whitelink">
Shortlist For Selection</div>
<div Id="menuItem7_4" class="menuItem" title="Selected" cmd="/people-
mgmt/servlet/applicant_test_dtl?dbopr=final_selected" class="whitelink">
Selected Candidate</div>
</div>
<div id="menu8" class="menu">
<div Id="menuItem8_1" class="menuItem" title="Leave Application" cmd="/people-
mgmt/servlet/leave_management?dbopr=leave_request" class="whitelink">Leave Apply
</div>
<div Id="menuItem8_2" class="menuItem" title="Approval Status" cmd="/people-
mgmt/servlet/leave_management?dbopr=leave_approve" class="whitelink">Leave
Approval</div>
    <div Id="menuItem8_3" class="menuItem" title="Approved/Rejected" cmd="/people-
    mgmt/servlet/leave_management?dbopr=approved_leave" class="whitelink">Approved
    Request</div>
</div>
</BODY>
</HTML>

```

In Listing 23.6, you can see that almost all the submenus, such as Create Profile, Edit Profile, New Applicant, Update Applicant, and Call for Test, invoke the appropriate servlet instead of calling JSP pages. According to the MVC architecture, this project also provides various servlets to handle user requests. The servlets process the requests and forward the response to the View page requested by the user.

Let's now explore the directory structure of the project in the next section.

Directory Structure of the Project

Let's now create a directory structure of the PeopleMgmt project to place all the files at a specific location. Perform the following steps to create the directory structure of the project:

1. Create a project folder and give it a name, say `people-mgmt`.
2. Create four more folders, namely `css`, `images`, `jsp`, and `WEB-INF`.
3. Place all the source files (`.java`) in the `WEB-INF/src` folder.
4. Put all the class files in the `WEB-INF/classes` folder within specific package folders, if any, and put all the jar files in the `WEB-INF/lib` folder.
5. Save all the JSP pages in the `people-mgmt/jsp` folder. The recently compiled files, `UserLoginDBObj.class` and `UserLoginDBMethods.class`, are placed in the `com/UserLogin` folder.

NOTE

The mapping of a servlet is not provided in the web.xml file, as it is already defined in the servlet by using annotations.

Figure 23.1 shows the directory structure of the PeopleMgmt project:

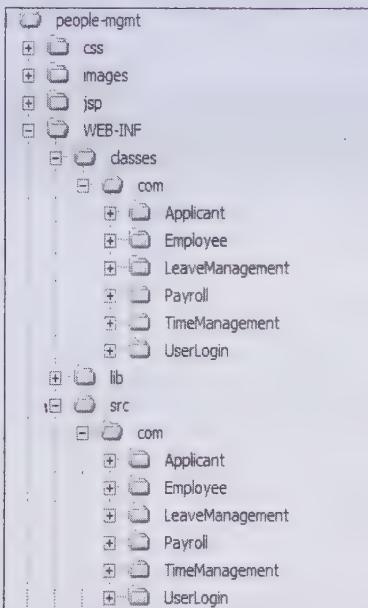


Figure 23.1: Showing the Directory Structure for the Project

The compiled Java source files (class files) are placed in the `classes` directory, along with their package directories. In the Login module, the `people_user_login` servlet is mapped within itself using the `@WebServlet` annotation. Use of this annotation removes the need of mapping a servlet in the `web.xml` file. In this project, the `web.xml` file is used for Web specification.

The code of the `web.xml` file is shown in Listing 23.7 (you can find this file in `PeopleMgmt\people-mgmt\WEB-INF` folder in CD):

Listing 23.7: Showing the Code of the web.xml File

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <display-name>PEOPLE</display-name>
        <description>
            Kogent Solutions
        </description>
</web-app>

```

After configuring the resources of the `PeopleMgmt` project in the `web.xml` file, you can execute the project. You must ensure that you have compiled all the source files of the project and placed them in the `classes` directory. Next, you also need to package the project in a war file and deploy the war file on the Glassfish application server.

Login and Navigating to Home Page

To run the `PeopleMgmt` project, you first need to ensure that the project has been packaged and deployed on the server. Then, start the Glassfish application server and open the `http://localhost:8080/people-mgmt/servlet/people_user_login` Uniform Resource Locator (URL) in the Internet Explorer. Next, you need to access the main login servlet, i.e. `people_user_login` to execute the project. This servlet automatically redirects you to the `people_user_login` JSP page.

Figure 23.2 shows the `people_user_login` JSP page:

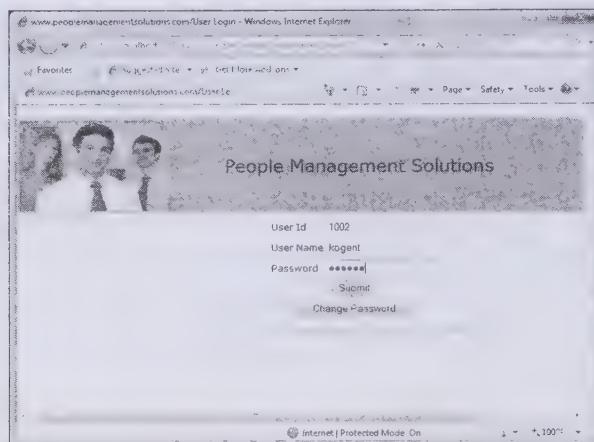


Figure 23.2: Displaying the people_user_login JSP Page with the Login User Interface

In the people_user_login JSP page, you need to enter a valid user id, user name, and password details, which are used for authentication purpose. In our case, we have entered 1002 as the user id and kogent as user name and password, as shown in Figure 23.2.

NOTE

In the `http://localhost:8080` URL, 8080 is the port number on which the Glassfish application server is running. This port number can vary depending on the port number on which the server is running on your system. You also need to create the required tables in the Oracle database from which the user id and password are retrieved to authenticate a user. The Structured Query Language (SQL) queries used to create the required tables are provided in the `PeopleMgmt/queries.txt` file in CD. You need to execute the following SQL query to insert a record in the `PEOPLE_USER_LOGIN` table of the Oracle database:

```
insert into PEOPLE_USER_LOGIN values ('1002', 'kogent',
'kogent', 'kogent', '02-Feb-2007', '02-Feb-08');
```

After entering the user id and password, click the Submit button. You should note that the form tag defined in the people_user_login JSP page does not contain the action attribute. Due to this, the form is submitted to the same servlet from where the user has forwarded the request to the people_user_login JSP page. If user id, user name, and password do not match with the details provided in the database, the people_user_login servlet redirects the user to the login page with an error message. However, if the user is authenticated, the request is forwarded to the home page of the project.

Figure 23.3 shows the home page of the project:

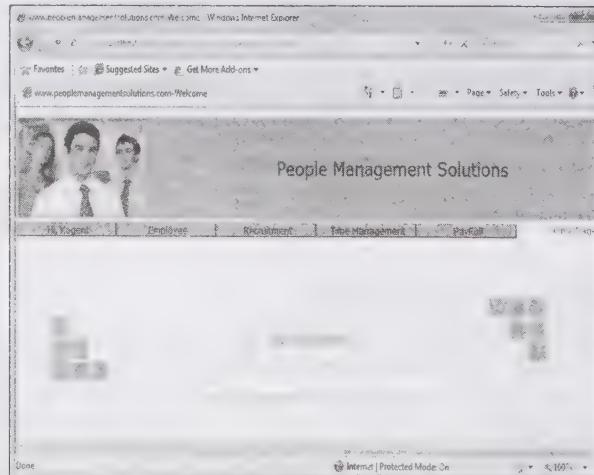


Figure 23.3: Displaying the Welcome Page after Validating All the Requested Login Data

The people_default_menu JSP page is inserted in the default page, as it contains all the navigation links for the application. This page is inserted in all the JSP pages of the application.

Changing Password

The Login module of the PeopleMgmt project also allows you to change the password of a user id. You need to create the people_user_login_pswd_change JSP page to implement the functionality of changing the password. The people_user_login_pswd_change JSP page provides five text boxes to enter the user id, user name, old and new password, and re-type the password. The business logic to change the password has been provided in the people_user_login servlet, UserLoginDBObj class, and UserLoginDBMethods class.

The code of the people_user_login_pswd_change.jsp file is shown in Listing 23.8 (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.8: Showing the Code of the people_user_login_pswd_change.jsp File

```
<html>
<head><title>www.peoplemanagementsolutions.com/Change Password</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<form name = "form1" method="post">
<table width="100%" >
<tr>
<td colspan="2">
    <%@ include file="\jsp\people_header.jsp" %>
</td>
</tr>
<tr >
<td colspan="2">
    <table border = "0"  align="center" >
        <tr><td>User Id</td>
        <td align="center" ><input type ="text" name="user_id" id="user_id" value="" /></td></tr>
        <tr><td>User Name</td>
        <td align="center" ><input type ="text" name="user_name" id="user_name" value="" /></td></tr>
        <tr><td>Old Password</td>
        <td align="center" ><input type ="password" name="old_pswd" id="old_pswd" value="" /></td></tr>
        <tr><td>New Password</td><td align="center" >
            <input type ="password" name="new_pswd" id="new_pswd" value="" /></td></tr>
        <tr><td>Retype Password</td><td align="center" >
            <input type ="password" name="retype_pswd" id="retype_pswd" value="" /></td></tr>
        <tr><td colspan="2" align="right">
            <input type ="submit" name="submit" id="submit" value="Submit" />
            <input type="hidden" name="action_submit" id="action_submit" value="login_pswd_change_submit"/>
        </td></tr>
        <% String msg = (String)session.getAttribute("lErrorMsg");
           if ( msg != null && msg.length() > 0){      %>
        <tr><td colspan="2" align="right">
            <%out.println("<div class=boldred align=center>" +msg+ "</div>"); %>
        </td></tr>
        <% } %>
    </table></td>
</tr>
<tr><td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td></tr>
</table></form></body></html>
```

In Listing 23.8, the hidden field value is used to specify the path of a servlet to which the request is redirected. Figure 23.4 shows the interface provided to change the password which is the output of the people_user_login_pswd_change JSP page:

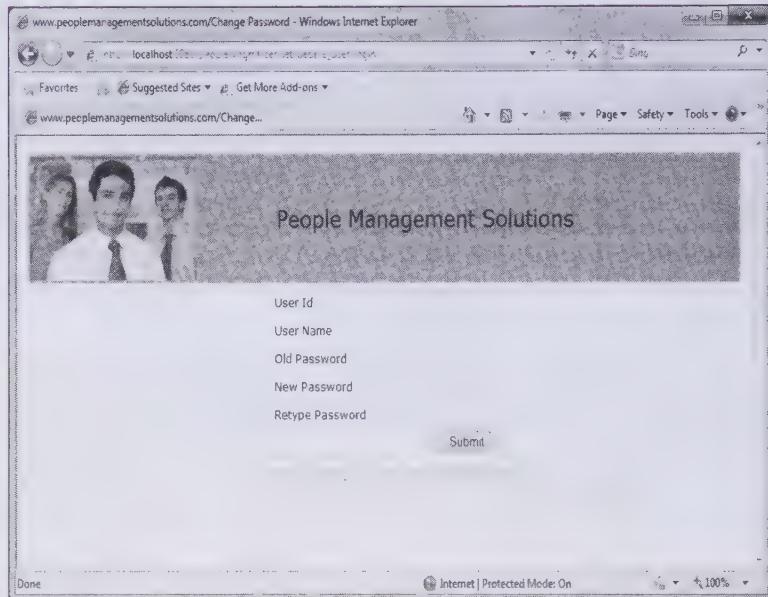


Figure 23.4: Displaying the Page Used to Change the password of a User

Figure 23.4 shows various fields to be filled to change the password.

This section has discussed how the MVC architecture is being used in the development of this project and how to implement the login process in the project. A single servlet, `people_user_login`, and two more Java classes, `UserLoginDBObj` and `serLoginDBMethods`, help the servlet to perform the functionality of user login. This servlet acts as a central controller as it intercepts all the requests. The `UserLoginDBObj` class is being used as a data transfer object, which represents Model in the MVC architecture. `people_default` is the home page of the application, which is designed for working with different modules.

The next section describes the development of module to handle profile details of all employees of the organization.

Section **B**

Developing the Profile Management Module

If you need an information on:

Implementing Logic with Servlet

See page:

1104

Creating Views

1112

The Profile Management module of the PeopleMgmt project enables an organization to maintain the profiles of all its employees. This module provides user interfaces to create or edit the profile of a person whose records are already maintained in the database. The profile-related information includes personal details, such as department name, designation, and date of joining. In this module, the details of a person are stored in the PEOPLE_EMPLOYEE table of the database. In addition, there is a search form used to search the records of the database on the basis of the employee id and the first name of an employee.

To develop this module, you need to create the following files:

- ❑ people_employee.java
- ❑ EmployeeDBObj.java
- ❑ EmployeeDBMethods.java
- ❑ GenerateId.java
- ❑ employee_insert.jsp
- ❑ employee_search.jsp
- ❑ employee_edit.jsp

Let's begin the discussion with the logic implementation using Java Servlet in the next section.

Implementing Logic with Servlet

The logic implementation of the Profile Management module is performed with the help of the people_employee servlet. The request is forwarded to the next JSP page on the basis of the parameters passed from the request page to a servlet. The people_employee servlet uses three more classes (EmployeeDBObj, EmployeeDBMethods, and GenerateId) to complete the task. This servlet decides the next JSP page according to the parameters passed from the page to the servlet. The servlet uses two more classes to complete this task, i.e. EmployeeDBObj and EmployeeDBMethods. The code for the GenerateId.java, people_employee.java, EmployeeDBObj.java, and EmployeeDBMethods.java files has been discussed in the next subsections.

Creating the people_employee Servlet

The people_employee servlet controls the navigation of the user from one JSP page to another while updating the profile of an employee. This servlet uses the `forward()` method of the `RequestDispatcher` object after setting values for target and action variables accordingly. The people_employee servlet is a controller servlet for this module.

Listing 23.9 shows the code of the `people_employee.java` file (you can find this file in the `PeopleMgmt\people-mgmt\WEB-INF\src` folder on CD):

Listing 23.9: Showing the Code of the `people_employee.java` File

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.ArrayList;
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.annotation.*;
import javax.servlet.annotation.WebServlet;
import com.Employee.EmployeeDBMethods;
import com.Employee.EmployeeDBObj;

@WebServlet(name="people_employee", urlPatterns="/servlet/people_employee")
public class people_employee extends HttpServlet
{
    String lDBUser = "";
    String lDBPswd = "";
    String lDBUrl = "";
    /**Initialize global variables*/
    @Override
    public void init(ServletConfig config) throws ServletException{
```

```

System.out.println("initializing controller servlet.");
ServletContext context = config.getServletContext();
lDBUser = "scott";
lDBPswd = "tiger";
lDBUrl = "jdbc:oracle:thin:@192.168.1.123:1521:"+"XE";
super.init(config);
}
}
/**Process the HTTP Get request*/
@Override
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException{
    doPost(request, response);
}
/**Process the HTTP Post request*/
@Override
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    HttpSession session = request.getSession();
    session.setAttribute("lErrorMsg",null);
    String target = "";
    String action = request.getParameter("action");
    String lDBOpr = "";
    EmployeeDBMethods employeeDBMethods = new
        EmployeeDBMethods(lDBUser,lDBPswd,lDBUrl);
    lDBOpr = (String)request.getParameter("dbopr");
    if( (lDBOpr != null && lDBOpr.length() > 0) && (lDBOpr.equals("create")) ){
        target = "/jsp/employee_insert.jsp";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) && (lDBOpr.equals("edit")) ){
        target = "/jsp/employee_search.jsp";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) && (lDBOpr.equals("show")) ){
        ArrayList employeeList = new ArrayList();
        String criteria = "";
        employeeList =
            (ArrayList)employeeDBMethods.selectEmployeeByCriteria(criteria);
        session.setAttribute("EmployeeList",employeeList);
        target = "/jsp/employee_list.jsp";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) &&
        (lDBOpr.equals("detail")) ){
        String lEmpId = "";
        String lEmpFName = "";
        lEmpId = (String)request.getParameter("emp_id");
        lEmpFName = (String)request.getParameter("emp_f_name");
        EmployeeDBObj employeeDBObj = new EmployeeDBObj();
        employeeDBObj =
            (EmployeeDBObj)employeeDBMethods.getRecordByPrimaryKey(lEmpId,lEmpFName);
        System.out.println("iiiii="+employeeDBObj.emp_id+"fffff="+employeeDBObj.emp_f_name);
        session.setAttribute("employeeDBObj",employeeDBObj);
        target = "/jsp/employee_profile.jsp";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) && (lDBOpr.equals("delete")) ){
        String emp_Id = "";
        emp_Id = (String)request.getParameter("emp_id");
        employeeDBMethods.deleteEmployee(emp_Id);
        String criteria = "";
        ArrayList employeeList = new ArrayList();
        employeeList =
            (ArrayList)employeeDBMethods.selectEmployeeByCriteria(criteria);
        session.setAttribute("EmployeeList",employeeList);
    }
}

```

```

target = "/jsp/employee_list.jsp";
}
String action_submit = request.getParameter("action_submit");
String action_edit = request.getParameter("action_edit");
System.out.println("action_submit=="+action_submit);
if ( action_submit != null || action_edit != null ){
    if ( request.getParameter("submit").equals("Submit") ){
        System.out.println("in the Submit");
        if ( action_submit.equals("people_employee_insert_submit") ){
            System.out.println("in the people_employee_insert_submit ");
            action = "people_employee_insert_submit";
        }
        else
            if (action_submit.equals("login_pswd_change_submit")){
                action = "login_pswd_change_submit";
            }
            else
                if (action_submit.equals("people_employee_search_submit")){
                    action = "people_employee_search_submit";
                }
    }
    else
        if ( request.getParameter("submit").equals("Edit") ){
            if ( action_edit.equals("people_employee_edit_submit") ){
                action = "people_employee_edit_submit";
                EmployeeDBObj popEmployeeDBObj = new EmployeeDBObj();
                popEmployeeDBObj =
(EmployeeDBObj)employeeDBMethods.populateEmployeeDBObjFromReq(request);
                int rval =
                    employeeDBMethods.updateEmployeeByPrimaryKey(popEmployeeDBObj);
                if ( rval > 0 ){
                    EmployeeDBObj employeeDBObj = new EmployeeDBObj();
                    employeeDBObj =
(EmployeeDBObj)employeeDBMethods.getRecordByPrimaryKey(popEmployeeDBObj.emp_id,pop
EmployeeDBObj.emp_f_name);
                    session.setAttribute("employeeDBObj",employeeDBObj);
                    String lErrorMsg = "Employee is Updated!!";
                    session.setAttribute("lErrorMsg",lErrorMsg);
                    target = "/jsp/employee_list.jsp";
                }
            }
        }
    if (action!=null){
        System.out.println("in the "+action);
        if (action.equals("people_employee_search_submit")){
            String lEmpId = "";
            String lEmpFName = "";
            lEmpId = (String)request.getParameter("emp_id");
            lEmpFName = (String)request.getParameter("emp_f_name");
            EmployeeDBObj employeeDBObj = new EmployeeDBObj();
            employeeDBObj =
(EmployeeDBObj)employeeDBMethods.getRecordByPrimaryKey(lEmpId,lEmpFName);
            System.out.println("iiiii="+employeeDBObj.emp_id+"fffff="+employeeDBObj.emp_f_name);
        }
        if ( (employeeDBObj.emp_id != null && employeeDBObj.emp_f_name != null) ){
            session.setAttribute("employeeDBObj",employeeDBObj);
            target = "/jsp/employee_edit.jsp";
        }
        else{
            String lErrorMsg = "Employee doesn't Exist";
            session.setAttribute("lErrorMsg",lErrorMsg);
            System.out.println("Employee:" + lErrorMsg);
            target = "/jsp/people_default.jsp";
        }
    }
}

```

```

}
else {
    if (action.equals("people_change_pswd_submit")){
        target = "/jsp/people_user_login_pswd_change.jsp";
    }
    else {
        if (action.equals("people_employee_insert_submit")){
            EmployeeDBObj popEmployeeDBObj = new EmployeeDBObj();
            popEmployeeDBObj =
                (EmployeeDBObj)employeeDBMethods.populateEmployeeDBObjFromReq(request);
            EmployeeDBObj employeeDBObj = new EmployeeDBObj();
            employeeDBObj =
                (EmployeeDBObj)employeeDBMethods.getRecordByPrimaryKey(popEmployeeDBObj.emp_id,pop
EmployeeDBObj.emp_f_name);
            if ( (popEmployeeDBObj.emp_id).equals(employeeDBObj) &
                (popEmployeeDBObj.emp_f_name).equals(employeeDBObj.emp_f_name) ){
                String lErrorMsg = "Employee Already Exist";
                session.setAttribute("lErrorMsg",lErrorMsg);
                System.out.println("Employee:" + lErrorMsg);
                target = "/jsp/people_default.jsp";
            }
        }
        else{
            int rval = employeeDBMethods.insertEmployee(popEmployeeDBObj);
            EmployeeDBObj sEmployeeDBObj = new EmployeeDBObj();
            sEmployeeDBObj =
                (EmployeeDBObj)employeeDBMethods.getRecordByPrimaryKey(popEmployeeDBObj.emp_id,pop
EmployeeDBObj.emp_f_name);
            session.setAttribute("employeeDBObj",sEmployeeDBObj);
            String lErrorMsg = "Employee is Added!!";
            session.setAttribute("lErrorMsg",lErrorMsg);
            target = "/jsp/people_inserted.jsp";
        }
    }
}
else
if (action.equals("people_employee_edit_submit")){
    EmployeeDBObj popEmployeeDBObj = new EmployeeDBObj();
    popEmployeeDBObj =
        (EmployeeDBObj)employeeDBMethods.populateEmployeeDBObjFromReq(request);
    int rval = employeeDBMethods.updateEmployeeByPrimaryKey(popEmployeeDBObj);
    if ( rval > 0 ){
        EmployeeDBObj employeeDBObj = new EmployeeDBObj();
        employeeDBObj =
            (EmployeeDBObj)employeeDBMethods.getRecordByPrimaryKey(popEmployeeDBObj.emp_id,pop
EmployeeDBObj.emp_f_name);
        session.setAttribute("employeeDBObj",employeeDBObj);
        String lErrorMsg = "Employee is Updated!!";
        session.setAttribute("lErrorMsg",lErrorMsg);
        target = "/jsp/employee_list.jsp";
    }
    else
        if (action.equals("people_employee_detail")){
            ArrayList employeeList = new ArrayList();
            String criteria="";
            employeeList =
                (ArrayList)employeeDBMethods.selectEmployeeByCriteria(criteria);
            session.setAttribute("EmployeeList",employeeList);
            target = "/jsp/employee_list.jsp";
        }
    else
        if (action.equals("people_employee_delete")){
            String emp_Id = "";
        }
}
} // (action== null )
/* forwarding the request/response to the targeted view */

```

```

RequestDispatcher requestDispatcher = null;
getServletContext().getRequestDispatcher(target);
requestDispatcher.forward(request, response);
} // doPost closed
}// class closed

```

In Listing 23.9, the @WebServlet annotation is used to define the servlet mapping, while the @Override annotation specifies that the methods, such as `doPost()`, and `doGet()` are overridden. The business logic embedded in this servlet uses various set of variables, such as `action` and `target` and the parameters passed with the request to find out the proper execution path. The value of the next page to be executed depends on the value held by the `dbopr` parameter. The two class files, i.e. `EmployeeDBObj` and `EmployeeDBMethods`, used in the `people_employee` servlet are discussed in the following sections.

Creating the EmployeeObj Class

`EmployeeDBObj` is a Java class having member variables matching the columns of the `PEOPLE_EMPLOYEE` table. It is used as a JavaBean to hold the data related to the profile of an employee. An instance of this class represents a single record in the `PEOPLE_EMPLOYEE` table.

Listing 23.10 shows the code of the `EmployeeDBObj.java` file (you can find this file in the `PeopleMgmt\people-mgmt\WEB-INF\src\com\Employee` folder on CD):

Listing 23.10: Showing the Code of the `EmployeeDBObj.java` File

```

package com.Employee;
public class EmployeeDBObj
{
    public String emp_id ;
    public String emp_f_name ;
    public String emp_m_name ;
    public String emp_l_name ;
    public String org_id ;
    public String level_id ;
    public String dept_id ;
    public String dob ;
    public String dojoin ;
    public String address_1;
    public String address_2 ;
    public String city ;
    public String state;
    public String nationality ;
}

```

Creating the EmployeeDBMethods Class

`EmployeeDBMethods` class provides various methods, such as `insertEmployee`, `updateEmployeeByPrimaryKey`, `populateEmployeeDBObjFromReq`, and `getRecordByPrimaryKey` to manipulate the records of the database directly. The `insertEmployee(EmployeeDBObj)` method inserts a new record in the `PEOPLE_EMPLOYEE` table. Similarly, the `updateEmployeeByPrimaryKey(EmployeeDBObj)` method updates the table with new information set with the `EmployeeDBObj` object passed to the method as an argument. Other methods are used to populate the `EmployeeDBObj` object with the data either fetched from the table (`getRecordByPrimaryKey()` method) or sent with the request (`populateEmployeeDBObjFromReq()` method).

Listing 23.11 shows the code of the `EmployeeDBMethods.java` file (you can find this file in the `PeopleMgmt\people-mgmt\WEB-INF\src\com\Employee` folder on CD):

Listing 23.11: Showing the Code of the `EmployeeDBMethods.java` File

```

package com.Employee;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.util.ArrayList;

```

```

import com.Employee.EmployeeDBObj;
public class EmployeeDBMethods
{
    public String DBUser;
    public String DBPswd;
    public String DBUrl;

    public EmployeeDBMethods()
    {
        DBUser = inDBUser;
        DBPswd = inDBPswd;
        DBUrl = inDBUrl;
    }

    public void initializeEmployeeDBObj(EmployeeDBObj inEmployeeDBObj)
    {
        inEmployeeDBObj.emp_id = "";
        inEmployeeDBObj.emp_f_name = "";
        inEmployeeDBObj.emp_m_name = "";
        inEmployeeDBObj.emp_l_name = "";
        inEmployeeDBObj.org_id = "";
        inEmployeeDBObj.level_id = "";
        inEmployeeDBObj.dept_id = "";
        inEmployeeDBObj.dob = "";
        inEmployeeDBObj dojojoin = "";
        inEmployeeDBObj.address_1 = "";
        inEmployeeDBObj.address_2 = "";
        inEmployeeDBObj.city = "";
        inEmployeeDBObj.state = "";
        inEmployeeDBObj.nationality = "";
    }

    public EmployeeDBObj getRecordByPrimaryKey(String inEmpId, String inEmpFName)
    {
        EmployeeDBObj employeeDBObj = new EmployeeDBObj();
        java.sql.Date date;
        try
        {
            System.out.println("DBUser=="+DBUser+",DBPswd=="+DBPswd+","
                    +DBUrl);
            DriverManager.registerDriver(new
                    oracle.jdbc.driver.OracleDriver());
            Connection conn= DriverManager.getConnection
                    (DBUrl,DBUser,DBPswd);
            Statement stmt = conn.createStatement();
            String lSqlString = "select * from PEOPLE_EMPLOYEE ";
            lSqlString = lSqlString + "where emp_id='"+inEmpId+"' ";
            if( inEmpFName != null && inEmpFName.length() > 0 )
                lSqlString = lSqlString + "and emp_f_name='"+inEmpFName+"' ";
            ResultSet rs = null;
            rs = stmt.executeQuery(lSqlString);
            System.out.println("lSqlString====trtrt==within"
                    +getRecordByPrimaryKey+" "+lSqlString);
            if( rs.next())
            {
                System.out.println("fffff=="+rs.getString("emp_id"));
                employeeDBObj.emp_id = rs.getString("emp_id");
                employeeDBObj.emp_f_name = rs.getString("emp_f_name");
                employeeDBObj.emp_m_name = rs.getString("emp_m_name");
                employeeDBObj.emp_l_name = rs.getString("emp_l_name");
                employeeDBObj.org_id = rs.getString("org_id");
                employeeDBObj.level_id = rs.getString("level_id");
                employeeDBObj.dept_id = rs.getString("dept_id");
                date=rs.getDate("dob");
                employeeDBObj.dob = date.toString();
                date=rs.getDate("dojoin");
                employeeDBObj.dojojoin = date.toString();
            }
        }
    }
}

```

```

employeeDBObj.dojoin = date.toString();
employeeDBObj.address_1 = rs.getString("address_1");
employeeDBObj.address_2 = rs.getString("address_2");
employeeDBObj.city = rs.getString("city");
employeeDBObj.state = rs.getString("state");
employeeDBObj.nationality = rs.getString("nationality");
System.out.println("fffff===="+rs.getString("emp_id"));
}
else
{
    initializeEmployeeDBObj(employeeDBObj);
}
System.out.println("fffff===="+employeeDBObj.emp_id);
}
catch(SQLException ex)
{
    ex.printStackTrace();
}
return employeeDBObj;
}
public int updateEmployeeByPrimaryKey(EmployeeDBObj inEmployeeDBObj)
{
    int recupd = 0;
    String lQuery = "";
    lQuery = lQuery + "update PEOPLE_EMPLOYEE set ";
    emp_m_name = "+inEmployeeDBObj.emp_m_name+" ";
    lQuery += ", emp_l_name = "+inEmployeeDBObj.emp_l_name+" ";
    lQuery += ", org_id = "+inEmployeeDBObj.org_id+" ";
    lQuery += ", level_id = "+inEmployeeDBObj.level_id+" ";
    lQuery += ", dept_id = "+inEmployeeDBObj.dept_id+" ";
    lQuery += ", dob = to_date('"+inEmployeeDBObj.dob+"', 'yyyy-mm-dd') ";
    lQuery += ", dojoin = to_date('"+inEmployeeDBObj.dojoin+"', 'yyyy-mm-dd') ";
    lQuery += ", address_1 = "+inEmployeeDBObj.address_1+" ";
    lQuery += ", address_2 = "+inEmployeeDBObj.address_2+" ";
    lQuery += ", city = "+inEmployeeDBObj.city+" ";
    lQuery += ", state = "+inEmployeeDBObj.state+" ";
    lQuery += ", nationality = "+inEmployeeDBObj.nationality+" ";
    lQuery += "where emp_id = "+inEmployeeDBObj.emp_id+" ";
    lQuery += "and emp_f_name = "+inEmployeeDBObj.emp_f_name+" ";
    System.out.println("lSqlString====:"+lQuery);
    try
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.oracleDriver());
        Connection conn = DriverManager.getConnection(DBUrl, DBUser, DBPswd);
        Statement stmt = conn.createStatement();
        recupd = stmt.executeUpdate(lQuery);
    }
    catch(SQLException ex)
    {
        ex.printStackTrace();
    }
    return recupd;
}
public EmployeeDBObj populateEmployeeDBObjFromReq(HttpServletRequest inReq)
{
    EmployeeDBObj employeeDBObj = new EmployeeDBObj();
    employeeDBObj.emp_id = (String)inReq.getParameter("emp_id");
    employeeDBObj.emp_f_name = (String)inReq.getParameter("emp_f_name");
    employeeDBObj.emp_m_name = (String)inReq.getParameter("emp_m_name");
    employeeDBObj.emp_l_name = (String)inReq.getParameter("emp_l_name");
    employeeDBObj.org_id = (String)inReq.getParameter("org_id");
    employeeDBObj.level_id = (String)inReq.getParameter("level_id");
    employeeDBObj.dept_id = (String)inReq.getParameter("dept_id");
    employeeDBObj.dob = (String)inReq.getParameter("dob");
}

```

```

employeeDBObj.dojoin = (String)inReq.getParameter("dojoin");
employeeDBObj.address_1 = (String)inReq.getParameter("address_1");
employeeDBObj.address_2 = (String)inReq.getParameter("address_2");
employeeDBObj.city = (String)inReq.getParameter("city");
employeeDBObj.state = (String)inReq.getParameter("state");
employeeDBObj.nationality = (String)inReq.getParameter("nationality");
return employeeDBObj;
}

public int insertEmployee(EmployeedBObj inEmployeeDBObj){
    int recupd = 0;
    String lQuery = "";
    lQuery = lQuery +"insert into PEOPLE_EMPLOYEE values ( ";
    lQuery = lQuery +"'"+inEmployeeDBObj.emp_id+"' ";
    lQuery = lQuery +"'"+inEmployeeDBObj.emp_f_name+"' ";
    lQuery = lQuery +"'"+inEmployeeDBObj.emp_m_name+"' ";
    lQuery = lQuery +"'"+inEmployeeDBObj.emp_l_name+"' ";
    lQuery = lQuery +"'"+inEmployeeDBObj.org_id+"' ";
    lQuery = lQuery +"'"+inEmployeeDBObj.level_id+"' ";
    lQuery = lQuery +"'"+inEmployeeDBObj.dept_id+"' ";
    lQuery = lQuery +", to_date('"+inEmployeeDBObj.dob+"',
'yyyy-mm-dd')";
    lQuery = lQuery +", to_date('"+inEmployeeDBObj.dojoin+"',
'yyyy-mm-dd')";
    lQuery = lQuery +", '"+inEmployeeDBObj.address_1+"' ";
    lQuery = lQuery +", '"+inEmployeeDBObj.address_2+"' ";
    lQuery = lQuery +", '"+inEmployeeDBObj.city+"' ";
    lQuery = lQuery +", '"+inEmployeeDBObj.state+"' ";
    lQuery = lQuery +", '"+inEmployeeDBObj.nationality+"' ";
    lQuery = lQuery +")";
    System.out.println("lsqlString==:" +lQuery);

    try
    {
        DriverManager.registerDriver(new
            oracle.jdbc.driver.OracleDriver());
        Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
        Statement stmt = conn.createStatement();
        recupd = stmt.executeUpdate(lQuery);
    }

    catch(SQLException ex)
    {
        ex.printStackTrace();
    }
    return recupd;
}
}

```

In Listing 23.11, there are several methods for adding, updating, and deleting profiles of people. These methods are then used in the `people_employee` servlet. The description of the `GenerateId` class is provided in the next subsection.

Creating the GenerateId Class

The `GenerateId` class is developed for the purpose of auto-generation of Employee Id, which is the primary key in the database. As you know that a primary key in the database should be unique; however, a user can enter duplicate Employee Id that causes an error or exception. This is because a record with duplicate primary key cannot be added into a database. To avoid this situation, the `GenerateId` class is designed, which automatically generates the value of the Employee Id field.

Listing 23.12 shows the code for the `GenerateId.java` file (you can find this file in the `PeopleMgmt\people-mgmt\WEB-INF\src\com\Employee` folder on CD):

Listing 23.12: Showing the Code of the GenerateId.java File

```

package com.Employee;
import java.sql.*;
import javax.sql.*;
public class GenerateId
{
    public int generateEmployeeId(){
        int emp_id=0;
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection conn=DriverManager.getConnection("jdbc:oracle:thin:@192.168.1.123:1521:"+XE,"scott","tiger");
            Statement stmt = conn.createStatement();
            String query="select max(emp_id) as emp_id from PEOPLE_EMPLOYEE ";
            ResultSet rs = null;
            rs = stmt.executeQuery(query);
            if(rs.next()){
                String id = rs.getString("emp_id");
                emp_id=Integer.parseInt(id);
            }
            emp_id = emp_id + 1;
        }
        catch(SQLException ex){
            ex.printStackTrace();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
        return emp_id;
    }
}

```

The employee id or the value of the `emp_id` column is automatically generated for the record of a new employee because the query executed in Listing 23.12 returns the maximum value of the `emp_id` column and then the value is incremented by 1.

Creating Views

There are three interfaces designed as view components in this module. The first interface (`employee_list`) is used to display the complete list of employees whose records are maintained in the database, and the other interface (`employee_search`) provides the fields on the basis of which the records of employees are searched. The third interface (`employee_edit`) is used to edit the details of an employee. These three interfaces are created by three JSP pages, namely `employee_insert`, `employee_edit`, and `employee_search`. The code for these JSP pages is discussed under the following sections.

Creating the employee_insert JSP Page

The `employee_insert` JSP page provides a form to be filled with the details of a new employee. The profile of the employee is normally created at the joining time of the employee. This form includes various fields, such as Employee Id, First Name, Middle Name, Last Name, Department, Date of Birth, Join Date, Address1, Address2, City, State, and Nationality. The client-side validation is used to specify the mandatory fields that must be filled by a user. If the user leaves any mandatory field as blank, a message is displayed to fill that field. The user can also view the `employee_insert` JSP page by clicking the Create Profile hyperlink, which calls the `people_employee` servlet. After clicking the hyperlink, the user's request is redirected to the `employee_insert` JSP page, which displays all the fields that are to be filled by the user.

Listing 23.13 shows the code of the `employee_insert.jsp` file (you can find this file in the `PeopleMgmt\people-mgmt\jsp` folder on CD):

Listing 23.13: Showing the Code of the employee_insert.jsp File

```

<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.Employee.*" %>
<% GenerateId gen=new GenerateId();
   int emp_id=gen.generateEmployeeId();%>
</%
<html>
<head>
<title>www.peoplemanagementsolutions.com/Create Profile</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
<script language="javascript">
    function validateForm() {
        var myName=document.form1.emp_f_name.value;
        if (myName == "") {
            alert ("First name cannot be blank");
            document.form1.emp_f_name.focus();
            return false;
        }
        myName=document.form1.emp_l_name.value;
        if (myName == "") {
            alert ("Last name cannot be blank");
            document.form1.emp_l_name.focus();
            return false;
        }
        myName=document.form1.dob.value;
        if (myName == "") {
            alert ("Date of Birth cannot be blank");
            document.form1.dob.focus();
            return false;
        }
        myName=document.form1.address_1.value;
        if (myName == "") {
            alert ("Address1 cannot be blank");
            document.form1.address_1.focus();
            return false;
        }
        myName=document.form1.city.value;
        if (myName == "") {
            alert ("City cannot be blank");
            document.form1.city.focus();
            return false;
        }
        myName=document.form1.state.value;
        if (myName == "") {
            alert ("State cannot be blank");
            document.form1.state.focus();
            return false;
        }
        myName=document.form1.level_id.value;
        if (myName == "Select Designation") {
            alert ("You Should select a Designation");
            document.form1.level_id.focus();
            return false;
        }
        myName=document.form1.dept_id.value;
        if (myName == "Select Department") {
            alert ("You Should select a Department");
            document.form1.dept_id.focus();
            return false;
        }
        myName=document.form1.dojoin.value;
        if (myName == "") {
            alert ("Date of Join field cannot be blank");
            document.form1.dojoin.focus();
            return false;
        }
        myName=document.form1.nationality.value;
        if (myName == "Select Nationality") {
            alert ("You Should select a Nationality");
            document.form1.nationality.focus();
        }
    }
</script>

```

```

        return false;
    }
    form1.submit();
}
</script>
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2"><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td width="900">
<%@ include file="../jsp/people_default_menu.jsp" %>
</td>
</tr><tr>
<td width="750" valign="top" align = "center">
<p>&nbsp;</p>
Enter Profile Detail for New Employee.
<hr width=100% color="#AAAAAA"/>
<table border="0" align="top" width=100% >
<form name="form1" method="post">
<input type='hidden' name='emp_id' id='emp_id' size ='10' value='<%=emp_id%>'/>
<tr>
<td>Employee Id</td>
<td align='left'><input type='text' disabled='disabled' name='emp_id' id='emp_id' size ='10' value='<%=emp_id%>' /></td>
<td>&nbsp;</td>
<td align='left'>&nbsp;</td>
</tr>
<tr>
<td>First Name<sup>*</sup></td>
<td><input type='text' name='emp_f_name' id='emp_f_name' size ='10' value=''/></td>
<td>
Middle Name</td><td><input type='text' name='emp_m_name' id='emp_m_name' size ='10' value=''/></td>
<td>Last Name<sup>*</sup></td><td><input type='text' name='emp_l_name' id='emp_l_name' size ='10' value=''/>
</td>
</tr>
<tr>
<td>Org Id</td><td><input type='hidden' name='org_id' id='org_id' size ='10' value="KLSI"/>
<td align='left'><input type='text' name='org_id_dup' id='org_id_dup' disabled='disabled' size ='10' value="KLSI"/>
</td>
<td>Designation<sup>*</sup></td>
<td align='left'><select name="level_id" id="level_id">
<option value='Select Designation' selected> Select Designation
<option value='CW'> Content Writer
<option value='TS'> Tester
<option value='HR'> Human Resource Manager
<option value='AC'> Accountant
<option value='AM'> Admin Manager
<option value='EM'> Event Manager
<option value='TL'> Team Leader
<option value='PM'> Project Manager
<option value='TR'> Trainer
</select></td></td>
</tr>
<tr>
<td>Department<sup>*</sup></td>
<td align='left'><select id='dept_id' name='dept_id'>
<option value='Select Department' selected> Select Department
<option value='CS'> Content Solutions
<option value='TS'> Testing
<option value='HR'> Human Resource
<option value='AC'> Accounts
<option value='AD'> Administration
<option value='EM'> Event Management
</select></td>

```

```

<td>DOB<sup>*</sup></td>
<td align='left'><input type='text' name='dob' id='dob' size ='10' value=''/>
(yyyy-mm-dd)</td>
</tr>
<tr>
<td>JoinDate<sup>*</sup></td>
<td align='left' colspan='2'><input type='text' name='dojoin' id='dojoin' size
='10' value=''/>(yyyy-mm-dd)</td>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td>Address1<sup>*</sup></td>
<td align='left' colspan='2'><input type='text' name='address_1' id='address_1'
size ='40' value=''/> </td>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td>Address2</td>
<td align='left' colspan='2'><input type='text' name='address_2' id='address_2'
size ='40' value=''/> </td>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td>City<sup>*</sup></td>
<td align='left'><input type='text' name='city' id='city' size ='10' value=''/>
</td>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td>State <sup>*</sup></td>
<td align='left'><input type='text' name='state' id='state' size ='10' value=''/>
</td>
<td>Nationality<sup>*</sup></td>
<td align='left'>
<select name='nationality' id='nationality'>
<option selected>Select Nationality
<option value='IN'> Indian
<option value='RS'> Russian
<option value='PK'> Pakistani
<option value='AM'> American
<option value='BR'> British
<option value='SR'> Srilankan
</select></td>
</tr>
<tr>
<td> All the ( <sup>*</sup>) marked are mandatory</td></tr><tr>
<td align='center' colspan='4'>
<input type="submit" name="submit" id="submit" size ="10" value="Submit"
onClick="return validateForm()"/>
<input type='hidden' name='action_submit' id='action_submit' size ='10'
value='people_employee_insert_submit' />
</td>
</tr>
</table>
<hr width=100% color=#AAAAAA>
</td>
</tr>
<tr>
<td colspan="2"><%@include file="..../jsp/people_footer.jsp"%></td>
</tr>
</table>
</body>
</html>

```

When you click the Create Profile submenu from the Employee menu, the employee_insert JSP page is displayed, as shown in Figure 23.5:

Figure 23.5: Displaying the Details for a New Employee in the employee_insert JSP Page

After filling up and submitting the employee_insert JSP page, the request is forwarded to the people_employee servlet. The value of the dbopr parameter in the query string is set as create, which invokes the insertEmployee() method of the EmployeeDBMethods Java class, resulting in the insertion of a new record to the PEOPLE_EMPLOYEE table after checking the existence of such an employee. No two employees in the table can have the same Employee Id.

Creating the employee_search JSP Page

The employee_search JSP page is designed to search a particular record of an employee on the basis of the employee id and first name of the employee entered in the page. When this JSP page is submitted, the people_employee servlet sets the action attribute and then calls the getRecordByPrimaryKey(1EmpId, 1EmpFName) method on the object of the EmployeeDBMethods Java class. The EmployeeDBObj object returned by this method is set as a session attribute that can be used in the next JSP page (employee_edit) to display information corresponding to a particular employee.

Listing 23.14 shows the code of the employee_search.jsp file (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.14: Showing the Code of the employee_search.jsp File

```
<%@ page language="java" %>
<%@ page session="true" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Search</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2">
<%@ include file="../jsp/people_header.jsp" %>
</td>
</tr>
<tr>
<td width="900">
```

```

<%@ include file="../jsp/people_default_menu.jsp" %>
</td></tr><tr>
<td width = "750" valign="top" align = "center">
<p>&nbsp;</p>
<div align=center>Enter Employee Id & First Name</div>
<hr bgcolor="#AAAAAA" width=500>
<table border="0" align="top" width=50% align="right">
<form name="form1" method="post">
<tr>
<td>Employee Id</td>
<td align='left'><input type='text' name='emp_id' id='emp_id' size ='10' value=''/></td>
</tr>
<tr>
<td>First Name</td>
<td align='left'><input type='text' name='emp_f_name' id='emp_f_name' size ='10' value=''/></td>
</tr>
<tr>
<td align='center' colspan='2' >
<input type='submit' name='submit' id='submit' size ='10' value='Submit' />
</td>
</tr>
<input type='hidden' name='action_submit' id='action_submit' size ='10' value='people_employee_search_submit' />
</form>
</table>
<hr bgcolor="#AAAAAA" width=500>
</td>
</tr>
<tr>
<td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr>
</table>
</body>
</html>

```

Click the **Edit Profile** link from the navigation bar; a search page appears containing two fields – Employee Id and First Name. You need to provide the employee id and first name of the employee whose details need to be searched, as shown in Figure 23.6:

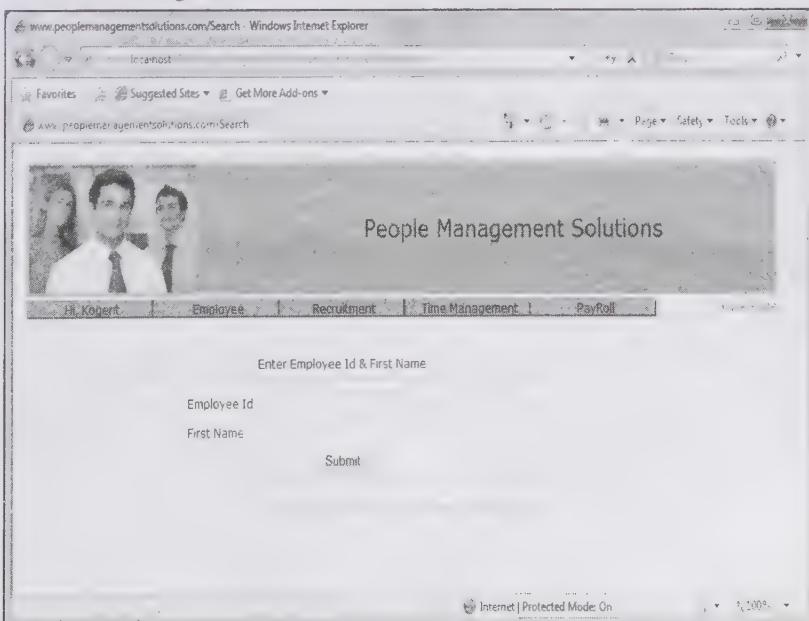


Figure 23.6: Displaying the employee_search JSP Page to Search an Employee

Figure 23.6 displays the employee_search JSP page with two fields, Employee Id and First Name, to be entered by a user. On the basis of the values entered in these fields, an employee record is searched and displayed for editing. If the record is found in the database, the employee id is forwarded to the employee_edit JSP page; otherwise, it is forwarded to the home page with the Employee does not exist error message.

The next section discusses about the employee_edit JSP page.

Creating the employee_edit JSP Page

The employee_edit JSP page is similar to the employee_create JSP page; except that the two fields, Employee Id and First Name, are disabled in the employee_edit JSP page. A user can enter the updated data in all other fields of the employee_edit page on the basis of which the record in the database is updated for new values. The code for the employee_edit JSP page is provided in Listing 23.15 in which the EmployeeDBObj object is used to retrieve the values set by the people_employee servlet.

Listing 23.15 shows the code of the employee_edit.jsp file (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.15: Showing the Code of the employee_edit.jsp File

```
<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.Employee.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Edit Employee</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2"><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td width="900">
    <%@ include file="../jsp/people_default_menu.jsp" %></td> </tr><tr>
<td width = "750" valign="top" align = "center">
<p>&nbsp;</p>
Employee Detail
<hr width=100% color=#AAAAAA>
<table border="0" align="top" width=100% >
<%
    EmployeeDBObj employeeDBObj = new EmployeeDBObj();
    employeeDBObj = (EmployeeDBObj)session.getAttribute("employeeDBObj");
%>
<form name="form1" method="post">
<input type='hidden' name='emp_id' id='emp_id' size ='10'
value="<%=employeeDBObj.emp_id%>" />
<input type='hidden' name='emp_f_name' id='emp_f_name' size ='10'
value="<%=employeeDBObj.emp_f_name%>" />
<tr>
<td>Employee Id</td>
<td align='left'>
<input type='text' disabled='disabled' name='emp_id_dup' id='emp_id_dup' size
='10' value="<%=employeeDBObj.emp_id%>" />
</td>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td>First Name</td>
<td align='left'>
<input type='text' disabled='disabled' name='emp_f_name_dup' id='emp_f_name_dup'
size ='10' value="<%=employeeDBObj.emp_f_name%>" />
```

```

</td>
<td colspan=2>
Middle Name  &nbsp;&nbsp;&nbsp; Last Name
<input type='text' name='emp_m_name' id='emp_m_name' size ='10'
value="<%>employeeDBObj.emp_m_name%>"/>
&nbsp;&nbsp;&nbsp;Last Name&nbsp;&nbsp;&nbsp;
<input type='text' name='emp_l_name' id='emp_l_name' size ='10'
value="<%>employeeDBObj.emp_l_name%>"/>
</td>
</tr>
<tr>
<td>Org Id</td>
<td align='left'>
<input type='text' name='org_id' id='org_id' size ='10'
value="<%>employeeDBObj.org_id%>"/>
</td>
<td>Designation</td>
<td align='left'>
<input type='text' name='level_id' id='level_id' size ='10'
value="<%>employeeDBObj.level_id%>"/>
</td>
</tr>
<tr>
<td>Dept</td>
<td align='left'>
<input type='text' name='dept_id' id='dept_id' size ='10'
value="<%>employeeDBObj.dept_id%>"/>
</td>
<td>DOB</td>
<td align='left'><input type='text' name='dob' id='dob' size ='10'
value="<%>employeeDBObj.dob%>"/>
(yyyy-mm-dd)</td>
</tr>
<tr>
<td>Join Date</td>
<td align='left'>
<input type='text' name='dojoin' id='dojoin' size ='10'
value="<%>employeeDBObj.dojoin%>"/>
(yyyy-mm-dd)</td>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td>Address1</td>
<td align='left' colspan='2'>
<input type='text' name='address_1' id='address_1' size ='40'
value="<%>employeeDBObj.address_1%>"/></td>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td>Address2</td>
<td align='left' colspan='2'>
<input type='text' name='address_2' id='address_2' size ='40'
value="<%>employeeDBObj.address_2%>"/>
</td>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td>City</td>
<td align='left'>
<input type='text' name='city' id='city' size ='10'
value="<%>employeeDBObj.city%>"/>
</td>

```

```

<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td>State</td>
<td align='left'>
<input type='text' name='state' id='state' size ='10'
value="<%employeeDBObj.state%>" />
</td>
<td>Nationality</td>
<td align='left'>
<input type='text' name='nationality' id='nationality' size ='10'
value="<%employeeDBObj.nationality%>" />
</td>
</tr>
<tr>
<td align='center' colspan='4'>
<input type='submit' name='submit' id='submit' size ='10' value='Edit' />
</td>
<input type='hidden' name='action_edit' id='action_edit' size ='10'
value='people_employee_edit_submit' />
</td>
</tr>
</table>
<hr width=100% color="#AAAAAA">
</td>
</tr>
<tr>
<td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr>
</table>
</body>
</html>

```

In Listing 23.15, the code is provided to disable the fields that are automatically generated using the GenerateId class.

Figure 23.7 shows the employee_edit form for updating a record of an employee with two disabled fields:

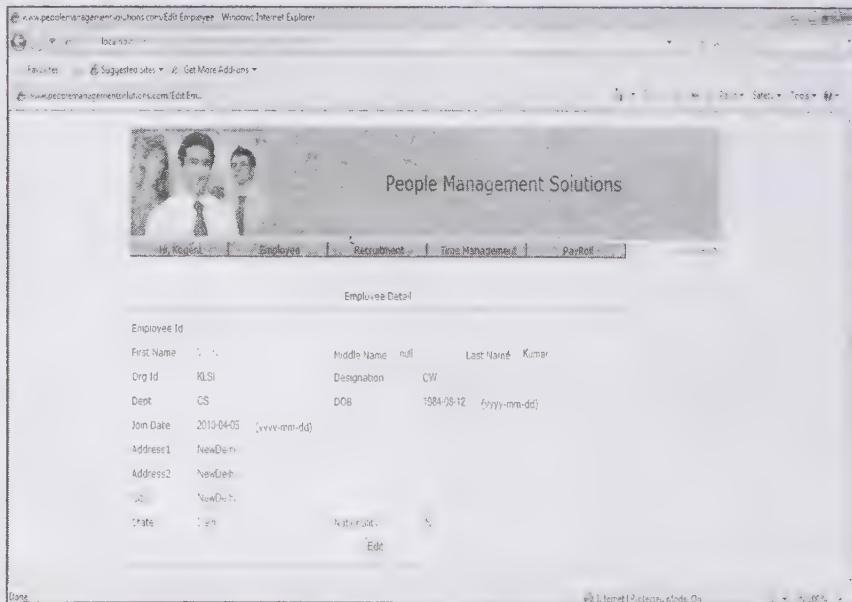


Figure 23.7: Displaying the employee_edit JSP Page for Editing an Employee Record

After filling the updated values in the employee_edit JSP page (Figure 23.7), click the Edit button, which submits the values of the hidden attributes, such as emp_id and emp_f_name, to a servlet. Then, the servlet determines the execution path on the basis of the hidden attribute; and accordingly sets the value for the action attribute.

In this section, you have learned how to develop the Profile Management module that handles profile of all the employees. The module helps you to create the profile of a new employee and edit the profile of an existing one, to incorporate the changes in the database. The entire module is composed of one servlet (people_employee), its two helper classes (EmployeeDBObj and EmployeeDBMethods), and three JSP pages, employee_insert, employee_search, and employee_edit.

Creating the employee_list JSP Page

The employee_list JSP page is designed to display the details of all the employees of the organization. This JSP page displays various fields, such as Employee Id, First Name, Last Name, Designation, and DOB. The page also provides the Edit, Delete, and Detail links for each employee record that is retrieved with the help of the people_employee servlet class.

Listing 23.16 shows the code of the employee_list.jsp file (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.16: Showing the Code of the employee_list.jsp File

```
<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.Employee.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.util.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Employee List</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2"><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td width="900"><%@ include file="../jsp/people_default_menu.jsp" %></td>
</tr><tr>
<td width = "750" valign="top">
<div align=center class=boldblack>List of Employees</div>
<hr bgcolor="#AAAAAA">
<table border="0" width=100% >
<%
    EmployeeDBObj employeeDBObj;
    ArrayList employeeList; // = new ArrayList();
%>
<%
    String dbopr = "";
    dbopr = (String)session.getAttribute("dbopr");
%>
<tr class="whitetext" height=20>
<td bgcolor ='#AAAAAA'>Emp Id</td>
<td bgcolor ='#AAAAAA'>F Name</td>
<td bgcolor ='#AAAAAA'>L Name</td>
<td bgcolor ='#AAAAAA'>Designation</td>
<td bgcolor ='#AAAAAA'>DOB</td>
<td bgcolor ='#AAAAAA' align='center' >Edit</td>
<td bgcolor ='#AAAAAA' align='center' >Delete</td>
<td bgcolor ='#AAAAAA' align='center' >Detail</td>
</tr>
<%
    employeeList = new ArrayList();
    employeeList = (ArrayList)session.getAttribute("EmployeeList");
    if ( employeeList != null && employeeList.size() > 0 ){
        for ( int size = 1; size <= employeeList.size() ; size++ ){
            employeeDBObj = new EmployeeDBObj();
            employeeDBObj.setEmpId(employeeList.get(size).getEmpId());
            employeeDBObj.setFName(employeeList.get(size).getFName());
            employeeDBObj.setLName(employeeList.get(size).getLName());
            employeeDBObj.setDesignation(employeeList.get(size).getDesignation());
            employeeDBObj.setDob(employeeList.get(size).getDob());
            employeeDBObj.setEditLink("Edit.jsp?emp_id=" + employeeDBObj.getEmpId());
            employeeDBObj.setDeleteLink("Delete.jsp?emp_id=" + employeeDBObj.getEmpId());
            employeeDBObj.setDetailLink("Detail.jsp?emp_id=" + employeeDBObj.getEmpId());
            employeeList.add(employeeDBObj);
        }
    }
%>
```

```

employeeDBObj = (EmployeeDBObj)employeeList.get(size-1);
%>
<form name="form1" method="post">
<tr bgcolor ='#AAAAAA' height=18>
<td align='left'><%=employeeDBObj.emp_id%></td>
<td align='left'><%=employeeDBObj.emp_f_name%></td>
<td align='left' ><%=employeeDBObj.emp_l_name%></td>
<td align='left' ><%=employeeDBObj.level_id%></td>
<td align='left' ><%=employeeDBObj.dob%></td>
<td align='center' bgcolor="#AAAAAA">
<a href='http://localhost:8080/people-mgmt/servlet/people_employee?dbopr=edit'
class="yellowlink">Edit </a>
</td >
<td align='center' bgcolor="#AAAAAA">
<a href='http://localhost:8080/people-
mgmt/servlet/people_employee?dbopr=detail&&emp_id=<%=employeeDBObj.emp_id%>&&emp_-
_name=<%=employeeDBObj.emp_f_name%>' class="yellowlink">Detail </a>
</td >
<td align='center' bgcolor="#AAAAAA">
<a href='http://localhost:8080/people-
mgmt/servlet/people_employee?dbopr=delete&&emp_id=<%=employeeDBObj.emp_id%>'
class="yellowlink">Delete </a>
</td >
<% } %>
%>
</tr>
</td>
</table>
</tr>
<tr>
<td colspan="2"><%@include file="..../jsp/people_footer.jsp"%></td>
</tr>
</table></body></html>

```

The Edit link is used to forward the request of a user to search a page where the user has to fill the details of an employee. The user can delete an employee record from the database by using the Delete link, and can view the details of the employee by clicking the Detail link in the employee_list JSP page.

Figure 23.8 shows the employee_list JSP page:

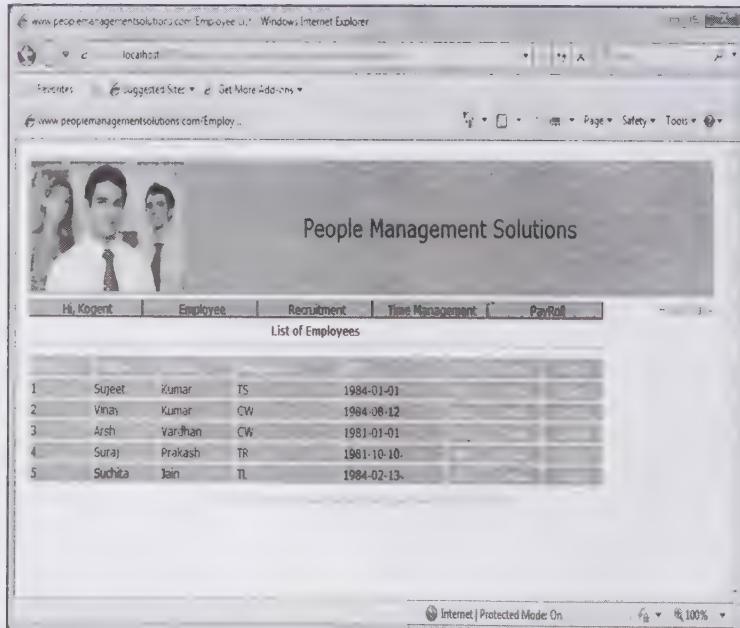


Figure 23.8: Displaying the List of Employees in the employee_list JSP Page

The records of all the employees added in this application are shown in Figure 23.8.

The next subsection describes the `employee_profile` JSP page.

Creating the employee_profile JSP Page

The `employee_profile` JSP page is designed to display the complete information of a particular employee. This JSP page contains all the fields that are used in the `employee_insert` JSP page. However, in the `employee_profile` JSP page all the fields are disabled. The employee records are retrieved on the basis of the values of the Employee ID and First Name fields.

Listing 23.17 shows the code of the `employee_profile.jsp` file (you can find this file in the `PeopleMgmt\people-mgmt\jsp` folder on CD):

Listing 23.17: Showing the Code of the `employee_profile` JSP Page

```

<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.Employee.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Employee Detail</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2"><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td width="900">
    <%@ include file="../jsp/people_default_menu.jsp" %></td>
</tr><tr>
<td width = "750" valign="top" align = "center">
<p>&nbsp;</p>
Employee Detail
<hr width=100% color="#AAAAAA">
<table border="0" align="top" width=100% >
<%
    EmployeedBObj employeedBObj = new EmployeedBObj();
    employeedBObj = (EmployeedBObj)session.getAttribute("employeedBObj");
%>
<input type='hidden' name='emp_id' id='emp_id' size ='10'
value="<%=employeedBObj.emp_id%>" />
<input type='hidden' name='emp_f_name' id='emp_f_name' size ='10'
value="<%=employeedBObj.emp_f_name%>" />
<input type='hidden' name='emp_m_name' id='emp_m_name' size ='10'
value="<%=employeedBObj.emp_m_name%>" />
<input type='hidden' name='emp_l_name' id='emp_l_name' size ='10'
value="<%=employeedBObj.emp_l_name%>" />
<input type='hidden' name='org_id' id='org_id' size ='10'
value="<%=employeedBObj.org_id%>" />
<input type='hidden' name='level_id' id='level_id' size ='10'
value="<%=employeedBObj.level_id%>" />
<input type='hidden' name='dept_id' id='dept_id' size ='10'
value="<%=employeedBObj.dept_id%>" />
<input type='hidden' name='dob' id='dob' size ='10'
value="<%=employeedBObj.dob%>" />
<input type='hidden' name='dojoin' id='dojoin' size ='10'
value="<%=employeedBObj.dojoin%>" />
<input type='hidden' name='address_1' id='address_1' size ='40'
value="<%=employeedBObj.address_1%>" />
<input type='hidden' name='address_2' id='address_2' size ='40'
value="<%=employeedBObj.address_2%>" />
<input type='hidden' name='city' id='city' size ='10'
value="<%=employeedBObj.city%>" />
<input type='hidden' name='state' id='state' size ='10'
value="<%=employeedBObj.state%>" />
<input type='hidden' name='nationality' id='nationality' size ='10'
value="<%=employeedBObj.nationality%>" />
<tr>

```

```

<td>Employee Id</td>
<td align='left'>
<input type='text' disabled='disabled' name='emp_id_dup' id='emp_id_dup' size
='10' value= "<%=employeeDBObj.emp_id%>"/>
</td>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td>First Name</td>
<td align='left'>
<input type='text' disabled='disabled' name='emp_f_name_dup' id='emp_f_name_dup'
size ='10' value="<%=employeeDBObj.emp_f_name%>"/>
</td>
<td colspan=2>
Middle Name&nbsp;&nbsp;&nbsp;
<input type='text' disabled='disabled' name='dup_emp_m_name' id='emp_m_name' size
='10' value="<%=employeeDBObj.emp_m_name%>"/>
&nbsp;&nbsp;&nbsp;Last Name&nbsp;&nbsp;&nbsp;
<input type='text' disabled='disabled' name='dup_emp_l_name' id='dup_emp_l_name'
size ='10' value="<%=employeeDBObj.emp_l_name%>"/>
</td>
</tr>
<tr>
<td>Org Id</td>
<td align='left'>
<input type='text' disabled='disabled' name='dup_org_id' id='dup_org_id' size
='10' value="<%=employeeDBObj.org_id%>"/>
</td>
<td>Designation</td>
<td align='left'>
<input type='text' disabled='disabled' name='dup_level_id' id='dup_level_id' size
='10' value="<%=employeeDBObj.level_id%>"/>
</td>
</tr>
<tr>
<td>Dept</td>
<td align='left'>
<input type='text' disabled='disabled' name='dup_dept_id' id='dup_dept_id' size
='10' value="<%=employeeDBObj.dept_id%>"/>
</td>
<td>DOB</td>
<td align='left'><input type='text' disabled='disabled' name='dup_dob'
id='dup_dob' size ='10' value="<%=employeeDBObj.dob%>"/>
(yyyy-mm-dd)</td>
</tr>
<tr>
<td>Join Date</td>
<td align='left'>
<input type='text' disabled='disabled' name='dup_dojoin' id='dup_dojoin' size
='10' value="<%=employeeDBObj.dojoin%>"/>
(yyyy-mm-dd)</td>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td>Address1</td>
<td align='left' colspan='2'>
<input type='text' name='dup_address_1' id='dup_address_1' disabled='disabled'
size ='40' value="<%=employeeDBObj.address_1%>"/>
</td>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td>Address2</td>
<td align='left' colspan='2'>
<input type='text' disabled='disabled' name='dup_address_2' id='dup_address_2'
size ='40' value="<%=employeeDBObj.address_2%>"/>
</td>
<td>&nbsp;</td>
<td>&nbsp;</td>

```

```

</tr>
<tr>
<td>City</td>
<td align='left'>
<input type='text' disabled='disabled' name='dup_city' id='dup_city' size ='10'
value="<%={employeeDBObj.city}%" />
</td>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td>State</td>
<td align='left'>
<input type='text' disabled='disabled' name='dup_state' id='dup_state' size ='10'
value="<%={employeeDBObj.state}%" />
</td>
<td>Nationality</td>
<td align='left'>
<input type='text' disabled='disabled' name='dup_nationality' id='dup_nationality'
size ='10' value="<%={employeeDBObj.nationality}%" />
</td>
</tr>
</table>
<br width=100% color=#AAAAAA>
</td>
</tr>
<tr>
<td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr>
</table>
</body>
</html>

```

Listing 23.17 specifies the code of the employee_profile JSP page that would be displayed after clicking the Detail link of the employee_list JSP page. In employee_profile.jsp file all the fields are disabled to prevent the profile from any modification. Figure 23.9 shows the employee_profile JSP page:



Figure 23.9: Displaying Complete Profile of a Selected Employee in the employee_profile JSP Page

Figure 23.9 shows the complete profile of an employee.

In the next section, you learn how to develop the Recruitment module, which deals with the recruitment process of a new candidate in an organization. The work of this module starts with registering a new candidate and updating the number of test details for different rounds.



Section C

Developing the Recruitment Module

If you need an information on:

Registering a New Applicant

See page:

1128

Conducting Rounds of Test

1149

Recruitment of a new employee is a process in which an organization tests the skills of an applicant on the basis of various tests and finally selects the appropriate applicant for the vacancy. The process starts with the registration of the applicants and calling the short listed candidates for written test. Next, the results of all the candidates are updated on the basis of the marks obtained by them. The candidates who clear the written test are called for the next round, in our case, technical round. This process continues till the final selection of the candidate happens. Let's now learn to develop the Recruitment module that provides various interfaces to handle the recruitment process of the new applicants.

Registering a New Applicant

The recruitment process starts with the registration of the applicants whose profiles have been searched or who have sent their resumes for consideration for a particular job opening in an organization. Any new candidate is registered by entering the necessary details, for which you need to design the applicant_register interface and update the respective data in the PEOPLE_APPLICANT table. Further, you can see the list of registered candidates in the applicant_list JSP page and can also modify or delete a particular candidate's information.

Let's now create the following files for the recruitment module:

- people_applicant.java
- ApplicantDBObj.java
- ApplicantDBMethods.java
- GenerateId.java

In addition to these Java classes, you also need to create the following JSP pages:

- applicant_register.jsp
- applicant_edit.jsp
- applicant_list.jsp

The next subsection discusses about the people_applicant servlet class.

Creating the people_applicant Servlet

The people_applicant servlet handles the process of registering an applicant with the help of the ApplicantDBObj object and the methods of the ApplicantDBMethods class. This servlet forwards the request of a user to the JSP page according to the action that has been set with the request. The object of the ApplicantDBObj class is used as a Data Transfer Object (DTO) for this module, and its values are initialized by using the methods of the ApplicantDBMethods class.

Listing 23.18 shows the code of the people_applicant servlet (you can find the people_applicant.java file in the PeopleMgmt\people-mgmt\WEB-INF\src folder on CD):

Listing 23.18: Showing the Code of the people_applicant.java File

```

import javax.servlet.*;
import javax.servlet.http.*;
import java.util.ArrayList;
import java.io.*;
import java.util.*;
import java.sql.*;
import com.Applicant.*;
import javax.servlet.annotation.*;
import javax.servlet.annotation.WebServlet;

@WebServlet(name="people_applicant", urlPatterns="/servlet/people_applicant")
public class people_applicant extends HttpServlet{
    String lDBUser = "";
    String lDBPswd = "";
    String lDBUrl = "";

    /**Initialize global variables*/
    @Override

```

```

public void init(ServletConfig config) throws ServletException{
    System.out.println("initializing controller servlet.");
    ServletContext context = config.getServletContext();
    lDBUser = "scott";
    lDBPswd = "tiger";
    lDBUrl = "jdbc:oracle:thin:@192.168.1.123:1521:+"XE";
    super.init(config);
}

/**Process the HTTP Get request*/
@Override
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{
    doPost(request, response);
}

/**Process the HTTP Post request*/
@Override
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    HttpSession session = request.getSession();
    session.setAttribute("lErrorMsg",null);
    String target = "";
    String action = request.getParameter("action");
    String lDBOpr = "";
    lDBOpr = (String)request.getParameter("dbopr");
    session.setAttribute("dbopr",lDBOpr);
    if( (lDBOpr != null & lDBOpr.length() > 0) &&
        (lDBOpr.equals("register")) ){
        target = "/jsp/applicant_register.jsp";
    }
    else
    if( (lDBOpr != null & lDBOpr.length() > 0) &&
        (lDBOpr.equals("edit")) ){
        action = "people_applicant_edit";
    }
    else
    if( (lDBOpr != null & lDBOpr.length() > 0) &&
        (lDBOpr.equals("delete")) ){
        action = "people_applicant_delete";
    }
    else
    if( (lDBOpr != null & lDBOpr.length() > 0) &&
        (lDBOpr.equals("detail")) ){
        action = "people_applicant_detail";
    }
    else{
        action = "people_applicant_select";
    }
    String action_submit = request.getParameter("action_submit");
    String action_edit = request.getParameter("action_edit");
    System.out.println("action_submit=="+action_submit);
    if( (action_submit != null || action_edit != null )&&
        (request.getParameter("submit").equals("Submit")) ){
        System.out.println("in the Submit");
        if( (action_submit.equals("people_applicant_register_submit")) ){
            System.out.println("in the people_applicant_register_submit ");
            action = "people_applicant_register_submit";
        }
        else
        if( (action_submit.equals("people_applicant_search_submit"))){
            action = "people_applicant_search_submit";
        }
    }
}
else
}

```

```

if ( request.getParameter("submit").equals("Update") ){
    if ( action_edit.equals("people_applicant_edit_submit") )
        action = "people_applicant_edit_submit";
    }
}
if (action!=null){
    System.out.println("in the "+action);
    if (action.equals("people_change_pswd_submit")){
        target = "/jsp/people_user_login_pswd_change.jsp";
    } else {
        if (action.equals("people_applicant_register_submit")){
            ApplicantDBObj popApplicantDBObj = new ApplicantDBObj();
            ApplicantDBMethods applicantDBMethods = new ApplicantDBMethods(
                lDBUser, lDBPswd, lDBUrl);
            popApplicantDBObj = (ApplicantDBObj) applicantDBMethods.
                populateApplicantDBObjFromReq(request);
            ApplicantDBObj applicantDBObj = new ApplicantDBObj();
            applicantDBObj = (ApplicantDBObj) applicantDBMethods.
                getRecordByPrimaryKey(popApplicantDBObj.applicant_id);
            if ( (popApplicantDBObj.applicant_id) != null &&
                (applicantDBObj.applicant_id) != null) &&
                (popApplicantDBObj.applicant_id).equals(applicantDBObj.applicant_id) ){
                String lErrorMsg = "Applicant Already Exist!!";
                session.setAttribute("lErrorMsg",lErrorMsg);
                target = "/jsp/applicant_register.jsp";
            }
            else{
                int rval = applicantDBMethods.insertApplicant(popApplicantDBObj);
                ArrayList ApplicantList = new ArrayList();
                String criteria = "";
                ApplicantList = (ArrayList) applicantDBMethods.
                    selectApplicantByCriteria (criteria);
                session.setAttribute("ApplicantList",ApplicantList);
                target = "/jsp/applicant_list.jsp";
            }
        }
    }
    else
        if (action.equals("people_applicant_select")){
            ApplicantDBMethods applicantDBMethods = new ApplicantDBMethods(lDBUser,
                lDBPswd,lDBUrl);
            ArrayList ApplicantList = new ArrayList();
            String criteria = "";
            ApplicantList =
                (ArrayList) applicantDBMethods.selectApplicantByCriteria(criteria);
            session.setAttribute("ApplicantList",ApplicantList);
            target = "/jsp/applicant_list.jsp";
        }
    else
        if (action.equals("people_applicant_edit")){
            String lApplicantId= "";
            lApplicantId = (String)request.getParameter("applicant_id");
            ApplicantDBMethods applicantDBMethods = new ApplicantDBMethods(lDBUser,
                lDBPswd,lDBUrl);
            ApplicantDBObj applicantDBObj = new ApplicantDBObj();
            applicantDBObj =
                (ApplicantDBObj) applicantDBMethods.getRecordByPrimaryKey(lApplicantId);
            if ( applicantDBObj != null && ( applicantDBObj.applicant_id != null &&
                applicantDBObj.applicant_id.length() > 0 ) ){
                session.setAttribute("applicantDBObj",applicantDBObj);
                target = "/jsp/applicant_edit.jsp";
            }
        else{
            target = "/jsp/applicant_list.jsp";
        }
    }
}

```

```

}
else
if (action.equals("people_applicant_edit_submit")){
    ApplicantDBObj popApplicantDBObj = new ApplicantDBObj();
    ApplicantDBMethods applicantDBMethods = new
        ApplicantDBMethods(1DBUser,1DBPswd,1DBUrl);
    popApplicantDBObj = (ApplicantDBObj)applicantDBMethods.
        populateApplicantDBObjFromReq(request);
    ApplicantDBObj applicantDBObj = new ApplicantDBObj();
    int rval = applicantDBMethods.updateApplicant(popApplicantDBObj);
    applicantDBObj = (ApplicantDBObj)applicantDBMethods.getRecordByPrimaryKey(
        popApplicantDBObj.applicant_id);
    session.setAttribute("applicantDBObj",applicantDBObj);
    String criteria = "";
    ArrayList ApplicantList = new ArrayList();
    ApplicantList = (ArrayList)applicantDBMethods.
        selectApplicantByCriteria(criteria);
    session.setAttribute("Applicantlist",ApplicantList);
    target = "/jsp/applicant_list.jsp";
}
else
if (action.equals("people_applicant_delete")){
    String lApplicationId = "";
    lApplicationId = (String)request.getParameter("applicant_id");
    ApplicantDBMethods applicantDBMethods = new ApplicantDBMethods(1DBUser,
        1DBPswd,1DBUrl);
    applicantDBMethods.deleteApplicant(lApplicationId);
    ArrayList ApplicantList = new ArrayList();
    String criteria = "";
    ApplicantList = (ArrayList)applicantDBMethods. selectApplicantByCriteria(
        criteria);
    session.setAttribute("ApplicantList",ApplicantList);
    target = "/jsp/applicant_list.jsp";
}
else
if (action.equals("people_applicant_detail")){
    String lApplicationId = "";
    lApplicationId = (String)request.getParameter("applicant_id");
    ApplicantDBMethods applicantDBMethods = new ApplicantDBMethods(1DBUser,
        1DBPswd,1DBUrl);
    ApplicantDBObj applicantDBObj = new ApplicantDBObj();
    applicantDBObj = (ApplicantDBObj)applicantDBMethods.
        getRecordByPrimaryKey(lApplicationId);
    session.setAttribute("applicantDBObj",applicantDBObj);
    target = "/jsp/applicant_edit.jsp";
}
/* forwarding the request/response to the targeted view */
RequestDispatcher requestDispatcher = getServletContext().
    getRequestDispatcher(target);
requestDispatcher.forward(request, response);
} // doPost closed
} // class closed

```

In Listing 23.18, the `@WebServlet` annotation is used for servlet mapping, which eliminates the need to define servlet mapping in the `web.xml` file. The `@Override` annotation used in Listing 23.18 specifies that a method is overridden in the current class.

The next subsection describes the `ApplicantDBObj` class, which is used as a DTO.

Creating the ApplicantDBObj Class

The `ApplicantDBObj` class contains member variables matching with the columns of the `PEOPLE_APPLICANT` table. The methods of the `ApplicantDBMethods` class use the `ApplicantDBObj` object to manipulate database with the data set being represented by the `ApplicantDBObj` object.

Listing 23.19 provides the code of the ApplicantDBObj.java file (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src\com\Applicant folder on CD):

Listing 23.19: Showing the Code of the ApplicantDBObj.java File

```
package com.Applicant;
public class ApplicantDBObj
{
    public String applicant_id;
    public String applicant_name;
    public String address_1;
    public String address_2;
    public String current_location;
    public String email;
    public long phone;
    public long mobile;
    public String dob;
    public String gender;
    public String nationality;
    public long work_exp;
    public String skill;
    public String industry;
    public String category;
    public String roles;
    public String current_employer;
    public double current_sal;
    public String highest_degree;
    public String second_highest_degree;
    public String domain;
}
```

Listing 23.19 provides code of the ApplicantDBObj class that contains all the variables with respect to the PEOPLE_APPLICANT table.

The next subsection discusses about the ApplicantDBMethods class.

Creating the ApplicantDBMethods Class

The ApplicantDBMethods class provides methods that are used to make the required changes in the details of an applicant, which is stored in the database. This implies that the methods and variables defined in the ApplicantDBMethods class are used to interact with the database. In the PeopleMgmt project, these methods separate the code responsible for the real interaction with the database. This class contains various methods that are used within other methods to perform various processes, such as:

- ❑ The insertApplicant() method: Inserts the details of a new applicant
- ❑ The updateApplicant() method: Updates the details of an existing applicant
- ❑ The deleteApplicant() method: Deletes the record of an applicant from the database
- ❑ The populateApplicantDBObjFromReq() method: Populates the ApplicantDBObj object with the values passed along with a request

Listing 23.20 provides the code of the ApplicantDBMethods.java file (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src\com\Applicant folder on CD):

Listing 23.20: Showing the Code of the ApplicantDBMethods.java File

```
package com.Applicant;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.util.ArrayList;
import com.Applicant.ApplicantDBObj;
public class ApplicantDBMethods
{
    public String DBUser;
    public String DBPswd;
```

```

public String DBurl ;
public ApplicantDBMethods() { }
public ApplicantDBMethods(String inDBUser, String inDBPswd, String inDBurl )
{
    DBUser = inDBUser ;
    DBPswd = inDBPswd;
    DBurl = inDBurl;
}
public void initializeApplicantDBObj(ApplicantDBObj inApplicantDBObj )
{
    inApplicantDBObj.applicant_id = "";
    inApplicantDBObj.applicant_name = "";
    inApplicantDBObj.address_1= "";
    inApplicantDBObj.address_2= "";
    inApplicantDBObj.current_location= "";
    inApplicantDBObj.email = "";
    inApplicantDBObj.phone = 0;
    inApplicantDBObj.mobile = 0;
    inApplicantDBObj.dob= "";
    inApplicantDBObj.gender = "";
    inApplicantDBObj.nationality= "";
    inApplicantDBObj.work_exp= 0;
    inApplicantDBObj.skill= "";
    inApplicantDBObj.industry= "";
    inApplicantDBObj.category= "";
    inApplicantDBObj.roles= "";
    inApplicantDBObj.current_employer= "";
    inApplicantDBObj.current_sal= 0;
    inApplicantDBObj.highest_degree= "";
    inApplicantDBObj.second_highest_degree= "";
    inApplicantDBObj.domain= "";
}
public ApplicantDBObj getRecordByPrimaryKey(String inApplicantID)
{
    . . . .
    . . . .
}
public ArrayList selectApplicantByCriteria(String inCriteria)
{
    . . . .
    . . . .
}
public int updateApplicant(ApplicantDBObj inApplicantDBObj)
{
    . . . .
    . . . .
}
public ApplicantDBObj populateApplicantDBObjFromReq(HttpServletRequest inReq)
{
    . . . .
    . . . .
}
public int insertApplicant(ApplicantDBObj inApplicantDBObj)
{
    . . . .
    . . . .
}
public void deleteApplicant(String inApplicantId)
{
    . . . .
    . . . .
}
}

```

Listing 23.20 defines the prototypes of each method depicting what a method accepts as an argument and what is being returned by the method.

Let's now discuss the code of each method.

The `getRecordByPrimaryKey()` method accepts a String as an argument, i.e. an applicant id of the applicant whose record from the `PEOPLE_APPLICANT` table is to be retrieved. This method executes a select query statement and the retrieved `ResultSet` object is saved to the `ApplicantDBObj` object. This object is then returned to the resultant JSP page.

Listing 23.21 provides the code of the `getRecordByPrimaryKey()` method:

Listing 23.21: Showing the Code of the `getRecordByPrimaryKey()` Method of the `ApplicantDBMethods.java` File

```
public ApplicantDBObj getRecordByPrimaryKey(String inApplicantId){
    ApplicantDBObj applicantDBObj = new ApplicantDBObj();
    java.sql.Date date;
    try{
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn= DriverManager.getConnection(DBURL,DBUSER,DBPswd);
        Statement stmt = conn.createStatement();
        String lsqlString = "select * from PEOPLE_APPLICANT ";
        lsqlString = lsqlString + "where applicant_id='"+inApplicantId+"' ";
        ResultSet rs = null;
        rs = stmt.executeQuery(lsqlString);
        System.out.println("lsqlString====trtrt==within getRecordByPrimaryKey=="
                           "+lsqlString);
        if( rs.next()){
            System.out.println("fffff=="+rs.getString("applicant_id"));
            applicantDBObj.applicant_id = rs.getString("applicant_id");
            applicantDBObj.applicant_name = rs.getString("applicant_name");
            applicantDBObj.address_1 = rs.getString("address_1");
            applicantDBObj.address_2 = rs.getString("address_2");
            applicantDBObj.current_location = rs.getString("current_location");
            applicantDBObj.email = rs.getString("email");
            applicantDBObj.phone = rs.getLong("phone");
            applicantDBObj.mobile = rs.getLong("mobile");
            date=rs.getDate("dob");
            applicantDBObj.dob = date.toString();
            applicantDBObj.gender = rs.getString("gender");
            applicantDBObj.nationality = rs.getString("nationality");
            applicantDBObj.work_exp = rs.getLong("work_exp");
            applicantDBObj.skill = rs.getString("skill");
            applicantDBObj.industry = rs.getString("industry");
            applicantDBObj.category = rs.getString("category");
            applicantDBObj.roles = rs.getString("roles");
            applicantDBObj.current_employer = rs.getString("current_employer");
            applicantDBObj.current_sal = rs.getDouble("current_sal");
            applicantDBObj.highest_degree = rs.getString("highest_degree");
            applicantDBObj.second_highest_degree =
                rs.getString("second_highest_degree");
            applicantDBObj.domain = rs.getString("domain");
            System.out.println("fffff=="+rs.getString("applicant_id"));
        }
        else{
            initializeApplicantDBObj(applicantDBObj);
        }
        System.out.println("fffff===="+applicantDBObj.applicant_id);
    }
    catch(SQLException ex){
        ex.printStackTrace();
    }
    return applicantDBObj;
}
```

The `getRecordByPrimaryKey()` method defined in Listing 23.21 returns the `ApplicantDBObj` object, as it searches the applicant on the basis of a primary key. Similarly, the `selectApplicantByCriteria()` method takes the `inCriteria` String variable containing the select query with the where clause used to obtain the desired result. The result may contain a list of the `ApplicantDBObj` objects; thereby, returning an object of the `ArrayList` class.

Listing 23.22: Providing the code of the `selectApplicantByCriteria()` method:

Listing 23.22: Showing the Code of the `selectApplicantByCriteria()` Method of the `ApplicantDBMethods.java` File

```
public ArrayList selectApplicantByCriteria(String inCriteria){
    ArrayList ApplicantList = new ArrayList();
    java.sql.Date date;
    try{
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
        Statement stmt = conn.createStatement();
        String lsqlString = "select * from PEOPLE_APPLICANT ";
        if( inCriteria != null && inCriteria.length() > 0 ){
            lsqlString = lsqlString +" "+inCriteria+" ";
        }
        lsqlString = lsqlString +" order by applicant_id" ;
        System.out.println("Criteria==== "+inCriteria+" and query="+lsqlString);
        ResultSet rs = null;
        rs = stmt.executeQuery(lsqlString);
        while( rs.next()){
            ApplicantDBObj applicantDBObj = new ApplicantDBObj();
            applicantDBObj.applicant_id = rs.getString("applicant_id");
            applicantDBObj.applicant_name = rs.getString("applicant_name");
            applicantDBObj.address_1 = rs.getString("address_1");
            applicantDBObj.address_2 = rs.getString("address_2");
            applicantDBObj.current_location = rs.getString("current_location");
            applicantDBObj.email = rs.getString("email");
            applicantDBObj.phone = rs.getLong("phone");
            applicantDBObj.mobile = rs.getLong("mobile");
            date=rs.getDate("dob");
            applicantDBObj.dob = date.toString();
            applicantDBObj.gender = rs.getString("gender");
            applicantDBObj.nationality = rs.getString("nationality");
            applicantDBObj.work_exp = rs.getLong("work_exp");
            applicantDBObj.skill = rs.getString("skill");
            applicantDBObj.industry = rs.getString("industry");
            applicantDBObj.category = rs.getString("category");
            applicantDBObj.roles = rs.getString("roles");
            applicantDBObj.current_employer = rs.getString("current_employer");
            applicantDBObj.current_sal = rs.getDouble("current_sal");
            applicantDBObj.highest_degree = rs.getString("highest_degree");
            applicantDBObj.second_highest_degree =
                rs.getString("second_highest_degree");
            applicantDBObj.domain = rs.getString("domain");
            ApplicantList.add(applicantDBObj);
        }
    }
    catch(SQLException ex){
        ex.printStackTrace();
    }
    return ApplicantList;
}
```

The code of Listing 23.22 is used to retrieve an applicant record on some criteria, such as `applicant_id` and the `test` status. The other important methods are `updateApplicant()`, `insertApplicant()`, and `deleteApplicant()`. These methods are used for updating, inserting, and deleting an applicant from the database.

Listing 23.23: Showing the Code of the `updateApplicant()`, `insertApplicant()`, `deleteApplicant()` Methods of the `ApplicantDBMethods.java` File

```

public int updateApplicant(ApplicantDBObj inApplicantDBObj){
    int recupd = 0;
    String lQuery = "";
    lQuery = lQuery + "update PEOPLE_APPLICANT set
        applicant_name='"+inApplicantDBObj.applicant_name+"' ";
    lQuery = lQuery + ", address_1='"+inApplicantDBObj.address_1+"' ";
    lQuery = lQuery + ", address_2='"+inApplicantDBObj.address_2+"' ";
    lQuery = lQuery + ", current_location='"+inApplicantDBObj.current_location+"' ";
    lQuery = lQuery + ", phone='"+inApplicantDBObj.phone+"' ";
    lQuery = lQuery + ", mobile='"+inApplicantDBObj.mobile+"' ";
    lQuery = lQuery + ", dob=to_date '"+inApplicantDBObj.dob+"','yyyy-mm-dd') ";
    lQuery = lQuery + ", gender='"+inApplicantDBObj.gender+"' ";
    lQuery = lQuery + ", nationality='"+inApplicantDBObj.nationality+"' ";
    lQuery = lQuery + ", work_exp='"+inApplicantDBObj.work_exp+"' ";
    lQuery = lQuery + ", skill='"+inApplicantDBObj.skill+"' ";
    lQuery = lQuery + ", industry='"+inApplicantDBObj.industry+"' ";
    lQuery = lQuery + ", category='"+inApplicantDBObj.category+"' ";
    lQuery = lQuery + ", roles='"+inApplicantDBObj.roles+"' ";
    lQuery = lQuery + ", current_employer='"+inApplicantDBObj.current_employer+"' ";
    lQuery = lQuery + ", current_sal='"+inApplicantDBObj.current_sal+"' ";
    lQuery = lQuery + ", highest_degree='"+inApplicantDBObj.highest_degree+"' ";
    lQuery = lQuery + ", second_highest_degree='"+inApplicantDBObj.second_highest_degree+"' ";
    lQuery = lQuery + ", domain='"+inApplicantDBObj.domain+"' ";
    lQuery = lQuery + " where applicant_id='"+inApplicantDBObj.applicant_id+"' ";
    System.out.println("lsqlString==:"+lQuery);
    try{
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
        Statement stmt = conn.createStatement();
        recupd = stmt.executeUpdate(lQuery);
    }
    catch(SQLException ex){
        ex.printStackTrace();
    }
    return recupd;
}
public int insertApplicant(ApplicantDBObj inApplicantDBObj){
    int recupd = 0;
    String lQuery = "";
    lQuery = lQuery + "insert into PEOPLE_APPLICANT values ( ";
    lQuery = lQuery + "+inApplicantDBObj.applicant_id+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.applicant_name+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.address_1+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.address_2+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.email+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.phone+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.mobile+"' ";
    lQuery = lQuery + ", to_date '"+inApplicantDBObj.dob+"','yyyy-mm-dd') ";
    lQuery = lQuery + ", "+inApplicantDBObj.gender+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.nationality+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.work_exp+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.skill+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.industry+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.category+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.roles+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.current_employer+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.current_sal+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.highest_degree+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.second_highest_degree+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.domain+"' ";
    lQuery = lQuery + ", "+inApplicantDBObj.current_location+"' ";
}

```

```

lQuery = lQuery + ")";
System.out.println("lSqlString==:"+lQuery);
try{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
    Statement stmt = conn.createStatement();
    recuperd = stmt.executeUpdate(lQuery);
}
catch(SQLException ex){
    ex.printStackTrace();
}
return recuperd;
}
public void deleteApplicant(String inApplicantId){
try{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
    Statement stmt = conn.createStatement();
    String lQuery = "";
    lQuery = lQuery +"delete from PEOPLE_APPLICANT ";
    lQuery = lQuery +" where applicant_id='"+inApplicantId+"' ";
    System.out.println("lSqlString==:"+lQuery);
    stmt.executeQuery(lQuery);
}
catch(SQLException ex){
    ex.printStackTrace();
}
}
}

```

In Listing 23.23, the `insertApplicant()` method inserts the details of an applicant; whereas, the `updateApplicant()` method updates the applicant profile, if edited. The `deleteApplicant()` method deletes an applicant profile. The last method, `populateApplicantDBObjFromReq()`, accepts the `HttpServletRequest` object as an argument and populates an object of the `ApplicantDBObj` class with the values fetched using the `request.getParameter()` method, and finally returns that object.

Listing 23.24 provides the code of the `populateApplicantDBObjFromReq()` method:

Listing 23.24: Showing the Code of the `populateApplicantDBObjFromReq()` Method of the `ApplicantDBMethods.java` File

```

public ApplicantDBObj populateApplicantDBObjFromReq(HttpServletRequest inReq){
    ApplicantDBObj applicantDBObj = new ApplicantDBObj();
    applicantDBObj.applicant_id = (String)inReq.getParameter("applicant_id");
    applicantDBObj.applicant_name =
        (String)inReq.getParameter("applicant_name");
    applicantDBObj.address_1 = (String)inReq.getParameter("address_1");
    applicantDBObj.address_2 = (String)inReq.getParameter("address_2");
    applicantDBObj.current_location =
        (String)inReq.getParameter("current_location");
    applicantDBObj.email = (String)inReq.getParameter("email");
    applicantDBObj.phone = Long.parseLong((String)inReq.getParameter("phone"));
    applicantDBObj.mobile =
        Long.parseLong((String)inReq.getParameter("mobile"));
    applicantDBObj.dob = (String)inReq.getParameter("dob");
    applicantDBObj.gender = (String)inReq.getParameter("gender");
    applicantDBObj.nationality = (String)inReq.getParameter("nationality");
    applicantDBObj.work_exp =
        Long.parseLong((String)inReq.getParameter("work_exp"));
    applicantDBObj.skill = (String)inReq.getParameter("skill");
    applicantDBObj.industry = (String)inReq.getParameter("industry");
    applicantDBObj.category = (String)inReq.getParameter("category");
    applicantDBObj.roles = (String)inReq.getParameter("roles");
    applicantDBObj.current_employer =
        (String)inReq.getParameter("current_employer");
    applicantDBObj.current_sal =
        Double.parseDouble((String)inReq.getParameter("current_sal"));
    applicantDBObj.highest_degree =

```

```

        (String)inReq.getParameter("highest_degree");
applicantDBObj.second_highest_degree =
        (String)inReq.getParameter("second_highest_degree");
applicantDBObj.domain = (String)inReq.getParameter("domain");
return applicantDBObj;
}

```

The code given in Listing 23.24 is used to populate the applicant details from the request page by using the object of the HttpServletRequest interface. Compile the Java files to get class files and place them in the WEB-INF/classes folder.

Creating the GenerateId Class

The GenerateId class is developed for the purpose of auto-generation of the applicant_id.

Listing 23.25 shows the code for the GenerateId.java file (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src\com\Applicant folder on CD):

Listing 23.25: Showing the Code of the GenerateId.java File

```

package com.Applicant;
import java.sql.*;
import javax.sql.*;
public class GenerateID
{
    public int generateApplicantId(){
        int applicant_id=0;
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection conn=
            DriverManager.getConnection("jdbc:oracle:thin:@192.168.1.123:1521:"+
                "XE","scott","tiger");
            Statement stmt = conn.createStatement();
            String query="select max(applicant_id) as applicant_id from
                PEOPLE_APPLICANT ";
            ResultSet rs = null;
            rs = stmt.executeQuery(query);
            if(rs.next()){
                String id = rs.getString("applicant_id");
                applicant_id=Integer.parseInt(id);
            }
            applicant_id = applicant_id + 1;
        }
        catch(SQLException ex){
            ex.printStackTrace();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
        return applicant_id;
    }
}

```

Listing 23.25 contains the code for auto-generation of the applicant_id field.

Let's now learn how to create the user interfaces in the next subsections.

Creating an Interface for Applicant Registration

After creating the required Java source files, let's now create all the required JSP pages of the recruitment module. There are three view pages designed for this module, namely, applicant_register, applicant_edit, and applicant_list. These JSP pages provide forms for entering details of the new applicants and display the details of all applicants in a tabular form.

Let's now discuss all these JSPs with their functionalities and codes.

Creating the applicant_register JSP Page

The recruitment process begins by registering a new applicant. The form required for a new registration is provided by the applicant_register JSP page. This page includes a form with fields, such as Applicant

Id, Name, Address1, Address2, City, Date of Birth (DOB), and Professional & Educational Details.

Listing 23.26 shows the code of the applicant_register JSP page, which creates a form with all the previously stated fields (you can find the applicant_register.jsp file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.26: Showing the Code of the applicant_register.jsp File

```
<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.Applicant.*" %>
<% GenerateId gen=new GenerateId();
   int applicant_id=gen.generateApplicantId();%>
</html>
<head>
<title>www.peoplemanagementsolutions.com/Register New Applicant</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css"/>
<script language="javascript">
    function validateForm() {
        var myName=document.form1.applicant_name.value;
        if (myName == "") {
            alert ("Applicant name cannot be blank");
            document.form1.applicant_name.focus();
            return false;
        }
        myName=document.form1.address_1.value;
        if (myName == "") {
            alert ("Address1 cannot be blank");
            document.form1.address_1.focus();
            return false;
        }
        myName=document.form1.current_location.value;
        if (myName == "") {
            alert ("Current Location cannot be blank");
            document.form1.current_location.focus();
            return false;
        }
        myName=document.form1.email.value;
        if (myName == "") {
            alert ("Email cannot be blank");
            document.form1.email.focus();
            return false;
        }
        myName=document.form1.phone.value;
        if (myName == "") {
            alert ("Phone Field cannot be blank");
            document.form1.phone.focus();
            return false;
        }
        myName=document.form1.mobile.value;
        if (myName == "") {
            alert ("Mobile cannot be blank");
            document.form1.mobile.focus();
            return false;
        }
        myName=document.form1.dob.value;
        if (myName == "") {
            alert ("You Should Provide Date of Birth");
            document.form1.dob.focus();
            return false;
        }
        myName=document.form1.skill.value;
```

```

        if (myName == "") {
            alert ("You Should provide the Skills, you know");
            document.form1.dept_id.focus();
            return false;
        }
        myName=document.form1.highest_degree.value;
        if (myName == "") {
            alert ("Provide Highest Qualification");
            document.form1.dojoin.focus();
            return false;
        }
        myName=document.form1.second_highest_degree.value;
        if (myName == "") {
            alert ("Provide your Second highest Qualification");
            document.form1.second_highest_degree.focus();
            return false;
        }
        myName=document.form1.domain.value;
        if (myName == "") {
            alert ("Provide the domain on which you can work");
            document.form1.domain.focus();
            return false;
        }
        form1.submit();
    }
</script>
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2" >
<%@ include file="../jsp/people_header.jsp" %>
</td>
</tr>
<tr>
<td width="900" valign="top">
    <%@ include file="../jsp/people_default_menu.jsp" %>
</td>
</tr>
<tr>
<td width ="750">
<table border="0" align="top" width=100%
<form name="form1" method="post">
<input type='hidden' name='applicant_id' id='applicant_id' size ='10' value='<%= applicant_id %>' />
<tr>
<td bgcolor ='#AAAAAA' colspan='4' align=center height=20><b>Contact Information</b></td>
</tr>
<tr><td width=150 >Applicant Id</td>
<td align='left' ><input type='text' disabled='disabled' name='dup_applicant_id' id='dup_applicant_id' size ='10' value='<%= applicant_id %>' /></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr>
<td>Name<sup>*</sup></td><td>
<input type='text' name='applicant_name' id='applicant_name' size ='40' value=''/></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr><td>Address1<sup>*</sup></td>
<td><input type='text' name='address_1' id='address_1' size ='40' value=''/></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr><td>Address2</td>

```

```

<td><input type='text' name='address_2' id='address_2' size ='40' value=''/></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr><td>Current Location<sup>*</sup></td>
<td>
<input type='text' name='current_location' id='current_location' size ='15'
value=''/>
</td><td>Email<sup>*</sup></td>
<td><input type='text' name='email' id='email' size ='25' value=''/></td>
</tr>
<tr><td>Phone<sup>*</sup></td>
<td><input type='text' name='phone' id='phone' size ='15' maxLength="8"
value=''/></td>
<td>Mobile<sup>*</sup></td>
<td><input type='text' name='mobile' id='mobile' maxLength="10" size ='15'
value=''/></td>
</tr>
<tr>
<td bgcolor ='#AAAAAA' colspan='4' align=center height=20><b>Personal
Information</b></td>
</tr>
<tr><td colspan='4'><br>
DOB<sup>*</sup></td><input type='text' name='dob' id='dob' size ='10' value=''/>(yyyy-
mm-dd)&nbsp;&nbsp;&nbsp;
Gender&nbsp;&nbsp;&nbsp;<input type='radio' name='gender' id='gender' size ='10'
value='Male'/'> Male &nbsp;&nbsp; <input type='radio' name='gender' id='gender'
size='10' value='Female'> Female
&nbsp;&nbsp;&nbsp;Nationality&nbsp;&nbsp;&nbsp;
<select name='nationality' id='nationality'>
<option selected>Select Nationality
<option value='IN'> Indian
<option value='RS'> Russian
<option value='PK'> Pakistani
<option value='AM'> American
<option value='BR'> British
<option value='SR'> Srilankan
</select><br></td>
<tr>
<td bgcolor ='#AAAAAA' colspan='4' align=center height=20><b>Professionals &
Educational Details</b></td>
</tr>
<tr><td>Work Exp</td>
<td><input type='text' name='work_exp' id='work_exp' size ='10' value=''/></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr><td>Skills<sup>*</sup></td>
<td><input type='text' name='skill' id='skill' size ='30' value=''/></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr><td>Industry</td>
<td><input type='text' name='industry' id='industry' size ='30' value=''/></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr>
<td>Category</td>
<td><input type='text' name='category' id='category' size ='10' value=''/></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr><td>Roles</td>
<td><input type='text' name='roles' id='roles' size ='10' value=''/></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr><td>Current Employer</td>
<td><input type='text' name='current_employer' id='current_employer' size ='30'
value=''/></td>

```

```

<td>Current Salary </td>
<td><input type='text' name='current_sal' id='current_sal' size ='10'
           value=''/></td>
</tr>
<tr><td>Highest Degree<sup>*</sup></td>
<td><input type='text' name='highest_degree' id='highest_degree' size ='10'
           value=''/></td>
<td colspan=2 align=right>Second Highest Degree<sup>*</sup></td>
&nbsp;&nbsp;&nbsp;
<input type='text' name='second_highest_degree' id='second_highest_degree' size
           ='10' value=''/>
</td></tr>
<tr><td bgcolor ='#AAAAAA' colspan='6' align=center height=20><b>Domain
Knowledge<sup>*</sup></b></td>
</tr>
<tr><td>Domain<sup>*</sup></td>
<td><input type='text' name='domain' id='domain' size ='10' value=''/></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr><td> All the ( <sup>*</sup>) marked are mandatory</td></tr><tr>
<td colspan=4 align=center><input type='submit' name='submit' id='submit' size
           ='10' value='Submit' onclick="return validateForm()"/></td>
</tr>
<input type='hidden' name='action_submit' id='action_submit' size ='10'
           value='people_applicant_register_submit'/> </table>
</td>
</tr>
<tr>
<td colspan="2"><%@include file="..../jsp/people_footer.jsp"%></td>
</tr>
</table></body></html>

```

In Listing 23.26, the value of the Applicant Id field is automatically generated and filled by the generateApplicantId() method of the GenerateId class. Due to this, the Applicant Id field is disabled. The client side validation is used for some fields, such as Name, Address1, Email, Phone, DOB, Mobile, and Skills to ensure that the user does not leave those fields blank.

Figure 23.10 shows the output of the applicant_register JSP page, in which the details of a new applicant are filled:

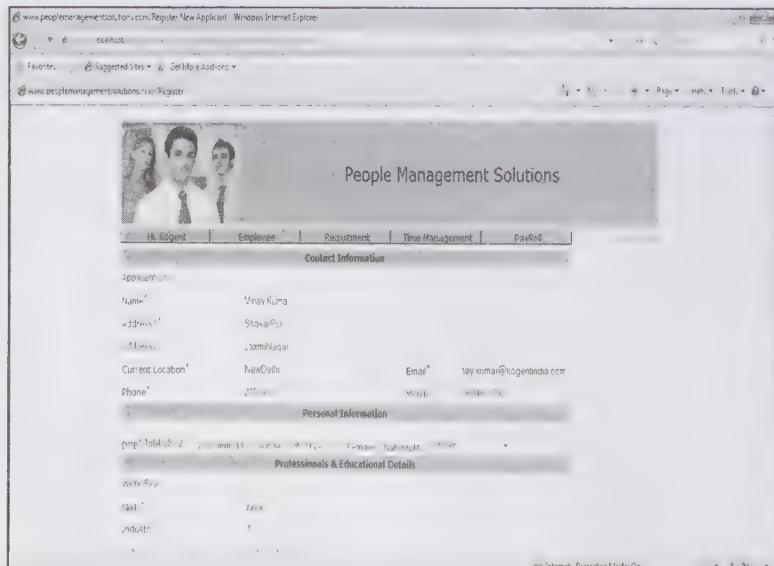


Figure 23.10: Displaying the Candidate Registration Form as the applicant_register JSP Page

Figure 23.10 shows the applicant registration form that needs to be filled up for all the applicants.

Creating the applicant_edit JSP Page

The applicant_edit JSP page allows you to edit the details of an applicant. You can update all the enabled fields with new values and submit the page to update the record for that applicant in the database.

The code of the applicant_edit JSP page has been shown in Listing 23.27 (you can find the applicant_edit.jsp file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.27: Showing the Code of the applicant_edit.jsp File

```

<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.Applicant.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Edit Applicant</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2">
    <%@ include file="../jsp/people_header.jsp" %>
</td>
</tr>
<tr>
<td width="900" valign="top"><%@ include file="../jsp/people_default_menu.jsp" %></td>
</tr>
<tr>
<td width = "750">
<table border="0" align="top" width=100% >
<%
    String dbopr = "";
    dbopr = (String)session.getAttribute("dbopr");
    ApplicantDBObj applicantDBObj = new ApplicantDBObj();
    applicantDBObj = (ApplicantDBObj)session.getAttribute("applicantDBObj");
%>
<form name="form1" method="post">
<tr>
<td bgcolor ='#AAAAAA' colspan='4' align=center height=20><b>Contact
Information</b></td>
</tr>
<tr>
<td width=150 >Applicant Id</td>
<td align='left' >
<input type='text' disabled='disabled' name='applicant_id_dup'
id='applicant_id_dup' size ='10' value='<%=applicantDBObj.applicant_id%>'/>
</td>
<td>
<input type='hidden' name='applicant_id' id='applicant_id' size ='20'
value='<%=applicantDBObj.applicant_id%>'/>
</td>
<td>&ampnbsp</td>
</tr>
<tr><td>Name</td>
<td><input type='text' name='applicant_name' id='applicant_name' size ='40'
value='<%=applicantDBObj.applicant_name%>' /></td>
<td>&ampnbsp</td><td>&ampnbsp</td>
</tr>
<tr><td>Address1</td>
<td><input type='text' name='address_1' id='address_1' size ='40'
value='<%=applicantDBObj.address_1%>' /></td>
<td>&ampnbsp</td><td>&ampnbsp</td>
</tr>
<tr><td>Address2</td>
<td><input type='text' name='address_2' id='address_2' size ='40'
value='<%=applicantDBObj.address_2%>' /></td>
<td>&ampnbsp</td><td>&ampnbsp</td>
</tr>

```

```

<tr><td>Current Location</td>
<td><input type='text' name='current_location' id='current_location' size ='10'
value='<%=applicantDBObj.current_location%>' />
</td>
<td>Email</td>
<td><input type='text' name='email' id='email' size ='25'
value='<%=applicantDBObj.email%>' /></td>
</tr>
<tr><td>Phone</td>
<td><input type='text' name='phone' id='phone' size ='15' maxLength="8"
value='<%=applicantDBObj.phone%>' /></td>
<td>Mobile</td><td><input type='text' name='mobile' id='mobile' size ='15'
maxLength="10" value='<%=applicantDBObj.mobile%>' /></td>
</tr>
<tr>
<td bgcolor ='#AAAAAA' colspan='4' align=center height=20><b>Personal
Information</b></td>
</tr>
<tr><td colspan='4'><br>
DOB&nbsp;&nbsp;&nbsp;<input type='text' name='dob' id='dob' size ='10'
value='<%=applicantDBObj.dob%>' />
(yyyy-mm-dd)&nbsp;&nbsp;&nbsp;Gender&nbsp;&nbsp;&nbsp;<input type='text'
name='gender' id='gender' size ='10' value='<%=applicantDBObj.gender%>' />
&nbsp;&nbsp;&nbsp;Nationality&nbsp;&nbsp;&nbsp;<input type='text'
name='nationality' id='nationality' size ='10'
value='<%=applicantDBObj.nationality%>' /><br>.
</td>
<tr>
<tr>
<td bgcolor ='#AAAAAA' colspan='4' align=center height=20><b>Professionals &
Educational Details</b></td>
</tr>
<tr><td>Work Exp</td>
<td><input type='text' name='work_exp' id='work_exp' size ='10'
value='<%=applicantDBObj.work_exp%>' /></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr><td>Skills</td>
<td><input type='text' name='skill' id='skill' size ='30'
value='<%=applicantDBObj.skill%>' /></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr><td>Industry</td>
<td><input type='text' name='industry' id='industry' size ='30'
value='<%=applicantDBObj.industry%>' /></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr><td>Category</td>
<td><input type='text' name='category' id='category' size ='10'
value='<%=applicantDBObj.category%>' /></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr><td>Roles</td>
<td><input type='text' name='roles' id='roles' size ='10'
value='<%=applicantDBObj.roles%>' /></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<tr><td>Current employer</td>
<td><input type='text' name='current_employer' id='current_employer',size ='30'
value='<%=applicantDBObj.current_employer%>' /></td>
<td>Current Salary </td>
<td><input type='text' name='current_sal' id='current_sal' size ='10'
value='<%=applicantDBObj.current_sal%>' /></td>
</tr>
<tr><td>Highest Degree</td>

```

```

<td><input type='text' name='highest_degree' id='highest_degree' size ='10'
value='<%=applicantDBObj.highest_degree%>'/></td>
<td colspan=2 align=right>Second Highest Degree &nbsp;&nbsp;&nbsp;
<input type='text' name='second_highest_degree' id='second_highest_degree' size
='10' value='<%=applicantDBObj.second_highest_degree%>'/>
</td>
</tr>
<tr>
<td bgcolor ='#AAAAAA' colspan='6' align=center height=20><b>Domain
Knowledge</b></td>
</tr>
<tr><td>Domain</td>
<td><input type='text' name='domain' id='domain' size ='10'
value='<%=applicantDBObj.domain%>'/></td>
<td>&nbsp;</td><td>&nbsp;</td>
</tr>
<% if (dbopr != null && !dbopr.equals("detail")) { %>
<tr>
<td align='center' colspan='6'>
<input type='submit' name='submit' id='submit' size ='10' value='Update' /> </td>
<input type='hidden' name='action_edit' id='action_edit' size ='10' value='people_applicant_edit_submit' /> </td>
</tr>
<% } %>
</table>
</td>
</tr>
<tr>
<td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr>
</table></form></body></html>

```

The applicant_edit JSP page (Listing 23.27) contains the fields similar to those defined in the applicant_register JSP page. The only difference is that some fields, such as Applicant_Id and Name, are disabled in the applicant_edit JSP page. In Figure 23.11, you can see that the Applicant_Id field is disabled and all the other fields can be filled up with the new values to be updated in the database.

Figure 23.11 shows the applicant_register JSP page:

The screenshot shows a web browser window with the URL <http://www.peoplemanagementsolutions.com/EditApplicant>. The page title is "Edit Applicant". The main content area is titled "People Management Solutions". It displays "Contact Information" and "Personal Information" sections. Under "Personal Information", it shows:

Name	Vinay Kumar
Address1	Shakerpur
Address2	LaxmiNagar
Current Location	New Delhi
Phone	21034312
Email	vnav.kumar@kognitivindia.cc
Mobile	9854515792

Below this is a "Professional & Educational Details" section with fields like Work Exp, Skills, Industry, Category, Roles, Current employer, and Current Salary.

Figure 23.11: Displaying the Edit Page of an Applicant

In Figure 23.11, the details of a new applicant are displayed in the applicant_edit JSP page.

The next subsection describes the applicant_list.jsp file.

Creating the applicant_list JSP Page

The result of the applicant_list JSP page displays the details of all the registered applicants that can be deleted or edited, along with the list of applicants that have been shortlisted for the written round.

Listing 23.28 shows the code of the applicant_list.jsp file (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.28: Showing the Code of the applicant_list.jsp File

```
<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.Applicant.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.util.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Applicant List</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2"><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td width="900"><%@ include file="../jsp/people_default_menu.jsp" %></td>
</tr>
<tr>
<td width = "750" valign="top">
<div align=center class=boldblack>List of Applicants</div>
<hr bgcolor="#AAAAAA">
<table border="0" width=100% >
<%
    String dbopr = "";
    dbopr = (String)session.getAttribute("dbopr");
%>
<tr class="whitetext" height=20>
<td bgcolor ='#AAAAAA' align='center' >Applicant Id</td>
<td bgcolor ='#AAAAAA' align='center' >Name</td>
<td bgcolor ='#AAAAAA' align='center' >Work Exp</td>
<td bgcolor ='#AAAAAA' align='center' colspan=2>Skills</td>
<td bgcolor ='#AAAAAA' align='center' >Highest Degree</td>
<%
    if( dbopr != null && (dbopr.equals("call_for_written") || dbopr.equals("call"))
    || dbopr.equals("remove")) ){
%>
<td bgcolor ='#AAAAAA' align='center' >Select</td>
<td bgcolor ='#AAAAAA' align='center' >Detail</td>
<%
    }
    else{ %
<td bgcolor ='#AAAAAA' align='center' >Edit</td>
<td bgcolor ='#AAAAAA' align='center' >Delete</td>
<td bgcolor ='#AAAAAA' align='center' >Detail</td>
<% }%>
</tr>
<%
    ArrayList ApplicantList = new ArrayList();
    ApplicantList = (ArrayList)session.getAttribute("ApplicantList");
    ArrayList applicantTestList = new ArrayList();
    applicantTestList = (ArrayList)session.getAttribute("applicantTestList");
    if ( ApplicantList != null && ApplicantList.size() > 0 ){

```

```

        for ( int size = 1; size <= ApplicantList.size() ; size++ ){
            ApplicantDBObj applicantDBObj = new ApplicantDBObj();
            applicantDBObj = (ApplicantDBObj)ApplicantList.get(size-1);
        }
    %>
    <form name="form1" method="post">
        <tr bgcolor ='#AAAAAA' height=18>
            <td align='center' ><%=applicantDBObj.applicant_id%></td>
            <td align='left' ><%=applicantDBObj.applicant_name%> </td>
            <td align='center' ><%=applicantDBObj.work_exp%></td>
            <td align='left' colspan='2' ><%=applicantDBObj.skill1%></td>
            <td align='center' ><%=applicantDBObj.highest_degree%></td>
        <%>
        if( dbopr != null && ( dbopr.equals("call_for_written") ||
            dbopr.equals("call") || dbopr.equals("remove") ) ){
    %>
        <td align='center' bgcolor="#AAAAAA">
            <a href='http://localhost:8080/people-
                mgmt/servlet/applicant_test_dt1?dbopr=call&&applicant_id=<%=applicantDBObj.a
                pplicant_id%>' class=yellowlink>Select</a></td >
        <td align='center' bgcolor="#AAAAAA">
            <a href='http://localhost:8080/people-
                mgmt/servlet/people_applicant?dbopr=detail&&applicant_id=<%=applicantDBObj.a
                pplicant_id%>' class=yellowlink>Detail </a></td >
        <%>
        }
        else{
    %>
        <td align='center' bgcolor="#AAAAAA">
            <a href='http://localhost:8080/people-mgmt/servlet/people_applicant?
                dbopr=edit&&applicant_id=<%=applicantDBObj.applicant_id%>' class="yellowlink">Edit
            </a>
        </td >
        <td align='center' bgcolor="#AAAAAA">
            <a href='http://localhost:8080/people-mgmt/servlet/people_applicant?
                dbopr=delete&&applicant_id=<%=applicantDBObj.applicant_id%>' 
                class="yellowlink">Delete </a></td >
        <td align='center' bgcolor="#AAAAAA">
            <a href='http://localhost:8080/people-mgmt/servlet/people_applicant?
                dbopr=detail&&applicant_id=<%=applicantDBObj.applicant_id%>' 
                class="yellowlink">Detail </a></td >
        <% } %>
    </tr>
    <% } %>
    if( applicantTestList != null && applicantTestList.size() > 0){
    %>
        <th bgcolor ='#AAAAAA' align='center' colspan='8'><font color=white>The Selected
            Applicant</font></th>
    <%
        for ( int size = 1; size <= applicantTestList.size() ; size++ ){
            ApplicantDBObj applicantDBObj = new ApplicantDBObj();
            applicantDBObj = (ApplicantDBObj)applicantTestList.get(size-1);
        }
    %>
        <tr bgcolor ='#AAAAAA'>
            <td align='center' ><%=applicantDBObj.applicant_id%></td>
            <td align='left' ><%=applicantDBObj.applicant_name%></td>
            <td align='center' ><%=applicantDBObj.work_exp%></td>
            <td align='left' colspan='2' ><%=applicantDBObj.skill1%></td>
            <td align='center' ><%=applicantDBObj.highest_degree%></td>
        <%>
        if( dbopr != null && ( dbopr.equals("call_for_written") ||
            dbopr.equals("call") || dbopr.equals("remove") ) ){
    %>
        <td align='center' colspan='2' bgcolor="#AAAAAA">
            <a href='http://localhost:8080/people-mgmt/servlet/applicant_test_dt1?
                dbopr=remove&&applicant_id=<%=applicantDBObj.applicant_id%>'>

```

```

        class=yellowlink>Remove </a></td >
<%} %>
</tr>
<% }%>
<% }
    if( dbopr != null && ( dbopr.equals("call") || dbopr.equals("remove") || (
        applicantTestList != null && applicantTestList.size() > 0 ) ){
%><tr>
<td align='center' colspan='8'><input type='submit' name='submit' id='submit' size
                ='10' value='Enter Test Detail' /> </td>
<input type='hidden' name='action_select' id='action_select' size = '10'
                value='applicant_call_for_wrtn_test_submit' />
</tr>
<% }
}
else{
    out.print("Applicant does not exist!!!!");
} %>
</table>
</td>
</tr>
<tr>
<td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr>
</table></body></html>

```

The applicant_list JSP page shows the details of all the candidates in the form of a list, along with the Edit, Delete, and Detail links for all the applicants. The Edit and Delete links are used to edit or delete a registered applicant profile, respectively. The third link, Detail, is used to display the complete profile (non-editable) of an applicant.

Figure 23.12 shows the applicant_list JSP page containing the list of candidates:

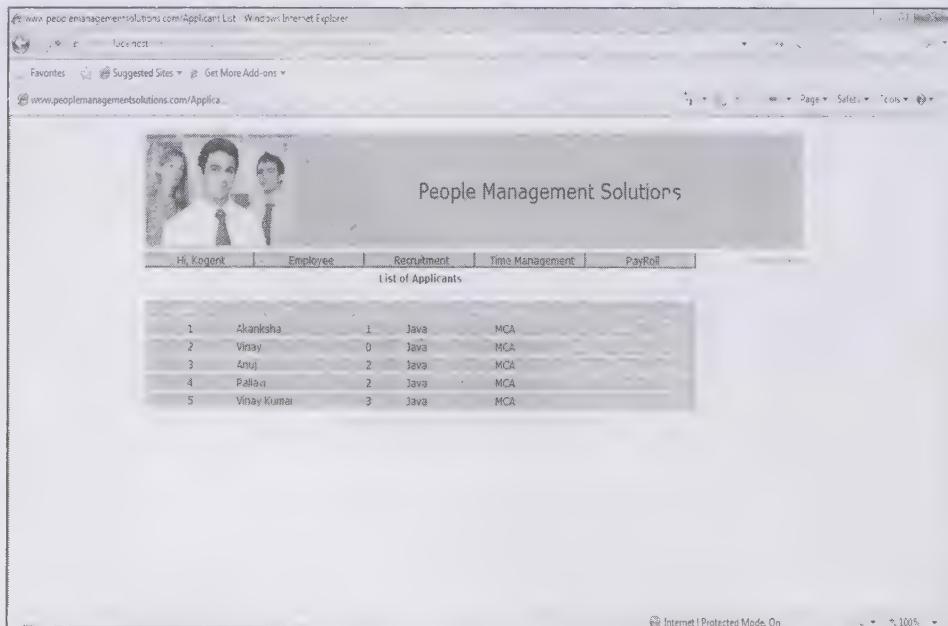


Figure 23.12: Displaying the List of Candidates in the applicant_list JSP Page

After registering all the details of the applicants, let's now provide the code to create the pages used to handle the details of different rounds of test during an interview, update the test result of each round, and select applicants for the next round.

Conducting Rounds of Test

After registering a new applicant, the shortlisted candidates are called for various rounds of test, which include written test, technical round, and finally the HR round. The Recruitment module deals with the process of shortlisting the selected candidates for further rounds; calling candidates for written test, technical round, and HR round by setting the test id, time, and date on which the test is to be conducted; and updating the results of the tests in each round. The process ends up with the issuing of the offer letter to the selected candidates.

The whole recruitment process is handled by a single servlet, applicant_test_dtl, which acts as a main controller servlet of the module. Let's now create the applicant_test_dtl servlet.

Creating the applicant_test_dtl Servlet

The applicant_test_dtl servlet is the most complex servlet, as it handles lots of requests and performs the desired task according to the request.

Note

As the code file for the applicant_test_dtl.java file is very large in size, we are not providing the full code in the section. You can find the complete code in the PeopleMgmt\people-mgmt\WEB-INF\src folder in CD.

When the people_applicant servlet is invoked, a parameter named dbopr is passed with the query string, which is used to select the required path of execution by the servlet.

Listing 23.29 shows the code for the applicant_test_dtl.java file (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src folder on CD):

Listing 23.29: Showing the Code of the applicant_test_dtl.java File

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.ArrayList;
import java.io.*;
import java.util.*;
import java.sql.*;
import com.Applicant.*;
import javax.servlet.annotation.*;
import javax.servlet.annotation.WebServlet;

@WebServlet(name="applicant_test_dtl", urlPatterns="/servlet/applicant_test_dtl")
public class applicant_test_dtl extends HttpServlet{
    String lDBUser = "";
    String lDBPswd = "";
    String lDBUrl = "";
    /**Initialize global variables*/

    @Override
    public void init(ServletConfig config) throws ServletException{
        System.out.println("initializing controller servlet.");
        ServletContext context = config.getServletContext();
        lDBUser = "scott";
        lDBPswd = "tiger";
        lDBUrl = "jdbc:oracle:thin:@192.168.1.123:1521:"+XE";
        super.init(config);
    }
    /**Process the HTTP Get request*/

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doPost(request, response);
    }
    /**Process the HTTP Post request*/

    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response)
```

```

throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    HttpSession session = request.getSession();
    session.setAttribute("lErrorMsg",null);
    String target = "";
    String action = request.getParameter("action");
    String lDBOpr = "";
    lDBOpr = (String)request.getParameter("dbopr");
    session.setAttribute("dbopr",lDBOpr);
    if( (lDBOpr != null && lDBOpr.length() > 0) &&
        (!lDBOpr.equals("call_for_written")) ){
        //call_for_written
        action = "people_applicant_select";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) &&
             (!lDBOpr.equals("call")) ){
        //call
        action = "people_applicant_call";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) &&
             (!lDBOpr.equals("remove")) ){
        action = "people_applicant_remove";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) &&
             (!lDBOpr.equals("detail")) ){
        action = "people_applicant_detail";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) &&
             (!lDBOpr.equals("upd_wrtn_performance"))){
        action = "upd_applicant_wrtn_performance";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) &&
             (!lDBOpr.equals("upd_tech_performance"))){
        action = "upd_applicant_tech_performance";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) &&
             (!lDBOpr.equals("upd_hr_performance"))){
        action = "upd_applicant_hr_performance";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) &&
             (!lDBOpr.equals("upd_wrtn_record"))){
        action = "upd_wrtn_record";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) &&
             (!lDBOpr.equals("upd_tech_record"))){
        action = "upd_tech_record";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) &&
             (!lDBOpr.equals("upd_hr_record"))){
        action = "upd_hr_record";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) &&
             (!lDBOpr.equals("shortlist_after_wrtn"))){
        action = "shortlist_after_wrtn";
    }
    if( (lDBOpr != null && lDBOpr.length() > 0) &&
        (!lDBOpr.equals("shortlist_after_tech"))){
        action = "shortlist_after_tech";
    }
    if( (lDBOpr != null && lDBOpr.length() > 0) &&
        (!lDBOpr.equals("shortlist_after_hr"))){
        action = "shortlist_after_hr";
    }
    else if( (lDBOpr != null && lDBOpr.length() > 0) &&
             (!lDBOpr.equals("applicant_call_for_tech"))){
        action = "applicant_call_for_tech";
    }
}

```

```

}
else if( lDBopr != null && lDBopr.length() > 0) &&
(lDBopr.equals("applicant_call_for_hr")){
    action = "applicant_call_for_hr";
}
else if( lDBopr != null && lDBopr.length() > 0) &&
(lDBopr.equals("applicant_call_for_final")){
    action = "applicant_call_for_final";
}
else if( lDBopr != null && lDBopr.length() > 0) &&
(lDBopr.equals("delete")) ){
    action = "applicant_remove_for_tech";
}
else if( lDBopr != null && lDBopr.length() > 0) &&
(lDBopr.equals("discard")) ){
    action = "applicant_remove_for_hr";
}
else if( lDBopr != null && lDBopr.length() > 0) &&
(lDBopr.equals("discard_for_final")) ){
    action = "applicant_remove_for_final";
}
else if( lDBopr != null && lDBopr.length() > 0) &&
(lDBopr.equals("final_selected")) ){
    action = "applicant_final_selected";
}
String action_submit = request.getParameter("action_submit");
String action_wrtn_dtl_submit = request.getParameter("action_wrtn_dtl_submit ");
String action_edit = request.getParameter("action_edit");
String action_select = request.getParameter("action_select");
.....
.....
.....
} // end of doPost
} // end of class

```

In Listing 23.29, two annotations, @WebServlet and @Override, are used. The use of the @WebServlet annotation has eliminated the need to define the servlet mapping in the web.xml file and the @Override annotation is used to depict that the method is overridden. All the further actions taken by the servlet depend upon the various checks applied on the value set for the action variable. Various if-else blocks are used in the code to test numerous combinations of conditions resulting in an execution of different lines of code.

The servlet and other JSP pages in this document are highly reusable by nature, i.e. the same code file can be used in a number of ways for different purposes. The other important variable, named criteria, is set throughout the code with different string values, as shown in Listing 23.26. The same variable is being used multiple times specifying the criteria being concatenated to the query executed by the methods of the ApplicantTestDt1DBMethods class. The following code snippet shows how to set a value for criteria to execute different types of queries:

```

.....
.....
.....
if(dbopr != null && dbopr.equals("upd_wrtn_record"))
criteria = "where test_status='W'";
else
if(dbopr != null && dbopr.equals("upd_tech_record"))
criteria = "where test_status='T' and applicant_id not in (select applicant_id
from APPLICANT_TEST_DETAIL where test_status in ('HR','confirm'))";
else
if(dbopr != null && dbopr.equals("upd_hr_record"))
criteria = "where test_status='HR' and applicant_id not in (select applicant_id
from APPLICANT_TEST_DETAIL where test_status in ('confirm'))";
.....
.....
.....

```

To view the full code of the servlet, you can get the file applicant_test_dtl.java from PeopleMgmt\people-mgmt\WEB-INF\src folder in the CD.

The next subsection discusses the ApplicantTestDtlDBObj.java file.

Creating the ApplicantTestDtlDBObj Class

The ApplicantTestDtlDBObj class is similar to other data access objects that define attributes corresponding to the columns of the APPLICANT_TEST_DETAIL table. The object of this class is used to retrieve data from the APPLICANT_TEST_DETAIL table and make the data available to the programmer.

Listing 23.30 shows the code of the ApplicantTestDtlDBObj.java file (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src\com\Applicant folder on CD):

Listing 23.30: Showing the Code of the ApplicantTestDtlDBObj.java File

```
package com.Applicant;
public class ApplicantTestDtlDBObj
{
    public String test_id;
    public String test_name;
    public String applicant_id;
    public String applicant_name;
    public String test_date;
    public String test_time;
    public String present_status;
    public long total_marks;
    public long marks_gained;
    public String test_status;
    public String pass_fail;
    public String next_round;
}
```

Creating the ApplicantTestDtlDBMethods Class

As the name suggests, the ApplicantTestDtlDBMethods class contains methods that are used by the applicant_test_dtl servlet to interact with the database and manipulate the data of the APPLICANT_TEST_DETAIL table.

Listing 23.31 shows various methods for the ApplicantTestDtlDBMethods class (you can find the ApplicantTestDtlDBMethods.java file in the PeopleMgmt\people-mgmt\WEB-INF\src\com\Applicant folder on CD):

Listing 23.31: Showing the Code of the ApplicantTestDtlDBMethods.java File

```
package com.Applicant;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.util.ArrayList;
import com.Applicant.*;
public class ApplicantTestDtlDBMethods
{
    public String DBUser;
    public String DBPswd;
    public String DBUrl ;
    public ApplicantTestDtlDBMethods(){ }

    public ApplicantTestDtlDBMethods(String inDBUser, String inDBPswd, String
        inDBUrl )
    {
        DBUser = inDBUser ;
        DBPswd = inDBPswd;
        DBUrl = inDBUrl;
    }
}
```

```

public void initializeApplicantTestDt1DBObj(ApplicantTestDt1DBObj
inApplicantTestDt1DBObj)
{
    . . .
    . . .
}

public ApplicantTestDt1DBObj getRecordByPrimaryKey(String inApplicantId)
{
    . . .
    . . .
}

public int updateApplicantTestDt1(ApplicantTestDt1DBObj
inApplicantTestDt1DBObj)
{
    . . .
    . . .
}

public ApplicantTestDt1DBObj populateApplicantTestDt1DBObjFromReq
(HttpServletRequest inReq)
{
    . . .
    . . .
}

public int insertApplicantTestDt1(ApplicantTestDt1DBObj
inApplicantTestDt1DBObj)
{
    . . .
    . . .
}

public void deleteApplicant(String inApplicantId)
{
    . . .
    . . .
}
}
}

```

The ApplicantTestDt1DBMethods class contains various methods similar to the methods of the xxxxxMethods classes developed earlier in the project. For example, the `insertApplicantTestDt1()` method is used to insert a new record, the `updateApplicantTestDt1()` method is used to update an existing record, and the `deleteApplicant()` method is used to delete a record from the APPLICANT_TEST_DETAIL table.

Let's now discuss the different JSP pages that are used for view.

Designing JSP Views

In this project, the servlets are created to process and control the flow of the requests; whereas, other classes are used to implement other functionalities, such as insert, edit, or delete records from the database. To implement the defined functionalities, you need to create user interfaces by using JSP pages.

Let's start the discussion with the `applicant_test_dt1_create` JSP page.

Creating the `applicant_test_dt1_create` JSP Page

The `applicant_test_dt1_create` JSP page is designed to provide the details of a new test. This JSP page contains various fields, such as Test Id, Test Name, Test Date (yy-mm-dd), Test Time (hh:mm), and Total Marks. The code of the `applicant_test_dt1_create` JSP page has been given in Listing 23.32 (you can find the `applicant_test_dt1_create.jsp` file in the `PeopleMgmt\people-mgmt\jsp` folder on CD):

Listing 23.32: Showing the Code of the applicant_test_dtl_create.jsp File

```

<%@ page language="java" %>
<%@ page session="true" %>
<html>
<head>
    <title>www.peoplemanagementsolutions.com</title>
    <link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
    <td colspan="2" ><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td width="900" valign="top"><%@ include file="../jsp/people_default_menu.jsp" %></td>
</tr>
<tr>
<td width = "750" valign="top">
<p>&nbsp;</p>
<hr width=500 color="#AAAAAA">
<table border="0" width=500 align=center>
<form name="form1" method="post">
<tr>
<td bgcolor ='#AAAAAA' colspan='4' align=center class=whitetext><b>Test
                Detail</b></td>
</tr>
<tr>
<td >Test Id</td>
<td align='left'><input type='text' name='test_id' id='test_id' size ='10'
                value=''/></td>
<td>Test Name</td>
<td align='left'><input type='text' name='test_name' id='test_name' size ='10'
                value=''/></td>
</tr>
<tr>
<td>Test Date(yyyy-mm-dd)</td>
<td align='left'><input type='text' name='test_date' id='test_date' size ='10'
                value=''/></td>
<td>Test Time(hh:mm)</td>
<td align='left'><input type='text' name='test_time' id='test_time' size ='10'
                value=''/></td>
</tr>
<tr>
<td>Total Marks </td>
<td align='left' ><input type='text' name='total_marks' id='total_marks' size
                ='10' value=''/></td>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td align='center' colspan=4>
<input type='submit' name='submit' id='submit' size ='10' value='Submit Detail' />
<input type='hidden' name='action_submit' id='action_submit' size ='10'
                value='people_applicant_wrtn_test_dtl_submit' />
</td>
</tr>
</table>
<hr width=500 color="#AAAAAA">
</td>
</tr>
<tr>
<td colspan="2" ><%@include file="../jsp/people_footer.jsp" %></td>
</tr>
</table></body></html>

```

In Listing 23.32, a hidden field is used to submit the test details.. Figure 23.13 displays the output of the applicant_test_dtl_create JSP page:

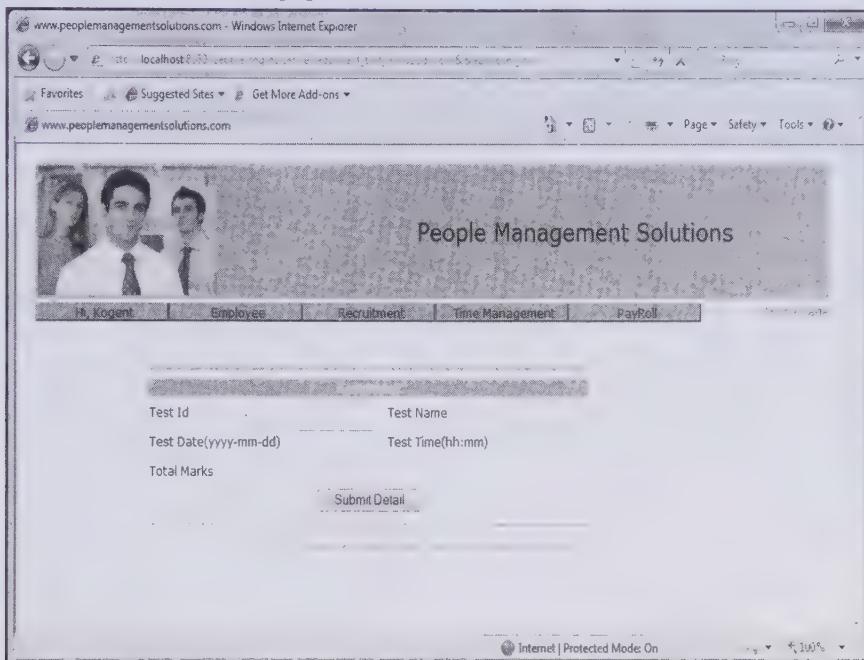


Figure 23.13: Displaying the applicant_test_dtl_create JSP Page to Add Values of New Fields of a New Test

In the applicant_test_dtl_create JSP page, you need to enter various details, such as test id, test name, test date, test time, and total marks.

Let's now discuss the other view page, the applicant_test_dtl_list JSP page.

Creating the applicant_test_dtl_list JSP Page

The applicant_test_dtl_list JSP page is used to provide a list of all the tests and their results. On the basis of the parameters that are passed with the query string, the details of an applicant for a round are displayed on the applicant_test_dtl_list JSP page. Listing 23.33 shows the use of the dbopr parameter to find out the proper if-block to be executed according to the specific round (written round, technical round, and HR round).

Listing 23.33 shows the code of the applicant_test_dtl_list.jsp file (you can find the applicant_test_dtl_list.jsp file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.33: Showing the Code of the applicant_test_dtl_list.jsp File

```
<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.Applicant.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.util.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Applicant Test Detail List</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2"><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
```

```

<tr>
<td width="900">
    <%@ include file="../jsp/people_default_menu.jsp" %>
</td>
</tr>
<tr>
<td width = "750" valign="top">
<table border="0" width=100% >
<%
    String dbopr = "";
    dbopr = (String)session.getAttribute("dbopr");
%>
<tr>
<td class=whitetext bgcolor ='#AAAAAA' align='center' >Test Id</th>
<td class=whitetext bgcolor ='#AAAAAA' align='center' >Test Name</th>
<td class=whitetext bgcolor ='#AAAAAA' align='center' >Applicant Id</th>
<td class=whitetext bgcolor ='#AAAAAA' align='center' colspan='2' >Applicant
    Name</th>
<td class=whitetext bgcolor ='#AAAAAA' align='center' >Test Date</th>
<td class=whitetext bgcolor ='#AAAAAA' align='center' >Test Time</th>
<td class=whitetext bgcolor ='#AAAAAA' align='center' >Present Status</th>
<td class=whitetext bgcolor ='#AAAAAA' align='center' >Total Marks</th>
<td class=whitetext bgcolor ='#AAAAAA' align='center' >Marks Gained</th>
<td class=whitetext bgcolor ='#AAAAAA' align='center' >Pass Fail</th>
<td class=whitetext bgcolor ='#AAAAAA' align='center' >--</th>
</tr>
<%
    ArrayList ApplicantTestDtlList = new ArrayList();
    ApplicantTestDtlList =
        (ArrayList)session.getAttribute("ApplicantTestDtlList");
    ArrayList selectApplicantTechList = new ArrayList();
    selectApplicantTechList =
        (ArrayList)session.getAttribute("selectApplicantTechList");
    if ( ApplicantTestDtlList != null && ApplicantTestDtlList.size() > 0 ){
        for ( int size = 1; size <= ApplicantTestDtlList.size() ; size++ ){
            ApplicantTestDtlDBObj applicantTestDtlDBObj = new
                ApplicantTestDtlDBObj();
            applicantTestDtlDBObj =
                (ApplicantTestDtlDBObj)ApplicantTestDtlList.get(size-
                    1);
        }
    }
<form name="form1" method="post">
<tr bgcolor ='#AAAAAA'>
<td align='center' ><%=applicantTestDtlDBObj.test_id%>
<input type='hidden' name='test_id' id='test_id' size ='20'
    value='<%=applicantTestDtlDBObj.test_id%>'>
</td>
<td align='left' ><%=applicantTestDtlDBObj.test_name%>
<input type='hidden' name='test_name' id='test_name' size ='20'
    value='<%=applicantTestDtlDBObj.test_name%>'>
</td>
<td align='center' ><%=applicantTestDtlDBObj.applicant_id%>
<input type='hidden' name='applicant_id' id='applicant_id' size ='20'
    value='<%=applicantTestDtlDBObj.applicant_id%>'>
</td>
<td align='left' colspan='2' ><%=applicantTestDtlDBObj.applicant_name%>
<input type='hidden' name='applicant_name' id='applicant_name' size ='20'
    value='<%=applicantTestDtlDBObj.applicant_name%>'>
</td>
<td align='center' ><%=applicantTestDtlDBObj.test_date%>
<input type='hidden' name='test_date' id='test_date' size ='20'
    value='<%=applicantTestDtlDBObj.test_date%>'></td>
<td align='center' ><%=applicantTestDtlDBObj.test_time%>
<input type='hidden' name='test_time' id='test_time' size ='20'
    value='<%=applicantTestDtlDBObj.test_time%>'>

```

```

</td>
<%
    if( applicantTestDt1DBObj.present_status != null )
        out.println("<td align='center'><input type='text' disabled='disabled' "
            + "name='present_status' id='present_status' size ='5' value='"
            + applicantTestDt1DBObj.present_status+"' /> </td>"); 
    else{
        out.println("<td align='center' ><input type='text' disabled='disabled' "
            + "name='present_status' id='present_status' size ='5' value=' ' /></td>"); 
    }
%>
<td align='center'><%=applicantTestDt1DBObj.total_marks%>
<input type='hidden' name='total_marks' id='total_marks' size = '20'
    value='<%=applicantTestDt1DBObj.total_marks%>' />
</td>
<td align='center'>
<input type='text' disabled='disabled' name='marks_gained' id='marks_gained' size=
    '5' value = '<%=applicantTestDt1DBObj.marks_gained%>' />
</td>
<%
    if( applicantTestDt1DBObj.pass_fail != null )
        out.println("<td align='center'><input type='text' disabled='disabled' "
            + "name='pass_fail' id='pass_fail' size ='5' value='"
            + applicantTestDt1DBObj.pass_fail+"' /> </td>"); 
    else{
        out.println("<td align='center'><input type='text' disabled='disabled' "
            + "name='pass_fail' id='pass_fail' size ='5' value=' ' /> </td>"); 
    }
    if( dbopr != null && ( dbopr.equals("shortlist_after_wrtn") ||
        dbopr.equals("applicant_call_for_tech") || dbopr.equals("delete") ) ){
        out.println("<td align='center' bgcolor ='#AAAAAA'>");
        out.println("<a href='http://localhost:8080/people-mgmt/servlet-
/applicant_test_dt1?dbopr=applicant_call_for_tech&applicant_id="+applicantT
estDt1DBObj.applicant_id+" class=yellowlink>Select For Tech </a>"); 
        out.println("</td >"); 
    }
    else{
        if( dbopr != null && ( dbopr.equals("shortlist_after_tech") ||
            dbopr.equals("applicant_call_for_hr") || dbopr.equals("discard") ) ){
            out.println("<td align='center' bgcolor ='#AAAAAA'>"); 
            out.println("<a href='http://localhost:8080/people-
mgmt/servlet/applicant_test_dt1?dbopr=applicant_call_for_hr&applicant_id
="+applicantTestDt1DBObj.applicant_id+" class=yellowlink>Select For HR
</a>"); 
            out.println("</td >"); 
        }
        else
            if( dbopr != null && ( dbopr.equals("shortlist_after_hr") ||
                dbopr.equals("applicant_call_for_final") || 
                dbopr.equals("discard_for_final") ) ){
                out.println("<td align='center' bgcolor ='#AAAAAA'>"); 
                out.println("<a href='http://localhost:8080/people-
mgmt/servlet/applicant_test_dt1?dbopr=applicant_call_for_final&&
applicant_id="+applicantTestDt1DBObj.applicant_id+" class=
yellowlink>Select For Final </a>"); 
                out.println("</td >"); 
            }
        else
            if( dbopr != null && ( dbopr.equals("upd_wrtn_performance") ||
                dbopr.equals("upd_wrtn_record") ) ){
                out.println("<td align='center' bgcolor ='#AAAAAA'>"); 
                out.println("<a href='http://localhost:8080/people-
");

```

```

mgmt/servlet/applicant_test_dt1?dbopr=upd_wrtn_record&&
applicantid='"+applicantTestDt1DBObj.applicant_id+" ' class=
yellowlink>Edit </a">);
out.println("</td >");
}
else if( dbopr != null && ( dbopr.equals("upd_tech_performance") || 
dbopr.equals("upd_tech_record") ) )
{
    out.println("<td align='center' bgcolor ='#AAAAAA'>");
    out.println("<a href='http://localhost:8080/people-
mgmt/servlet/applicant_test_dt1?dbopr=upd_tech_record&&
applicantid='"+applicantTestDt1DBObj.applicant_id+" ' 
class=yellowlink>Edit </a">);
    out.println("</td >");
}
else if( dbopr != null && ( dbopr.equals("upd_hr_performance") || 
dbopr.equals("upd_hr_record") ) )
{
    out.println("<td align='center' bgcolor ='#AAAAAA'>");
    out.println("<a href='http://localhost:8080/people-
mgmt/servlet/applicant_test_dt1?dbopr=upd_hr_record&&
applicantid='"+applicantTestDt1DBObj.applicant_id+" ' class=
yellowlink>Edit </a">);
    out.println("</td >");
}
}
}
else{
    out.println("Applicant does not exist!!!");
}
if( selectApplicantTechList != null && selectApplicantTechList.size() > 0)
{
out.println("<tr>");
out.println("<td class=whitetext bgcolor ='#AAAAAA' align='center' 
colspan='13'>The Selected Applicant</th>"); 
out.println("</tr>"); 
for( int size = 1; size <= selectApplicantTechList.size() ; size++ ){
    ApplicantTestDt1DBObj applicantTestDt1DBObj = new ApplicantTestDt1DBObj();
    applicantTestDt1DBObj =
        (ApplicantTestDt1DBObj)selectApplicantTechList.get(size-1);
    out.println("<tr bgcolor ='#AAAAAA'>"); 
    out.println("<td align='center' >"+applicantTestDt1DBObj.test_id+"</td>"); 
    out.println("<td align='center' >"+applicantTestDt1DBObj.test_name+" 
</td>"); 
    out.println("<td align='center' >"+applicantTestDt1DBObj.applicant_id+
" </td>"); 
    out.println("<td align='center' colspan='2' >"+applicantTestDt1DBObj.
applicant_name+ "</td>"); 
    out.println("<td align='center' >"+applicantTestDt1DBObj.test_date+"</td>"); 
    out.println("<td align='center' >"+applicantTestDt1DBObj.test_time+"</td>"); 
    out.println("<td align='center'>"+applicantTestDt1DBObj.present_status
+ "</td>"); 
    out.println("<td align='center' >"+applicantTestDt1DBObj.total_marks
+ "</td>"); 
    out.println("<td align='center' >"+applicantTestDt1DBObj.marks_gained
+ "</td>"); 
    out.println("<td align='center' >"+applicantTestDt1DBObj.pass_fail+"</td>"); 
    if( dbopr != null && ( dbopr.equals("shortlist_after_wrtn") || 
dbopr.equals("shortlist_after_tech") || dbopr.equals
("applicant_call_for_tech") || dbopr.equals("delete") )){ 
        out.println("<td align='center' colspan='2' bgcolor ='#AAAAAA'>"); 
        out.println("<a href='http://localhost:8080/people-mgmt/servlet/
applicant_test_dt1?dbopr=delete&&applicant_id='"+applicantTestDt1DBObj.

```

```

applicant_id+" ' class=yellowlink>Delete</a">;
out.println("</td >");
}
else
if( dbopr != null && ( dbopr.equals("shortlist_after_tech") || dbopr.equals("applicant_call_for_hr") || dbopr.equals("discard") )){
    out.println("<td align='center' colspan='2' bgcolor ='#AAAAAA'>"); 
    out.println("<a href='http://localhost:8080/people-mgmt/servlet/applicant_test_dt1?dbopr=discard&&applicant_id="+applicantTestDt1DBObj
    .applicant_id+" ' class=yellowlink>Discard</a">"); 
    out.println("</td >"); 
}
else
if( dbopr != null && ( dbopr.equals("shortlist_after_hr") || dbopr.equals("discard_for_final") || dbopr.equals("applicant_call_for_final") )){
    out.println("<td align='center' colspan='2' bgcolor ='#AAAAAA'>"); 
    out.println("<a href='http://localhost:8080/people-mgmt/servlet/applicant_test_dt1?dbopr=discard_for_final&&applicant_id="+
    applicantTestDt1DBObj.applicant_id+" ' class=yellowlink
    >Remove</a>"); 
    out.println("</td >"); 
}
out.println("</tr>"); 
}
if( dbopr != null && ( dbopr.equals("delete") || dbopr.equals("applicant_call_for_tech") )){
    out.println("<tr>"); 
    out.println("<td align='center' colspan='13'><input type='submit'
    name='submit' id='submit' size ='10' value='Call For Tech'/> </td>"); 
    out.println("<input type='hidden' name='action_submit' id='action_submit'
    size ='10' value='applicant_call_tech_dt1_submit'/> "); 
    out.println("</tr>"); 
}
else
if( dbopr != null && ( dbopr.equals("discard") || dbopr.equals("applicant_call_for_hr") )){
    out.println("<tr>"); 
    out.println("<td align='center' colspan='13'><input type='submit'
    name='submit' id='submit' size ='10' value='Call For HR'/> </td>"); 
    out.println("<input type='hidden' name='action_submit' id='action_submit'
    size ='10' value='applicant_call_hr_dt1_submit'/> "); 
    out.println("</tr>"); 
}
else
if( dbopr != null && ( dbopr.equals("discard_for_final") || dbopr.equals("applicant_call_for_final") )){
    out.println("<tr>"); 
    out.println("<td align='center' colspan='13'><input type='submit'
    name='submit' id='submit' size ='10' value='Select Final'/> </td>"); 
    out.println("<input type='hidden' name='action_submit' id='action_submit'
    size ='10' value='applicant_select_for_final_submit'/> "); 
    out.println("</tr>"); 
}
}
out.println("</tr>"); 
%>
</table>
</td>
</tr>
<tr>
    <td colspan="2"><%@include file="..../jsp/people_footer.jsp"%></td>
</tr>
</table>
</body>
</html>

```

The applicant_test_dtl_list JSP page shows different rounds of results on the basis of the value of the dbopr variable, and can be executed by clicking the Update Result hyperlink provided for different rounds, such as written, technical, and HR.

Figure 23.14 displays the output of the applicant_test_dtl_list JSP page, showing the list of applicants to be updated for the written round:

Id	Subject	Round	Name	Test Date	Test Time	Marks Obtained	Marks Scored	Action
101	Java	1	Akanksha	2010-07-09	12:00	300	300	Edit
102	Java	3	Vinay Kumar	2010-07-09	12:00	300	300	Edit

Figure 23.14: Displaying the Test Details for Written Round in the applicant_test_dtl_list JSP Page

The applicant_test_dtl_list JSP page is used to show the test details for all rounds and allows you to edit the test details by clicking the Edit link. This results in the invocation of the applicant_test_dtl_update JSP page.

Creating the applicant_test_dtl_update JSP Page

The applicant_test_dtl_update JSP page provides a form to update a test result. The same JSP page is used to update the results of all rounds of tests. Listing 23.34 shows the code of the applicant_test_dtl_update.jsp file (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.34: Showing the Code of the applicant_test_dtl_update.jsp File

```
<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.Applicant.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.util.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Test Detail Update</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
```

```

<table width="900" border="0" align="center">
<tr>
    <td colspan="2" ><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
    <td width="900" valign="top"><%@ include file="../jsp/people_default_menu.jsp" %></td>
</tr>
<tr>
    <td width="750" valign="top">
<p>&nbsp;</p>
<div align=center class=boldblack>Update Result</div>
<hr width=500 color="#AAAAAA">
<table border="0" width=300 align=center>
<form name="form1" method="post">
<%
    ApplicantTestDt1DBObj applicantTestDt1DBObj = new ApplicantTestDt1DBObj();
    applicantTestDt1DBObj =
        (ApplicantTestDt1DBObj)session.getAttribute("applicantTestDt1DBObj");
<%
    <input type='hidden' name='applicant_id' id='applicant_id' size ='20'
    value='<%=applicantTestDt1DBObj.applicant_id%>' />
    <input type='hidden' name='test_id' id='test_id' size ='20'
    value='<%=applicantTestDt1DBObj.test_id%>' />
    <input type='hidden' name='test_name' id='test_name' size ='20'
    value='<%=applicantTestDt1DBObj.test_name%>' />;
    <tr><td>Status in Test</td>
    <td>
        <select name='present_status'>
            <option value=Absent >Absent</option>
            <option value=Present selected>Present</option>
        </select>
    </td></tr>
    <tr><td>Total Marks</td>
    <td>
        <input type='hidden' name='total_marks' id='total_marks' size ='20'
               value=''+applicantTestDt1DBObj.total_marks+''/>
        <%=applicantTestDt1DBObj.total_marks%>
    </td></tr>
    <tr>
        <td> Marks Gained</td>
        <td><%
            if( applicantTestDt1DBObj.marks_gained != 0 )
                out.println("<input type='text' name='marks_gained' id='marks_gained'
                           size= '5' value = '" + applicantTestDt1DBObj.marks_gained + "' />");
            else
                out.println("<input type='text' name='marks_gained' id='marks_gained'
                           size= '5' value = '0' />");<%
        </td></tr>
    <tr><td>Result</td>
    <td>
        <%
            if( applicantTestDt1DBObj.pass_fail != null )
                out.println("<input type='text' name='pass_fail' id='pass_fail' size
                           = '5' value ='" + applicantTestDt1DBObj.pass_fail + "' />");<%
            else{
                out.println("<SELECT name='pass_fail' ><option value=></option>
                            <option value=Pass>Pass</option><option value=Fail>
                            Fail</option></SELECT>");<%
            }<%
        </td></tr>
    <tr><td colspan=2 align=center>
        <input type='hidden' name='action_submit' id='action_submit' size = '10'
               value='people_applicant_wrtn_test_dt1_upd_submit'/>
    </td></tr>

```

```

<input type='submit' name='submit' id='submit' value='Update Detail' />
</td></tr>
</table></td>
</tr>
<tr>
    <td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr>
</table></body></html>

```

The code given in Listing 23.34 is used to update the results of an applicant. Figure 23.15 shows the output of the applicant_test_dtl_update JSP page:

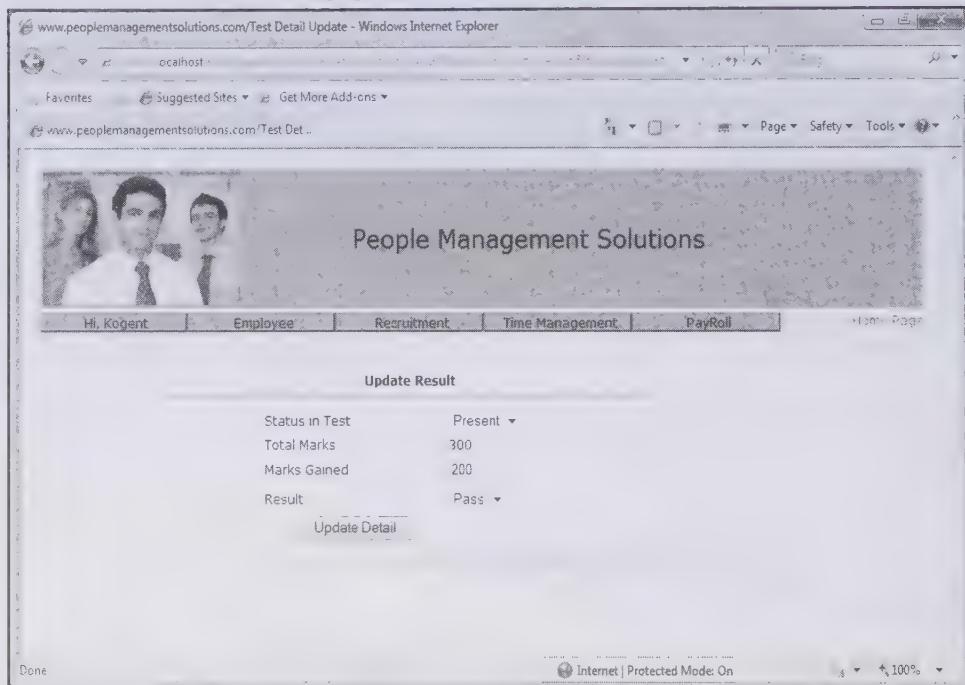


Figure 23.15: Displaying the Update Result Form as the Output of the applicant_test_dtl_update JSP Page

The applicant_test_dtl_update JSP page is used to update the results of the applicants.

Let's now discuss the applicant_final_selected_list JSP page.

Creating the applicant_final_selected_list JSP Page

The applicant_final_selected_list JSP page shows the list of finally selected candidates. In this page, the ApplicantTestDtlList ArrayList is used as a container to store the details of all selected candidates who have passed all the tests and cleared all the rounds.

Listing 23.35 shows the code of the applicant_final_selected_list.jsp file (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.35: Showing the Code of the applicant_final_selected_list.jsp File

```

<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.Applicant.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.util.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Selected Candidates</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />

```

```

</head>
<body>
<table width="900" border="0" align="center">
<tr>
    <td colspan="2"><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
    <td width="900"><%@ include file="../jsp/people_default_menu.jsp" %></td>
</tr>
<tr>
    <td width = "750" valign="top">
        <p>&nbsp;</p>
        <div align=center class=boldblack>List of Selected Candidates to be
            Appointed.</div>
        <hr width=400 color="#AAAAAA">
        <table border="0" width=100% >
        <%
            String dbopr = "";
            dbopr = (String)session.getAttribute("dbopr");
        %>
        <tr height=20>
            <td class=whitetext bgcolor ='#AAAAAA' align='center' colspan='2'>Applicant
                Id</td>
            <td class=whitetext bgcolor ='#AAAAAA' align='center' colspan='2'>Applicant
                Name</td>
            <td class=whitetext bgcolor ='#AAAAAA' align='center' colspan='2'></td>
        </tr>
        <%
            ArrayList ApplicantTestDtlList = new ArrayList();
            ApplicantTestDtlList =
                (ArrayList)session.getAttribute("ApplicantTestDtlList");
            ArrayList selectApplicantTechList = new ArrayList();
            selectApplicantTechList =
                (ArrayList)session.getAttribute("selectApplicantTechList");
            if (ApplicantTestDtlList != null && ApplicantTestDtlList.size() > 0 ){
                for ( int size = 1; size <= ApplicantTestDtlList.size() ; size++ ){
                    ApplicantTestDtlDBObj applicantTestDtlDBObj = new
                        ApplicantTestDtlDBObj();
                    applicantTestDtlDBObj =
                        (ApplicantTestDtlDBObj)ApplicantTestDtlList.get(size-1);
                }
            }
        <form name="form1" method="post">
        <tr bgcolor ='#AAAAAA'>
            <input type='hidden' name='test_id' id='test_id' size ='20'
                value='<%=applicantTestDtlDBObj.test_id%>'>
            <input type='hidden' name='test_name' id='test_name' size ='20'
                value='<%=applicantTestDtlDBObj.test_name%>'>
            <input type='hidden' name='applicant_id' id='applicant_id' size ='20'
                value='<%=applicantTestDtlDBObj.applicant_id%>'>
            <td align='center' colspan='2'><%=applicantTestDtlDBObj.applicant_id%></td>
            <input type='hidden' name='applicant_name' id='applicant_name' size ='20'
                value='<%=applicantTestDtlDBObj.applicant_name%>'>
            <td align='center' colspan='2'><%=applicantTestDtlDBObj.applicant_name%></td>
            <input type='hidden' name='test_date' id='test_date' size ='20'
                value='<%=applicantTestDtlDBObj.test_date%>'>
            <input type='hidden' name='test_time' id='test_time' size ='20'
                value='<%=applicantTestDtlDBObj.test_time%>'>
            <td align='center' bgcolor="#AAAAAA">
                <a href='../jsp/selected_applicant_dtls.jsp' class=yellowlink>Detail </a>
            </td >
            <td align='center' bgcolor="#AAAAAA">
                <a href='../jsp/offerletter.jsp' class=yellowlink>Issue Joining Letter </a>
            </td >
        <%
        }
    
```

```

    }
    out.println("</tr>");
%>
</table>
</td></tr>
<tr>
    <td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr></table></body></html>

```

The output of the code provided in Listing 23.35 for the applicant_final_selected_list JSP page is shown in Figure 23.16:

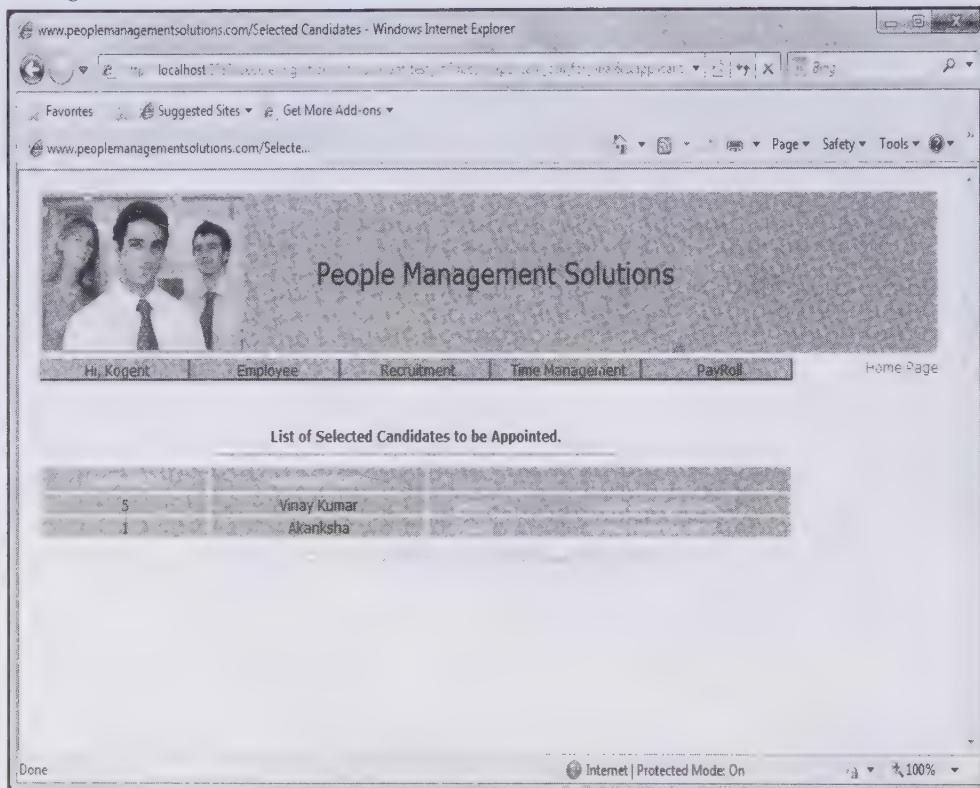


Figure 23.16: Displaying the List of Selected Candidates in the applicant_final_selected_list JSP Page

Figure 23.16 shows the list of all the selected candidates.

Let's now discuss the functioning of the recruitment module.

Working of the Recruitment Module

The recruitment process starts with the registration of new applicants. To register a new applicant, you need to click the New Applicant submenu from the Recruitment menu, and fill all the details of the applicant, such as name, address, qualifications in the relevant fields.

The applicant_list JSP page, shown in Figure 23.12, also contains the Update Applicant link that allows you to view the details of an applicant, delete an applicant, and update the details of the applicant. To find the list of all the candidates who can be called for the written test, click the Call for Test link under the Written Round section. The applicant_list JSP page appears, from where you can click the Select link to select an applicant for the written test, as shown in Figure 23.17:

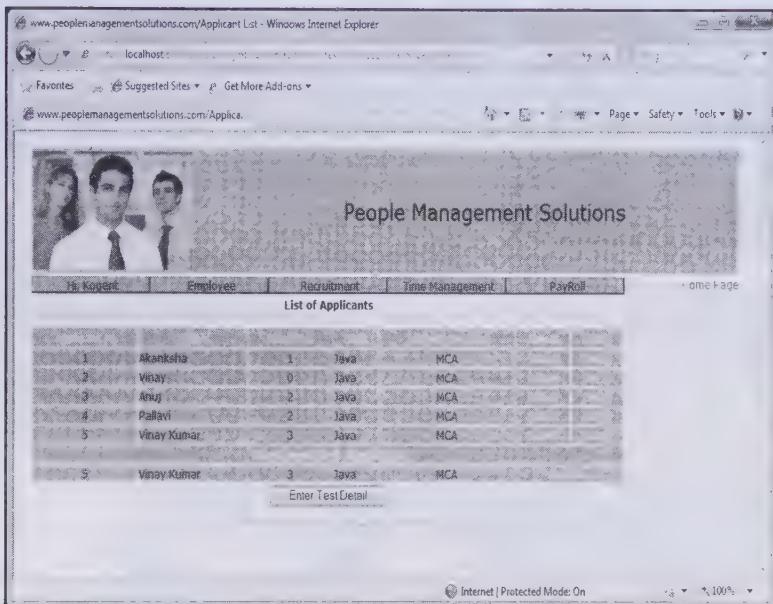


Figure 23.17: Displaying the Applicants Available for Written Test in the applicant_list JSP Page

When you click the Enter Test Detail button, the request is forwarded to the applicant_test_dtl_create JSP page, where you need to enter the details for a new test, such as Test Id, Test Name, Test Date (yy-mm-dd), Test Time (hh:mm), and Total Marks, as shown in Figure 23.13. The names of the applicants who have cleared the written round are displayed for the technical round, as shown in Figure 23.18:

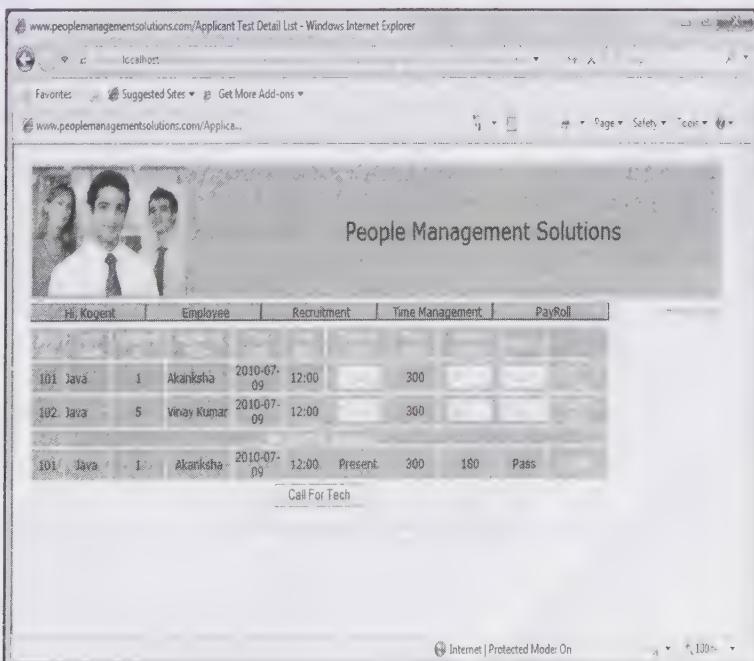


Figure 23.18: Displaying the List of Applicants Selected for the Technical Round

After the applicants clear the technical round, they are called for the HR round. The list of the applicants selected for the HR round is displayed by clicking the Shortlist for HR Round link under the HR Round section.

Figure 23.19 shows the list of the applicants shortlisted for the HR round:

The screenshot shows a web browser window for Internet Explorer displaying a recruitment application interface. The title bar reads "www.peoplemanagementsolutions.com/Applicant Test Detail List - Windows Internet Explorer". The main content area features a banner with three people's faces and the text "People Management Solutions". Below the banner is a navigation menu with links: Hi, Kognent, Employee, Recruitment, Time Management, PayRoll, and Home Page. The main content area displays a table of applicant data:

ID	Subject	Score	Name	Date	Time	Total Marks	Obtained Marks	Status
104	Java	5	Vinay Kumar	2010-07-09	14:00	250	250	Pass
103	Java	1	Akanksha	2010-07-09	13:00	250	150	Fail
103	Java	1	Akanksha	2010-07-09	13:00	Present	250	Pass

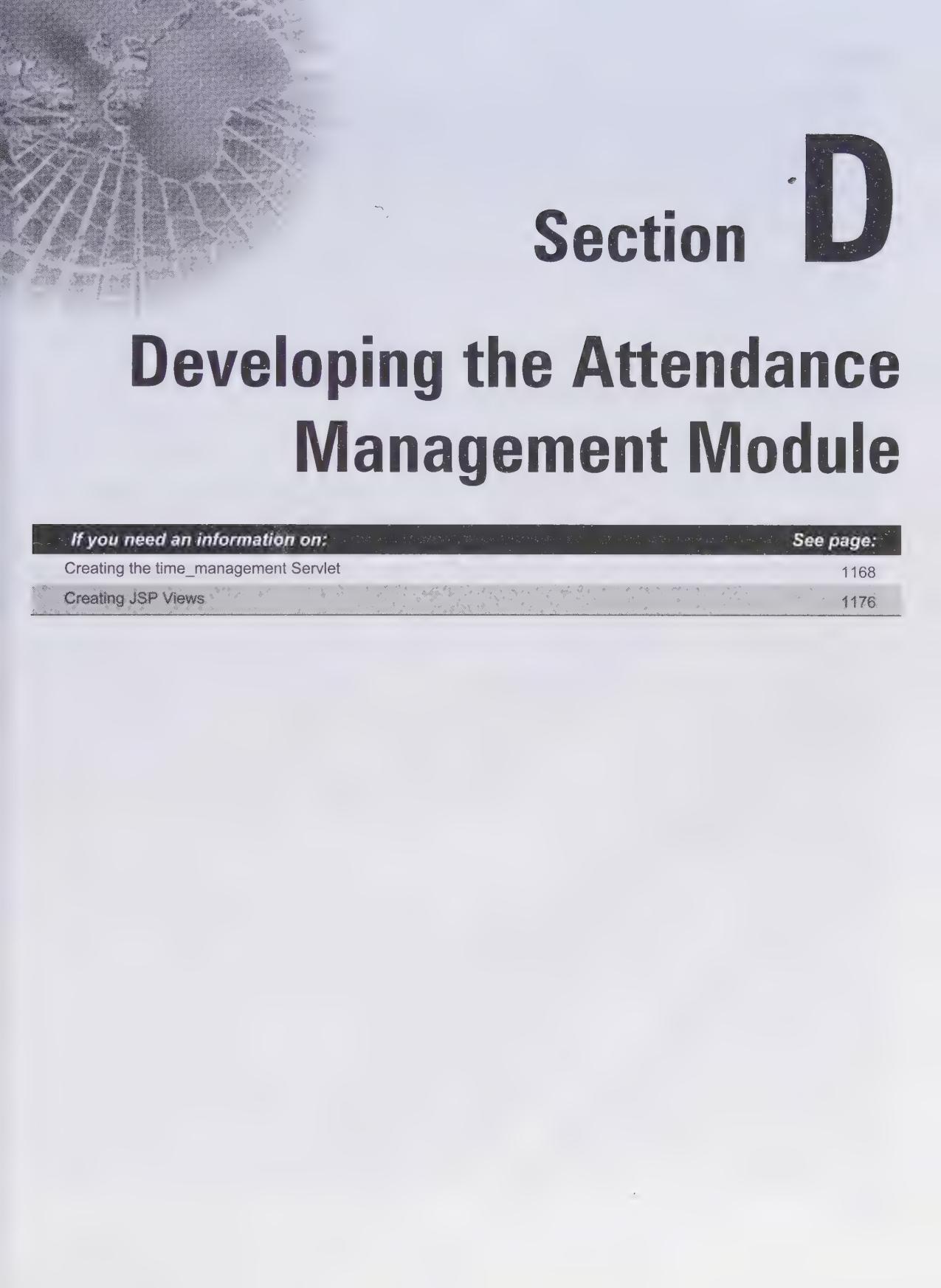
At the bottom of the table, there is a button labeled "Call For HR". The status bar at the bottom of the browser window indicates "Internet | Protected Mode: On" and shows the zoom level as "100%".

Figure 23.19: Displaying the List of Candidates Selected for the HR Round

The appropriate applicant is finally selected on the basis of the tests conducted in the HR round. You can check the list of selected candidates by clicking the Selected Candidate link under the HR Round section.

The recruitment module described in this section handles the registration of a new applicant and maintains tests details of the shortlisted applicants. The test results can be updated and the applicants who have cleared a round are automatically listed for the next round. The recruitment process ends with the selection of the appropriate applicant in the organization.

The next section describes the development and functioning of the Attendance Management module.



Section D

Developing the Attendance Management Module

If you need an information on:

See page:

Creating the time_management Servlet

1168

Creating JSP Views

1176

The Attendance Management module is designed for handling the daily attendance details of every employee; for example, time of entering and leaving the office and number of leaves taken in a month. These details help the HR department in deciding the number of working hours of an employee. This module has been developed using the MVC pattern, similar to the earlier modules.

The following code files need to be created for developing the Attendance Management module:

- ❑ time_management.java
- ❑ DateYearMonthDayDBObj.java
- ❑ EmpDailyAttendanceDBObj.java
- ❑ TimeManagementDBMethods.java
- ❑ employee_daily_attendance.jsp
- ❑ employee_daily_attendance_summary.jsp

Let's start the discussion with the time_management servlet.

Creating the time_management Servlet

The time_management servlet class is invoked when any one of the two links—Enter/Update Attendance or Daily Attendance Summary is clicked. This servlet sets the values for the action and target variables and forwards a user request to a specific JSP page.

Listing 23.36 shows the code of the time_management.java file (you can find the time_management.java file in the PeopleMgmt\people-mgmt\WEB-INF\src folder on CD):

Listing 23.36: Showing the Code of the time_management.java File

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.ArrayList;
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.annotation.*;
import javax.servlet.annotation.WebServlet;
import com.Employee.EmployeeDBMethods;
import com.Employee.EmployeeDBObj;
import com.TimeManagement.*;

@WebServlet(name="time_management", urlPatterns="/servlet/time_management")

public class time_management extends HttpServlet{
    String lDBUser = "";
    String lDBPswd = "";
    String lDBurl = "";

    /**Initialize global variables*/
    @Override
    public void init(ServletConfig config) throws ServletException{
        System.out.println("initializing controller servlet.");
        ServletContext context = config.getServletContext();
        lDBUser = "scott";
        lDBPswd = "tiger";
        lDBurl = "jdbc:oracle:thin:@192.168.1.123:1521:"+"XE";
        super.init(config);
    }
    /**Process the HTTP Get request*/
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException{
        doPost(request, response);
    }
    /**Process the HTTP Post request*/
    @Override
```

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
HttpSession session = request.getSession();
session.setAttribute("lErrorMsg",null);
String target = "";
String action = request.getParameter("action");
String lDBOpr = "";
lDBOpr = (String)request.getParameter("dbopr");
if( lDBOpr != null && lDBOpr.length() > 0) &&
(lDBOpr.equals("daily_attendance_entry")) ){
target = "/jsp/employee_search.jsp";
}
else
if( lDBOpr != null && lDBOpr.length() > 0) && (lDBOpr.equals
("daily_attendance_summary")) ){
action = "daily_attendance_summary";
}
else
if( lDBOpr != null && lDBOpr.length() > 0) &&
(lDBOpr.equals("edit")) ){
action = "daily_attendance_summary_edit";
}
String action_submit = request.getParameter("action_submit");
String action_edit = request.getParameter("action_edit");
System.out.println("action_submit=="+action_submit);
if( action_submit != null || action_edit != null ){
if( request.getParameter("submit").equals("Submit") ){
System.out.println("in the Submit");
if( action_submit.equals("people_employee_search_submit") ){
System.out.println("in the people_employee_insert_submit ");
action = "people_employee_search_submit";
}
}
else
if( request.getParameter("submit").equals("Submit Detail") ){
if( action_submit.equals("emp_daily_att_dtl_submit") )
action = "emp_daily_att_dtl_submit";
}
}
if( action!=null){
System.out.println("in the "+action);
if( action.equals("people_employee_search_submit")){
String lEmpId = "";
String lEmpFName = "";
lEmpId = (String)request.getParameter("emp_id");
lEmpFName = (String)request.getParameter("emp_f_name");
TimeManagementDBMethods timeManagementDBMethods = new
TimeManagementDBMethods(lDBUser,lDBPswd,lDBUrl);
DateYearMonthDayDBObj dateYearMonthDayDBObj = new
DateYearMonthDayDBObj();
dateYearMonthDayDBObj = (DateYearMonthDayDBObj)timeManagement
DBMethods.getCurDateYearMonthDayDBObj();
session.setAttribute("dateYearMonthDayDBObj",dateYearMonth
DayDBObj);
EmployeeDBObj employeeDBObj = new EmployeeDBObj();
EmployeeDBMethods employeeDBMethods = new EmployeeDBMethods(
lDBUser,lDBPswd,lDBUrl);
employeeDBObj = (EmployeeDBObj)employeeDBMethods.getRecord
ByPrimaryKey(lEmpId,lEmpFName);
EmpDailyAttendanceDBObj empDailyAttendanceDBObj = new
EmpDailyAttendanceDBObj();
empDailyAttendanceDBObj = (EmpDailyAttendanceDBObj)time
ManagementDBMethods.getRecordByPrimaryKey(lEmpId,dateYear
}
```

```

        MonthDayDBObj.today_date);
if ( (employeeDBObj.emp_id != null &&
employeeDBObj.emp_f_name.equals(lEmpFName) ) ){
    session.setAttribute("empDailyAttendanceDBObj",
                        empDailyAttendanceDBObj);
    session.setAttribute("employeeDBObj",employeeDBObj);
    target = "/jsp/employee_daily_attendance.jsp";
}
else{
    String lErrorMsg = "Employee doesn't Exist";
    session.setAttribute("lErrorMsg",lErrorMsg);
    target = "/jsp/people_default.jsp";
}
}
else
if (action.equals("daily_attendance_summary_edit")){
    String lEmpId = "";
    String lTodayDate = "";
    String lEmpFName = "";
    lEmpId = (String)request.getParameter("emp_id");
    lTodayDate = (String)request.getParameter
                  ("today_date");
    TimeManagementDBMethods timeManagementDBMethods =
    new TimeManagementDBMethods (lDBUser,lDBPswd,lDBurl);
    DateYearMonthDayDBObj dateYearMonthDayDBObj = new
        DateYearMonthDayDBObj();
    dateYearMonthDayDBObj = (DateYearMonthDayDBObj)time
    ManagementDBMethods.getCurDateYearMonthDayDBObj();
    EmployeeDBObj employeeDBObj = new EmployeeDBObj();
    EmployeeDBMethods employeeDBMethods = new
        EmployeeDBMethods(lDBUser,lDBPswd,lDBurl);
    employeeDBObj = (EmployeeDBObj)employeeDBMethods.get
                    RecordByPrimaryKey(lEmpId,lEmpFName);
    EmpDailyAttendanceDBObj empDailyAttendanceDBObj = new
        EmpDailyAttendanceDBObj();
    empDailyAttendanceDBObj = (EmpDailyAttendanceDBObj)
    timeManagementDBMethods.getRecordByPrimaryKey
        (lEmpId,lTodayDate);
    session.setAttribute("empDailyAttendanceDBObj",
                        empDailyAttendanceDBObj);
    session.setAttribute("employeeDBObj",employeeDBObj);
    session.setAttribute("dateYearMonthDayDBObj",
                        dateYearMonthDayDBObj);
    target = "/jsp/employee_daily_attendance.jsp";
}
else
if (action.equals("daily_attendance_summary")){
    TimeManagementDBMethods timeManagementDBMethods =
    new TimeManagementDBMethods(lDBUser,lDBPswd,lDBurl);
    DateYearMonthDayDBObj dateYearMonthDayDBObj = new
        DateYearMonthDayDBObj();
    dateYearMonthDayDBObj = (DateYearMonthDayDBObj)timeManagement
    DBMethods.getCurDateYearMonthDayDBObj();
    session.setAttribute("dateYearMonthDayDBObj"
                        ,dateYearMonthDayDBObj);
    ArrayList empDailyAttendanceList = new ArrayList();
    String criteria = "";
    criteria = " where today_date='"+dateYear
        MonthDayDBObj.today_date+"'";
    empDailyAttendanceList = (ArrayList)timeManagement
        .selectEmpDailyAttendanceByCriteria(criteria);
    session.setAttribute("empDailyAttendanceList"
                        ,empDailyAttendanceList);
    target = "/jsp/employee_daily_attendance_summary.jsp";
}
}

```

```

    else
        if (action.equals("emp_daily_att_dtl_submit")){
            EmpDailyAttendanceDBObj popEmpDailyAttendanceDBObj =
                new EmpDailyAttendanceDBObj();
            TimeManagementDBMethods timeManagementDBMethods =
                new TimeManagementDBMethods(lDBUser, lDBPswd, lDBurl);
            popEmpDailyAttendanceDBObj = (EmpDailyAttendanceDBObj)timeManagementDBMethods.populateEmpDailyAttendanceDBObjFromReq(request);
            EmpDailyAttendanceDBObj empDailyAttendanceDBObj =
                new EmpDailyAttendanceDBObj();
            empDailyAttendanceDBObj = (EmpDailyAttendanceDBObj) timeManagementDBMethods.getRecordByPrimaryKey(popEmpDailyAttendanceDBObj.emp_id,
                popEmpDailyAttendanceDBObj.today_date);
            if ((empDailyAttendanceDBObj.emp_id != null &
                (popEmpDailyAttendanceDBObj.emp_id).equals(empDailyAttendanceDBObj.emp_id)) && (popEmpDailyAttendanceDBObj.today_date).equals(
                    empDailyAttendanceDBObj.today_date)){
                int rval=timeManagementDBMethods.updateEmpDailyAttendanceDBObjByPrimaryKey(popEmpDailyAttendanceDBObj);
                empDailyAttendanceDBObj = (EmpDailyAttendanceDBObj)timeManagementDBMethods.getRecordByPrimaryKey(popEmpDailyAttendanceDBObj.emp_id,
                    popEmpDailyAttendanceDBObj.today_date);
                session.setAttribute("empDailyAttendanceDBObj",
                    empDailyAttendanceDBObj);
            }
            else{
                int rval = timeManagementDBMethods.insertEmpDailyAttendanceDBObj(popEmpDailyAttendanceDBObj);
                empDailyAttendanceDBObj= (EmpDailyAttendanceDBObj)timeManagementDBMethods.getRecordByPrimaryKey(popEmpDailyAttendanceDBObj.emp_id,
                    popEmpDailyAttendanceDBObj.today_date);
                session.setAttribute("empDailyAttendanceDBObj",
                    empDailyAttendanceDBObj);
            }
            DateYearMonthDayDBObj dateYearMonthDayDBObj = new DateYearMonthDayDBObj();
            dateYearMonthDayDBObj = (DateYearMonthDayDBObj)
                timeManagementDBMethods.getCurDateYearMonthDayDBObj();
            session.setAttribute("dateYearMonthDayDBObj",dateYearMonthDayDBObj);
            ArrayList empDailyAttendanceList = new ArrayList();
            String criteria = "";
            criteria = " where today_date='"+dateYearMonthDayDBObj
                .today_date+"'";
            empDailyAttendanceList = (ArrayList)timeManagementDBMethods.selectEmpDailyAttendanceByCriteria(criteria);
            session.setAttribute("empDailyAttendanceList",
                empDailyAttendanceList);
            target = "/jsp/employee_daily_attendance_summary.jsp";
        }
    }/* forwarding the request/response to the targeted view */
    RequestDispatcher requestDispatcher = getServletContext().getRequestDispatcher(target);
    requestDispatcher.forward(request, response);
} // doPost closed
} // class closed

```

In Listing 23.36, the @WebServlet annotation is used to provide the servlet mapping in the time_management servlet.

Let's now create the classes in the com.TimeManagement package.

Creating the Classes in the com.TimeManagement Package

The com.TimeManagement package contains three classes, namely DateYearMonthDayDBObj, EmpDailyAttendanceDBObj, and TimeManagementDBMethods. These classes support the time_management servlet to complete the entire process of attendance management.

Creating the DateYearMonthDayDBObj Class

The DateYearMonthDayDBObj is a simple Java class that contains various member variables, such as today_date, month, day, and year to store attendance-related information of an employee.

Listing 23.37 shows the code of the DateYearMonthDayDBObj.java file (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src\com\TimeManagement folder on CD):

Listing 23.37: Showing the Code of the DateYearMonthDayDBObj.java File

```
package com.TimeManagement;
public class DateYearMonthDayDBObj {
    public String today_date ;
    public String month ;
    public String day ;
    public long year ;
}
```

Creating the EmpDailyAttendanceDBObj Class

The EmpDailyAttendanceDBObj class is used as a DAO. The member variables of this class are used to store information to be added in the EMPLOYEE_DAILY_ATTENDANCE table, which is used to maintain the attendance records. An attendance record consists of the emp_id, emp_name, today_date, month, day, year, in_time, out_time, and remark fields.

Listing 23.38 shows the code of the EmpDailyAttendanceDBObj.java file (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src\com\TimeManagement folder on CD):

Listing 23.38: Showing the Code of the EmpDailyAttendanceDBObj.java File

```
package com.TimeManagement;
public class EmpDailyAttendanceDBObj {
    public String emp_id ;
    public String emp_name ;
    public String today_date ;
    public String month ;
    public String day ;
    public long year ;
    public String in_time ;
    public String out_time ;
    public String remark ;
}
```

Creating the TimeManagementDBMethods Class

The TimeManagementDBMethods class defines various methods, such as getCurDateYearMonthDayDBObj, initializeEmpDailyAttendanceDBObj, and getRecordByPrimaryKey, which are used to retrieve the desired result from the EMPLOYEE_DAILY_ATTENDANCE table. An object of the EmpDailyAttendanceDBObj class is used either as an argument type or as a return type.

Listing 23.39 shows the code of the TimeManagementDBMethods.java file (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src\com\TimeManagement folder on CD):

Listing 23.39: Showing the Code of the TimeManagementDBMethods.java File

```
package com.TimeManagement;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.util.ArrayList;
import com.Employee.EmployeeDBObj;
```

```

import com.TimeManagement.EmpDailyAttendanceDBObj;
import com.TimeManagement.DateYearMonthDayDBObj;

public class TimeManagementDBMethods{
    public String DBUser;
    public String DBPswd;
    public String DBUrl ;
    public TimeManagementDBMethods(){ }
    public TimeManagementDBMethods(String inDBUser, String inDBPswd, String
        inDBUrl ){
        DBUser = inDBUser;
        DBPswd = inDBPswd;
        DBUrl = inDBUrl;
    }
    public DateYearMonthDayDBObj getCurDateYearMonthDayDBObj(){
        DateYearMonthDayDBObj dateYearMonthDayDBObj = new
            DateYearMonthDayDBObj();
        GregorianCalendar calendar = new GregorianCalendar();
        String month = Integer.toString((calendar.get(Calendar.MONTH) + 1));
        String day = Integer.toString(calendar.get(Calendar.DATE));
        String year = Integer.toString(calendar.get(Calendar.YEAR));
        if( month != null && month.length() < 2 ) month = "0"+month;
        if( day != null && day.length() < 2 ) day = "0"+day;
        String date = year+"-"+month+"-"+day;
        dateYearMonthDayDBObj.today_date = date;
        dateYearMonthDayDBObj.month = getMonth(calendar.get
            (Calendar.MONTH));
        dateYearMonthDayDBObj.day = getDay(calendar.get(Calendar
            .DAY_OF_WEEK));
        dateYearMonthDayDBObj.year = calendar.get(Calendar.YEAR);
        System.out.println("YEAR: " + calendar.get(Calendar.YEAR));
        System.out.println("MONTH: " + calendar.get(Calendar.MONTH));
        System.out.println("DATE: " + calendar.get(Calendar.DATE));
        System.out.println("DAY_OF_WEEK: " +
            calendar.get(Calendar.DAY_OF_WEEK));
        return dateYearMonthDayDBObj;
    }
    public String getDay( int day ){
        String strDay= "";
        if(day == 1) strDay = "SUN";
        else if(day == 2) strDay = "MON";
        else if(day == 3) strDay = "TUS";
        else if(day == 4) strDay = "WED";
        else if(day == 5) strDay = "THU";
        else if(day == 6) strDay = "FRI";
        else if(day == 7) strDay = "SAT";
        return strDay;
    }
    public String getMonth( int month ){
        String strMonth = "";
        if(month == 0) strMonth = "JAN";
        else if(month == 1) strMonth = "FEB";
        else if(month == 2) strMonth = "MAR";
        else if(month == 3) strMonth = "APR";
        else if(month == 4) strMonth = "MAY";
        else if(month == 5) strMonth = "JUN";
        else if(month == 6) strMonth = "JUL";
        else if(month == 7) strMonth = "AUG";
        else if(month == 8) strMonth = "SEP";
        else if(month == 9) strMonth = "OCT";
        else if(month == 10) strMonth = "NOV";
        else if(month == 11) strMonth = "DEC";
        return strMonth;
    }
    public void initializeEmpDailyAttendanceDBObj(EmpDailyAttendanceDBObj
        inEmpDailyAttendanceDBObj ){
        inEmpDailyAttendanceDBObj.emp_id = "";
        inEmpDailyAttendanceDBObj.emp_name = "";
        inEmpDailyAttendanceDBObj.today_date = "";
        inEmpDailyAttendanceDBObj.month = "";
        inEmpDailyAttendanceDBObj.day = "";
        inEmpDailyAttendanceDBObj.year = 0;
    }
}

```

```

    inEmpDailyAttendanceDBObj.in_time = "";
    inEmpDailyAttendanceDBObj.out_time = "";
    inEmpDailyAttendanceDBObj.remark = "";
}
public EmpDailyAttendanceDBObj getRecordByPrimaryKey(String inEmpId, String
    inTodayDate){
    EmpDailyAttendanceDBObj empDailyAttendanceDBObj = new EmpDaily
        AttendanceDBObj();
try{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    Connection conn= DriverManager.getConnection(DBurl,DBUser,DBPswd);
    Statement stmt = conn.createStatement();
    String lsqlString = "select * from EMPLOYEE_DAILY_ATTENDANCE ";
    lsqlString = lsqlString + "where emp_id='"+inEmpId+"' ";
    lsqlString = lsqlString + "and today_date='"+inTodayDate+"' ";
    ResultSet rs = null;
    rs = stmt.executeQuery(lsqlString);
    System.out.println("lsqlString===="+lsqlString);
    getRecordByPrimaryKey== "+lsqlString);
    if( rs.next()){
        System.out.println("fffff=="+rs.getString("emp_id"));
        empDailyAttendanceDBObj.emp_id =
            (String)rs.getString("emp_id");
        empDailyAttendanceDBObj.emp_name =
            (String)rs.getString("emp_name");
        empDailyAttendanceDBObj.today_date =
            (String)rs.getString("today_date");
        empDailyAttendanceDBObj.month =
            (String)rs.getString("month");
        empDailyAttendanceDBObj.day = (String)rs.getString("day");
        empDailyAttendanceDBObj.year = rs.getLong("year");
        String intime=rs.getString("in_time");
        if(intime!=null)
            empDailyAttendanceDBObj.in_time = intime.substring(11,16);
        String outtime=rs.getString("out_time");
        if(outtime!=null)
            empDailyAttendanceDBObj.out_time = outtime.substring(11,16);
        empDailyAttendanceDBObj.remark =
            (String)rs.getString("remark");
        System.out.println("fffff=="+rs.getString("emp_id"));
    }
    else{
        initializeEmpDailyAttendanceDBObj(empDailyAttendanceDBObj);
    }
    System.out.println("fffff===="+empDailyAttendanceDBObj.emp_id);
}
catch(SQLException ex){
    ex.printStackTrace();
}
return empDailyAttendanceDBObj;
}
public ArrayList selectEmpDailyAttendanceByCriteria(String inCriteria){
    ArrayList EmpDailyAttendanceList = new ArrayList();
    try{
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn= DriverManager.getConnection(DBurl,DBUser,DBPswd);
        Statement stmt = conn.createStatement();
        String lsqlString = "select * from EMPLOYEE_DAILY_ATTENDANCE ";
        if( inCriteria != null && inCriteria.length() > 0 ){
            lsqlString = lsqlString +" "+inCriteria+"";
        }
        System.out.println("Criteria===== "+inCriteria+" and query="+lsqlString);
        ResultSet rs = null;
        rs = stmt.executeQuery(lsqlString);
        while( rs.next()){
            EmpDailyAttendanceDBObj empDailyAttendanceDBObj = new
                EmpDailyAttendanceDBObj();
            empDailyAttendanceDBObj.emp_id = (String)rs.getString("emp_id");
            empDailyAttendanceDBObj.emp_name = (String)rs.getString("emp_name");
            empDailyAttendanceDBObj.today_date = (String)rs.getString
                ("today_date");
            empDailyAttendanceDBObj.month = (String)rs.getString("month");
        }
    }
}

```

```

        empDailyAttendanceDBObj.day = (String)rs.getString("day");
        empDailyAttendanceDBObj.year = rs.getLong("year");
        String intime=rs.getString("in_time");
        if(intime!=null)
            empDailyAttendanceDBObj.in_time = intime.substring(11,16);
        String outtime=rs.getString("out_time");
        if(outtime!=null)
            empDailyAttendanceDBObj.out_time = outtime.substring(11,16);
        empDailyAttendanceDBObj.remark = (String)rs.getString
            ("remark");
        EmpDailyAttendanceList.add(empDailyAttendanceDBObj);
    }
}
catch(SQLException ex){
    ex.printStackTrace();
}
return EmpDailyAttendanceList;
}
public int updateEmpDailyAttendanceDBObjByPrimaryKey(EmpDailyAttendanceDBObj
    inEmpDailyAttendanceDBObj){
    int recupd = 0;
    String lQuery = "";
    lQuery = lQuery +"update EMPLOYEE_DAILY_ATTENDANCE set
        emp_name='"+inEmpDailyAttendanceDBObj.emp_name+"'";
    lQuery = lQuery +", month='"+inEmpDailyAttendanceDBObj.month+"'";
    lQuery = lQuery +", day='"+inEmpDailyAttendanceDBObj.day+"'";
    lQuery = lQuery +", year='"+inEmpDailyAttendanceDBObj.year+"'";
    lQuery = lQuery +", in_time=to_date('"+inEmpDailyAttendanceDBObj.today_date+
        "'"+inEmpDailyAttendanceDBObj.in_time+"','yyyy -mm-dd HH24:MI')";
    lQuery = lQuery +", out_time=to_date('"+inEmpDailyAttendanceDBObj
        .today_date+"','"+inEmpDailyAttendanceDBObj.out_time+"','yyyy-mm-
        dd HH24:MI')";
    lQuery = lQuery +", remark='"+inEmpDailyAttendanceDBObj.remark+"'";
    lQuery = lQuery + "where emp_id='"+inEmpDailyAttendanceDBObj
        .emp_id+"'";
    lQuery = lQuery + "and today_date='"+inEmpDailyAttendanceDBObj
        .today_date+"'";
    System.out.println("lQueryString==:"+lQuery);
    try{
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
        Statement stmt = conn.createStatement();
        recupd = stmt.executeUpdate(lQuery);
    }
    catch(SQLException ex){
        ex.printStackTrace();
    }
    return recupd;
}
public EmpDailyAttendanceDBObj populateEmpDailyAttendanceDBObjFromReq
    (HttpServletRequest inReq){
    EmpDailyAttendanceDBObj empDailyAttendanceDBObj = new
        EmpDailyAttendanceDBObj();
    empDailyAttendanceDBObj.emp_id = (String)inReq.getParameter
        ("emp_id");
    empDailyAttendanceDBObj.emp_name = (String)inReq.getParameter
        ("emp_name");
    empDailyAttendanceDBObj.today_date = (String)inReq.getParameter
        ("today_date");
    empDailyAttendanceDBObj.month = (String)inReq.getParameter
        ("month");
    empDailyAttendanceDBObj.day = (String)inReq.getParameter("day");
    empDailyAttendanceDBObj.year = Long.parseLong((String)inReq.
        getParameter("year"));
    empDailyAttendanceDBObj.in_time = (String)inReq.getParameter
        ("in_time");
    empDailyAttendanceDBObj.out_time = (String)inReq.getParameter
        ("out_time");
    empDailyAttendanceDBObj.remark = (String)inReq.getParameter
        ("remark");
    return empDailyAttendanceDBObj;
}

```

```

} public int insertEmpDailyAttendanceDBObj(EmpDailyAttendanceDBObj
    inEmpDailyAttendanceDBObj){
    int recupd = 0;
    String lQuery = "";
    lQuery = lQuery + "insert into EMPLOYEE_DAILY_ATTENDANCE values ( ";
    lQuery = lQuery + "'"+inEmpDailyAttendanceDBObj.emp_id+"', ";
    lQuery = lQuery + "'"+inEmpDailyAttendanceDBObj.emp_name+"', ";
    lQuery = lQuery + "'"+inEmpDailyAttendanceDBObj.today_date+"', ";
    lQuery = lQuery + "'"+inEmpDailyAttendanceDBObj.month+"', ";
    lQuery = lQuery + "'"+inEmpDailyAttendanceDBObj.day+"', ";
    lQuery = lQuery + "'"+inEmpDailyAttendanceDBObj.year+"';

    lQuery = lQuery + ", to_date('" + inEmpDailyAttendanceDBObj
        .today_date+ "'"+inEmpDailyAttendanceDBObj.in_time+"', 'yyyy-mm-dd
        HH24:MI') ";
    lQuery = lQuery + ", to_date('" + inEmpDailyAttendanceDBObj
        .today_date+ "'"+inEmpDailyAttendanceDBObj.out_time+"', 'yyyy-mm-dd
        HH24:MI') ";
    lQuery = lQuery + ", "+inEmpDailyAttendanceDBObj.remark+"'";
    lQuery = lQuery + ")";
    System.out.println("lSqlString==:" + lQuery);
    try{
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
        Statement stmt = conn.createStatement();
        recupd = stmt.executeUpdate(lQuery);
    }
    catch(SQLException ex){
        ex.printStackTrace();
    }
    return recupd;
}
}

```

The TimeManagementDBMethods.java file contains the methods that are used to insert, update, and delete records from the database.

Let's now create the view pages for the Attendance Management module.

Creating JSP Views

In the Attendance Management module, the two JSP pages required to update and monitor daily attendance records are employee_daily_attendance and employee_daily_attendance_summary. The employee_daily_attendance JSP page allows the employees to provide various details, such as the incoming time and outgoing time on a particular day; whereas, the employee_daily_attendance_summary page displays the details of the attendance of all the employees in a summarized format.

Creating the employee_daily_attendance JSP Page

To mark daily attendance, an employee needs to click the Enter/Update Attendance link in the Time Management menu. This displays the employee_search JSP page, in which an employee needs to enter his Employee Id and First Name. On submitting the details entered in the employee_search JSP page, the employee_daily_attendance JSP page is displayed, where the employee needs to enter the In Time and Out Time for the current date. Listing 23.40 shows the code of the employee_daily_attendance JSP page (you can find the employee_daily_attendance.jsp file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.40: Showing the Code of the employee_daily_attendance.jsp File

```

<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.Employee.*" %>
<%@ page import="com.TimeManagement.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Attendance</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>

```

```

<body>
<table width="900" border="0" align="center">
<tr>
    <td colspan="2" ><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td width="900" valign="top"><%@ include file="../jsp/people_default_menu.jsp"
    %></td>
</tr><tr>
<td width = "750" valign="top">
<table border="0" width=100% >
<p>&nbsp;</p>
<hr width=100% color="#AAAAAA>
<%
    String dbopr = "";
    dbopr = (String)session.getAttribute("dbopr");
    EmployeeDBObj employeeDBObj = new EmployeeDBObj();
    employeeDBObj = (EmployeeDBObj)session.getAttribute("employeeDBObj");
    DateYearMonthDayDBObj dateYearMonthDayDBObj = new DateYearMonthDayDBObj();
    dateYearMonthDayDBObj = (DateYearMonthDayDBObj)session.getAttribute
        ("dateYearMonthDayDBObj");
    EmpDailyAttendanceDBObj empDailyAttendanceDBObj = new EmpDailyAttendanceDBObj();
    empDailyAttendanceDBObj = (EmpDailyAttendanceDBObj)session.getAttribute
        ("empDailyAttendanceDBObj");
%>
<form name="form1" method="post">
<tr>
<td bgcolor ='#AAAAAA' colspan='4' class=whitetext height=20 align=center><b>Enter
    In/Out Time</b></td>
</tr>
<tr>
<td>Employee Id</td>
<td>
<%=employeeDBObj.emp_id%>
<input type='hidden' name='emp_id' id='emp_id' size ='10' value='<%=employeeDBObj
    .emp_id%>' />
</td>
<td>Date</td>
<td>
<%=dateYearMonthDayDBObj.today_date%>
<input type='hidden' name='today_date' id='today_date' size ='10' value='<%=date
    YearMonthDayDBObj.today_date%>' />
</td>
</tr>
<tr>
<td>Employee Name</td>
<td><input type='hidden' name='emp_name' id='emp_name' size ='10' value='<%=employ
    eeDBObj.emp_f_name%> <%=employeeDBObj.emp_m_name%> <%=employeeDBObj.emp_l_name%>/>
<%=employeeDBObj.emp_f_name%>
<%
    if(employeeDBObj.emp_m_name!=null)
        out.print(employeeDBObj.emp_m_name);
%>
<%=employeeDBObj.emp_l_name%>
</td>
<td>Day</td>
<td>
<%=dateYearMonthDayDBObj.day%>
<input type='hidden' name='day' id='day' size ='10' value='<%=dateYearMonthDay
    DBObj.day%>' />
</td>
</tr>
<tr><td>Department</td><td><%=employeeDBObj.dept_id%></td>
<td>Month</td>
<td><input type='hidden' name='month' id='month' size ='10' value='<%=dateYearM
    onthDayDBObj.month%>' />
<%=dateYearMonthDayDBObj.month%>
</td></tr>
<tr>
<td>&nbsp;</td><td>&nbsp;</td><td>Year</td>
<td><%=dateYearMonthDayDBObj.year%>
<input type='hidden' name='year' id='year' size ='10' value='<%=dateYearMon
    th%>' />
</td>

```

```

        thDayDBObj.year%>'/>
</td></tr>
<tr><td>In Time</td>
<%
    if( empDailyAttendanceDBObj.in_time != null )
    out.print("<td><input type='text' name='in_time' id='in_time' size ='10'
              value='"+empDailyAttendanceDBObj.in_time+"'/>(HH:MM) </td>");
    else
    out.print("<td align='left'><input type='text' name='in_time' id='in_time'
              size ='10' value=''/>(HH:MM) </td>");

%>
<td>Out Time</td>
<%
    if( empDailyAttendanceDBObj.out_time != null )
    out.print("<td><input type='text' name='out_time' id='out_time' size ='10'
              value='"+empDailyAttendanceDBObj.out_time+"'/>(HH:MM)</td>");
    else
    out.print("<td><input type='text' name='out_time' id='out_time' size ='10'
              value=''/>(HH:MM) </td>");

%>
</tr>
<tr><td>Remark</td>
<td colspan=3><input type='text' name='remark' id='remark' size = '85'
              value=''/></td></tr>
<tr><td colspan=4 align=center>
<input type='submit' name='submit' id='submit' size ='10' value='Submit Detail' />
<input type='hidden' name='action_submit' id='action_submit' size ='10' value=
              'emp_daily_att_dtl_submit' />
</td>
</tr>
</table>
<p>&nbsp;</p>
<hr width=100% color=#AAAAAA>
</td>
</tr>
<tr>
<td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr>
</table>
</body>
</html>

```

Figure 23.20 shows the employee_daily_attendance JSP page:

The screenshot shows a web browser window for 'www.peoplemanagementsolutions.com/Attendance'. The page title is 'People Management Solutions'. At the top, there's a navigation menu with links for Home, Employee, Recruitment, Time Management, and PayRoll. Below the menu, there's a header featuring three people's faces. The main content area contains an attendance form. The form fields are as follows:

Employee ID	1	Date	2010-07-14
Employee Name	Super Admin	Day	WED
Department	IT	Month	JUL
In Time	9:30 (HH:MM)	Year	2010
Out Time	19:30 (HH:MM)		
Remark	Leave	<input type="button" value="Submit Detail"/>	

Figure 23.20: Displaying the Attendance Form of an Employee

Let's now create the employee_daily_attendance_summary JSP page.

Creating the employee_daily_attendance_summary JSP Page

The employee_daily_attendance_summary JSP page shows the daily attendance records of all the employees of an organization. The details displayed in the employee_daily_attendance_summary JSP page include various fields, such as Employee Id, Employee Name, In Time, Out Time, and Remark.

Listing 23.41 shows the code for the employee_daily_attendance_summary JSP page (you can find the employee_daily_attendance_summary.jsp file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.41: Showing the Code of the employee_daily_attendance_summary.jsp File

```
<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.TimeManagement.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.util.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Attendance Summary...</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2"><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td><%@ include file="../jsp/people_default_menu.jsp" %></td>
</tr><tr>
<td width ="750" valign="top">
<p>&nbsp;</p>
<div align=center class=boldblack>Daily Attendance Summary</div>
<hr width=100% color="#AAAAAA">
<table border="0" width=100% >
<%
    String dbopr = "";
    dbopr = (String)session.getAttribute("dbopr");
%>
<tr>
<td bgcolor ='#AAAAAA' align='center' class=whitetext>Employee Id</td>
<td bgcolor ='#AAAAAA' align='center' class=whitetext>Employee Name</td>
<td bgcolor ='#AAAAAA' align='center' class=whitetext>In Time</td>
<td bgcolor ='#AAAAAA' align='center' class=whitetext>Out Time</td>
<td bgcolor ='#AAAAAA' align='center' class=whitetext>Remark</td>
<td bgcolor ='#AAAAAA' align='center' class=whitetext>Opr</td>
</tr>
<%
    ArrayList empDailyAttendanceList = new ArrayList();
    empDailyAttendanceList = (ArrayList)session.getAttribute("empDailyAttendanceList");
    if ( empDailyAttendanceList != null && empDailyAttendanceList.size() > 0 ){
        for ( int size = 1; size <= empDailyAttendanceList.size() ; size++ ){
            EmpDailyAttendanceDBObj  empDailyAttendanceDBObj = new EmpDailyAttendanceDBObj();
            empDailyAttendanceDBObj = (EmpDailyAttendanceDBObj)empDailyAttendanceList.get(size-1);
        }
    }
%>
<form name="form1" method="post">
<tr bgcolor ='#AAAAAA'>
<td align='center'><%=empDailyAttendanceDBObj.emp_id%></td>
<td align='center'><%=empDailyAttendanceDBObj.emp_name%> </td>
<td align='center'><%=empDailyAttendanceDBObj.in_time%></td>
<td align='center'><%=empDailyAttendanceDBObj.out_time%></td>
<td align='center'>
<%
    if(empDailyAttendanceDBObj.remark!=null)
        out.print(empDailyAttendanceDBObj.remark);
    else
        out.print("--");
%>
```

```

%>
<td align='center' bgcolor=#AAAAAA >
<a href='http://localhost:8080/people-mgmt/servlet/time_management?dbopr=edit&&
emp_id=<%=empDailyAttendanceDBObj.emp_id%>&&today_date=<%=empDailyAttendanceDBObj.
today_date%>' class=yellowlink>Edit</a></td >
</tr>
<%
}
}
else{
    out.println("Employee does not exist!!!");
}
%>
</table>
</tr>
</td>
</tr>
<td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr>
</table>
</body>
</html>

```

In Listing 23.41, the `ArrayList` class is used to fetch the data from the database and show the attendance of all the employees for the current day. The `employee_daily_attendance_summary` JSP page also contains a link, named `Edit`, to edit the attendance record.

Figure 23.21 shows the daily attendance record of all employees for the current date:

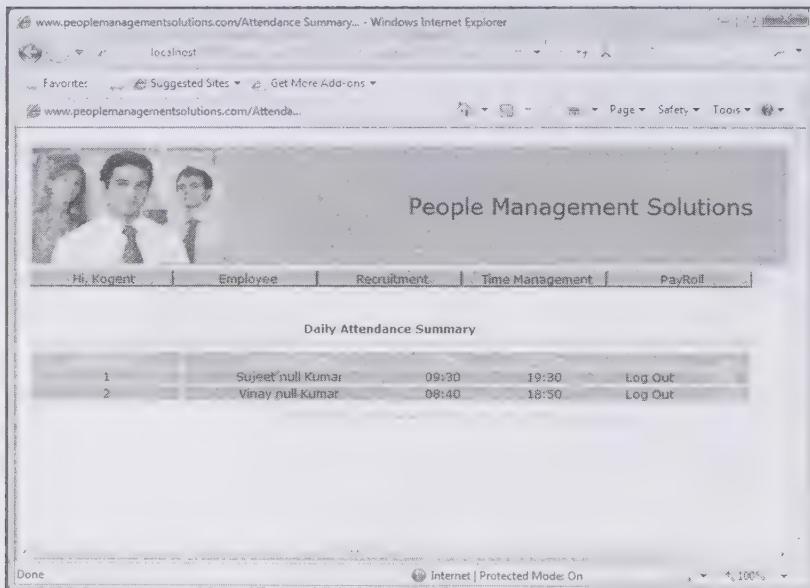


Figure 23.21: Displaying the employee_daily_attendance_summary JSP Page Containing Employees Attendance List

In this section, the Attendance Management module has been designed to handle attendance details of employees of an organization. You have also developed several Java classes as well as servlets used to make this module functional. In addition, this module contains two JSP pages, one for entering in and out time and another for showing daily attendance records of all employees.

The next section deals with the development of the Leave Management module.

Section



Developing the Leave Management Module

If you need an information on:

Creating the leave_management Servlet

See page:

1182

Designing JSP Views

1190

Every organization needs to keep a record of the leaves taken by its employees. The responsibility of granting and managing leaves comes in the purview of the HR department. The HR department can approve or reject the leave of an employee on the basis of reason for leave, projects assigned to the employee, and number of leaves available. In this section, let's develop the Leave Management module to facilitate the HR management in managing the leaves of the employees. This module provides an interface to fill the leave request details and submitting the details for approval. In addition, this module provides another interface, displaying the leave requests approved by the HR management in a summarized format.

The following Java source files are to be created in this module:

- leave_management.java
- LeaveRequest.java
- LeaveMgmtBeanMethods.java
- GenerateId.java

In addition to these Java classes, the following JSP files are needed:

- leave_request.jsp
- leave_request_edit.jsp
- leave_request_reject.jsp
- leave_request_list.jsp

Let's now create the leave_management servlet.

Creating the leave_management Servlet

The leave_management servlet is the controller servlet for this module. Any request for the interaction with the Leave Management module is handled by this servlet, which guides the user to the required JSP page after executing appropriate segments of code. This servlet class uses three other Java classes, i.e. LeaveRequest, GenerateId, and LeaveMgmtBeanMethods, which are discussed later in this section.

Listing 23.42 shows the code of the leave_management.java file (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src folder on CD):

Listing 23.42: Showing the Code of the leave_management.java File

```

import javax.servlet.*;
import javax.servlet.http.*;
import java.util.ArrayList;
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.annotation.*;
import javax.servlet.annotation.WebServlet;
import com.LeaveManagement.LeaveMgmtBeanMethods;
import com.LeaveManagement.LeaveRequest;
@WebServlet(name="leave_management", urlPatterns="/servlet/leave_management")
public class leave_management extends HttpServlet{
    String dbuser = "";
    String dbpswd = "";
    String dburl = "";
    /**Initialize global variables*/
    @Override
    public void init(ServletConfig config) throws ServletException{
        System.out.println("initializing controller servlet.");
        ServletContext context = config.getServletContext();
        dbuser = "scott";
        dbpswd = "tiger";
        dburl = "jdbc:oracle:thin:@192.168.1.123:1521:"+"XE";
        super.init(config);
    }
    /**Process the HTTP Get request*/
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {

```

```

        doPost(request, response);
    }
    /**Process the HTTP Post request*/
    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        HttpSession session = request.getSession();
        session.setAttribute("lErrorMsg",null);
        String target = "";
        String action = request.getParameter("action");
        String dbopr = "";
        dbopr = (String)request.getParameter("dbopr");
        session.setAttribute("dbopr",dbopr);
        if( (dbopr != null && dbopr.length() > 0) && (dbopr.equals("leave_request")))
        {
            target = "/jsp/leave_request.jsp";
        }
        else
        if( (dbopr != null && dbopr.length() > 0) && (dbopr.equals("leave_approve")))
        {
            action = "select_leave_request";
        }
        else
        if( (dbopr != null && dbopr.length() > 0) && (dbopr.equals("approve")) ){
            action = "leave_request_approve";
        }
        else
        if( (dbopr != null && dbopr.length() > 0) &&
        (dbopr.equals("approved_Leave")) ){
            action = "approved_leave_request";
        }
        String action_submit = request.getParameter("action_submit");
        System.out.println("action_submit=="+action_submit);
        if ( action_submit != null ){
            if ( request.getParameter("submit").equals("Submit") ){
                System.out.println("in the Submit");
            }
            if ( action_submit.equals("employee_leave_req_submit") ){
                System.out.println("in the employee_leave_req_submit ");
                action = "employee_leave_req_submit";
            }
        }
        else
        if ( request.getParameter("submit").equals("Approve") ){
            if ( action_submit.equals("employee_leave_req_edit_submit") ) {
                action = "employee_leave_req_edit_submit";
            }
        }
        if (action!=null){
            System.out.println("in the "+action);
            if (action.equals("leave_request_approve")){
                String reqId = "";
                reqId = (String)request.getParameter("req_id");
                String empId = "";
                empId = (String)request.getParameter("emp_id");
                LeaveMgmtBeanMethods leaveMgmtBeanMethods = new LeaveMgmtBean
                    Methods(dbuser,dbpswd,dburl);
                LeaveRequest leaveRequest = new LeaveRequest();
                leaveRequest = (LeaveRequest)leaveMgmtBeanMethods.getRecordBy
                    PrimaryKey(reqId,empId);
                session.setAttribute("leaveRequest",leaveRequest);
                target = "/jsp/leave_request_edit.jsp";
            }
        }
    }
}

```

```

if (action.equals("select_leave_request") || action.equals("
approved_leave_request")){
LeaveMgmtBeanMethods leaveMgmtBeanMethods = new LeaveMgmtBean
Methods(dbuser,dbpswd,dburl);
ArrayList leaveRequestList = new ArrayList();
String criteria = "";
if( action.equals("select_leave_request") )
    criteria = " where leave_status='Req' ";
else
    criteria = " where leave_status='Aprv' ";
LeaveRequestList = (ArrayList)leaveMgmtBeanMethods
    .selectLeaveRequestByCriteria(criteria);
session.setAttribute("leaveRequestList",leaveRequestList);
target = "/jsp/leave_request_list.jsp";
}
else
if(action.equals("employee_leave_req_submit")){
LeaveRequest popLeaveRequest = new LeaveRequest();
LeaveMgmtBeanMethods leaveMgmtBeanMethods = new LeaveMgmtBeanM
ethods(dbuser,dbpswd,dburl);
popLeaveRequest = (LeaveRequest)leaveMgmtBeanMethods
    .populateLeaveRequestFromReq(request);
popLeaveRequest.leave_status = "Req";
LeaveRequest leaveRequest = new LeaveRequest();
leaveRequest = (LeaveRequest)leaveMgmtBeanMethods.getRecordBy
    PrimaryKey(popLeaveRequest.req_id,popLeaveRequest.emp_id);
int rval = leaveMgmtBeanMethods.insertLeave
    Request(popLeaveRequest);
LeaveRequest sLeaveRequest = new LeaveRequest();
sLeaveRequest = (LeaveRequest)leaveMgmtBeanMethods.getRecord
    ByPrimaryKey(popLeaveRequest.req_id,popLeaveRequest.emp_id);
session.setAttribute("leaveRequest",sLeaveRequest);
if ( rval > 0 ){
    LeaveRequest leaveRequest1 = new LeaveRequest();
    leaveRequest1 = (LeaveRequest)leaveMgmtBeanMethods.getRecord
        ByPrimaryKey(popLeaveRequest.req_id,popLeaveRequest.emp_id);
    session.setAttribute("LeaveRequest",leaveRequest1);
    ArrayList leaveRequestList = new ArrayList();
    String criteria = "";
    criteria = " where leave_status='Req' ";
    LeaveRequestList = ( ArrayList)leaveMgmtBeanMethods.
        selectLeaveRequestByCriteria(criteria);
    session.setAttribute("leaveRequestList",leaveRequestList);
    target = "/jsp/leave_request_list.jsp";
}
}
else
if (action.equals("employee_leave_req_edit_submit")){
LeaveRequest popLeaveRequest = new LeaveRequest();
LeaveMgmtBeanMethods leaveMgmtBeanMethods = new LeaveMgmtBeanMet
hods(dbuser,dbpswd,dburl);
String ldbopr = (String)session.getAttribute("ldbopr");
System.out.println("ldbopr????????????????//"+ldbopr);
popLeaveRequest = (LeaveRequest)leaveMgmtBeanMethods.populate
    LeaveRequestFromReq(request);
if(ldbopr != null && ldbopr.equals("approve"))
    popLeaveRequest.leave_status = "Aprv";
else
    popLeaveRequest.leave_status = "Req";
System.out.println("ldbopr????????????//"+popLeaveRequest.leave_status);
int rval = leaveMgmtBeanMethods .updateLeaveRequestByPrimaryKey
    (popLeaveRequest);
if ( rval > 0 ){
    LeaveRequest LeaveRequest = new LeaveRequest();
    LeaveRequest = (LeaveRequest)leaveMgmtBeanMethods.getRecordByPrimary
}
}

```

```

        Key(popLeaveRequest.req_id,popLeaveRequest.emp_id);
        session.setAttribute("LeaveRequest",LeaveRequest);
        ArrayList leaveRequestList = new ArrayList();
        String criteria = "";
        criteria = " where leave_status='Aprv' ";
        leaveRequestList = (ArrayList)leaveMgmtBeanMethods.selectLeave
            RequestByCriteria(criteria);
        session.setAttribute("leaveRequestList",leaveRequestList);
        session.setAttribute("dbopr","approved_leave");
        target = "/jsp/leave_request_list.jsp";
    }
    else{
        target = "/jsp/employee_edit.jsp";
    }
}
/* forwarding the request/response to the targeted view */
RequestDispatcher requestDispatcher = getServletContext().getRequest
    Dispatcher(target);
requestDispatcher.forward(request, response);
} // doPost closed
}// class closed

```

In Listing 23.42, the servlet uses objects of the `LeaveRequest` and `LeaveMgmtBeanMethods` classes to perform the operations according to the parameter passed and the values of the `action` and `target` variables. The methods defined in the `LeaveMgmtBeanMethods` class perform the actual database interaction on behalf of the servlet and return the object of the `LeaveRequest` class. This object is used as a carrier of data from one method to another.

Creating the LeaveRequest Class

The member attributes of the `LeaveRequest` class are set with the information associated with a leave request, including whether it has been approved or not. This information is updated in the `LEAVE_REQUEST` table. The `LeaveRequest` class has member variables matching with the table attributes. In other words, these member variables are being used as form beans without any setter and getter methods, and data is being set and fetched from the corresponding object manually.

Listing 23.43 shows the code for the `LeaveRequest.java` file (you can find this file in the `PeopleMgmt\people-mgmt\WEB-INF\src\com\LeaveManagement` folder on CD):

Listing 23.43: Showing the Code of the `LeaveRequest.java` File

```

package com.LeaveManagement;
public class LeaveRequest {
    public String req_id;
    public String emp_id;
    public String emp_name;
    public String today_date;
    public String level_id;
    public String dept_id;
    public String from_date;
    public String to_date;
    public int days;
    public String reason;
    public String leave_type;
    public String activity_1;
    public String activity_2;
    public String activity_3;
    public String person_1;
    public String person_2;
    public String person_3;
    public String detail_1;
    public String detail_2;
    public String detail_3;
    public String leave_status;
    public String remark;
}

```

```
    public String address;
}
```

In Listing 23.43, the attributes of the LeaveRequest class are same as those of the leave request form. The object of this class is used to store the data related to the leaves of employees in the database.

Let's now create the LeaveMgmtBeanMethods class in the next subsection.

Creating the LeaveMgmtBeanMethods Class

The LeaveMgmtBeanMethods class defines various methods, such as updateLeaveRequestByPrimaryKey(), insertLeaveRequest(), and populateLeaveRequestFromReq(), which are used to insert, retrieve, and update data in the LEAVE_REQUEST table. All these methods are used to execute one or another SQL query to interact with the database and perform the particular operation.

Listing 23.44 shows the code of the LeaveMgmtBeanMethods.java file (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src\com\LeaveManagement folder on CD):

Listing 23.44: Showing the Code of the LeaveMgmtBeanMethods.java File

```
package com.LeaveManagement;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.util.ArrayList;
import com.LeaveManagement.LeaveRequest;
public class LeaveMgmtBeanMethods{
    public String DBUser;
    public String DBPswd;
    public String DBUrl ;
    public LeaveMgmtBeanMethods(){ }
    public LeaveMgmtBeanMethods(String inDBUser, String inDBPswd, String inDBUrl ){
        DBUser = inDBUser ;
        DBPswd = inDBPswd;
        DBUrl = inDBUrl;
    }
    public void initializeLeaveRequest(LeaveRequest inLeaveRequest ){
        inLeaveRequest.req_id="";
        inLeaveRequest.emp_id="";
        inLeaveRequest.emp_name="";
        inLeaveRequest.today_date="";
        inLeaveRequest.level_id="";
        inLeaveRequest.dept_id="";
        inLeaveRequest.from_date="";
        inLeaveRequest.to_date="";
        inLeaveRequest.days= 0;
        inLeaveRequest.reason="";
        inLeaveRequest.leave_type="";
        inLeaveRequest.activity_1="";
        inLeaveRequest.activity_2="";
        inLeaveRequest.activity_3="";
        inLeaveRequest.person_1="";
        inLeaveRequest.person_2="";
        inLeaveRequest.person_3="";
        inLeaveRequest.detail_1="";
        inLeaveRequest.detail_2="";
        inLeaveRequest.detail_3="";
        inLeaveRequest.leave_status="";
        inLeaveRequest.remark="";
        inLeaveRequest.address="";
    }
    public LeaveRequest getRecordByPrimaryKey(String inReqId, String inEmpId){
        LeaveRequest leaveRequest = new LeaveRequest();
        java.sql.Date date;
        try{
```

```

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
Statement stmt = conn.createStatement();
String lsqlString ="select * from LEAVE_REQUEST ";
lsqlString +="where req_id='"+inReqId+"' ";
lsqlString +="and emp_id='"+inEmpId+"' ";
ResultSet rs = null;
rs = stmt.executeQuery(lsqlString);
if( rs.next()){
    System.out.println("fffff=="+rs.getString("emp_id"));
    leaveRequest.req_id = (String)rs.getString("req_id");
    leaveRequest.emp_id = (String)rs.getString("emp_id");
    leaveRequest.emp_name = (String)rs.getString("emp_name");
    date=rs.getDate("today_date");
    leaveRequest.today_date = date.toString();
    date=rs.getDate("from_date");
    leaveRequest.from_date = date.toString();
    date=rs.getDate("to_date");
    leaveRequest.to_date = date.toString();
    leaveRequest.level_id = (String)rs.getString("level_id");
    leaveRequest.dept_id = (String)rs.getString("dept_id");
    leaveRequest.days = rs.getInt("days");
    leaveRequest.reason = (String)rs.getString("reason");
    leaveRequest.leave_type = (String)rs.getString("leave_type");
    leaveRequest.activity_1 = (String)rs.getString("activity_1");
    leaveRequest.activity_2 = (String)rs.getString("activity_2");
    leaveRequest.activity_3 = (String)rs.getString("activity_3");
    leaveRequest.person_1 = (String)rs.getString("person_1");
    leaveRequest.person_2 = (String)rs.getString("person_2");
    leaveRequest.person_3 = (String)rs.getString("person_3");
    leaveRequest.detail_1 = (String)rs.getString("detail_1");
    leaveRequest.detail_2 = (String)rs.getString("detail_2");
    leaveRequest.detail_3 = (String)rs.getString("detail_3");
    leaveRequest.leave_status = (String)rs.getString("leave_status");
    leaveRequest.address = (String)rs.getString("address");
    leaveRequest.remark = (String)rs.getString("remark");
    System.out.println("fffff=="+rs.getString("emp_id"));
}
else{
    initializeLeaveRequest(leaveRequest);
}
System.out.println("fffff===="+leaveRequest.emp_id);
}
catch(SQLException ex){
    ex.printStackTrace();
}
return leaveRequest;
}

public ArrayList selectLeaveRequestByCriteria(String inCriteria) {
ArrayList LeaveRequestList = new ArrayList();
java.sql.Date date;
try{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
    Statement stmt = conn.createStatement();
    String lsqlString = "select * from LEAVE_REQUEST ";
    if( inCriteria != null && inCriteria.length() > 0 ){
        lsqlString +=" "+inCriteria+" ";
    }
    lsqlString +=" order by req_id" ;
    System.out.println("Criteria==== "+inCriteria+" and
query="+lsqlString);
    ResultSet rs = null;
    rs = stmt.executeQuery(lsqlString);
    while( rs.next()){

}

```

```

LeaveRequest leaveRequest = new LeaveRequest();
leaveRequest.req_id = (String)rs.getString("req_id");
leaveRequest.emp_id = (String)rs.getString("emp_id");
leaveRequest.emp_name = (String)rs.getString("emp_name");
date=rs.getDate("today_date");
leaveRequest.today_date = date.toString();
date=rs.getDate("from_date");
leaveRequest.from_date = date.toString();
date=rs.getDate("to_date");
leaveRequest.to_date = date.toString();
leaveRequest.level_id = (String)rs.getString("level_id");
leaveRequest.dept_id = (String)rs.getString("dept_id");
leaveRequest.days = rs.getInt("days");
leaveRequest.reason = (String)rs.getString("reason");
leaveRequest.leave_type = (String)rs.getString("leave_type");
leaveRequest.activity_1 = (String)rs.getString("activity_1");
leaveRequest.activity_2 = (String)rs.getString("activity_2");
leaveRequest.activity_3 = (String)rs.getString("activity_3");
leaveRequest.person_1 = (String)rs.getString("person_1");
leaveRequest.person_2 = (String)rs.getString("person_2");
leaveRequest.person_3 = (String)rs.getString("person_3");
leaveRequest.detail_1 = (String)rs.getString("detail_1");
leaveRequest.detail_2 = (String)rs.getString("detail_2");
leaveRequest.detail_3 = (String)rs.getString("detail_3");
leaveRequest.leave_status = (String)rs.getString("leave_status");
leaveRequest.address = (String)rs.getString("address");
leaveRequest.remark = (String)rs.getString("remark");
LeaveRequestList.add(leaveRequest);
}
}
catch(SQLException ex){
    ex.printStackTrace();
}
return LeaveRequestList;
}

public int updateLeaveRequestByPrimaryKey(LeaveRequest inLeaveRequest){
int recupd = 0;
String lQuery = "";
lQuery = lQuery +"update LEAVE_REQUEST set emp_name='"+inLeaveRequest.emp_name+"' ";
lQuery = lQuery +", today_date=to_date('" +inLeaveRequest
        .today_date+"','yyyy-mm-dd') ";
lQuery = lQuery +", level_id='"+inLeaveRequest.level_id+"' ";
lQuery = lQuery +", dept_id='"+inLeaveRequest.dept_id+"' ";
lQuery = lQuery +", from_date=to_date('" +inLeaveRequest.from_date+"', 'yyyy-mm-dd') ";
lQuery = lQuery +", to_date=to_date('" +inLeaveRequest.to_date+"', 'yyyy-mm-dd') ";
lQuery = lQuery +", days='"+inLeaveRequest.days+"' ";
lQuery = lQuery +", reason='"+inLeaveRequest.reason+"' ";
lQuery = lQuery +", leave_type='"+inLeaveRequest.leave_type+"' ";
lQuery = lQuery +", activity_1='"+inLeaveRequest.activity_1+"' ";
lQuery = lQuery +", activity_2='"+inLeaveRequest.activity_2+"' ";
lQuery = lQuery +", activity_3='"+inLeaveRequest.activity_3+"' ";
lQuery = lQuery +", person_1='"+inLeaveRequest.person_1+"' ";
lQuery = lQuery +", person_2='"+inLeaveRequest.person_2+"' ";
lQuery = lQuery +", person_3='"+inLeaveRequest.person_3+"' ";
lQuery = lQuery +", detail_1='"+inLeaveRequest.detail_1+"' ";
lQuery = lQuery +", detail_2='"+inLeaveRequest.detail_2+"' ";
lQuery = lQuery +", detail_3='"+inLeaveRequest.detail_3+"' ";
lQuery = lQuery +", address='"+inLeaveRequest.address+"' ";
lQuery = lQuery +", remark='"+inLeaveRequest.remark+"' ";
lQuery = lQuery +", leave_status='"+inLeaveRequest.leave_status+"' ";
lQuery = lQuery + "where req_id='"+inLeaveRequest.req_id+"' ";
lQuery = lQuery + "and emp_id='"+inLeaveRequest.emp_id+"' ";
System.out.println("lsqlString==:"+lQuery);
try{
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
}

```

```

Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
Statement stmt = conn.createStatement();
recupd = stmt.executeUpdate(lQuery);
}
catch(SQLException ex){
    ex.printStackTrace();
}
return recupd;
}

public LeaveRequest populateLeaveRequestFromReq(HttpServletRequest inReq){
LeaveRequest leaveRequest = new LeaveRequest();
leaveRequest.req_id = (String)inReq.getParameter("req_id");
leaveRequest.emp_id = (String)inReq.getParameter("emp_id");
leaveRequest.emp_name = (String)inReq.getParameter("emp_name");
leaveRequest.today_date = (String)inReq.getParameter("today_date");
leaveRequest.from_date = (String)inReq.getParameter("from_date");
leaveRequest.to_date = (String)inReq.getParameter("to_date");
leaveRequest.level_id = (String)inReq.getParameter("level_id");
leaveRequest.dept_id = (String)inReq.getParameter("dept_id");
leaveRequest.days = Integer.parseInt((String)inReq.getParameter("days"));
leaveRequest.reason = (String)inReq.getParameter("reason");
leaveRequest.leave_type = (String)inReq.getParameter("leave_type");
leaveRequest.activity_1 = (String)inReq.getParameter("activity_1");
leaveRequest.activity_2 = (String)inReq.getParameter("activity_2");
leaveRequest.activity_3 = (String)inReq.getParameter("activity_3");
leaveRequest.person_1 = (String)inReq.getParameter("person_1");
leaveRequest.person_2 = (String)inReq.getParameter("person_2");
leaveRequest.person_3 = (String)inReq.getParameter("person_3");
leaveRequest.detail_1 = (String)inReq.getParameter("detail_1");
leaveRequest.detail_2 = (String)inReq.getParameter("detail_2");
leaveRequest.detail_3 = (String)inReq.getParameter("detail_3");
leaveRequest.leave_status = (String)inReq.getParameter("leave_status");
leaveRequest.address = (String)inReq.getParameter("address");
leaveRequest.remark = (String)inReq.getParameter("remark");
return leaveRequest;
}

public int insertLeaveRequest(LeaveRequest inLeaveRequest){
int recupd = 0;
String lQuery = "";
lQuery = lQuery +"insert into LEAVE_REQUEST values ( ";
lQuery = lQuery +"'"+inLeaveRequest.req_id+"'";
lQuery = lQuery +", '"+inLeaveRequest.emp_id+"'";
lQuery = lQuery +", '"+inLeaveRequest.emp_name+"'";
lQuery = lQuery +", to_date('"+inLeaveRequest.today_date+"','yyyy-mm-dd') ";
lQuery = lQuery +", '"+inLeaveRequest.level_id+"'";
lQuery = lQuery +", '"+inLeaveRequest.dept_id+"'";
lQuery = lQuery +", to_date('"+inLeaveRequest.from_date+"','yyyy-mm-dd') ";
lQuery = lQuery +", to_date('"+inLeaveRequest.to_date+"','yyyy-mm-dd') ";
lQuery = lQuery +", '"+inLeaveRequest.days+"'";
lQuery = lQuery +", '"+inLeaveRequest.reason+"'";
lQuery = lQuery +", '"+inLeaveRequest.leave_type+"'";
lQuery = lQuery +", '"+inLeaveRequest.activity_1+"'";
lQuery = lQuery +", '"+inLeaveRequest.activity_2+"'";
lQuery = lQuery +", '"+inLeaveRequest.activity_3+"'";
lQuery = lQuery +", '"+inLeaveRequest.person_1+"'";
lQuery = lQuery +", '"+inLeaveRequest.person_2+"'";
lQuery = lQuery +", '"+inLeaveRequest.person_3+"'";
lQuery = lQuery +", '"+inLeaveRequest.detail_1+"'";
lQuery = lQuery +", '"+inLeaveRequest.detail_2+"'";
lQuery = lQuery +", '"+inLeaveRequest.address+"'";
lQuery = lQuery +", '"+inLeaveRequest.remark+"'";
lQuery = lQuery +", '"+inLeaveRequest.leave_status+"'";
lQuery = lQuery +")";
System.out.println("lsqlString==:"+lQuery);
try {
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
    Statement stmt = conn.createStatement();
    recupd = stmt.executeUpdate(lQuery);
}

```

```
        }
        catch(SQLException ex){
            ex.printStackTrace();
        }
        return recupd;
    }
}
```

In Listing 23.44, the `insertLeaveRequest()` method inserts a leave request in the `LEAVE_REQUEST` table; whereas, the `updateLeaveRequestByPrimaryKey()` method updates the status of the leave request, such as approved or rejected, in the `LEAVE_REQUEST` table.

Creating the GenerateId Class

The `GenerateId` class is developed for the auto-generation of the value of the `req_id` field in the database. Listing 23.45 shows the code for the `GenerateId.java` file (you can find this file in the `PeopleMgmt\people-mgmt\WEB-INF\src\com\LeaveManagement` folder on CD):

Listing 23.45: Showing the Code of the `GenerateId.java` File

```
package com.LeaveManagement;
import java.sql.*;
import javax.sql.*;
public class GenerateId
{
    public int generateRequestId(){
        int req_id=0;
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection conn= DriverManager.getConnection("jdbc:oracle:thin:
                @192.168.1.123:1521;"+"XE","scott","tiger");
            Statement stmt = conn.createStatement();
            String query="select max(req_id) as req_id from LEAVE_REQUEST ";
            ResultSet rs = null;
            rs = stmt.executeQuery(query);
            if(rs.next()){
                String id = rs.getString("req_id");
                req_id=Integer.parseInt(id);
            }
            req_id = req_id + 1;
        }
        catch(SQLException ex){
            ex.printStackTrace();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
        return req_id;
    }
}
```

In Listing 23.45, the `req_id` field is generated automatically, as the select query returns the maximum value of the `req_id` column. This maximum value retrieved from the database is incremented by 1 to allocate a unique id for a new leave request.

This ends up the discussion on the creation of the Java files. You should compile all the Java files and save the generated .class files in their respective folders. You do not need to specify servlet-mapping in the `web.xml` file as the `@WebServlet` annotation has been used to map a servlet to a url-pattern.

Let's now design the JSP pages for this module.

Designing JSP Views

After writing code for servlet and other Java classes, let's design the required JSP pages for user interface. There are three JSP pages designed for this module, i.e. `leave_request`, `leave_request_edit`, and `leave_request_list`. The first JSP page, `leave_request`, provides a form for a leave request; and the second JSP page, `leave_request_edit`, is used to set the status of the leave, i.e. approved or not. The third JSP page, `leave_request_list`, lists all the leave requests of all the employees of the organization.

Creating the leave_request JSP Page

The leave_request JSP page provides a form containing the fields required to apply for a leave. The form consists of data entry fields that are categorized into various sections, such as employee details, leave details, and activity details. These details include employee id, name, designation, department, the number of days worked, activities the employee is currently responsible for, and the name of the person who will be responsible for the respective activities in the absence of this employee.

Listing 23.46 shows the code for the leave_request.jsp file (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.46: Showing the Code of the leave_request JSP Page

```
<%@ page language="java" import="java.util.*, java.text.SimpleDateFormat" %>
<%@ page session="true" %>
<%@ page import="com.LeaveManagement.*" %>
<% GenerateId gen=new GenerateId();
    int req_id=gen.generateRequestId();%>
</html>
<head>
<title>www.peoplemanagementsolutions.com/Leave Request</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
<script language="javascript">
    function validateForm() {
        var myName=document.form1.emp_id.value;
        if (myName == "") {
            alert ("Employee Id field cannot be blank");
            document.form1.emp_id.focus();
            return false;
        }
        myName=document.form1.emp_name.value;
        if (myName == "") {
            alert ("Employee Name cannot be blank");
            document.form1.emp_name.focus();
            return false;
        }
        myName=document.form1.from_date.value;
        if (myName == "") {
            alert ("Provide the starting Date of Leave");
            document.form1.from_date.focus();
            return false;
        }
        myName=document.form1.to_date.value;
        if (myName == "") {
            alert ("Provide the date to which Leave applied");
            document.form1.to_date.focus();
            return false;
        }
        myName=document.form1.reason.value;
        if (myName == "") {
            alert ("Provide the Valid reason for Leave");
            document.form1.reason.focus();
            return false;
        }
        myName=document.form1.address.value;
        if (myName == "") {
            alert ("Address Field cannot be blank");
            document.form1.address.focus();
            return false;
        }
        form1.submit();
    }
</script>
</head>
<body>
```

```


|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |  |  |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |  |  |                                                                                                                                                                                                                                                |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |  |  |                                                                                                                                                                                                                                                                                                                                                               |  |  |  |                                                                                                       |  |  |  |                                                                                            |  |  |  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|--|--|--|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|--|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|--|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|--|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|--|-------------------------------------------------------------------------------------------------------|--|--|--|--------------------------------------------------------------------------------------------|--|--|--|
| <%@ include file="../jsp/people_header.jsp" %></td>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |  |  |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |  |  |                                                                                                                                                                                                                                                |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |  |  |                                                                                                                                                                                                                                                                                                                                                               |  |  |  |                                                                                                       |  |  |  |                                                                                            |  |  |  |
| <%@ include file="../jsp/people_default_menu.jsp" %></td>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |  |  |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |  |  |                                                                                                                                                                                                                                                |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |  |  |                                                                                                                                                                                                                                                                                                                                                               |  |  |  |                                                                                                       |  |  |  |                                                                                            |  |  |  |
| <form name="form1" method="post"> <input type='hidden' name='req_id' id='req_id' size ='10' value='<%= req_id %>' /> <table border="0" align="top" width=600> <tr> <td <="" align="center&gt;Leave" application&lt;="" bgcolor="#AAAAAA" class="whitetext" colspan="4" height="20" td="" td&gt;=""> </td></tr> <tr> <td colspan="4"> Request ID&lt;/td&gt; &lt;td align='left'&gt;&lt;input type='text' disabled='disabled' name='dup_req_id' id='dup_req_id' size ='10' value='&lt;%= req_id %&gt;' /&gt; &lt;/td&gt; &lt;td&gt;Date(YYYY-MM-DD)&lt;/td&gt;&lt;input type='hidden' name='today_date' id='today_date' size ='10' value='&lt;%= new SimpleDateFormat("yyyy-MM-dd").format(new Date()) %&gt;' /&gt; &lt;td align='left'&gt;&lt;input type='text' name='dup_today_date' disabled='disabled' id='dup_today_date' size ='10' value='&lt;%= new SimpleDateFormat("yyyy-MM-dd").format(new Date()) %&gt;' /&gt; &lt;/td&gt; </td> </tr> <tr> <td colspan="4"> Employee Id&lt;sup&gt;*&lt;/sup&gt;&lt;/td&gt; &lt;td align='left'&gt;&lt;input type='text' name='emp_id' id='emp_id' size ='10' value=' /&gt; &lt;/td&gt; &lt;td&gt;Name&lt;sup&gt;*&lt;/sup&gt;&lt;/td&gt; &lt;td align='left'&gt;&lt;input type='text' name='emp_name' id='emp_name' size ='20' value=' /&gt; &lt;/td&gt; </td> </tr> <tr> <td colspan="4"> Designation&lt;/td&gt; &lt;td align='left'&gt;&lt;select name="level_id" id="level_id"&gt; &lt;option value='select Designation' selected&gt; Select Designation &lt;option value='CW'&gt; Content Writer &lt;option value='TS'&gt; Tester &lt;option value='HR'&gt; Human Resource Manager &lt;option value='AC'&gt; Accountant &lt;option value='AM'&gt; Admin Manager &lt;option value='EM'&gt; Event Manager &lt;option value='TL'&gt; Team Leader &lt;option value='PM'&gt; Project Manager &lt;option value='TR'&gt; Trainer &lt;/select&gt;&lt;/td&gt; </td> </tr> <tr> <td colspan="4"> Department&lt;/td&gt; &lt;td align='left'&gt;&lt;select id='dept_id' name='dept_id'&gt; &lt;option value='select Department' selected&gt; Select Department &lt;option value='CS'&gt; Content Solutions &lt;option value='TS'&gt; Testing &lt;option value='HR'&gt; Human Resource &lt;option value='AC'&gt; Accounts &lt;option value='AD'&gt; Administration &lt;option value='EM'&gt; Event Management &lt;/select&gt; &lt;/td&gt; </td> </tr> <tr> <td colspan="4"> &lt;td colspan='4' height=20 height=20 bgcolor ='#AAAAAA' class=whitetext align=center&gt;Leave Detail&lt;/td&gt; </td> </tr> <tr> <td colspan="4"> &lt;td&gt;From(YYYY-MM-DD)&lt;sup&gt;*&lt;/sup&gt;&lt;/td&gt;&amp;ampnbsp&amp;ampnbsp; &lt;td&gt;To(YYYY-MM-DD)&lt;sup&gt;*&lt;/sup&gt;&lt;/td&gt; </td> </tr> |  |  |  |  |  | Request ID</td> <td align='left'><input type='text' disabled='disabled' name='dup_req_id' id='dup_req_id' size ='10' value='<%= req_id %>' /> </td> <td>Date(YYYY-MM-DD)</td><input type='hidden' name='today_date' id='today_date' size ='10' value='<%= new SimpleDateFormat("yyyy-MM-dd").format(new Date()) %>' /> <td align='left'><input type='text' name='dup_today_date' disabled='disabled' id='dup_today_date' size ='10' value='<%= new SimpleDateFormat("yyyy-MM-dd").format(new Date()) %>' /> </td> |  |  |  | Employee Id<sup>*</sup></td> <td align='left'><input type='text' name='emp_id' id='emp_id' size ='10' value=' /> </td> <td>Name<sup>*</sup></td> <td align='left'><input type='text' name='emp_name' id='emp_name' size ='20' value=' /> </td> |  |  |  | Designation</td> <td align='left'><select name="level_id" id="level_id"> <option value='select Designation' selected> Select Designation <option value='CW'> Content Writer <option value='TS'> Tester <option value='HR'> Human Resource Manager <option value='AC'> Accountant <option value='AM'> Admin Manager <option value='EM'> Event Manager <option value='TL'> Team Leader <option value='PM'> Project Manager <option value='TR'> Trainer </select></td> |  |  |  | Department</td> <td align='left'><select id='dept_id' name='dept_id'> <option value='select Department' selected> Select Department <option value='CS'> Content Solutions <option value='TS'> Testing <option value='HR'> Human Resource <option value='AC'> Accounts <option value='AD'> Administration <option value='EM'> Event Management </select> </td> |  |  |  | <td colspan='4' height=20 height=20 bgcolor ='#AAAAAA' class=whitetext align=center>Leave Detail</td> |  |  |  | <td>From(YYYY-MM-DD)<sup>*</sup></td>&ampnbsp&ampnbsp; <td>To(YYYY-MM-DD)<sup>*</sup></td> |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |  |  |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |  |  |                                                                                                                                                                                                                                                |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |  |  |                                                                                                                                                                                                                                                                                                                                                               |  |  |  |                                                                                                       |  |  |  |                                                                                            |  |  |  |
| Request ID</td> <td align='left'><input type='text' disabled='disabled' name='dup_req_id' id='dup_req_id' size ='10' value='<%= req_id %>' /> </td> <td>Date(YYYY-MM-DD)</td><input type='hidden' name='today_date' id='today_date' size ='10' value='<%= new SimpleDateFormat("yyyy-MM-dd").format(new Date()) %>' /> <td align='left'><input type='text' name='dup_today_date' disabled='disabled' id='dup_today_date' size ='10' value='<%= new SimpleDateFormat("yyyy-MM-dd").format(new Date()) %>' /> </td>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |  |  |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |  |  |                                                                                                                                                                                                                                                |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |  |  |                                                                                                                                                                                                                                                                                                                                                               |  |  |  |                                                                                                       |  |  |  |                                                                                            |  |  |  |
| Employee Id<sup>*</sup></td> <td align='left'><input type='text' name='emp_id' id='emp_id' size ='10' value=' /> </td> <td>Name<sup>*</sup></td> <td align='left'><input type='text' name='emp_name' id='emp_name' size ='20' value=' /> </td>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |  |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |  |  |                                                                                                                                                                                                                                                |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |  |  |                                                                                                                                                                                                                                                                                                                                                               |  |  |  |                                                                                                       |  |  |  |                                                                                            |  |  |  |
| Designation</td> <td align='left'><select name="level_id" id="level_id"> <option value='select Designation' selected> Select Designation <option value='CW'> Content Writer <option value='TS'> Tester <option value='HR'> Human Resource Manager <option value='AC'> Accountant <option value='AM'> Admin Manager <option value='EM'> Event Manager <option value='TL'> Team Leader <option value='PM'> Project Manager <option value='TR'> Trainer </select></td>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |  |  |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |  |  |                                                                                                                                                                                                                                                |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |  |  |                                                                                                                                                                                                                                                                                                                                                               |  |  |  |                                                                                                       |  |  |  |                                                                                            |  |  |  |
| Department</td> <td align='left'><select id='dept_id' name='dept_id'> <option value='select Department' selected> Select Department <option value='CS'> Content Solutions <option value='TS'> Testing <option value='HR'> Human Resource <option value='AC'> Accounts <option value='AD'> Administration <option value='EM'> Event Management </select> </td>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |  |  |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |  |  |                                                                                                                                                                                                                                                |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |  |  |                                                                                                                                                                                                                                                                                                                                                               |  |  |  |                                                                                                       |  |  |  |                                                                                            |  |  |  |
| <td colspan='4' height=20 height=20 bgcolor ='#AAAAAA' class=whitetext align=center>Leave Detail</td>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |  |  |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |  |  |                                                                                                                                                                                                                                                |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |  |  |                                                                                                                                                                                                                                                                                                                                                               |  |  |  |                                                                                                       |  |  |  |                                                                                            |  |  |  |
| <td>From(YYYY-MM-DD)<sup>*</sup></td>&ampnbsp&ampnbsp; <td>To(YYYY-MM-DD)<sup>*</sup></td>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |  |  |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |  |  |                                                                                                                                                                                                                                                |  |  |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |  |  |                                                                                                                                                                                                                                                                                                                                                               |  |  |  |                                                                                                       |  |  |  |                                                                                            |  |  |  |


```

```

<td>Days</td>
<td>&nbsp;</td>
</tr>
<tr>
<td align='left'><input type='text' name='from_date' id='from_date' size ='10' value=''/> </td>
<td align='left'><input type='text' name='to_date' id='to_date' size ='10' value=''/> </td>
<td align='left'><input type='text' name='days' id='days' size ='5' value=''/> </td>
<td>&nbsp;</td>
</tr>
<tr>
<td colspan=2>Reason For Leave(100 letters only)<sup>*</sup></td>
<td align='left' colspan=2><textarea name='reason' id='reason' col='80' rows='4' value=''/></textarea></td>
</tr>
<tr>
<td colspan=2>Type Of Leave Applied For</td>
<td align='left' colspan=2><SELECT name='leave_type'>
<OPTION VALUE=></OPTION>
<OPTION VALUE=CL>CL</OPTION>
<OPTION VALUE=SL>SL</OPTION></SELECT>
</td>
</tr>
<tr>
<td colspan='4' height=20 bgcolor ='#AAAAAA' class=whitetext align=center>Major Activity to Be Handled During The Leave Applied</td>
</td>
</tr>
<tr>
<td align=right>S.No.</td>
<td>Activity Details</td>
<td>Name Of the Person Responsible</td>
<td>Details</td>
</tr>
<tr>
<td align=right>1.</td>
<td align='left'><input type='text' name='activity_1' id='activity_1' size ='20' value=''/> </td>
<td align='left'><input type='text' name='person_1' id='person_1' size ='20' value=''/> </td>
<td align='left'><input type='text' name='detail_1' id='detail_1' size ='20' value=''/> </td>
</tr><tr>
<td align=right>2.</td>
<td align='left'><input type='text' name='activity_2' id='activity_2' size ='20' value=''/> </td>
<td align='left'><input type='text' name='person_2' id='person_2' size ='20' value=''/> </td>
<td align='left'><input type='text' name='detail_2' id='detail_2' size ='20' value=''/> </td>
</tr>
<tr>
<td align=right>3.</td>
<td align='left'><input type='text' name='activity_3' id='activity_3' size ='20' value=''/> </td>
<td align='left'><input type='text' name='person_3' id='person_3' size ='20' value=''/> </td>
<td align='left'><input type='text' name='detail_3' id='detail_3' size ='20' value=''/> </td>
</tr>
<tr>
<td colspan='4' height=20 bgcolor ='#AAAAAA' class=whitetext align=center>Contact Details of the applicant during the leave period</td>

```

```

</tr>
<tr>
<td>Address<sup>*</sup></td>
<td align='left' colspan='2'><input type='text' name='address' id='address' size ='20' value=''/></td>
<td></td>
</tr>
<tr>
<td>Remark</td>
<td align='left' colspan='2'><input type='text' name='remark' id='remark' size ='20' value=''/> </td>
<td></td>
</tr>
<tr><td> All the ( <sup>*</sup>) marked are mandatory</td></tr><tr>
<td align='center' colspan='4'>
<input type='submit' name='submit' id='submit' size ='10' value='Submit' onclick="return validateForm()"/>
<input type='hidden' name='action_submit' id='action_submit' size ='10' value='employee_leave_req_submit' />
</td>
</tr>
</table>
</td>
</tr>
<tr>
<td colspan="2"><%@include file="..../jsp/people_footer.jsp"%></td>
</tr>
</table>
</body>
</html>

```

In Listing 23.46, the `req_id` field is disabled because it is generated automatically with the help of the `GenerateId` class. The validations are imposed on some fields, such as name, designation, and department, as these fields cannot be left blank. When you click the `Leave Apply` link under the `Leave Management` section, the `leave_management` servlet redirects you to the `leave_request` JSP page, showing various fields to be filled to apply for a leave.

Figure 23.22 shows the leave request form:

The screenshot shows a web browser displaying a JSP page titled "Leave Request". The URL in the address bar is "http://www.peoplemanagementsolutions.com/leave_Request". The page has a header with navigation links: Home, Employee, Recruitment, Time Management, and PayRoll. Below the header, there are several input fields and dropdown menus. The "Employee Id" field contains "1". The "Name" field contains "Sheetal Kunar". The "Designation" field contains "Content Writer" and the "Department" dropdown menu shows "Contact Solutions". There are two date pickers: "From Date" set to "2010-07-14" and "To Date" set to "2010-07-15". A text area labeled "Reason For Leave(100 letters only)" is present. A dropdown menu "Type Of Leave Applied For" is shown with one option selected. At the bottom, there is a "Done" button and a status bar indicating "Internet Explorer Protected Mode On" and "100%".

Figure 23.22: Displaying the `leave_request` JSP Page

Creating the leave_request_edit JSP Page

The leave_request_edit JSP page is similar to the leave_request JSP page, except that all the fields provided in the leave_request_edit JSP page are already filled with the leave details. The fields can be updated for new values and submitted to the servlet for performing the update operations in the table.

Listing 23.47 shows the code of the leave_request_edit.jsp file (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.47: Showing the Code of the leave_request_edit.jsp File

```
<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.LeaveManagement.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Leave Request</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2"><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td width="900"><%@ include file="../jsp/people_default_menu.jsp" %></td>
</tr><tr>
<td width = "730" valign="top" align = "center">
<table border="0" align="top" width=600 >
<%
    LeaveRequest leaveRequest = new LeaveRequest();
    leaveRequest = (LeaveRequest)session.getAttribute("leaveRequest");
%>
<form name="form1" method="post">
<tr>
<td colspan='4' height=20 height=20 bgcolor ='#AAAAAA' class=whitetext
align=center>Leave Application</td>
</tr>
<tr>
<td >Request ID</td>
<td align='left'><input type='text' name='req_id' id='req_id' size ='10'
value='<%=leaveRequest.req_id%>' /> </td>
<td >Date(YYYY-MM-DD)</td>
<td align='left'><input type='text' name='today_date' id='today_date' size ='10'
value='<%=leaveRequest.today_date%>' /> </td>
</tr>
<tr>
<td>Employee Id</td>
<td align='left'><input type='text' name='emp_id' id='emp_id' size ='10'
value='<%=leaveRequest.emp_id%>' /> </td>
<td> Name</td>
<td align='left'><input type='text' name='emp_name' id='emp_name' size ='20'
value='<%=leaveRequest.emp_name%>' /> </td>
</tr>
<tr>
<td>Designation</td>
<td align='left'><input type='text' name='level_id' id='level_id' size ='10'
value='<%=leaveRequest.level_id%>' /> </td>
<td>Department</td>
<td align='left'><input type='text' name='dept_id' id='dept_id' size ='10'
value='<%=leaveRequest.dept_id%>' /> </td>
</tr>
<tr>
<td colspan='4' height=20 bgcolor ='#AAAAAA' class=whitetext align=center>Leave
Detail</td>
</tr>
<tr>
<td>From<br>(YYYY-MM-DD)</td>
```

```

<td>To(YYYY-MM-DD)</td>
<td>Days</td>
</tr>
<tr>
<td align='left'>
<input type='text' name='from_date' id='from_date' size ='10' value=
'<%=leaveRequest.from_date%>'/> </td>
<td align='left'><input type='text' name='to_date' id='to_date' size ='10'
value='<%=leaveRequest.to_date%>'/> </td>
<td align='left'><input type='text' name='days' id='days' size ='5'
value='<%=leaveRequest.days%>'/> </td>
</tr>
<tr>
<td colspan='2'>Reason For Leave(100 letters only)</td>
<td align='left' colspan='2'><input type='text' name='reason' id='reason'
col='80' rows='4' value='<%=leaveRequest.reason%>'/>
</textarea>
</td>
</tr>
<tr>
<td colspan='2'>Type Of Leave Applied For</td>
<%
    if( leaveRequest.reason != null )
        out.print("<td align='left'><input type='text' name='leave_type'
id='leave_type' size ='10' value='"+leaveRequest.leave_type+"'/></td>"); 
    else
        out.print("<td align='left'><SELECT name='leave_type' > <OPTION
VALUE=></OPTION> <OPTION VALUE=CL>CL</OPTION>
<OPTION VALUE=SL>SL</OPTION></SELECT></td>"); 
%>
</tr>
<tr>
<td colspan='4' height=20 bgcolor ='#AAAAAA' class=whitetext align=center>Major
Activity to Be Handled During The Leave Applied</td>
</td>
</tr>
<tr>
<td>S.NO</td>
<td>Activity Details</td>
<td>Name of the Person Responsible</td>
<td>Details</td>
</tr>
<tr>
<td>1.</td>
<td align='left'><input type='text' name='activity_1' id='activity_1' size ='20'
value='<%=leaveRequest.activity_1%>'/> </td>
<td align='left'><input type='text' name='person_1' id='person_1' size ='20'
value='<%=leaveRequest.person_1%>'/> </td>
<td align='left'><input type='text' name='detail_1' id='detail_1' size ='20'
value='<%=leaveRequest.detail_1%>'/> </td>
</tr>
<tr>
<td>2.</td>
<td align='left'><input type='text' name='activity_2' id='activity_2' size ='20'
value='<%=leaveRequest.activity_2%>'/> </td>
<td align='left'><input type='text' name='person_2' id='person_2' size ='20'
value='<%=leaveRequest.person_2%>'/> </td>
<td align='left'><input type='text' name='detail_2' id='detail_2' size ='20'
value='<%=leaveRequest.detail_2%>'/> </td>
</tr>
<tr>
<td>3.</td>
<td align='left'><input type='text' name='activity_3' id='activity_3' size ='20'
value='<%=leaveRequest.activity_3%>'/> </td>
<td align='left'><input type='text' name='person_3' id='person_3' size ='20'
value='<%=leaveRequest.person_3%>'/> </td>
<td align='left'><input type='text' name='detail_3' id='detail_3' size ='20'
value='<%=leaveRequest.detail_3%>'/> </td>

```

```

</tr>
<tr>
<td colspan='4' height=20 bgcolor ='#AAAAAA' class=whitetext align=center>Contact
Details of the applicant during the leave period</td>
</tr>
<tr>
<td>Address</td>
<td align='left' colspan='3'><input type='text' name='address' id='address' size
='20' value='<%=leaveRequest.address%>'/> </td>
</tr>
<tr>
<td>Remark</td>
<td align='left' colspan='3'><input type='text' name='remark' id='remark' size
='20' value='<%=leaveRequest.remark%>'/> </td>
</tr>
<tr>
<th colspan='4' bgcolor ='#AAAAAA'></th>
</tr>
<tr>
<td align='center' colspan='4'><input type='submit' name='submit' id='submit' size
='10' value='Approve'/'> </td>
<input type='hidden' name='action_submit' id='action_submit' size ='10'
value='employee_leave_req_edit_submit'/'> </td>
</tr>
</table>
</td></tr>
<tr>
<td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr>
</table>
</body>
</html>

```

In Listing 23.47, a hidden input field, submit, is used to differentiate between the leave requests submitted by the employees, and those approved or rejected by the HR department.

Let's now create the leave_request_reject JSP page.

Creating the leave_request_reject JSP Page

The leave_request_reject JSP page is similar to the leave_request_edit JSP page, except that this page has the submit button with value Reject. Listing 23.48 shows the code of the leave_request_reject.jsp file (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.48: Showing the Code of the leave_request_reject.jsp File

```

<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.LeaveManagement.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Leave Request</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2"><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td width="900"><%@ include file="../jsp/people_default_menu.jsp" %></td>
</tr><tr>
<td width ="730" valign="top" align = "center">
<table border="0" align="top" width=600 >
<%
LeaveRequest leaveRequest = new LeaveRequest();
LeaveRequest = (LeaveRequest)session.getAttribute("leaveRequest");
%>

```

```

<form name="form1" method="post">
<tr>
<td colspan='4' height=20 bgcolor ='#AAAAAA' class=whitetext align=center>Leave Application</td>
</tr>
<tr>
<td>Request ID</td>
<td align='left'><input type='text' name='req_id' id='req_id' size ='10' value='<%=leaveRequest.req_id%>' /> </td>
<td>Date(YYYY-MM-DD)</td>
<td align='left'><input type='text' name='today_date' id='today_date' size ='10' value='<%=leaveRequest.today_date%>' /> </td>
</tr>
<tr>
<td>Employee Id</td>
<td align='left'><input type='text' name='emp_id' id='emp_id' size ='10' value='<%=leaveRequest.emp_id%>' /> </td>
<td> Name</td>
<td align='left'><input type='text' name='emp_name' id='emp_name' size ='20' value='<%=leaveRequest.emp_name%>' /> </td>
</tr>
<tr>
<td>Designation</td>
<td align='left'><input type='text' name='level_id' id='level_id' size ='10' value='<%=leaveRequest.level_id%>' /> </td>
<td>Department</td>
<td align='left'><input type='text' name='dept_id' id='dept_id' size ='10' value='<%=leaveRequest.dept_id%>' /> </td>
</tr>
<tr>
<td colspan='4' height=20 bgcolor ='#AAAAAA' class=whitetext align=center>Leave Detail</td>
</tr>
<tr>
<td>From<br>(YYYY-MM-DD)</td>
<td>To(YYYY-MM-DD)</td>
<td>Days</td>
</tr>
<tr>
<td align='left' >
<input type='text' name='from_date' id='from_date' size ='10' value='<%=leaveRequest.from_date%>' /> </td>
<td align='left' ><input type='text' name='to_date' id='to_date' size ='10' value='<%=leaveRequest.to_date%>' /> </td>
<td align='left'><input type='text' name='days' id='days' size ='5' value='<%=leaveRequest.days%>' /> </td>
</tr>
<tr>
<td colspan='2'>Reason For Leave(100 letters only)</td>
<td align='left' colspan='2'><input type='text' name='reason' id='reason' col='80' rows='4' value='<%=leaveRequest.reason%>' /> </td>
</tr>
<tr>
<td colspan='2'>Type Of Leave Applied For</td>
<%
if( leaveRequest.reason != null )
    out.print("<td align='left'><input type='text' name='leave_type' id='leave_type' size ='10' value='"+leaveRequest.leave_type+"'/></td>");
else
    out.print("<td align='left'><SELECT name='leave_type' > <OPTION VALUE=></OPTION>
<OPTION VALUE=CL>CL</OPTION><OPTION VALUE=SL>SL</OPTION></SELECT></td>");%
</tr>
<tr>
<td colspan='4' height=20 bgcolor ='#AAAAAA' class=whitetext align=center>Major Activity to Be Handled During The Leave Applied</td>
</td>

```

```

</tr>
<tr>
<td>S.NO</td>
<td>Activity Details</td>
<td>Name Of the Person Responsible</td>
<td>Details</td>
</tr>
<tr>
<td>1.</td>
<td align='left'><input type='text' name='activity_1' id='activity_1' size ='20' value='<%=leaveRequest.activity_1%>' /> </td>
<td align='left'><input type='text' name='person_1' id='person_1' size ='20' value='<%=leaveRequest.person_1%>' /> </td>
<td align='left'><input type='text' name='detail_1' id='detail_1' size ='20' value='<%=leaveRequest.detail_1%>' /> </td>
</tr>
<tr>
<td>2.</td>
<td align='left'><input type='text' name='activity_2' id='activity_2' size ='20' value='<%=leaveRequest.activity_2%>' /> </td>
<td align='left'><input type='text' name='person_2' id='person_2' size ='20' value='<%=leaveRequest.person_2%>' /> </td>
<td align='left'><input type='text' name='detail_2' id='detail_2' size ='20' value='<%=leaveRequest.detail_2%>' /> </td>
</tr>
<tr>
<td>3.</td>
<td align='left'><input type='text' name='activity_3' id='activity_3' size ='20' value='<%=leaveRequest.activity_3%>' /> </td>
<td align='left'><input type='text' name='person_3' id='person_3' size ='20' value='<%=leaveRequest.person_3%>' /> </td>
<td align='left'><input type='text' name='detail_3' id='detail_3' size ='20' value='<%=leaveRequest.detail_3%>' /> </td>
</tr>
<tr>
<td colspan='4' height=20 bgcolor ='#AAAAAA' class=whitetext align=center>Contact Details of the applicant during the leave period</td>
</tr>
<tr>
<td>Address</td>
<td align='left' colspan='3'><input type='text' name='address' id='address' size ='20' value='<%=leaveRequest.address%>' /> </td>
</tr>
<tr>
<td>Remark</td>
<td align='left' colspan='3'><input type='text' name='remark' id='remark' size ='20' value='<%=leaveRequest.remark%>' /> </td>
</tr>
<tr>
<th colspan='4' bgcolor ='#AAAAAA'></th>
</tr>
<tr>
<td align='center' colspan='4'><input type='submit' name='submit' id='submit' size ='10' value='Reject' />
<input type='hidden' name='action_submit' id='action_submit' size ='10' value='employee_leave_req_reject_submit' />
</td>
</tr>
</table>
</td></tr>
<tr>
<td colspan="2"><%@include file="..../jsp/people_footer.jsp"%></td>
</tr>
</table></body></html>

```

In Listing 23.48, a hidden input field, submit, is used to differentiate between the leave requests submitted by the employees, and those rejected by the HR department.

Let's now create the leave_request_list JSP page.

Creating the leave_request_list JSP Page

The leave_request_list JSP page displays the names of employees who have requested for leave in a list format. The code for the leave_request_list JSP page has been shown in Listing 23.48 (you can find the leave_request_list.jsp file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.49: Showing the Code of the leave_request_list.jsp File

```

<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.LeaveManagement.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.util.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Leave Request List</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
    <td colspan="2"><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
    <td width="900"><%@ include file="../jsp/people_default_menu.jsp" %></td>
</tr><tr>
    <td width = "750" valign="top">
<p>&nbsp;</p>
<%
    String dbopr = "";
    dbopr = (String)session.getAttribute("dbopr");
%>
<%
    if(dbopr != null && !dbopr.equals("rejected_leave")){
%>
<div align=center class=boldblack>List of Approved Leave Requests</div>
<% } else { %>
<div align=center class=boldblack>List of Rejected Leave Requests</div>
<% } %>
<hr width=100% color="#AAAAAA">
<table border="0" width=600 align=center>
    <tr>
        <td class=whitetext bgcolor ='#AAAAAA' align='center'>Request Id</td>
        <td class=whitetext bgcolor ='#AAAAAA' align='center'>Employee Id</td>
        <td class=whitetext bgcolor ='#AAAAAA' align='center'>Employee Name</td>
        <td class=whitetext bgcolor ='#AAAAAA' align='center'>Request Date</td>
        <td class=whitetext bgcolor ='#AAAAAA' align='center'>From Date </td>
        <td class=whitetext bgcolor ='#AAAAAA' align='center'>To Date </td>
        <td class=whitetext bgcolor ='#AAAAAA' align='center'>Days</td>
    <%
    if(dbopr != null && !dbopr.equals("approved_leave") && !dbopr.equals("rejected_leave"))
    out.println("<td class=whitetext colspan='3' bgcolor ='#AAAAAA' align='center'>Approve/Reject</td>");
%>
    </tr>
<%
ArrayList leaveRequestList = new ArrayList();
leaveRequestList = (ArrayList)session.getAttribute("leaveRequestList");
if ( leaveRequestList != null && leaveRequestList.size() > 0 ){
    for ( int size = 1; size <= leaveRequestList.size() ; size++ ){
        LeaveRequest leaveRequest = new LeaveRequest();
        leaveRequest = (LeaveRequest)leaveRequestList.get(size-1);
    %}
    <form name="form1" method="post">
        <tr bgcolor ='#AAAAAA'>
            <td align='center'><%=leaveRequest.req_id%></td>
            <td align='center'><%=leaveRequest.emp_id%></td>
            <td align='center'><%=leaveRequest.emp_name%> </td>
    
```

```

<td align='center' ><%=leaveRequest.today_date%> </td>
<td align='center' ><%=leaveRequest.from_date%></td>
<td align='center' ><%=leaveRequest.to_date%></td>
<td align='center' ><%=leaveRequest.days%></td>
<%
if(dbopr != null && !dbopr.equals("approved_leave") &&
!dbopr.equals("rejected_leave") ){
out.println("");
%>
<td align='center' bgcolor='#AAAAAA'><a href='http://localhost:8080/people-
mgmt/servlet/leave_management?dbopr=approve&&req_id=<%=leaveRequest.req_id%>&&emp_id=<%=leaveR
equest.emp_id%>' class=yellowlink>Approve</a></td>
<td align='center' bgcolor='#AAAAAA'><a href='http://localhost:8080/people-
mgmt/servlet/leave_management?dbopr=reject&&req_id=<%=leaveRequest.req_id%>&&emp_id=<%=leaveRe
quest.emp_id%>' class=yellowlink>Reject</a>
</td>
<%>
</tr>
<%
}
}
else{
    out.println("Request does not exist!!!");
}
%>
</table>
<hr width=100% color=#AAAAAA>
</td>
</tr>
<tr>
    <td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr>
</table>
</body>
</html>

```

In Listing 23.48, the Approve and Reject link is provided beside the name of every leave request. The HR department can approve the leave by just clicking Approve link or can reject the leave by clicking Reject link. Figure 23.23 shows all the leave requests that are not yet approved or rejected by the HR department:

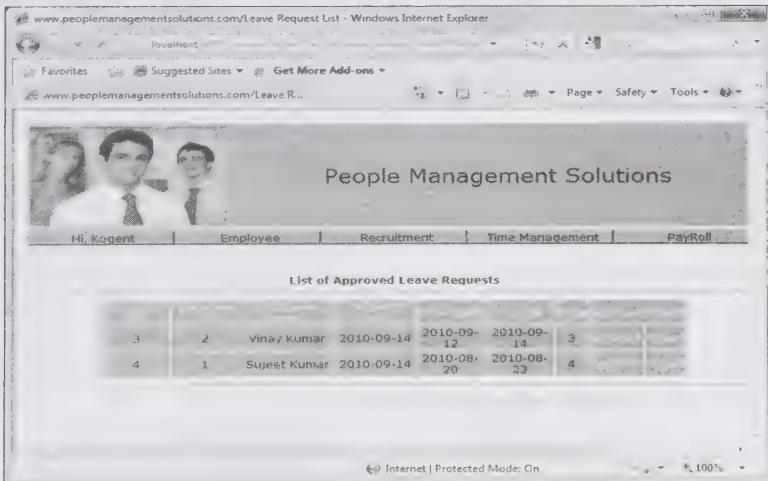


Figure 23.23: Displaying the List of Leave Requests in the leave_request_list JSP Page

When you click on Approve link shown in Figure 23.23, then after approving the leave request a Web page of approved leave requests is displayed. Figure 23.24 displays approved leave requests in the leave_request_list JSP page:

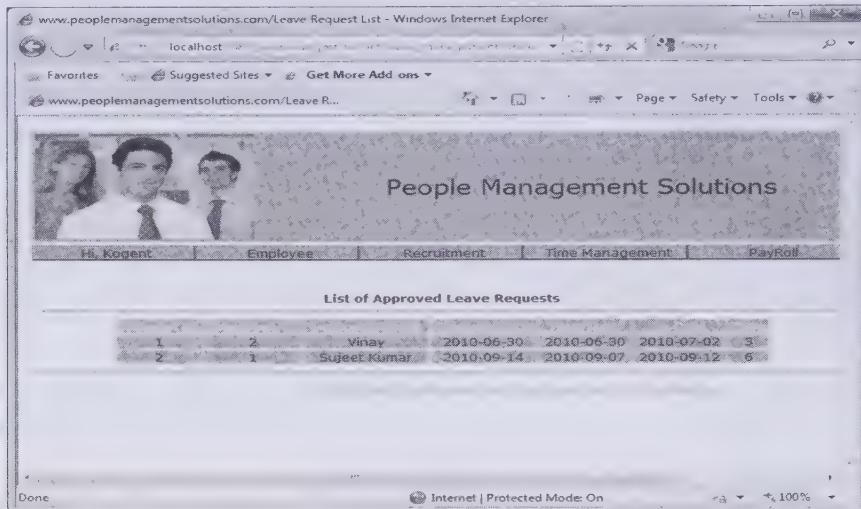


Figure 23.24: Displaying the Approved Leave Requests in the leave_request_list JSP Page

When you click on Reject link (Figure 23.23), then after rejecting the leave request a list of rejected leave requests is displayed. Figure 23.25 displays rejected leave requests in the leave_request_list JSP page:

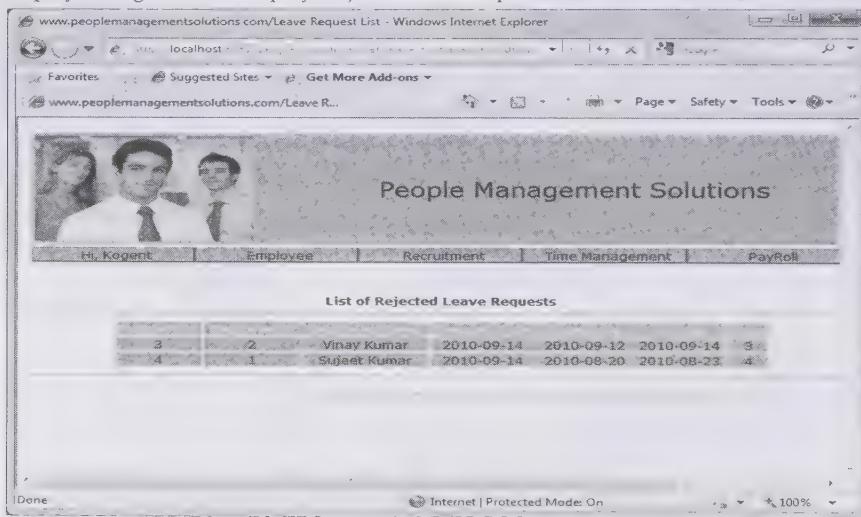


Figure 23.25: Displaying the Rejected Leave Requests in the leave_request_list JSP Page

The Leave Management module has been designed to manage the leave requests of the employees of an organization. In this module, the leave_management servlet acts as the controller servlet, and the other helper classes include LeaveRequest and LeaveMgmtBeanMethods. In this section, you have also designed various JSP views used to handle the details of the leave requests.

In the next section, we discuss the development of another module, the payroll module.

Section F

Developing the Payroll Module

If you need an information on:

See page:

Updating Salary Statement

1204

Creating people_payroll Servlet

1204

Designing JSP Views

1218

Disbursing salaries on time and generating pay slips after calculating the monthly salaries of all employees of an organization is one of the most complex functions of the HR department. There are a number of things that need to be considered while generating monthly salary of an employee, such as leave balance, attendance, and salary package. Every employee has a salary package that is divided into many allowance types, such as basic salary, House Rent Allowance (HRA), Dearness Allowance (DA), Transportation Allowance (TA), and Provident Fund (PF). In this module, you learn to create applications to update the salary statement and calculate the salary of employees for every month.

Similar to other modules, this module also implements the MVC architecture with a servlet, Java classes, and JSP pages. The list of Java source code files to be created for this module is as follows:

- people_payroll.java
- EmpSal.java
- EmployeeAgreement.java
- PayrollBeanMethods.java

The following JSP pages also need to be created in this module:

- salary_search.jsp
- employee_agreement.jsp
- employee_agreement_edit.jsp
- salary_slip.jsp

The next section discusses how to update salary statement of an employee.

Updating Salary Statement

Salary statement can be described as the information containing the details of the salary package of an employee that includes different heads and the amount being paid under that head, such as basic salary and other allowances. A new statement is made for every new employee who joins the organization. This statement gets updated on every salary revision (increment/decrement) of the employee. This employee statement is used to calculate monthly salary of a particular employee.

Let's now create the people_payroll servlet.

Creating people_payroll Servlet

The people_payroll servlet plays the role of the controller servlet for this module. This implies that all the requests for payroll-related functions are handled by this servlet. This servlet is similar to other servlets developed in this project, and forwards the request to the appropriate JSP view after performing the required operations. This servlet can be used in various ways by going through different execution paths. Different parts of the servlet can be executed according to the values passed by the dbopr and action parameters to the servlet.

Listing 23.49 shows the code of the people_payroll.java file (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src folder on CD):

Listing 23.49: Showing the Code of the people_payroll.java File

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.ArrayList;
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.annotation.*;
import javax.servlet.annotation.WebServlet;
import com.Employee.*;
import com.Payroll.*;
import com.LeaveManagement.*;
import com.TimeManagement.*;
@WebServlet(name="people_payroll", urlPatterns="/servlet/people_payroll")
public class people_payroll extends HttpServlet{
```

```

String dbuser = "";
String dbpswd = "";
String dburl = "";
/**Initialize global variables*/
@Override
public void init(ServletConfig config) throws ServletException{
    System.out.println("initializing controller servlet.");
    ServletContext context = config.getServletContext();
    dbuser = "scott";
    dbpswd = "tiger";
    dburl = "jdbc:oracle:thin:@192.168.1.123:1521:"+"XE";
    super.init(config);
}
/**Process the HTTP Get request*/
@Override
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException{
    doPost(request, response);
}
/**Process the HTTP Post request*/
@Override
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    HttpSession session = request.getSession();
    session.setAttribute("lErrorMsg",null);
    String target = "";
    String action = request.getParameter("action");
    String dbopr = "";
    dbopr = (String)request.getParameter("dbopr");
    session.setAttribute("dbopr",dbopr);
    if( (dbopr != null && dbopr.length() > 0) && (dbopr.equals
        ("employee_agreement")) ){
        target = "/jsp/employee_search.jsp";
    }
    else
    if( (dbopr != null && dbopr.length() > 0) && (dbopr.equals("edit_head")) ){
        action = "edit_sal_head";
    }
    else
    if( (dbopr != null && dbopr.length() > 0) && (dbopr.equals("delete_head")) )
    {
        action = "employee_sal_head_delete";
    }
    else
    if( (dbopr != null && dbopr.length() > 0) && (dbopr.equals
        ("calc_employee_salary")) ){
        target= "/jsp/salary_search.jsp";
    }
    String action_submit = request.getParameter("action_submit");
    System.out.println("action_submit=="+action_submit);
    if ( action_submit != null ){
        if ( request.getParameter("submit").equals("Submit") ){
            System.out.println("in the Submit");
            if ( action_submit.equals("people_employee_search_submit") ){
                System.out.println("in the people_employee_search_submit ");
                action = "people_employee_search_submit";
            }
        }
    }
    else
    if ( request.getParameter("submit").equals("Edit") ){
        if ( action_submit.equals("employee_sal_head_edit_submit") )
        {
            action = "employee_sal_head_edit_submit";
        }
    }
}

```

```

    else
      if ( request.getParameter("submit").equals("Submit Detail") ){
      if ( action_submit.equals("emp_agreement_dtl_submit") )
        action = "emp_agreement_dtl_submit";
      }
    else
      if ( request.getParameter("submit").equals("Calc") ){
      if ( action_submit.equals("salary_calc_submit") )
        action = "salary_calc_submit";
      }
    }
  }

  if (action!=null){
    System.out.println("in the "+action);
    if (action.equals("people_employee_search_submit")){
      String lEmpId = "";
      String lEmpFName = "";
      lEmpId = (String)request.getParameter("emp_id");
      lEmpFName = (String)request.getParameter("emp_f_name");
      EmployeeDBObj employeeDBObj = new EmployeeDBObj();
      EmployeeDBMethods employeeDBMethods = new
      EmployeeDBMethods(dbuser,dbpswd,dburl);
      employeeDBObj = (EmployeeDBObj)employeeDBMethods.
      getRecordByPrimaryKey(lEmpId,lEmpFName);
      System.out.println("iiiii="+employeeDBObj.emp_id+"fffff=
      "+employeeDBObj.emp_f_name);
      if ( (employeeDBObj.emp_id != null && employeeDBObj.emp_f_name != null) ){
        session.setAttribute("employeeDBObj",employeeDBObj);
        PayrollBeanMethods payrollBeanMethods = new PayrollBeanMethods
        (dbuser,dbpswd,dburl);
        ArrayList employeeAgreementList = new ArrayList();
        String criteria = "";
        criteria = " where emp_id='"+employeeDBObj.emp_id+"' ";
        employeeAgreementList = (
        ArrayList)payrollBeanMethods.selectEmployee
        AgreementByCriteria(criteria);
        session.setAttribute("employeeAgreementList"
        ,employeeAgreementList);
        target = "/jsp/employee_agreement.jsp";
      }
      else{
        String lErrorMsg = "Employee doesn't Exist";
        session.setAttribute("lErrorMsg",lErrorMsg);
        target = "/jsp/people_default.jsp";
      }
    }
    else
      if (action.equals("edit_sal_head")){
        String empId = "";
        String allowanceName = "";
        empId = (String)request.getParameter("emp_id");
        allowanceName = (String)request.getParameter("allowance_name");
        PayrollBeanMethods payrollBeanMethods = new PayrollBeanMethods
        (dbuser,dbpswd,dburl);
        EmployeeAgreement employeeAgreement = new EmployeeAgreement();
        employeeAgreement =(EmployeeAgreement)payrollBeanMethods.getEmployee
        AgreementRecord(empid,allowanceName);
        session.setAttribute("employeeAgreement",employeeAgreement);
        target = "/jsp/employee_agreement_edit.jsp";
      }
      else
        if (action.equals("salary_calc_submit")){
          PayrollBeanMethods payrollBeanMethods = new PayrollBeanMethods
          (dbuser,dbpswd,dburl);
          String empId = (String)request.getParameter("emp_id");
          int year = Integer.parseInt((String)request.getParameter("year"));
        }
      }
    }
  }
}

```

```

int month = Integer.parseInt((String)request.getParameter("month"));
String lEmpFName = "";
EmployeeDBObj employeeDBObj = new EmployeeDBObj();
EmployeeDBMethods employeeDBMethods = new EmployeeDBMethods
(dbuser,dbpswd,dburl);
employeeDBObj = (EmployeeDBObj)employeeDBMethods.getRecord
ByPrimaryKey(empId,lEmpFName);
int totalAttendance = 0;
totalAttendance = getTotalAttendance(empId,year,month);
System.out.println("totalAttendance===="+totalAttendance);
session.setAttribute("totalAttendance",Integer.toString
(totalAttendance));
int totalLeave = 0;
totalLeave = getTotalLeave(empId,year,month);
System.out.println("totalLeave=====+totalLeave");
session.setAttribute("totalLeave",Integer.toString(totalLeave));
session.setAttribute("employeeDBObj",employeeDBObj);
session.setAttribute("year",Integer.toString(year));
session.setAttribute("month",Integer.toString(month));
ArrayList empSalExist = new ArrayList();
String criteria = "";
criteria = " where emp_id='"+empId+"' and year='"+year+"' and
month='"+month+"'";
empSalExist = ( ArrayList)payrollBeanMethods.select
EmpsAlByCriteria(criteria);
if ( empSalExist != null && empSalExist.size() > 0 ){
    ArrayList empsalList = new ArrayList();
    criteria = "";
    criteria = " where emp_id='"+empId+"' and year='"+year+"' and
month='"+month+"'";
    empsalList = ( ArrayList)payrollBeanMethods.selectEmpSal
ByCriteria(criteria);
    session.setAttribute("empsalList",empsalList);
    System.out.println("empsalList.size()====="
+empsalList.size());
}
else
if( totalAttendance > 0 ){
    ArrayList employeeAgreementList = new ArrayList();
    criteria = "";
    criteria = " where emp_id='"+empId+"'";
    employeeAgreementList = ( ArrayList)payrollBeanMethods.select
EmployeeAgreementByCriteria(criteria);
    for ( int rec =1; rec <= employeeAgreementList.size() ; rec++ )
    {
        EmployeeAgreement employeeAgreement = new
            EmployeeAgreement();
        Empsal empsal = new Empsal();
        employeeAgreement = (EmployeeAgreement)employeeAgree
        mentList.get(rec-1);
        empsal.emp_id = employeeAgreement.emp_id;
        empsal.year = year ;
        empsal.month = month;
        empsal.allowance_name = employeeAgreement.allowance_name;
        empsal.allowance_type = employeeAgreement.allowance_type;
        empsal.amt = employeeAgreement.amt;
        empsal.taxable = employeeAgreement.taxable;
        empsal.percentage = employeeAgreement.percentage;
        payrollBeanMethods.insertEmpSal(empsal);
    }
    ArrayList empsalList = new ArrayList();
    criteria = "";
    criteria = " where emp_id='"+empId+"' and year='"+year+"' and
month='"+month+"'";
    empsalList = ( ArrayList)payrollBeanMethods.selectEmpSalByCriteria

```

```

        (criteria);
        session.setAttribute("empSalList", empSalList);
        System.out.println("empSalList.size()===="+empSalList.size());
    }
    else{
        String lErrorMsg = "your criteria is not correct .. salary is
                           not prepared!!!";
        session.setAttribute("lErrorMsg", lErrorMsg);
        if(session.getAttribute("empSalList")!=null)
            session.removeAttribute("empSalList");
    }
    target = "/jsp/salary_slip.jsp";
}
else
if (action.equals("emp_agreement_dtl_submit")){
    EmployeeAgreement popEmployeeAgreement = new
        EmployeeAgreement();
    PayrollBeanMethods payrollBeanMethods = new
        PayrollBeanMethods(dbuser,dbpswd,dburl);
    popEmployeeAgreement = (EmployeeAgreement)payrollBeanMethods.
        populateEmployeeAgreementFromReq(request);
    EmployeeAgreement employeeAgreement = new
        EmployeeAgreement();
    employeeAgreement = (EmployeeAgreement)payrollBeanMethods.get
        EmployeeAgreementRecord(popEmployeeAgreement,
        emp_id,popEmployeeAgreement.allowance_name);
    if ( (popEmployeeAgreement.emp_id).equals(employeeAgreement.emp_id)
    && (popEmployeeAgreement.allowance_name).equals(employee
    Agreement.allowance_name ) ){
        String lErrorMsg = "Allowance Already Exist";
        session.setAttribute("lErrorMsg", lErrorMsg);
        target = "/jsp/employee_agreement.jsp";
    }
    else{
        int rval = payrollBeanMethods.insertEmployeeAgreement
        (popEmployeeAgreement);
        EmployeeAgreement sEmployeeAgreement = new
        EmployeeAgreement();
        sEmployeeAgreement = (EmployeeAgreement)payrollBeanMethods.get
            EmployeeAgreementRecord(popEmployeeAgreement.emp_id
            ,popEmployeeAgreement.allowance_name);
        ArrayList employeeAgreementList = new ArrayList();
        String criteria = "";
        criteria = " where emp_id='"+popEmployeeAgreement.emp_id+"'";
        employeeAgreementList = (ArrayList)payrollBeanMethods.
        selectEmployeeAgreementByCriteria(criteria);
        session.setAttribute("employeeAgreementList",
        employeeAgreementList);
        session.setAttribute("employeeAgreement",sEmployeeAgreement);
        target = "/jsp/employee_agreement.jsp";
    }
}
else
if (action.equals("employee_sal_head_edit_submit")){
    EmployeeAgreement popEmployeeAgreement = new EmployeeAgreement();
    PayrollBeanMethods payrollBeanMethods = new PayrollBeanMethods
        (dbuser,dbpswd,dburl);
    popEmployeeAgreement = (EmployeeAgreement)payrollBeanMethods.
        populateEmployeeAgreementFromReq(request);
    int rval = payrollBeanMethods.updateEmployeeAgreement
    ByPrimaryKey(popEmployeeAgreement);
    if ( rval > 0){
        ArrayList employeeAgreementList = new ArrayList();
        String criteria = "";
        criteria = " where emp_id='"+popEmployeeAgreement.emp_id';
}

```

```

employeeAgreementList = ( ArrayList ) payrollBeanMethods.select
EmployeeAgreementByCriteria( criteria );
session.setAttribute("employeeAgreementList",
employeeAgreementList);
target = "/jsp/employee_agreement.jsp";
}
else{
target = "/jsp/employee_agreement_edit.jsp";
}
}
else if (action.equals("employee_sal_head_delete")){
String empId = "";
String allowanceName = "";
empId = (String) request.getParameter("emp_id");
allowanceName = (String) request.getParameter
("allowance_name");
PayrollBeanMethods payrollBeanMethods = new
PayrollBeanMethods(dbuser, dbpwd, dburl);
payrollBeanMethods.deleteEmployeeAgreement
(empId, allowanceName);
ArrayList employeeAgreementList = new ArrayList();
String criteria = "";
criteria = " where emp_id='"+empId+"'";
employeeAgreementList = ( ArrayList ) payrollBeanMethods.select
EmployeeAgreementByCriteria( criteria );
session.setAttribute("employeeAgreementList", employee
AgreementList);
target = "/jsp/employee_agreement.jsp";
}
}

/* forwarding the request/response to the targeted view */
RequestDispatcher requestDispatcher = getServletContext().getRequest
Dispatcher(target);
requestDispatcher.forward(request, response);
} // doPost closed
public int getTotalAttendance( String inEmpId, int inYear,int inMonth ){
.....
.....
.....
}
public int getTotalLeave( String inEmpId, int inYear,int inMonth ){
.....
.....
.....
}
public String getMonth( int month ){
String strMonth = "";
if(month == 0) strMonth = "JAN";
else if(month == 1) strMonth = "FEB";
else if(month == 2) strMonth = "MAR";
else if(month == 3) strMonth = "APR";
else if(month == 4) strMonth = "MAY";
else if(month == 5) strMonth = "JUN";
else if(month == 6) strMonth = "JUL";
else if(month == 7) strMonth = "AUG";
else if(month == 8) strMonth = "SEP";
else if(month == 9) strMonth = "OCT";
else if(month == 10) strMonth = "NOV";
else if(month == 11) strMonth = "DEC";
return strMonth;
}
}// class closed

```

Section F

In Listing 23.49, four packages, such as com.Employee, com.Payroll, com.LeaveManagement, and com.TimeManagement are imported, as some of the classes from these packages are used to implement an operation. Apart from the standard init(), doGet(), and doPost() methods, this servlet has two more methods—getTotalAttendance() and getTotalLeave(), which help in calculating the salary of an employee by providing the details of the total attendance and the number of leaves taken in a month.

The getTotalAttendance() method described in Listing 23.50 returns total attendance of a particular employee for a specific month of a given year. The people_payroll.java file defines the selectEmpDailyAttendanceByCriteria() method of the TimeManagementDBMethods class, as shown in Listing 23.50:

Listing 23.50: Showing the Code of the getTotalAttendance() Method in the people_payroll.java File

```
public int getTotalAttendance( String inEmpId, int inYear,int inMonth )
{
    TimeManagementDBMethods timeManagementDBMethods = new
        TimeManagementDBMethods(dbuser,dbpswd,dburl);
    int totalAttendance = 0;
    ArrayList attendanceList = new ArrayList();
    String criteria = "where emp_id ='"+inEmpId+"' and year='"+inYear+"' and
        month='"+timeManagementDBMethods.getMonth(inMonth-1)+"'";
    attendanceList =
        (ArrayList)timeManagementDBMethods.selectEmpDailyAttendanceBy
            Criteria(criteria);
    if( attendanceList != null && attendanceList.size() > 0)
        totalAttendance = attendanceList.size();
    return totalAttendance;
}
```

The getTotalAttendance () method provided in Listing 23.50 returns the total attendance of an employee. This method invokes the selectEmployeeAttendanceByCriteria() method of the com.TimeManagement.TimeManagementDBMethods class that returns total attendance of the employee. Similarly, the getTotalLeave() method returns the total number of leaves of an employee in a given month of a given year by using the selectLeaveRequestByCriteria() method of the LeaveMgmtBeanMethods class.

Listing 23.51 provides the code for the getTotalLeave() method:

Listing 23.51: Showing the Code of the getTotalLeave() Method in the people_payroll.java File

```
public int getTotalLeave( String inEmpId, int inYear,int inMonth )
{
    LeaveMgmtBeanMethods leaveMgmtBeanMethods = new
        LeaveMgmtBeanMethods(dbuser,dbpswd,dburl);
    int totalLeave = 0;
    ArrayList leaveList = new ArrayList();
    String strtdate = "";
    String enddate = "";
    strtdate = "01/"+inMonth+"/"+inYear+"";
    enddate = "31/"+inMonth+"/"+inYear+"";
    String criteria = "where emp_id ='"+inEmpId+"' and leave_status='Aprv' and
        from_date >='"+strtdate+"' and to_date <='"+enddate+"' ";
    leaveList = (ArrayList)leaveMgmtBeanMethods.
        selectLeaveRequestByCriteria(criteria);
    if( leaveList != null && leaveList.size() > 0)
    {
        for (int i = 1;i<=leaveList.size() ;i++ )
        {
            LeaveRequest leaveRequest = new LeaveRequest();
            leaveRequest = (LeaveRequest)leaveList.get(i-1);
            totalLeave = totalLeave + leaveRequest.days;
        }
    }
    return totalLeave;
}
```

You should note that the people_payroll.java file uses the @WebServlet annotation for servlet-mapping. The use of the @WebServlet annotation removes the need of specifying the servlet-mapping in the web.xml file.

Let's now create the EmpSal class.

Creating the EmpSal Class

The EmpSal class is a simple java class having a set of member variables matching the EMP_SAL table. An object of this class is used to represent the salary of an employee for a month. The details include emp_id, year, month, and amt.

Listing 23.52 provides the code of the EmpSal.java file (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src\com\Payroll folder on CD):

Listing 23.52: Showing the Code of the EmpSal.java File

```
package com.Payroll;
public class EmpSal
{
    public String emp_id ;
    public int year;
    public int month;
    public String allowance_type;
    public String allowance_name;
    public double amt;
    public String taxable;
    public double percentage;
}
```

In Listing 23.52, the datafields are specified corresponding to the EMP_SAL table.

Creating the EmployeeAgreement Class

The EmployeeAgreement class represents a salary statement showing details of the entire salary package of an employee. An object of this class represents the details of the allowance given to an employee, such as the head name and amount.

Listing 23.53 provides the code of the EmployeeAgreement.java file (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src\com\Payroll folder on CD):

Listing 23.53: Showing the Code of the EmployeeAgreement.java File

```
package com.Payroll;
public class EmployeeAgreement
{
    public String emp_id;
    public String emp_name;
    public String level_id;
    public String allowance_type;
    public String allowance_name;
    public double amt;
    public String taxable;
    public double percentage;
    public String agreement_date;
}
```

Listing 23.53 represents an entity corresponding to the fields of the employee_agreement JSP page.

Let's now create the PayrollBeanMethods class.

Creating the PayrollBeanMethods Class

The PayrollBeanMethods class contains the methods that directly interact with different tables to perform various functions, such as fetching and updating records. This class uses the fields that are specified in the EmpSal and EmployeeAgreement classes with the EMP_SAL and EMPLOYEE AGREEMENT tables, respectively.

Listing 23.54 provides the code of the PayrollBeanMethods.java file (you can find this file in the PeopleMgmt\people-mgmt\WEB-INF\src\com\Payroll folder on CD):

Listing 23.54: Showing the Code of the PayrollBeanMethods.java File

```

package com.Payroll;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.util.ArrayList;
import com.Payroll.EmployeeAgreement;
public class PayrollBeanMethods
{
    public String DBUser;
    public String DBPswd;
    public String DBUrl ;
    public PayrollBeanMethods() { }

    public PayrollBeanMethods(String inDBUser, String inDBPswd, String inDBUrl )
    {
        DBUser = inDBUser ;
        DBPswd = inDBPswd;
        DBUrl = inDBUrl;
    }
    public void initializeEmployeeAgreement(EmployeeAgreement inEmployeeAgreement)
    {
        inEmployeeAgreement.emp_id = "";
        inEmployeeAgreement.emp_name = "";
        inEmployeeAgreement.level_id = "";
        inEmployeeAgreement.allowance_type = "";
        inEmployeeAgreement.allowance_name = "";
        inEmployeeAgreement.amt = 0;
        inEmployeeAgreement.taxable = "";
        inEmployeeAgreement.percentage = 0;
        inEmployeeAgreement.agreement_date = "";
    }
    public EmployeeAgreement getEmployeeAgreementRecord(String inEmpId, String
        inAllowanceName)
    {
        EmployeeAgreement employeeAgreement = new EmployeeAgreement();
        java.sql.Date date;
        try
        {
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
            Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
            Statement stmt = conn.createStatement();
            String lsqlString = "...select * from EMPLOYEE AGREEMENT";
            lsqlString = lsqlString + "where emp_id='"+inEmpId+"' ";
            lsqlString = lsqlString + "and allowance_name='"+
            inAllowanceName+"' ";
            ResultSet rs = null;
            rs = stmt.executeQuery(lsqlString);
            System.out.println("lsqlString===="+lsqlString);
            if( rs.next())
            {
                System.out.println("fffff=="+rs.getString("emp_id"));
                employeeAgreement.emp_id =
                    (String)rs.getString("emp_id");
                employeeAgreement.emp_name =
                    (String)rs.getString("emp_name");
                employeeAgreement.level_id =
                    (String)rs.getString("level_id");
                employeeAgreement.allowance_type =
                    (String)rs.getString("allowance_type");
                employeeAgreement.allowance_name =
                    rs.getString("allowance_name");
                employeeAgreement.amt = rs.getDouble("amt");
            }
        }
    }
}

```

```

employeeAgreement.taxable =
    (String)rs.getString("taxable");
employeeAgreement.percentage = rs.getDouble("percentage");
date=rs.getDate("agreement_date");
if(date!=null)
    employeeAgreement.agreement_date = date.toString();
System.out.println("fffff===="+rs.getString("emp_id"));
}
else
{
    initializeEmployeeAgreement(employeeAgreement);
}
System.out.println("fffff===="+employeeAgreement.emp_id);
if(rs != null)
    rs.close();
if( conn != null)
    conn.close();
}
catch(SQLException ex)
{
    ex.printStackTrace();
}
return employeeAgreement;
}

public ArrayList selectEmployeeAgreementByCriteria(String inCriteria)
{
    ArrayList EmployeeAgreementList = new ArrayList();
    java.sql.Date date;
    try
    {
        DriverManager.registerDriver(new
            oracle.jdbc.driver.OracleDriver());
        Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
        Statement stmt = conn.createStatement();
        String lsqlString = "select * from EMPLOYEE AGREEMENT ";
        if( inCriteria != null && inCriteria.length() > 0 )
        {
            lsqlString = lsqlString +" "+inCriteria+" ";
        }
        System.out.println("Criteria===== "+inCriteria+" and
            query="+lsqlString);
        ResultSet rs = null;
        rs = stmt.executeQuery(lsqlString);
        while( rs.next())
        {
            EmployeeAgreement employeeAgreement = new
                EmployeeAgreement();
            employeeAgreement.emp_id =
                (String)rs.getString("emp_id");
            employeeAgreement.emp_name =
                (String)rs.getString("emp_name");
            employeeAgreement.level_id =
                (String)rs.getString("level_id");
            employeeAgreement.allowance_type =
                (String)rs.getString("allowance_type");
            employeeAgreement.allowance_name =
                rs.getString("allowance_name");
            employeeAgreement.amt = rs.getDouble("amt");
            employeeAgreement.taxable =
                (String)rs.getString("taxable");
            employeeAgreement.percentage = rs.getDouble("percentage");
            date=rs.getDate("agreement_date");
            if(date!=null)
                employeeAgreement.agreement_date = date.toString();
            EmployeeAgreementList.add(employeeAgreement);
        }
    }
}

```

```

        if(rs != null)
            rs.close();
        if( conn != null)
            conn.close();
    }
    catch(SQLException ex)
    {
        ex.printStackTrace();
    }
    return EmployeeAgreementList;
}

public int updateEmployeeAgreementByPrimaryKey(EmployeeAgreement
    inEmployeeAgreement)
{
    int recupd = 0;
    String lQuery = "";
    lQuery = lQuery +"update EMPLOYEE AGREEMENT set
        emp_name='"+inEmployeeAgreement.emp_name+"' ";
    lQuery = lQuery +", level_id='"+inEmployeeAgreement.level_id+"' ";
    lQuery = lQuery +",
        allowance_type='"+inEmployeeAgreement.allowance_type+"' ";
    lQuery = lQuery +", amt='"+inEmployeeAgreement.amt+" ";
    lQuery = lQuery +", taxable='"+inEmployeeAgreement.taxable+"' ";
    lQuery = lQuery +", percentage='"+inEmployeeAgreement.percentage+"' ";
    lQuery = lQuery +",
        agreement_date=to_date('" +inEmployeeAgreement.agreement_date+"', 'yyyy-mm-dd') ";
    lQuery = lQuery +", where emp_id='"+inEmployeeAgreement.emp_id+"' ";
    lQuery = lQuery + "and
        allowance_name='"+inEmployeeAgreement.allowance_name+"' ";
    System.out.println("lSqlString==:" +lQuery);
    try
    {
        DriverManager.registerDriver(new
            oracle.jdbc.driver.OracleDriver());
        Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
        Statement stmt = conn.createStatement();
        recupd = stmt.executeUpdate(lQuery);
        if( conn != null)
            conn.close();
    }
    catch(SQLException ex)
    {
        ex.printStackTrace();
    }
    return recupd;
}

public EmployeeAgreement populateEmployeeAgreementFromReq(HttpServletRequest
    inReq)
{
    EmployeeAgreement employeeAgreement = new EmployeeAgreement();
    employeeAgreement.emp_id = (String)inReq.getParameter("emp_id");
    employeeAgreement.emp_name = (String)inReq.getParameter("emp_name");
    employeeAgreement.level_id = (String)inReq.getParameter("level_id");
    employeeAgreement.allowance_type =
        (String)inReq.getParameter("allowance_type");
    employeeAgreement.allowance_name =
        inReq.getParameter("allowance_name");
    if( (String)inReq.getParameter("amt") != null &&
        ((String)inReq.getParameter("amt")).length() > 0 )
        employeeAgreement.amt =
            Double.parseDouble((String)inReq.getParameter("amt")) ;
    else
        employeeAgreement.amt = 0;
    employeeAgreement.taxable = (String)inReq.getParameter("taxable");
}

```

```

if( (String)inReq.getParameter("percentage") != null &&
    ((String)inReq.getParameter("percentage")).length() > 0)
    employeeAgreement.percentage =
        Double.parseDouble((String)inReq.getParameter("percentage"));
else
    employeeAgreement.percentage = 0;
employeeAgreement.agreement_date =
    (String)inReq.getParameter("agreement_date");
return employeeAgreement;
}
public int insertEmployeeAgreement(EmployeeAgreement inEmployeeAgreement)
{
    ...
    ...
}

public int insertEmpSal(EmpSal inEmpSal)
{
    ...
    ...
}

public ArrayList selectEmpSalByCriteria(String inCriteria)
{
    ...
    ...
}

public void deleteEmployeeAgreement(String inEmpId , String inAllowanceName)
{
    ...
    ...
}
}

```

In Listing 23.54, the `getEmployeeAgreementRecord()` method returns the `EmployeeAgreement` object. Other methods, such as `selectEmployeeAgreementByCriteria()` and `updateEmployeeAgreementByPrimaryKey()`, are used to retrieve and update an employee's salary, respectively. The `selectEmployeeAgreementByCriteria()` method returns the salary of an employee in the form of the elements of the `ArrayList` object. The `updateEmployeeAgreementByPrimaryKey()` method returns an `int` value on the basis of the number of records updated in the database. The `PayrollBeanMethods` class also provides methods to insert or delete a record from the table. All these methods accept arguments to create queries and return an object containing the result of the query. For example, the `insertEmpSal()` method takes an object of the `EmpSal` class and inserts a new record in the `EMP_SAL` table. This method returns an `int` value, depending upon the number of records updated by the execution of the insert query. In our case, the `1` value is returned by the `insertEmpSal()` method.

Listing 23.55 provides the code of the `insertEmpSal()` method of the `PayrollBeanMethods` class:

Listing 23.55: Showing the Code of the `insertEmpSal()` Method of the `PayrollBeanMethods` Class

```

public int insertEmpSal(EmpSal inEmpSal)
{
    int recupd = 0;
    String lQuery = "";
    lQuery = lQuery +"insert into EMP_SAL values ( ";
    lQuery = lQuery +""+inEmpSal.emp_id+" ";
    lQuery = lQuery +", "+inEmpSal.year+" ";
    lQuery = lQuery +", "+inEmpSal.month+" ";
    lQuery = lQuery +", '"+inEmpSal.allowance_type+"' ";
    lQuery = lQuery +", '"+inEmpSal.allowance_name+"' ";
    lQuery = lQuery +", "+inEmpSal.amt+" ";
    lQuery = lQuery +", '"+inEmpSal.taxable+"' ";
    lQuery = lQuery +", '"+inEmpSal.percentage+"' ";
    lQuery = lQuery +")";
    System.out.println("lsqlstring====:"+lQuery);
}

```

```

try
{
    DriverManager.registerDriver(new
        oracle.jdbc.driver.OracleDriver());
    Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
    Statement stmt = conn.createStatement();
    recupd = stmt.executeUpdate(lQuery);
    if( conn != null)
        conn.close();
}
catch(SQLException ex)
{
    ex.printStackTrace();
}
return recupd;
}

```

The code provided in Listing 23.55 is used to insert the salary of an employee in the EMP_SAL table. The other method named selectEmpSalByCriteria() accepts a string as an argument to execute the query to find salary details of an employee. The passed string is used as a criteria and contains the where clause. The method returns an ArrayList containing objects of the EmpSal type. Each object in that ArrayList represents the details of allowances given to a particular employee.

The code for the selectEmpSalByCriteria() method of the PayrollBeanMethods class is shown in Listing 23.56:

Listing 23.56: Showing the Code of the selectEmpSalByCriteria() Method of the PayrollBeanMethods Class

```

public ArrayList selectEmpSalByCriteria(String inCriteria)
{
    ArrayList EmpSalList = new ArrayList();
    try
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
        Statement stmt = conn.createStatement();
        String lsqlString ="select * from EMP_SAL ";
        if( inCriteria != null && inCriteria.length() > 0 )
        {
            lsqlString = lsqlString +" "+inCriteria+"";
        }
        System.out.println("Criteria==== "+inCriteria+" and
                           query="+lsqlString);
        ResultSet rs = null;
        rs = stmt.executeQuery(lsqlString);
        while( rs.next())
        {
            EmpSal empSal = new EmpSal();
            empSal.emp_id = (String)rs.getString("emp_id");
            empSal.year = rs.getInt("year");
            empSal.month = rs.getInt("month");
            empSal.allowance_type =
                (String)rs.getString("allowance_type");
            empSal.allowance_name = rs.getString("allowance_name");
            empSal.amt = rs.getDouble("amt");
            empSal.taxable = (String)rs.getString("taxable");
            empSal.percentage = rs.getDouble("percentage");
            EmpSalList.add(empsal);
        }
        if(rs != null)
            rs.close();
        if( conn != null)
            conn.close();
    }
    catch(SQLException ex)
    {
        ex.printStackTrace();
    }
}

```

```

        return EmpSalList;
    }
}

```

The `selectEmpSalByCriteria()` method provided in Listing 23.56 returns the salary of an employee as an object of the `ArrayList` class. The `insertEmployeeAgreement()` method takes an object of the `EmployeeAgreement` type and inserts the details of the salary statement in the `EMPLOYEE AGREEMENT` table. The values to be inserted are obtained from the passed object.

Listing 23.57 provides the code of the `insertEmployeeAgreement()` method of the `PayrollBeanMethods` class:

Listing 23.57: Showing the Code of the `insertEmployeeAgreement()` Method of the `PayrollBeanMethods` Class

```

public int insertEmployeeAgreement(EmployeeAgreement inEmployeeAgreement)
{
    int recupd = 0;
    String lQuery = "";
    lQuery = lQuery + "insert into EMPLOYEE AGREEMENT values ( ";
    lQuery = lQuery + " "+inEmployeeAgreement.emp_id+" ";
    lQuery = lQuery + " , '"+inEmployeeAgreement.emp_name+"'";
    lQuery = lQuery + " , '"+inEmployeeAgreement.level_id+"'";
    lQuery = lQuery + " , '"+inEmployeeAgreement.allowance_type+"'";
    lQuery = lQuery + " , '"+inEmployeeAgreement.allowance_name+"'";
    lQuery = lQuery + " , "+inEmployeeAgreement.amt+"";
    lQuery = lQuery + " , '"+inEmployeeAgreement.taxable+"'";
    lQuery = lQuery + " , "+inEmployeeAgreement.percentage+"";
    lQuery = lQuery + " , '"+inEmployeeAgreement.agreement_date+"'";
    lQuery = lQuery + ")";
    System.out.println("lSqlString==:"+lQuery);
    try
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
        Statement stmt = conn.createStatement();
        recupd = stmt.executeUpdate(lQuery);
        if( conn != null)
            conn.close();
    }
    catch(SQLException ex)
    {
        ex.printStackTrace();
    }
    return recupd;
}

```

The `insertEmployeeAgreement()` method described in Listing 23.57 takes the `EmployeeAgreement` object as an argument and inserts employee details in the `EMP_SAL` table. The other method, `deleteEmployeeAgreement()`, deletes a salary statement record from the `EMPLOYEE AGREEMENT` table. Listing 23.58 provides the code of the `deleteEmployeeAgreement()` method of the `PayrollBeanMethods` class:

Listing 23.58: Showing the Code of the `deleteEmployeeAgreement()` Method of the `PayrollBeanMethods` Class

```

public void deleteEmployeeAgreement(String inEmpId , String inAllowanceName)
{
    try
    {
        DriverManager.registerDriver(new
            oracle.jdbc.driver.OracleDriver());
        Connection conn= DriverManager.getConnection(DBUrl,DBUser,DBPswd);
        Statement stmt = conn.createStatement();
        String lQuery = "";
        lQuery = lQuery +"delete from EMPLOYEE AGREEMENT ";
        lQuery = lQuery +" where emp_id='"+inEmpId+"' and "
            " allowance_name='"+inAllowanceName+"' ";
        System.out.println("lSqlString==:"+lQuery);
        stmt.executeQuery(lQuery);
        if( conn != null)

```

```

        conn.close();
    }
    catch(SQLException ex)
    {
        ex.printStackTrace();
    }
}

```

The deleteEmployeeAgreement() method, specified in Listing 23.58, deletes the salary details of a specific employee from the EMP_SAL table.

After creating the required Java source files, you need to compile all the .java files and place their .class files in the WEB-INF/classes folder along with their package directory.

Let's now design the JSP views for the Payroll module.

Designing JSP Views

In this section, let's design the JSP pages for the Payroll module. These JSP pages provide forms to enter the details related to the salary of the employees, such as salary statement of a particular employee and allowances granted to an employee. In this module, you need to create the employee_agreement, employee_agreement_edit, salary_search, and salary_slip JSP pages. Let's first create the employee_agreement JSP page.

Creating the employee_agreement JSP Page

The employee_agreement JSP page displays the values of various fields, such as employee id, employee name, department, and the designation of the employee. In addition, this page allows you to enter the values in the Allowance Name and Allowance Type fields. You also need to specify whether the salary lies in the taxable slab or not; if the tax needs to be deducted from the salary, you need to specify the percentage for the same.

Listing 23.59 provides the code of the employee_agreement.jsp file (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.59: Showing the Code of the employee_agreement.jsp File

```

<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.Employee.*" %>
<%@ page import="com.Payroll.*" %>
<%@ page import="java.util.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Breakup of Salary</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2" ><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td width="900" valign="top"><%@ include file="../jsp/people_default_menu.jsp" %></td>
</tr>
<tr>
<td width = "750" valign="top">
<p>&nbsp;</p>
<table border="0" width=100% align=center>
<%
    String dbopr = "";
    dbopr = (String)session.getAttribute("dbopr");
    EmployeeDBObj employeeDBObj = new EmployeeDBObj();
    employeeDBObj = (EmployeeDBObj)session.getAttribute("employeeDBObj");
%>
<tr>

```

```

<td bgcolor ='#AAAAAA' class=whitetext align=center height=20><b>Salary Heads
Distribution</b></td>
</tr>
<form name="form1" method="post">
<%
  if(dbopr != null && ( dbopr.equals("employee_agreement") || 
  dbopr.equals("edit_head") || dbopr.equals("delete_head") )){ 
%>
<tr>
<td>
<table border='0' width=400 align=center>
<tr><td>Employee Id</td>
<td align='left' >
<%=employeeDBObj.emp_id%>
<input type='hidden' name='emp_id' id='emp_id' size='10' value=
'<%=employeeDBObj.emp_id%>' />
</td></tr>
<tr><td>Employee Name</td>
<td align='left'>
<input type='hidden' name='emp_name' id='emp_name' size = '10' value='<%=employee
DBObj.emp_f_name%> <%=employeeDBObj.emp_m_name%> <%=employeeDBObj.emp_l_name%>' />
<%=employeeDBObj.emp_f_name%> <%=employeeDBObj.emp_m_name%> <%=employee
DBObj.emp_l_name%>
</td></tr>
<tr><td>Department</td>
<td align='left'><%=employeeDBObj.dept_id%></td></tr>
<tr><td>Designation</td>
<td align='left'><%=employeeDBObj.level_id%>
<input type='hidden' name='level_id' id='level_id' size = '10' value='<%=employee
DBObj.level_id%>' />
</td></tr>
<tr><td>Agreement Date</td>
<td align='left'> <input type='hidden' name='agreement_date' id='agreement_date'
size = '10' value='<%=employeeDBObj.dojoin%>' />
<input type='text' disabled='disabled' name='d_agreement_date' id='d_agreement_
date' size = '10' value='<%=employeeDBObj.dojoin%>' /> (yyyy-mm-dd)</td></tr>
<tr><td colspan=2>
<table border=0>
<tr>
<td>Allowance Name</td>
<td align='left'>
<SELECT name='allowance_name' >
<OPTION VALUE=></OPTION>
<OPTION VALUE=Basic>Basic</OPTION>
<OPTION VALUE=HRA>HRA</OPTION>
<OPTION VALUE=PF>PF</OPTION>
<OPTION VALUE=SPLAL>SPLAL</OPTION>
<OPTION VALUE=DA>DA</OPTION>
</SELECT>
</td>
<td>Allowance Type</td>
<td align='left'>
<SELECT name='allowance_type' >
<OPTION VALUE=></OPTION>
<OPTION VALUE=Income>Income</OPTION><OPTION VALUE=Deduction>Deduction</OPTION>
</SELECT>
</td>
</tr>
<tr>
<td>Amount</td>
<td align='left'><input type='text' name='amt' id='amt' size = '10' value=''/></td>
<td>Taxable</td>
<td align='left'>
<SELECT name='taxable' >
<OPTION VALUE=></OPTION>

```

Section F

```
<OPTION VALUE=Yes>Yes</OPTION><OPTION VALUE=No>No</OPTION>
</SELECT>
</td>
</tr>
<tr>
<td>Percentage</td>
<td align='left'><input type='text' name='percentage' id='percentage' size ='10' value=''/></td>
<td></td>
<td></td>
</tr>
</table>
</td>
</tr>
<tr>
<td>
<input type='submit' name='submit' id='submit' size ='10' value='Submit Detail' />
<input type='hidden' name='action_submit' id='action_submit' size ='10' value='emp_agreement_dtl_submit' />
</td>
</tr>
</table>
</td></tr><tr><td>
<%
}
ArrayList employeeAgreementList = new ArrayList();
employeeAgreementList = (ArrayList)session.getAttribute("employeeAgreementList");
if ( employeeAgreementList != null && employeeAgreementList.size() > 0){
out.println("<table border=0 align=center>");
out.println("<tr bgcolor='#AAAAAA'>");
if(dbopr != null && !( dbopr.equals("employee_agreement") ||
dbopr.equals("edit_head") || dbopr.equals("delete_head")) ){
    out.println("<td class=whitetext align=center>Employee Id</td>");
    out.println("<td class=whitetext >Employee Name</td>");
}
%>
<td class=whitetext align=center>Allowance Type</td>
<td class=whitetext align=center>Allowance Name</td>
<td class=whitetext align=center>Amount</td>
<td class=whitetext align=center>Taxable</td>
<td class=whitetext align=center>Percentage</td>
<td class=whitetext align=center>Opr</td>
<td class=whitetext align=center>Opr</td>
</tr>
<%
double totalsalary = 0;
for ( int rec = 1; rec <= employeeAgreementList.size(); rec++ ){
    EmployeeAgreement employeeAgreement = new EmployeeAgreement();
    employeeAgreement = (EmployeeAgreement)employeeAgreementList.
    get(rec-1);
    out.println("<tr bgcolor ='#AAAAAA'>");
    if(dbopr != null && !( dbopr.equals("employee_agreement") ||
dbopr.equals("edit_head") || dbopr.equals("delete_head")) ){
        out.println("<td align='center'>"+employeeAgreement.emp_id+"</td>");
        out.println("<td align='center'>"+employeeAgreement.emp_name+"</td>");
    }
%>
<td align='center' ><%=employeeAgreement.allowance_name%> </td>
<td align='center' ><%=employeeAgreement.allowance_type%> </td>
<td align='center' ><%=employeeAgreement.amt%></td>
<%
    if ( employeeAgreement.allowance_type.equals("Income")){
        totalsalary      =      totalsalary + employeeAgreement.amt;
    }
}
```

```

%>
<td align='center' ><%=employeeAgreement.taxable%></td>
<td align='center' ><%=employeeAgreement.percentage%></td>
<td align='center' bgcolor="#AAAAAA">
<a href='http://localhost:8080/people-mgmt/servlet/people_payroll?dbopr=edit_head
&&emp_id=<%=employeeAgreement.emp_id%>&&allowance_name=<%=employeeAgreement.allowa
nce_name%>' class=yellowlink>Edit</a>
</td >
<td align='center' bgcolor="#AAAAAA">
<a href='http://localhost:8080/people-mgmt/servlet/people_payroll?dbopr=delete_he
ad&&emp_id=<%=employeeAgreement.emp_id%>&&allowance_name=<%=employeeAgreement.allo
wance_name%>' class=yellowlink>Delete</a>
</td >
</tr>
<% } %>
</table>
</td>
</tr>
<tr>
<td align=center class=boldblack>Gross Salary:<%=totalSalary%></td>
</tr>
<% } %>
</table>
<hr bgcolor="#AAAAAA">
</td>
</tr>
<tr>
<td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr>
</table>
</body>
</html>

```

The code provided in Listing 23.59 is used to design a salary slip for an employee. All employees have their own salary statements according to which their salary is calculated. To enter a new salary statement, click the Employee Agreement submenu under the Payroll menu. The `employee_agreement` JSP page is displayed, in which you need to enter values for various fields, such as Basic, HRA, Provident Fund, and Medical.

Figure 23.24 shows the output form of the `employee_agreement` JSP page:

www.peoplemanagementsolutions.com/BreakUp of Salary Windows Internet Explorer

Favorites Suggested Sites Get More Add-ons

www.peoplemanagementsolutions.com/BreakU...

People Management Solutions

Hi, Kogent | Employee | Recruitment | Time Management | PayRoll |

Employee Id	1
Employee Name	Sujeet null Kumar
Department	TS
Designation	TS
Agreement Date	(yyyy-mm-dd)
Allowance Name	Basic
Allowance Type	Income
Amount	8000
Taxable	Yes
Percentage	
<input type="button" value="Submit Details"/>	

Done Internet | Protected Mode: On

Figure 23.24: Displaying the `employee_agreement` JSP Page Containing the Salary Statement Details

In the employee_agreement JSP page, you need to enter the respective details and click the Submit Detail button. As a result, the people_payroll servlet re-executes and updates the data in the database. Further, the user is redirected to the employee_agreement JSP page that shows salary details.

Figure 23.25 shows the two salary heads that have been added for a particular employee:

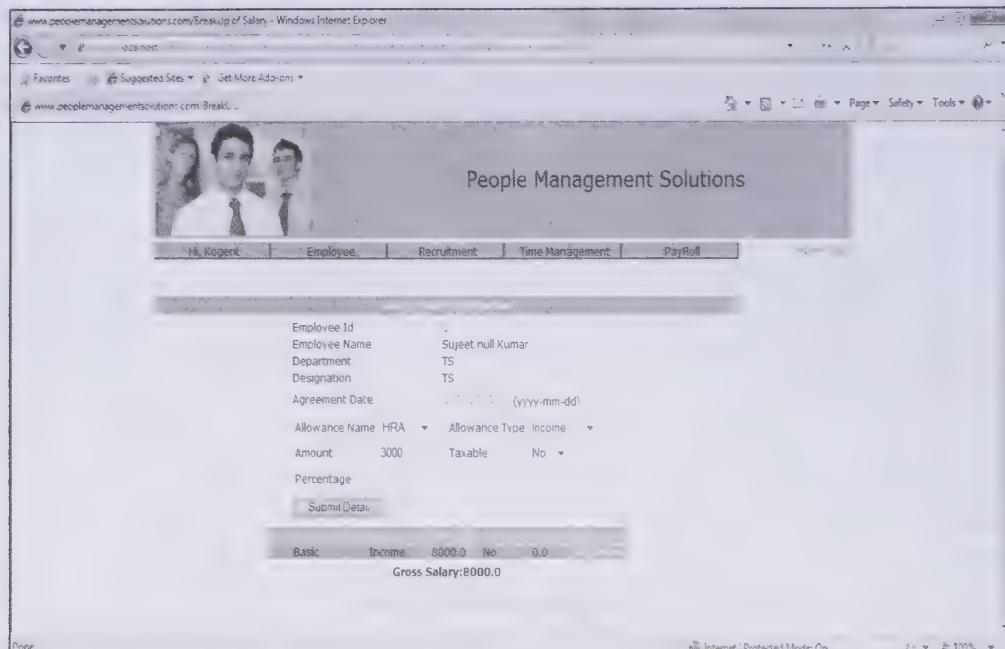


Figure 23.25: Displaying the employee_agreement JSP Page with Salary Head Details

Let's now create the employee_agreement_edit JSP page.

Creating the employee_agreement_edit JSP Page

When you click the Edit link in the employee_agreement JSP page (Figure 23.25), the request is forwarded to the employee_agreement_edit JSP page. This page appears as a form filled with data to be edited for a particular salary statement.

Listing 23.60 shows the code of the employee_agreement_edit.jsp file (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.60: Showing the Code of the employee_agreement_edit.jsp File

```
<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.Payroll.*" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Employee Salary Edit</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2"><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td width="900"><%@ include file="../jsp/people_default_menu.jsp" %></td>
</tr>
<tr>
```

```

<td width = "730" valign="top" align = "center">
<table border="0" align="top" width=100%>
<%
EmployeeAgreement employeeAgreement = new EmployeeAgreement();
employeeAgreement = (EmployeeAgreement)session.getAttribute("employeeAgreement")
%>
<form name="form1" method="post">
<p>&nbsp;</p>
<table align=center>
<tr>
<td colspan=4 bgcolor=#AAAAAA class=whitetext align=center height=20><b>Employee
Salary BreakUp</b></td>
</tr>
<tr><td>Employee Id</td>
<td><input type='text' name='emp_id' id='emp_id' size ='10' value='<%=employee
Agreement.emp_id%>'/></td>
<td>Name</td>
<td><input type='text' name='emp_name' id='emp_name' size ='30' value='<%=employee
Agreement.emp_name%>' /></td>
</tr>
<tr><td>Designation</td>
<td><input type='text' name='level_id' id='level_id' size ='10' value='<%=employee
Agreement.level_id%>'/></td>
<td></td><td></td></tr>
<tr>
<td>Allowance Name</td>
<td>
<input type='hidden' name='allowance_name' id='allowance_name' size ='5' value='
<%=employeeAgreement.allowance_name%>'/>
<input type='text' disabled='disabled' name='allowance_name_dup' id='allowance_
name_dup' size ='5' value='<%=employeeAgreement.allowance_name%>'/>
</td>
<td>Allowance Type</td>
<td><SELECT name='allowance_type' > <OPTION VALUE=></OPTION> <OPTION VALUE=Income>
Income</OPTION><OPTION VALUE=Deduction>Deduction</OPTION></SELECT></td>
</tr>
<tr>
<td>Amount</td>
<td><input type='text' name='amt' id='amt' size ='5' value='<%=employee
Agreement.amt%>'/></td>
<td>Taxable</td>
<td><SELECT name='taxable'> <OPTION VALUE=></OPTION> <OPTION VALUE=Yes>Yes
</OPTION><OPTION VALUE=No>No</OPTION></SELECT></td>
</tr>
<tr>
<td>Percentage</td>
<td><input type='text' name='percentage' id='percentage' size ='5' value='
<%=employeeAgreement.percentage%>'/></td>
<td>Date</td>
<td><input type='text' name='agreement_date' id='agreement_date' size ='10'
value='<%=employeeAgreement.agreement_date%>'/> (yyyy-mm-dd)</td>
</tr>
<tr>
<td colspan=4><input type='submit' name='submit' id='submit' size ='10' value=
'Edit'/>
<input type='hidden' name='action_submit' id='action_submit' size ='10' value=
'employee_sal_head_edit_submit'/'> </td>
</tr>
</table>
</td></tr>
<tr>
<td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr>
</table></body></html>

```

Figure 23.26 displays the output of the `employee_agreement_edit.jsp` page for a particular employee:

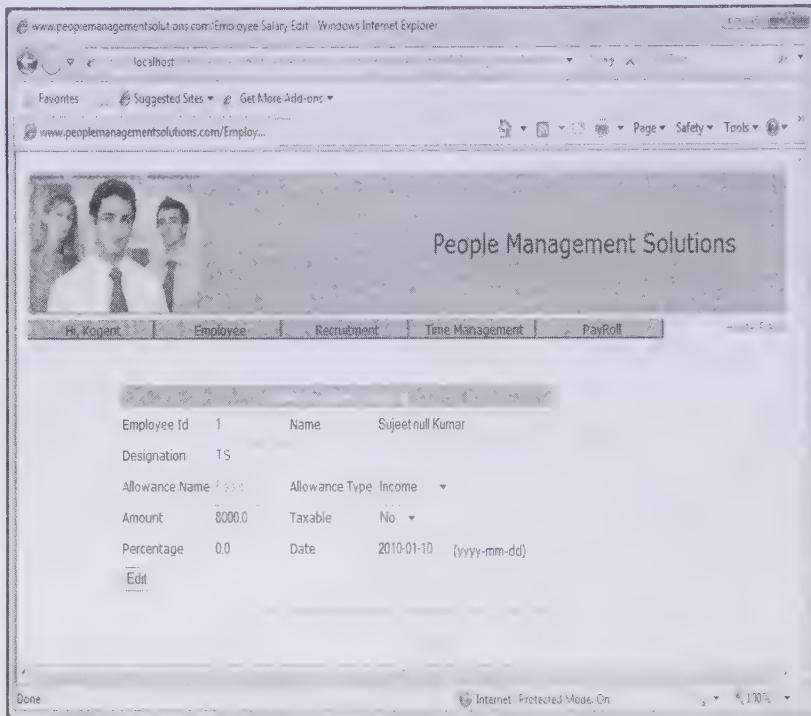


Figure 23.26: Displaying the employee_agreement_edit JSP Page to Edit Salary Head Details

Figure 23.26 shows the salary details of an employee whose employee id is 1.

Let's now create the salary_search JSP page.

Creating the salary_search.jsp File

The salary_search JSP page displays a search form used to search the salary details of a particular employee. The form submission consequently generates a salary slip of that employee for a given month, showing all the salary details.

Listing 23.61 provides the code for the salary_search.jsp file (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.61: Showing the Code of the salary_search.jsp File

```
<%@ page language="java" %>
<%@ page session="true" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Salary</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2"><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td width="900"><%@ include file="../jsp/people_default_menu.jsp" %></td>
</tr><tr>
<td width ="750" valign="top" align = "center">
<p>&nbsp;</p>
<div align=center class=boldblack>Calculate Salary</div>
<hr width=400 color="#AAAAAA">
<table border="0" align="top" width=200 align="right">
```

```

<form name="form1" method="post">
<tr><td>Employee Id</td>
<td align='left'><input type='text' name='emp_id' id=emp_id size=10 value=''/>
</td></tr>
<tr><td>Year</td>
<td align='left'><select name='year' >
<option value=></option>
<%
    for(int i=2010;i>2000;i--)
        out.println("<option value='"+i+"'>"+i+"</option>");
    %>
</select></td></tr>
<tr><td>Month</td>
<td align='left'><SELECT name='month'>
<option value=></option>
<%
    for(int i=12;i>0;i--)
        out.println("<option value='"+i+"'>"+i+"</option>");
    %>
</SELECT></td></tr>
<tr><td align='center' colspan='2' >
<input type='submit' name='submit' id='submit' size ='10' value='Calc' />
<input type='hidden' name='action_submit' id='action_submit' size ='10' value='
salary_calc_submit' />
</td></tr>
</table><hr width=400 color=#AAAAAA>
</td>
</tr>
<tr>
<td colspan="2"><%@include file="../jsp/people_footer.jsp"%></td>
</tr>
</table></body></html>

```

In Listing 23.61, the salary_search JSP page is designed, which contains three fields, Employee Id, Year, and Month. When the user fills these fields and clicks the Calc button, the data is retrieved from the EMPLOYEE AGREEMENT table.

Figure 23.27 displays the salary_search JSP page:

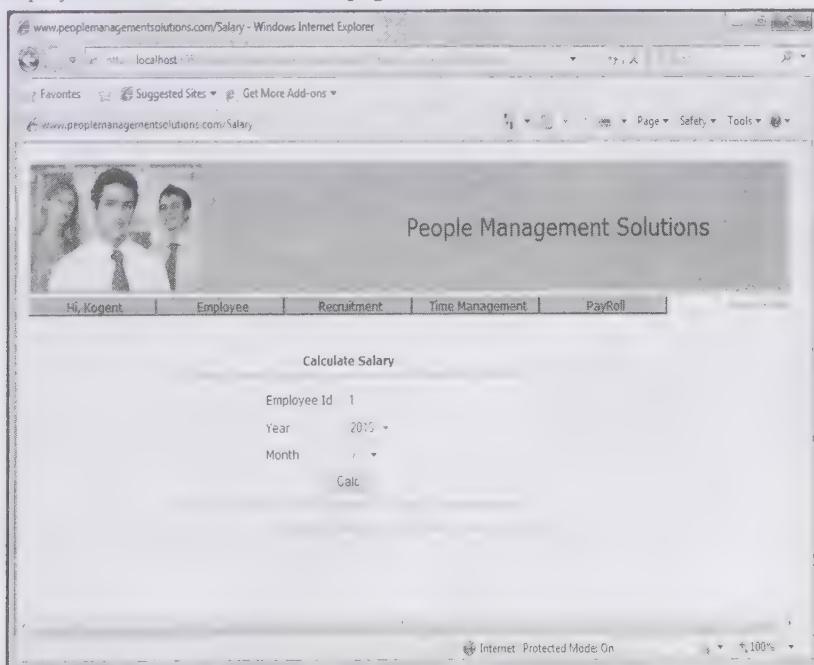


Figure 23.27: Displaying the salary_search JSP Page with Fields for Salary Slip Detail

Figure 23.27 shows the salary_search JSP page used to find the salary details of the employee having the Employee Id, 1.

Let's now create the salary_slip JSP page.

Creating the salary_slip JSP Page

The salary_slip JSP page displays a salary slip of the employee whose details have been provided in the salary_search JSP page. In addition to the employee id, employee name, department, and designation, the generated salary slip shows various allowances given to the employee, amount of each allowance, total attendance of the employee, total number of leaves taken by the employee in that month, and the total salary for that particular month.

Listing 23.62 provides the code for the salary_slip.jsp file (you can find this file in the PeopleMgmt\people-mgmt\jsp folder on CD):

Listing 23.62: Showing the Code of the salary_slip.jsp File

```
<%@ page language="java" %>
<%@ page session="true" %>
<%@ page import="com.Employee.*" %>
<%@ page import="com.Payroll.*" %>
<%@ page import="java.util.*" %>
<%@ page import="java.text.DecimalFormat" %>
<html>
<head>
<title>www.peoplemanagementsolutions.com/Salary_Slip</title>
<link rel="stylesheet" href="../css/mystyle.css" type="text/css" />
</head>
<body>
<table width="900" border="0" align="center">
<tr>
<td colspan="2" ><%@ include file="../jsp/people_header.jsp" %></td>
</tr>
<tr>
<td width="900" valign="top"><%@ include file="../jsp/people_default_menu.jsp" %></td></tr>
<tr>
<td width = "750" valign="top">
<table border="0" width=500 align=center>
<%
    String dbopr = "";
    dbopr = (String)session.getAttribute("dbopr");
    EmployeeDBObj employeeDBObj = new EmployeeDBObj();
    employeeDBObj = (EmployeeDBObj)session.getAttribute("employeeDBObj");
    int totalAttendance = 0;
    int totalLeave = 0;
    int year = 0;
    int month = 0;
    totalAttendance= Integer.parseInt((String)session.getAttribute(totalAttendance));
    totalLeave = Integer.parseInt((String)session.getAttribute("totalLeave"));
    year = Integer.parseInt((String)session.getAttribute("year"));
    month = Integer.parseInt((String)session.getAttribute("month"));
%>
<p>&nbsp;</p>
<hr bgcolor="#AAAAAA" width=500>
<form name="form1" method="post">
<tr>
<td colspan=4 align=center bgcolor="#AAAAAA" height=20 class=whitetext>Salary Slip
For <%=month%>/<%=year%></td>
</tr>
<tr><td>Employee Id</td>
<td><%=employeeDBObj.emp_id%>
<input type='hidden' name='emp_id' id='emp_id' size ='10' value='<%=employeeDBObj.emp_id%>'/></td>
</tr>
<tr><td>Employee Name</td>
<td>
<%=employeeDBObj.emp_f_name%>
<%
    if(!"null".equals(employeeDBObj.emp_m_name))
        out.print(employeeDBObj.emp_m_name);
%>
```

```

%>
<%=employeeDBObj.emp_l_name%>
<input type='hidden' name='emp_name' id='emp_name' size ='10' value=
'<%=employeeDBObj.emp_f_name%>
<%=employeeDBObj.emp_m_name%> <%=employeeDBObj.emp_l_name%>' /></td>
</tr>
<tr><td>Department</td><td><%=employeeDBObj.dept_id%></td></tr>
<tr><td>Desination</td><td>
<%=employeeDBObj.level_id%>
<input type='hidden' name='level_id' id='level_id' size ='10' value=
'<%=employeeDBObj.level_id%>' /></td>
</tr>
</table>
<%
 ArrayList empsalList = new ArrayList();
 empsalList = (ArrayList)session.getAttribute("empsalList");
 if ( empsalList != null && empsalList.size() > 0){
%>


|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <%  double totalsalary = 0;  double taxAmt = 0 ;  for ( int rec = 1; rec <= empsalList.size(); rec++ ){     EmpSal empsal = new EmpSal();     empsal = (EmpSal)empsalList.get(rec-1);     out.println("<tr bgcolor ='#AAAAAA'>");     out.println("<td align='center' >"+empsal.allowance_name+"</td>");     if ( empsal.allowance_type.equals("Income") ){         out.println("<td align='center' >(+)" +empsal.amt+"</td>");         totalsalary += totalsalary + empsal.amt;     }     else     if ( empsal.allowance_type.equals("Deduction") ){         out.println("<td align='center' >(-)" +empsal.amt+"</td>");         totalsalary -= totalsalary - empsal.amt;     }     if ( empsal.taxable.equals("Yes") ){         taxAmt = taxAmt + (empsal.amt * empsal.percentage/100);     }     out.println("</tr>"); } %> <tr> <td class=boldblack>Total Present: <%=totalAttendance%></td> <td class=boldblack>Total Leave: <%=totalLeave%></td> </tr> <tr> <td class=boldblack>Total Salary: <%=totalsalary%></td> <td class=boldblack>Tax: <%=taxAmt%></td> <%  double monthSalary=0.0;  double deduction=0.0;  String output="";  String monthsalary="";  if(totalLeave != 0) {     deduction= totalsalary/30;     deduction = deduction*totalLeave;     monthSalary = totalsalary - deduction;     DecimalFormat myFormatter = new DecimalFormat("#####.##");     monthsalary=myFormatter.format(monthSalary);     output = myFormatter.format(deduction); } %> </tr> <tr> <td class=boldblack>Deduction : <%=output%> Rs.</td> <td class=boldblack>Month's Salary: <%=monthsalary%> Rs.</td> </tr> </table> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|


```

```

<%}
String lErrorMsg="";
lErrorMsg = (String)session.getAttribute("lErrorMsg");
out.println("<div align=center class=boldred"+lErrorMsg+"</div>");
%>
</table>
</td></tr>
<tr>
<td colspan="2">
<%@include file="../jsp/people_footer.jsp"%>
</td>
</tr></table></body>
</html>

```

Listing 23.62 shows the code to design the salary slip of the specified employee.

Figure 23.28 displays the salary_slip JSP page:

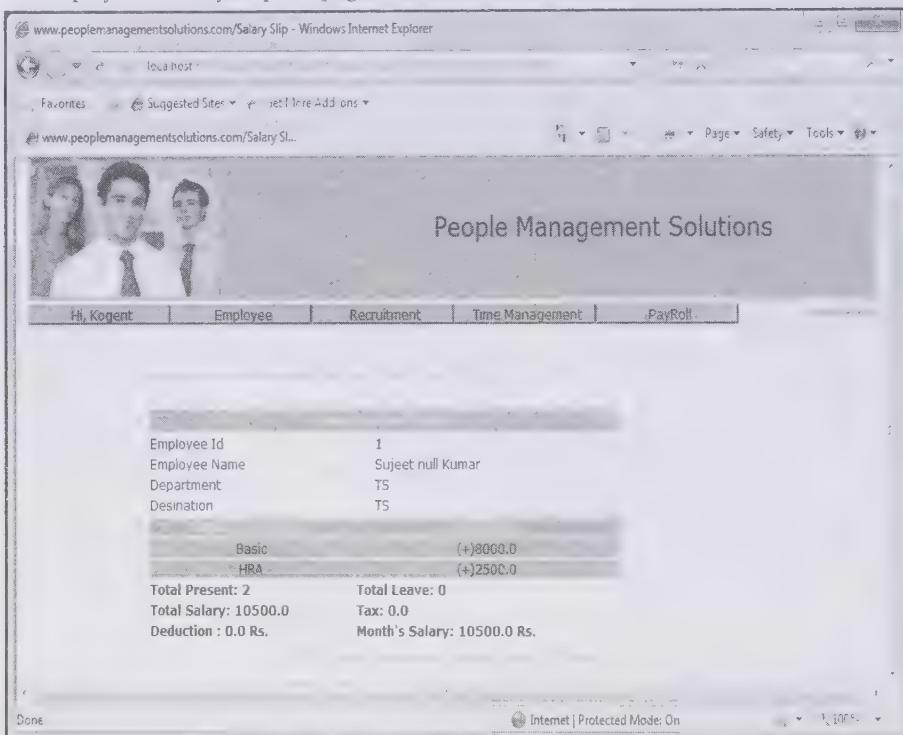


Figure 23.28: Displaying the salary_slip JSP Page

Figure 23.28 displays the salary slip of an employee for the 7th Month of the 2010 year.

This section has explained the logic behind the development of the Payroll module that can be used to handle salary statements of all employees of an organization. This module generates salary slips of all the employees for a given month by considering number of leaves and number of attendances.

As all modules of the ProjectMgmt project are designed in similar manner; therefore, a reader can go through a single module to understand the implementation logic of the MVC architecture. Every module of this project has its own controller servlet, helper Java classes, and the JSP pages that are used as user interfaces. The model part has been implemented using data transfer objects that are being populated manually using ResultSet. The JSP pages have been used as View in the architecture. You should note that the project is designed to maintain the reusability of the code throughout the modules. In other words, all servlets and almost all JSPs are being reused in some way; thereby, decreasing the number of files created in the project.

A

AJAX

Asynchronous JavaScript and XML (AJAX) is a new technique, which describes how other technologies, such as JavaScript, Document Object Model (DOM), and EXtensible Markup Language (XML), can be used together to create interactive Web applications. In early days, a Web application created by using these technologies was not efficient and platform-independent, as compared to a desktop based application. To overcome this, Jesse James Garrett of Adaptive Path combined JavaScript, XML, and DOM to form a new technique, called AJAX.

In this technique, the request to the Web server is sent by using the XMLHttpRequest object. This object, a part of the JavaScript technology, helps in sending asynchronous requests to the server. With this request, Web applications can now interact with the Web server asynchronously. The time taken to refresh the page is also minimized, which enhances the efficiency of the Web application.

In this appendix, we trace the evolution of Web applications and the technologies used for developing them. We then discuss the problems associated with these technologies that were used to create Web applications in early days, and how these problems led to the development of the AJAX technique. Finally, we create a sample AJAX-based application.

Evolution of Web Applications

In earlier times, applications had their own client-based program that required to be configured on the client machine. Both the server and the client needed an upgrade when there were any changes made in the application. However, with the advent of Web-based applications, client-server applications changed a lot. A Web application provides a series of Web pages to the client, which is accessed by all types of clients using any browser of their choice. There is no need to install a separate client program on all client machines, as the Web browser interprets and displays all Web pages and works as a common client for a Web application. Therefore, we can now develop Web-based client-server application without spending time on creating separate client programs for different client machines.

In earlier times, we could access simple, static Hypertext Markup Language (HTML) pages using a Web browser, which sends a request to a Web server. The Web server then sends the requested web page, which is stored at the server using Hypertext Transfer Protocol (HTTP). These Web pages display static content, i.e. a constant state or a text file that does not change. The ever evolving technology lead to the requirement for designing a Web application that processes the data given by the client and presents dynamic content to the client arose, which was not possible through static Web pages. To overcome this problem, the evolution of Web application took place, which led to the development of technologies for processing the client request and generating response content dynamically. With the emergence of this new concept, new technologies also took birth. Therefore, the evolution of a Web application lead to the development of new technologies, which helped in creating Web applications. The following are few of the technologies, which can be used to create a Web application:

- Common Gateway Interface (CGI)

- Applets
- JavaScript
- Servlets
- JavaServer Pages (JSP)
- Active Server Pages (ASP)
- Hypertext Processor (PHP)
- Dynamic HTML (DHTML)
- XML

Common Gateway Interface

CGI is a standard protocol for interfacing the external application software with an information server, commonly known as Web server. Figure A.1 shows the processing of CGI:

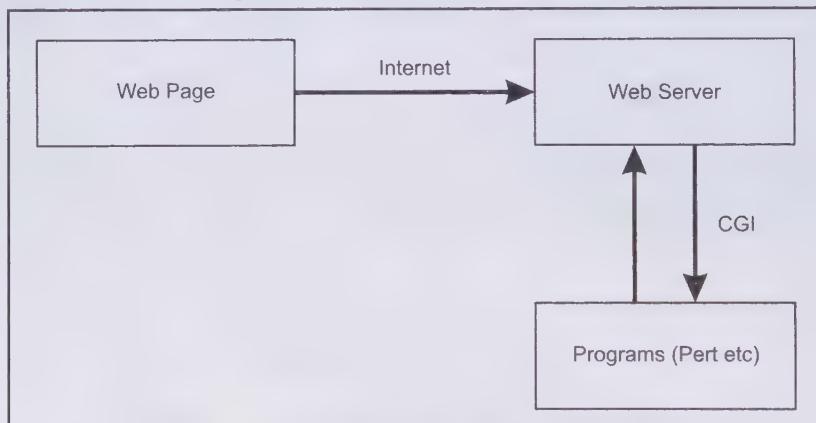


Figure A.1: Displaying How CGI works

CGI is not a programming language; rather it is a protocol that defines a set of rules on how the Web server communicates with the program. This functionality allows the server to pass the request from the client Web browser to the external application. In fact, CGI is a specification used to transfer information between the Web server and the CGI program. A CGI program can be written in any programming language, such as C, Perl, and Java, etc. CGI programs help the Web server to interact dynamically with Web users, e.g. a CGI program is used to process the form's data when it is submitted once. CGI also allows HTML pages to interact with applications rather than a static Web page. Some of the advantages of CGI are as follows:

- It is simple and quick to develop.
- It is rich in libraries. Therefore, there is no need to provide code to create the required class; instead, the developers need to provide code to implement the classes of the libraries provided by CGI.

The following are some of the limitations of CGI:

- CGI is slow; being slow is the flaw of CGI itself and not of the particular programming language used for scripting.
- Each time the process must be launched from the beginning and that takes a lot of time.
- The resources, such as the database connections, must be created and reloaded every time.
- The state is not persistent. On every request, a new state of a user is built.

Now let's move on to another technology called applets, related to Web applications.

Applets

An applet is a program written in Java programming language, which can be included in the HTML page and run within a Web browser. Java applets are normally used to include small, interactive components to a Web

page. The applets are mainly used to provide dynamic user-interface and a number of graphical effects for the Web pages. When the Web client or the Web browser opens the Web page, the applet automatically is downloaded, similar to an image. However, with the applet, you cannot control the amount of space that the applet takes up on screen.

JavaScript

Initially, Netscape invented a simple scripting language known as LiveScript. LiveScript was a proprietary add-on to the HTML. When Sun's new programming language, Java, became popular, Netscape quickly switched over and came up with a new scripting language called JavaScript. The first four alphabets of the two technologies, Java and JavaScript, are quite similar; otherwise both are entirely different from each other.

JavaScript is the scripting language, which is used in many websites by the Web designer to create a client-side application with very less effort. The scripting language is interpreted at runtime and not compiled like other languages, such as C++, C#. Therefore, JavaScript is an interpreted language, which means that the scripts execute without compilation. JavaScript is also the client-side language, as it runs on the client browser. JavaScript can be used in almost all the Web browsers, such as Internet Explorer, Mozilla Firefox, Netscape and it can easily interact with the HTML elements.

Basically, JavaScript is designed to create interactivity with HTML pages. The following are the uses of JavaScript:

- ❑ Allows anyone to put the small snippets of the code into their HTML pages, as JavaScript is simpler to understand.
- ❑ Enables writing of dynamic text into HTML page. The variable text can also be written in HTML page, e.g. `document.write(" <h1>" + name + " </h1>")`. This command writes the text of the name variable into the HTML page.
- ❑ Enables you to read and change the content of HTML controls. For example, the text inserted in the text field of an HTML page can be read with the help of JavaScript.
- ❑ Allows you to perform certain validations on the client-side. For example, ensuring that no text field is left blank, matching the passwords, and confirming the entries in the password fields while setting a new password, can be checked at client-side by using JavaScript as the scripting language.
- ❑ Allows you to create cookies that can be used to either store or retrieve relevant information on the client's computer.
- ❑ Enables you to load a specific page depending upon the client's request.
- ❑ Allows functions that are embedded in or included from HTML pages and interact with the DOM of the page.
- ❑ Allows you to change an image as the mouse cursor moves over it.
- ❑ Helps to call the new Web page, according to the client or user's action.

Till now, you were getting acquainted with the basic technologies related to Web application programming. Now, let's understand the technologies required at the time of Web application development.

Servlets

Java Servlet is an alternative to CGI programs. The major difference between CGI and servlets is that the servlets are persistent. In other words, once loaded, it stays to fulfill other subsequent requests. On the contrary, CGI disappears after fulfilling the request once. Moreover, similar to the applets that run on the browser, the servlets run on Java-enabled Web server.

The Java Servlet Application Programming Interface (API) allows the developer to add dynamic content to a Web server using Java platform. The servlets maintain the state across multiple requests by using HTTP cookies, session variables, Uniform Resource Locator (URL) rewriting, or the hidden fields. The Servlet API contains the `javax.servlet` package hierarchy. The Servlet API also specifies the expected interaction of the Web container and a servlet. The Web container is part of the Web server and performs numerous tasks. Some of these tasks include interacting with the servlets, managing the life cycle of the servlets, and mapping the URL to a particular servlet, if the URL requester has the correct access rights.

The main purpose of the servlets is to serve HTML to the client, mainly through HTTP protocol. Servlets are created, managed, and destroyed by the Web server, where they run. The servlets are recognized in the context of the Web server.

JavaServer Pages

JSP is an enhancement to the Java Servlet technology, offered by Sun Microsystems. The JSP technology provides a technique to dynamically generate Web pages. JSP also simplifies the process of creating or developing web-based applications.

The JSP technology allows HTML to be combined with Java on the same page. In response to the Web client's request, JSP allows software developers to dynamically generate HTML documents. The JSP technology allows Java code and certain pre-defined actions to be embedded into static content. The Java code is added with the use of the JSP directives, JSP scripting elements, and JSP actions. The JSP syntax adds additional XML-like tags, called JSP actions. The JSP compiler compiles JSP into servlets. The JSP compiler generates a servlet in Java code, which is then compiled by the Java compiler. The extension of the JSP files is .jsp.

Figure A.2 shows the different clients connecting to the Web server through the Internet:

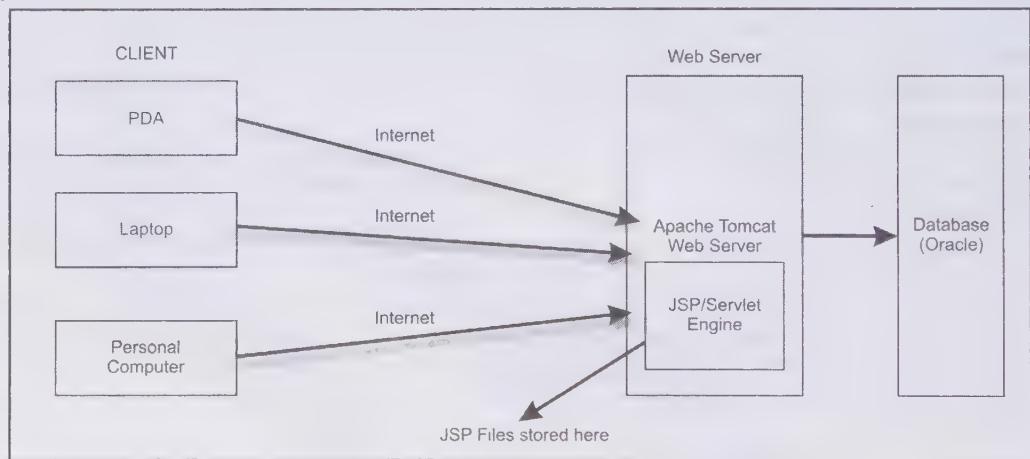


Figure A.2: Different Clients Connected to the Web Server

Figure A.2 shows the most popularly used Web server, Apache Tomcat Web server, running on Windows. As shown in Figure A.2, the JSP files run on the Web server in the JSP Servlet engine. The JSP Servlet engine dynamically generates the HTML output and sends it to the client's browser.

Active Server Pages

ASP is the Microsoft's server-side scripting engine for dynamically generating HTML or the Web pages for the Web browser. The default scripting language used to write ASP is VBScript. The simple approach to understand ASP is that ASP is a program that runs inside Internet Information Server (IIS). Similar to JSP, i.e. a server-side technology given by Sun Microsystems, ASP is the server-side technology provided by Microsoft.

The ASP page, though similar to the HTML page, also contains text, XML, and scripts. The scripts in an ASP file are executed on the server. The extension of the ASP files is .asp. When the Web browser sends the request for an ASP file, the IIS passes the request to the ASP engine, which then executes the scripts in the file after reading it line by line. Later, in the plain HTML format, the ASP file is returned to the Web browser.

The modification and additions to the contents of the Web page can be done dynamically with the help of ASP. The data from the database can also be accessed with the help of ASP. In addition, the result is returned to the Web browser. ASP, in comparison to CGI and Perl, is simpler and faster.

Hypertext Processor

Till now, you were aware of Sun Microsystems and Microsoft technologies, which are JSP and ASP, respectively. Let's move on to understand another server-side scripting technology called PHP, which is an alternative to ASP and JSP. Hypertext Processor, more commonly known as PHP, helps to create dynamic Web pages. You can embed PHP into HTML pages and generate a dynamic Web page. PHP programs can be deployed on a Web server. PHP uses the concept of CGI for server-side programming. It acts as a filter for displaying the dynamic content and can be used for extracting data from a database.

Dynamic HTML

Before studying DHTML in detail, let's first expand the term DHTML. DHTML stands for Dynamic Hypertext Markup Language and is the art of making the HTML pages dynamic. DHTML is a combination of technologies used to create dynamic and interactive websites and Web applications. In standard HTML, once the page is loaded from the server it will not change until another request is made to the server. On the contrary, dynamic HTML provides more control over HTML elements. It allows the HTML elements to change at any time without returning to the Web server; and uses the DOM, Cascading Style Sheet (CSS), HTML, and JavaScript to develop interactive Web applications.

Document Object Model

DOM allows changing any part of the Web page using DHTML. DOM is the API that serves as glue for binding a scripting language, such as JavaScript, with the markup language, such as HTML. HTML DOM defines the standard set of objects for HTML and allows access and manipulation of HTML objects in a standard way. It helps in specifying each part of the Web page and allows access by using naming conventions.

Cascading Style Sheets

CSS is used in DHTML to control the look and feel of a Web page. CSS is used to style the HTML elements. The font color, font text, background color, images, and the placements of objects on the Web page are defined in the CSS files. CSS allows the designer to control the style and layout of various Web pages. This can be done by defining the style for each HTML element and then can be applied to multiple Web pages at a time. This also enables a quick global change. In other words, if the same style is applied to all the HTML elements, all the elements will get automatically updated by simply making changes in the style. Therefore, DHTML helps in making the Web pages dynamic and provides a good look and feel of the Web page with the help of CSS.

The HTML elements or objects are placed in the Web page by using XHTML. With the help of DOM specified in the Web page, these objects, placed in the Web page, can be accessed or manipulated at any time. Now, let's understand XML.

XML

XML is the extensible markup language used for the exchange of information between applications or organizations. XML allows the designers to create their own user-defined tags and enable the transmission, validation, and interpretation of data between applications or the data between the organizations. In simpler words, it allows the designer to provide data in tags by creating meaningful tags. Some of the XML-related technologies are as follows:

- XHTML
- XML DOM
- eXtensible Stylesheet Language Transformations (XSLT)
- XML Parser

Let's have an overlook to the listed XML-technologies.

XHTML

As discussed earlier, XHTML is a cleaner but stricter version of HTML. XHTML is similar to HTML 4.x, and is defined in the form of an XML application. It consists of elements in HTML 4.01, combined with the syntax of XML. As previously stated, XML is the markup language that results in well-formatted documents. Therefore,

Appendix A

XHTML provides the privilege of writing well-formed documents, which work in all the Web browsers. Certain rules to be followed while using XHTML are as follows:

- XHTML elements should be properly nested
- XHTML elements should always be closed
- XHTML elements should be in lowercase
- XHTML documents must have a single root element

XML DOM

DOM refers to the Document Object Model and presents the XML document as the tree-structure having the Root node as the parent element and the Elements, Attributes, and Text defined as the child nodes. Therefore, XML DOM defines the standard way for accessing and manipulating XML documents. The elements containing the text and the attributes, with the help of the DOM tree, can be manipulated and accessed. The contents of these elements can be modified, new elements can be created, or the unwanted elements can be removed from the DOM tree. The most important thing to be noted is that all the Elements, their Text, and their Attributes are known as the nodes.

In the DOM structure, the entire document is considered as the Document node; XML tag or the XML element is recognized as the Element node; the text in the XML elements are referred to as the Text node; attributes are considered the Attribute nodes; and the comments are considered the Comment node. In the DOM tree-structure, the nodes have a hierarchical relationship with each other. The terms parent and child are used to describe the relationships between the nodes.

Let's consider an example of a XML file and look at its DOM tree-structure. The code for the `products.xml` file containing the data related to various products is provided in Listing A.1:

Listing A.1: Displaying the Code for the `products.xml` File

```
<?xml version="1.0" encoding="UTF-8"?>
<PRODUCTDATA>
    <PRODUCT PRODID="P001">
        <PRODUCTNAME>Barbie Doll</PRODUCTNAME>
        <DESCRIPTION>This is a toy for children in the age group below 5 years</DESCRIPTION>
        <PRICE>$24.00</PRICE>
        <QUANTITY>12</QUANTITY>
    </PRODUCT>
    <PRODUCT PRODID="P002">
        <PRODUCTNAME>Mini Bus</PRODUCTNAME>
        <DESCRIPTION>This is a toy for children in the age group of 5-10 years</DESCRIPTION>
        <PRICE>$42.00</PRICE>
        <QUANTITY>6</QUANTITY>
    </PRODUCT>
    <PRODUCT PRODID="P003">
        <PRODUCTNAME>Car</PRODUCTNAME>
        <DESCRIPTION>This is a toy for children in the age group of 10-15 years</DESCRIPTION>
        <PRICE>$60.00</PRICE>
        <QUANTITY>21</QUANTITY>
    </PRODUCT>
</PRODUCTDATA>
```

In Listing A.1, `<PRODUCTDATA>` is the root element of the document. As all the other elements are within the `<PRODUCTDATA>` element, it is considered as the root element. The root element has three `<PRODUCT>` nodes and an Attribute node named, `PRODID`. Each of the Element nodes has a Text node as well.

Figure A.3 shows the DOM tree-structure for `products.xml`:

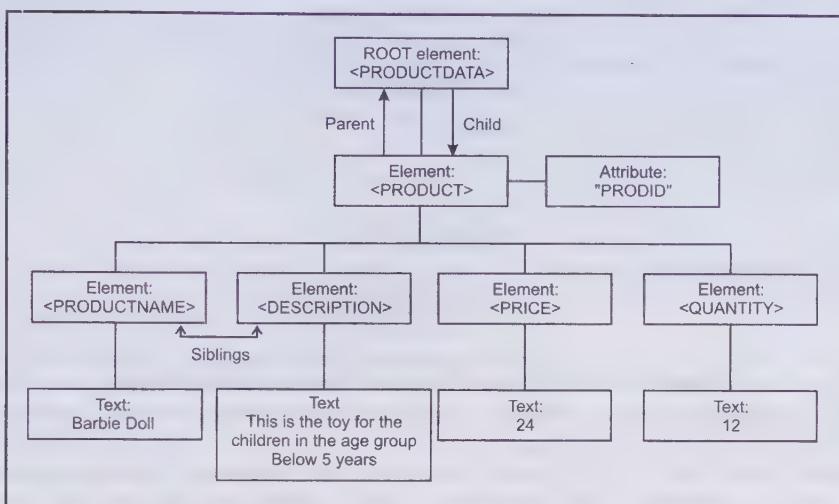


Figure A.3: Displaying the DOM Node Tree-structure

Figure A.3 shows only one child node, <PRODUCT>, of the parent node <PRODUCTDATA>. The <PRODUCT> child node has four Element nodes and an Attribute node. Each Element node has the respective Text node, as shown in Figure A.3.

XSLT

XSLT stands for eXtensible Stylesheet Language Transformations, and represents a language used for transforming XML documents into XHTML or other XML documents. XSLT uses XPath for navigating through XML documents and finding information in it. In the transformation process, XSLT uses the XPath searches for those parts of the source document that match the pre-defined template. When a match is found, XSLT transforms the source document into the resultant document by applying the pre-defined template.

XML Parser

The XML parser is used to read, update, create, and manipulate XML documents. For manipulating the XML document, the XML parser loads the document into the computer's memory and then manipulates data by using the DOM node-tree-structure. The XML parser is the part of the software that reads the XML files and tests whether the XML document is well-formed against the given Document Type Definitions (DTD) or the XML schema. Moreover, the XML parser also makes the XML files available to the application with the use of the DOM.

Till now, the appendix has dealt with the explanation of almost all the technologies used to create Web applications. All these technologies discussed earlier share a common problem, and AJAX proves to be the solution for these problems. Read on to know about them.

Problems of the Traditional Technologies

All the previously mentioned technologies use the Classical or traditional Web application model. In the Classical or the traditional Web application model, the nature of interaction between the client and the server is of start-stop. In a traditional Web application model, the browser responds to the user action by discarding the current HTML page. Then, the request is sent back to the Web server and when the server completes the processing of request, it returns the response page to the Web browser. Finally, the browser refreshes the screen and displays the new HTML page. You will be surprised to note that the user is bound not to do anything on the Web pages until the entire process is completed.

In technical terms, the problem with the classical Web applications model was the synchronous request-response communication model. This can be explained with the help of Figure A.4:

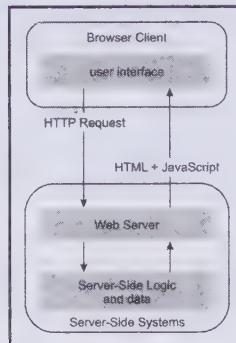


Figure A.4: Displaying the Classical Web Application Model

The Classical Web Application model, shown in Figure A.4, makes technical sense; however it does not provide the best user experience. As this classical application model keeps the user waiting, it does not provide the best user experience. To overcome this, the developer noticed a technical approach that Google used. After analyzing the facts of Google, on February 18, 2005 Jesse James Garrett, President, and founder of the Adaptive Path, came out with a new technique AJAX—that was based on the approach used by Google.

Let us move on further to learn about AJAX.

AJAX—the Solution

AJAX, a new approach to Web applications, is based on several technologies that help to develop applications with better user experience. It uses JavaScript and XML as the main technology for developing interactive Web applications. These applications are based on AJAX Web application model, which uses JavaScript and XMLHttpRequest object for asynchronous data exchange. The JavaScript uses XMLHttpRequest object to exchange data asynchronously over the client and server. Let's move further to have a detail study on the AJAX Web application Model.

AJAX Web Application Model

You already know that the major issue with regard to the Classical Web application model was resolved through AJAX. The AJAX application eradicates the start-stop-start-stop nature or the click, wait, and refresh criteria of the client-server interaction. Figure A.5 shows how the intermediary layer is introduced between the user and the Web server:

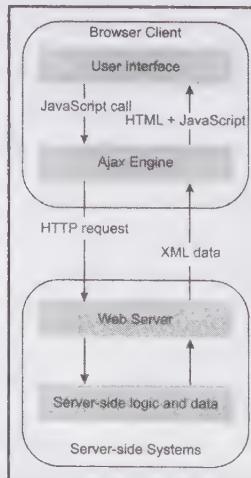


Figure A.5: Displaying the AJAX Web Application Model

Instead of loading the Web page during the beginning of the session, the browser loads the Ajax engine, written in JavaScript. As shown in Figure A.5, the Web page sends its requests using a JavaScript function. This JavaScript code makes a request to the server. The server response comprises of data and not the presentation, which implies that the data required by the page is provided by the server as the response, and the style or presentation is implemented on that data with the help of the markup language. Most of the page does not change. The parts of the page that need to change are updated. In other words, JavaScript dynamically updates the Web page, without redrawing everything. For the Web server, nothing has changed; it still responds to each request, just as it did earlier!

Though JavaScript makes a request to the server, you can still type in Web forms and even click buttons, while the Web server is still working in the background. Then, when the server completes its processing, your code updates just the part of the page that has changed. This way, you do not have to wait around for the entire cycle to be completed, which reflects the power of asynchronous requests.

AJAX engine, between the user and the application, irrespective of the server, does asynchronous communication. This prevents the user from waiting for the server to complete its processing. The AJAX engine takes care of displaying the user interface and the interaction with the server on the user's behalf.

However, in traditional Web applications, the synchronous mode of communication existed between the client and the server, as shown in Figure A.6:

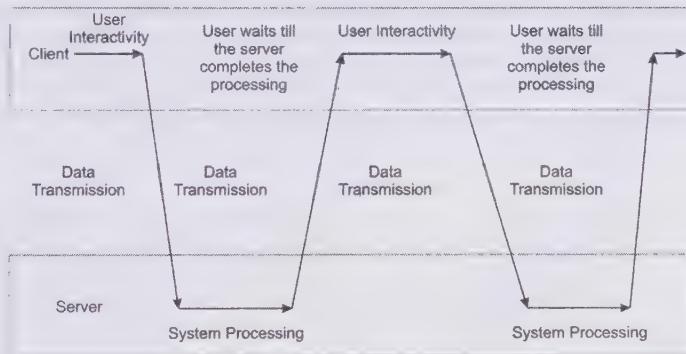


Figure A.6: Displaying the Synchronous Mode of Communication

As the essence of AJAX is a partial screen update and the asynchronous communication, the programming model, shown in Figure A.7, is not bound to a specific data exchange format or the specific programming language or the specific communication mechanism. Figure A.7 shows the asynchronous mode of communication:

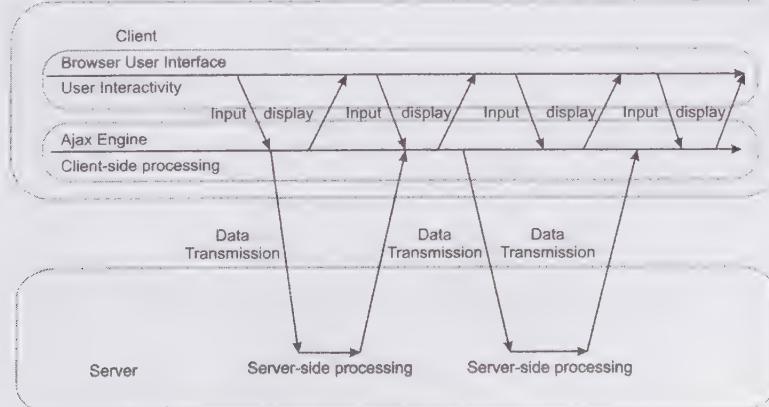


Figure A.7: Displaying Asynchronous Mode of Communication

As shown in Figure A.7, every user action generates an HTTP request that takes the form of a JavaScript to call the AJAX engine. Any response to the user action does not require the trip back to the server, unlike the Classical Web application model. Rather, the AJAX engine handles on its own; i.e., the data validation, some navigational functions, editing data in memory, and so on, are handled by the AJAX engine.

If the AJAX engine needs to retrieve new data from the server or load additional interface code, the engine makes an asynchronous interaction with the server, using JavaScript and XMLHttpRequest object for asynchronous data exchange. The engine's interaction with the server does not interrupt the user's interaction with the application. In this way, asynchronous communication is done with the help of the AJAX engine.

Comparing Figures A.6 and A.7, it can be seen that in the asynchronous mode of interaction, there is no scope for the user to wait until the server-side processing gets over. The AJAX Web application model allows users to continue working, and simultaneously, if necessary, the AJAX engine interacts with the server without interrupting the user's interaction with the application.

After learning how AJAX works and how the problems or shortcomings of traditional technologies are overcome by using AJAX, let's create an AJAX application by using JavaScript.

Creating a Sample AJAX Application

Let's consider a scenario of a Jewelry showroom. The owner wants to design a Web page that would display the different jewelry items, along with some information, to its various customers. When the user points to an image whose information he wants to know, the details will be displayed on the Web page without the page being refreshed repeatedly. Let's create a jewelry application. In this application, we first need to create a Jewelery.html page, which shows information about the different jewelry in the showroom.

The code for the Jewelery.html page of the application is provided in Listing A.2 (you can find this file on CD in code\JavaEE\AppendixA\jewelry folder):

Listing A.2: Displaying the Code for the Jewelery.html File

```
<html>
<head>

<title>First AJAX Application</title>
<script language = "javascript">
    var XMLHttpRequestObj = false;
    if (window.XMLHttpRequest)
    {
        XMLHttpRequestObj = new XMLHttpRequest();
    }
    else if (window.ActiveXObject)
    {
        XMLHttpRequestObj = new
        ActiveXObject("Microsoft.XMLHTTP");
    }
    function getData(dataSource, divID)
    {
        if(XMLHttpRequestObj)
        {
            var obj = document.getElementById(divID);
            XMLHttpRequestObj.open("GET", dataSource);

            XMLHttpRequestObj.onreadystatechange = function()
            {
                if (XMLHttpRequestObj.readyState == 4 &&
                    XMLHttpRequestObj.status == 200)
                {
                    obj.innerHTML = XMLHttpRequestObj.responseText;
                }
            }
        }
    }
</script>
</head>
<body>
    <div id="div1">This is a sample text</div>
    
</body>

```

```

        }

        XMLHttpRequestObj.send(null);
    }

}

</script>
</head>
<body>
<H1>First Application using AJAX</H1>



<div id="targetDiv">
    <h1>Welcome to my Jewelery Showroom!</h1>
</div>
</body>
</HTML>

```

In Listing A.2, an HTML page is designed, displaying the images of the various jewelry items available in the showroom. As soon as the user points the mouse pointer on any of the three images, the text saved in the respective text files are displayed on the Web browser. The three text files are as follows:

- bangles.txt (Listing A.3)
- rings.txt (Listing A.4)
- necklaces.txt (Listing A.5)

You also need to add a web.xml file in the WEB-INF folder of the application in which Jewelery.html page is mapped as the welcome page. Now, let's understand how the Jewelery.html page uses AJAX. When the mouse moves over any of the three images, the onmouseover event is generated and the JavaScript method, getData, is called, as shown in the following code snippet:

```

<body>
    <H1>First Application using AJAX</H1>
    
    
    
    <div id="targetDiv">
        <h1>Welcome to my Jewelery Showroom!</h1>
    </div>
.</body>

```

The getData method passes two text strings—first, the name of the text file, such as bangles.txt, rings.txt, or necklaces.txt, and secondly the name of the <div> element.

The text provided in the bangles.txt file is given in Listing A.3 (you can find this file on CD in the code\JavaEE\AppendixA\jewelry folder):

Listing A.3: Showing the Text Displayed on Hovering the Mouse on Bangles Image

```

We offer too many bangles to list!
Gold Bangles
Diamond bangles

```

The text provided in the rings.txt file is given in Listing A.4 (you can find this file on CD in the code\JavaEE\AppendixA\jewelry folder):

Listing A.4: Showing the Text Displayed on Hovering the Mouse on Rings Image

```

Rings: Amazing rings with wonderful designing.
Heart-shaped rings
Diamond rings
gold rings
Gold plated rings

```

The text provided in the necklaces.txt file is given in Listing A.5 (you can find this file on CD in the code\JavaEE\AppendixA\jewelry folder):

Listing A.5: Showing the Text Displayed on Hovering the Mouse on Necklace Image

The all kind of diamond and gold necklaces are also available.

Diamond necklace

Gold necklace

The necklaces are also designed according to your choice.

Apart from the preceding text files, three image files of the bangles, rings, and necklaces are also displayed on the Web page.

Any of these three text files is downloaded by the browser from the server, which is in the background, while the user is working with the rest of the Web page. Read on further to understand how this is done.

To run the Jewelry application, ensure that a Web server is configured on your machine. Here, we are using Glassfish as a Web server for running AJAX-based Web application. Package and deploy the Jewelry application on Glassfish Application Server. Follow these steps to run the application:

- Open the Internet Explorer (IE).
- Type the following address (`http://<Ip address of Web Server>:8080/Jewelery/`) in the address bar of the IE and press enter to open the page (Figure A.8).

Figure A.8 shows the output of the jewelry application:

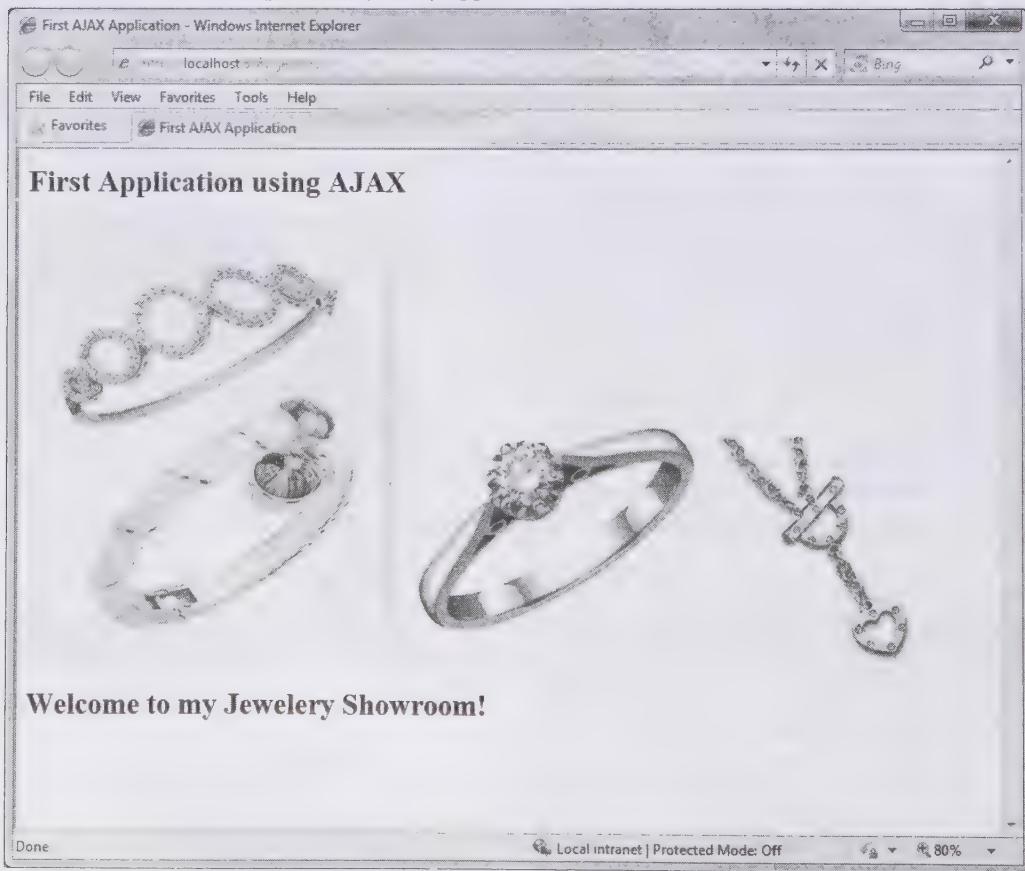


Figure A.8: Displaying the Output of the Jewelry Application

- When you move the mouse pointer on any of the images, the text related to that image get displayed, as shown in the Figure A.9:

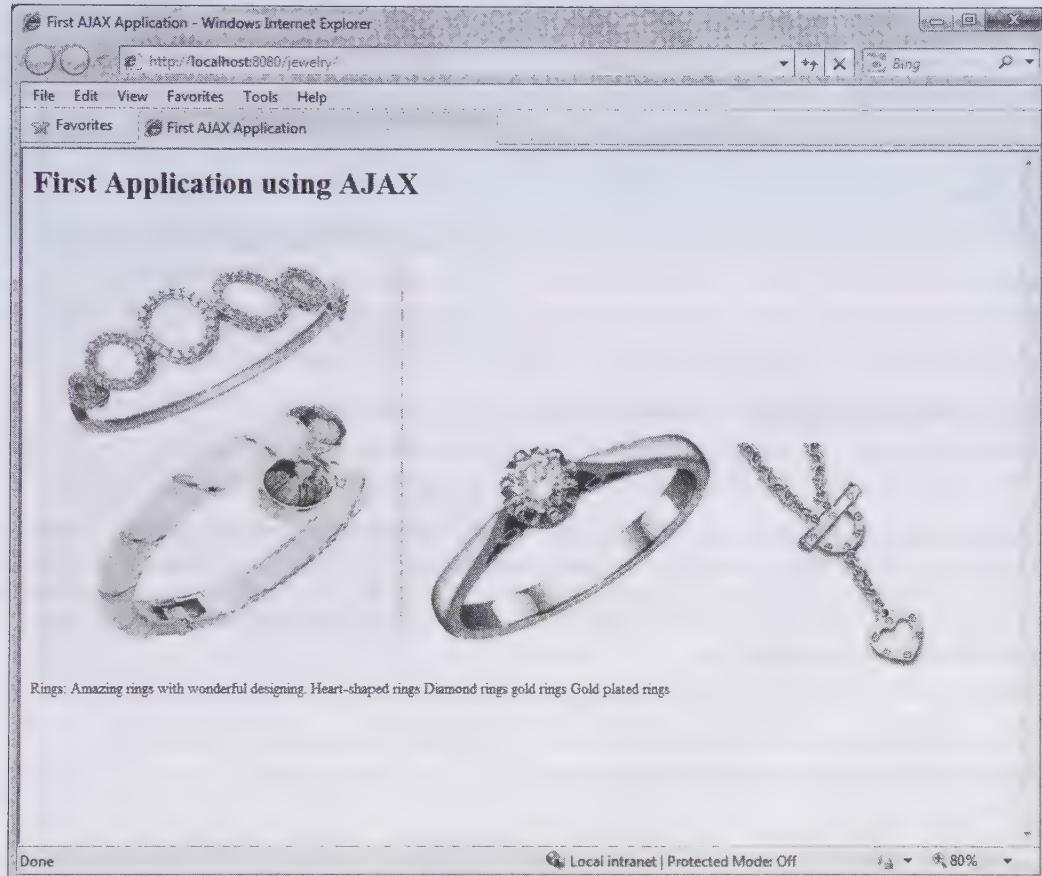


Figure A.9: Displaying a Simple AJAX Example

As the mouse moves from one image to another, the JavaScript in the page fetches some new text and replaces the older text, without even a screen flicker or page fetch or fuss.

Creating the XMLHttpRequest Object

The application created in this subsection requires the XMLHttpRequest object. In Listing A.2, towards the beginning of Jewelry.html code, locate the following code snippet:

```
<script language = "javascript">
var XMLHttpRequestObj = false;
```

The preceding code snippet declares a variable XMLHttpRequestObj. As this code snippet is outside any function, it runs immediately when a JSP page is loaded. This variable is set to false, so that the script can check later whether the variable is created or not. In case of browsers, such as Netscape, FireFox, and Opera, the XMLHttpRequest object is usually the part of the browser's window object and can be accessed as window.XMLHttpRequest. If the window.XMLHttpRequest returns true, an XMLHttpRequest object is created with the following code snippet:

```
if (window.XMLHttpRequest)
{
    XMLHttpRequestObj = new XMLHttpRequest();
}
```

On the contrary, a different perspective is required for the Internet Explorer Web browser. The ActiveX object in the Internet Explorer (version 5 and above) is used to create the XMLHttpRequest object, as shown in the following code snippet:

```
if (window.ActiveXObject)
{
    XMLHttpRequestObj = new
    ActiveXObject ("Microsoft.XMLHTTP");
}
```

Therefore, depending upon the browser you are using, an XMLHttpRequest object is created.

When the user moves the mouse over the images, an “onmouseover” event is generated, which calls the getData() function. When the getData () function is called, the XMLHttpRequest object is first checked to see whether it is valid, and then further processing is done.

Opening the XMLHttpRequest Object for Asynchronous Downloads

When the valid object of the XMLHttpRequest is created, the object calls its open () method. You can configure the object to use the URL you want, by using the object’s open method. The syntax of the open () method is as follows:

```
req.open("GET",URL,true);
```

In the preceding code snippet, the first parameter indicates the type of HTTP method that is used for sending request; the second parameter is the URL of the requested resource and; the third parameter is optional, which shows whether the request is synchronous or asynchronous. The default value of third parameter is true, which indicates an asynchronous request.

In this application, the URL from which the data you want to fetch is passed from the getData function as the dataSource argument. The URL can be opened with the standard HTTP techniques, such as GET, POST, or PUT. The following code snippet uses the GET method to request the respective text file on the server:

```
XMLHttpRequestObj.open("GET", dataSource);
```

When you open the XMLHttpRequest object, the XMLHttpRequest object contains the property named onreadystatechange, which allows handling the asynchronous loading operations. If this property is assigned to any JavaScript function, this function will be called each time the XMLHttpRequest object’s state changes.

This JavaScript function is also known as “Callback” function. When the server returns with the information, the callback function is invoked. In turn, the callback function can display the new information to the user. We have defined the callback function with the following JavaScript code snippet:

```
XMLHttpRequestObj.onreadystatechange = function ()
{
    if (XMLHttpRequestObj.readyState == 4 &&
        XMLHttpRequestObj.status == 200)
    {
        obj.innerHTML =
        XMLHttpRequestObj.responseText;
    }
}
```

Finally, when the XMLHttpRequest object is in its ready state and the status is equal to 200, then the data is fetched. The readyState value “0” indicates that the request is completed and the status 200 refers to the ‘Ok’ state of the XMLHttpRequest object, which means that the request resource is completely downloaded. The five states of the XMLHttpRequest object are as follows:

- 0 for uninitialized state
- 1 for loading state

- 2 for loaded state
- 3 for interactive state
- 4 for the complete state

The status property holds the status of the download. Table A.1 provides some possible values of the status property:

Table A.1: Possible Values for the status Property of XMLHttpRequest Object

Status	Possible values
Ok	200
Created	201
No Content	204
Reset Content	205
Partial Content	206
Bad request	400
Unauthorized status	401
Forbidden status	403
Not Found status	404
Method Not Allowed	405
Not Acceptable	406
Proxy Authentication Required	407
Request Timeout	408
Length Required	411
Requested Entity Too Large	413
Requested URL Too Long	414
Unsupported Media Type	415
Internal Server Error	500
Not Implemented	501
Bad Gateway	502
Service Unavailable	503
Gateway Timeout	504
HTTP Version Not Supported	505

Therefore, to make sure that the data is completely downloaded, the value for the status property must be 200. Finally, when the data is completely downloaded, it is retrieved in either the standard HTML or the XML format. The `responseText` property is used to retrieve the data in standard HTML format. However, if your data is formatted as XML, then `responseXML` property is used.

After retrieving the data, you have to display the data on the Web page. The data is displayed by using the HTML `<div>` element, as shown in the following code snippet:

```
<div id="targetDiv">  
  <h1>Welcome to my Jewelery Showroom!</h1>  
</div>
```

The `<div>` element shows the location where you want to display the data. The `id` attribute is used to identify the `<div>` element and it is passed as an argument to the `getData()` function for `bangles.txt` file. The following code snippet shows the implementation of the `getData()` function passing `bangles.txt` and `targetDiv` as arguments:

```
getData ('bangles.txt', 'targetDiv')
```



Installing Java EE 6 SDK

The Java EE 6 SDK distributions provide a free integrated development kit to build, test, and deploy Java Enterprise Edition (Java EE) based applications. You should note that SDK stands for Software Development Kit. The SDK also supports the newly released Java platform, Standard Edition 6 (Java SE 6). With this all in one bundle, the developers can quickly install, develop, and deploy new enterprise Java technologies. You can download this SDK bundle with or without Java Development Kit (JDK) from the <http://java.sun.com/javase/downloads/index.jsp> URL.

This appendix discusses the system requirements to install the Java EE 6 SDK. In addition, you learn how to install the Java EE 6 SDK on your system.

System Requirements for Installing Java EE 6 SDK

You can install Java EE SDK 6 on various operating systems, such as Windows, Solaris, Linux, and Macintosh. The installation of Java EE 6 SDK on the operating systems supports the root and non-root user installation. Windows users should have administrator rights to install the Java EE 6 SDK bundle, which includes the following components:

- Glassfish v3
- Glassfish v3 Web profile
- Java EE 6 samples
- Tutorial
- API documentation (Java docs)

The developers can also download the NetBeans 6.8 Integrated Development Environment (IDE) that contains the Glassfish v3 application server and allows you to build Java EE 6 applications.

You should ensure that your system satisfies the minimum system requirements for Java EE 6 SDK installation. Table B.1 describes the minimum system requirements for Operating Systems (OS) supporting Java EE 6 SDK:

Table B.1: System Requirements for Supported Operating Systems in SDK Distributions

Operating System	Minimum Memory	Recommended Memory	Minimum Disk Space	Recommended Disk Space	JVM
Sun Solaris 9, 10 (SPARC)	512 MB	512 MB	250 MB free	500 MB free	Java SE 5 and 6
Sun Solaris 9, 10 (x86)					
Redhat	512 MB	1 GB	250 MB free	500 MB free	Java SE 5

Table B.1: System Requirements for Supported Operating Systems in SDK Distributions

Operating System	Minimum Memory	Recommended Memory	Minimum Disk Space	Recommended Disk Space	JVM
Enterprise Linux 4.0					and 6
Macintosh (Intel, Power)	512 MB	512 MB	250 MB free	500 MB free	Java SE 5 and 6
Windows Server 2003, Windows XP, Windows Vista, and Windows 7	1 GB	2 GB	500 MB free	1 GB free	Java SE 5 and 6

After ensuring that your system meets the preceding mentioned requirements and after downloading Java EE 6 SDK, follow the steps discussed in the following section to install the SDK bundle on Windows.

Installing Java EE 6 SDK

When you download the Java EE 6 SDK distributions, you are prompted to save the java_ee_sdk-6-windows.exe file on your system. After the downloading process is complete, perform the following steps to install Java EE 6 SDK on your system:

1. Run the java_ee_sdk-6-windows.exe file to initiate the installation program. The Welcome page of the Java EE 6 SDK wizard appears after a few moments.

Figure B.1 shows the Welcome page of the Java EE 6 SDK wizard:

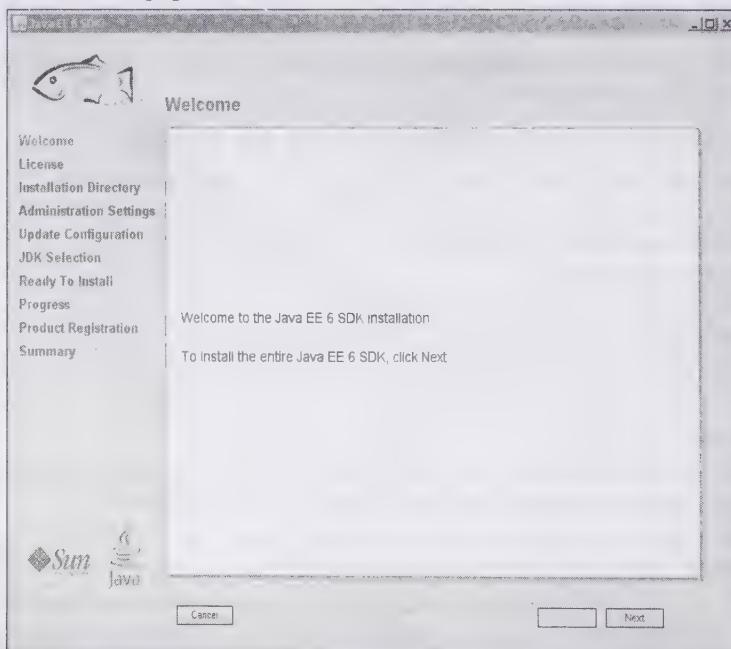


Figure B.1: Displaying the Welcome Screen while Installing Java EE 6 SDK

2. Click the Next button. The License page is displayed (Figure B.2).
3. Read the license agreement and select the I accept the terms in the license agreement radio button, as shown in Figure B.2:

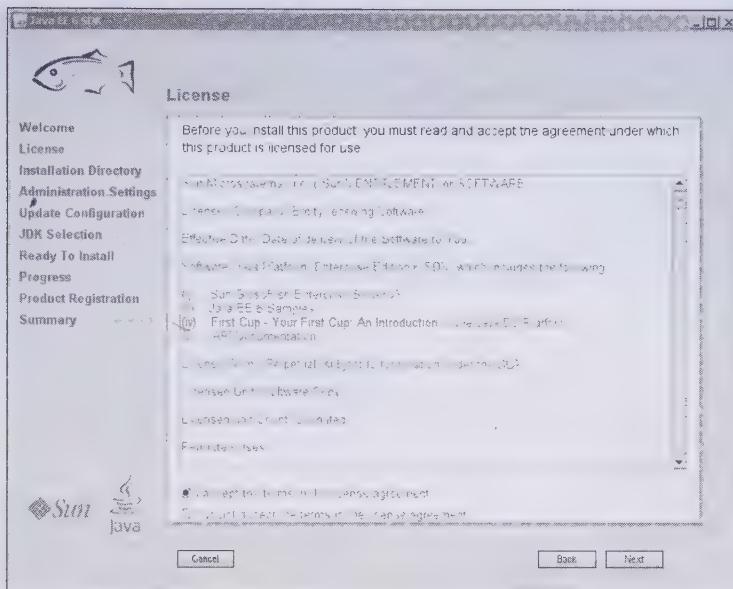


Figure B.2: Displaying the License Page

- Click the Next button (Figure B.2) to continue installation. By default, Java EE 6 SDK is installed in the C:/glassfishv3 directory. You can change the path of installation directory by clicking the Browse button. Then, locate the path where you want to install the SDK bundle. In our case, we select the default directory, as shown in Figure B.3:

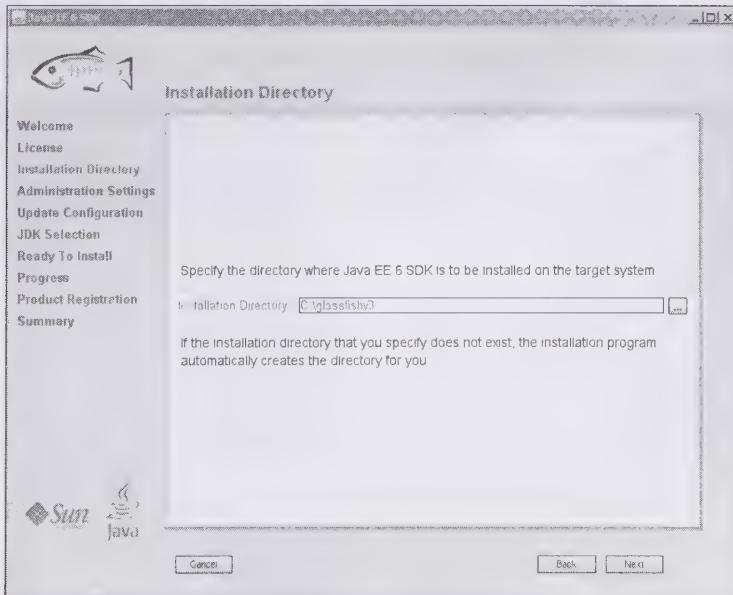


Figure B.3: Displaying the Selection of Installation Directory

- Click the Next button to resume the installation process. The Administration Settings page is displayed, where you need to enter a username and password. You also need to provide the admin and HTTP port numbers, as shown in Figure B.4:

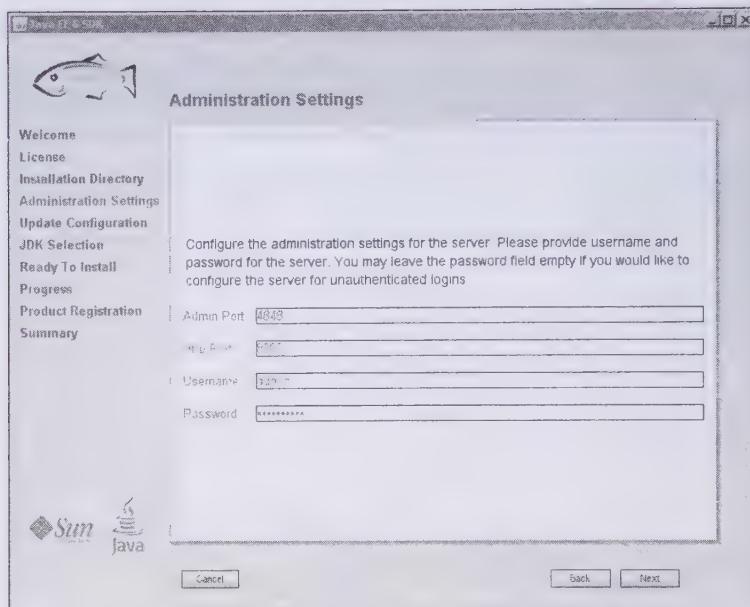


Figure B.4: Displaying the Administration Settings Page

Figure B.4 displays the default port numbers and the default user name as admin.

It is optional to enter the password For the admin user. By default, the blank password is accepted.

After entering the correct password, the JDK Selection page for the server appears.

6. Select the proper location of JDK and by default, if JDK is installed on your system, the installed JDK location is selected. In our case, we continue with the default location and click the Next button, as shown in Figure B.5:

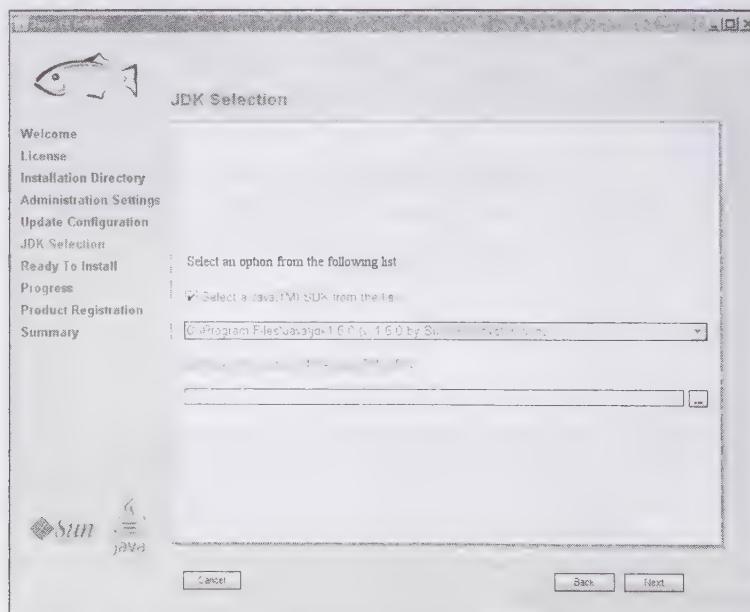


Figure B.5: Displaying the JDK Selection Folder

The Ready to Install page appears, listing the features that are to be installed, as shown in Figure B.6:

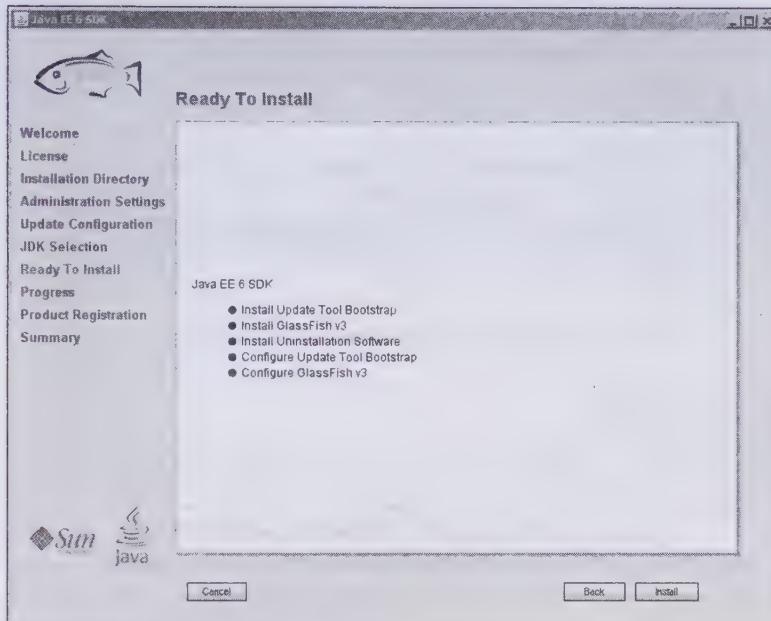


Figure B.6: Displaying the Features to be Installed

- Click the Install button (Figure B.6) to begin the installation process, as shown in Figure B.7.

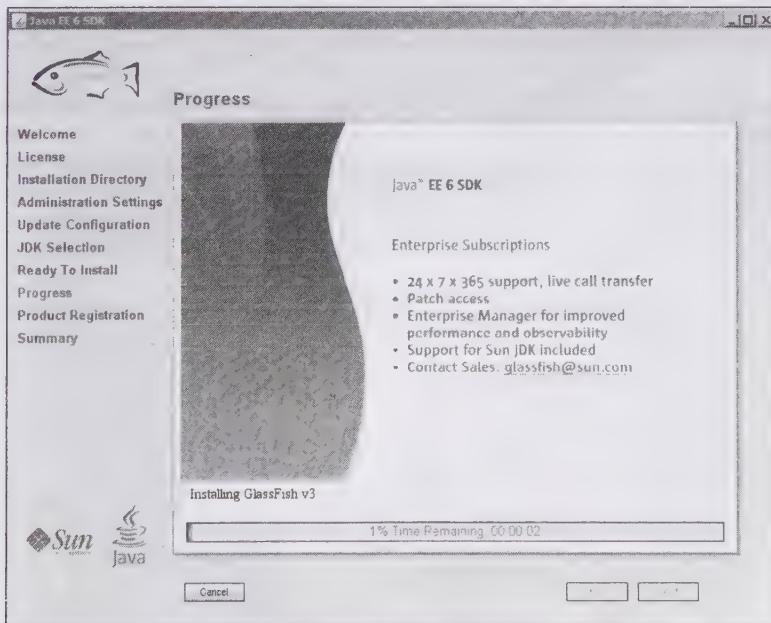


Figure B.7: Displaying the Installation Progress

Figure B.7 displays the progress of the installation process, on the completion of which the Product Registration page appears (Figure B.8). If you want to get the updates about the product from the vendor, you can create a new account on the vendor's site, as shown in Figure B.8:

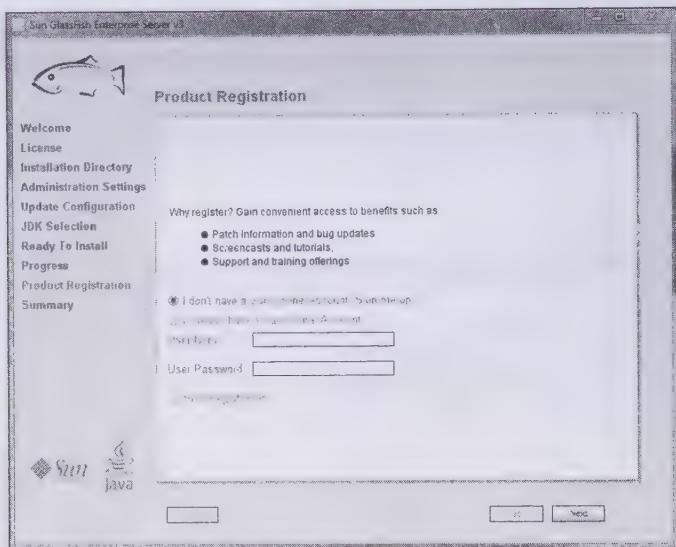


Figure B.8: Displaying the Product Registration Page

Depending upon your choice, you can select the appropriate radio button from the Product Registration page (Figure B.8). For example, if you do not want to register the product, select the Skip Registration radio button. If you already have a Sun online account, select the I already have a Sun Online Account radio button and enter the user name along with the password. In our case, we have selected the I don't have a Sun Online Account radio button, which displays the page where you need to enter the personal details required to create an online account on the Sun Microsystems' website.

8. Enter the personal details, such as email address, phone number, and name. Your Sun online account is created.
9. Click the Next button (Figure B.8). The Summary page appears, containing the information about the installed features, as shown in Figure B.9:

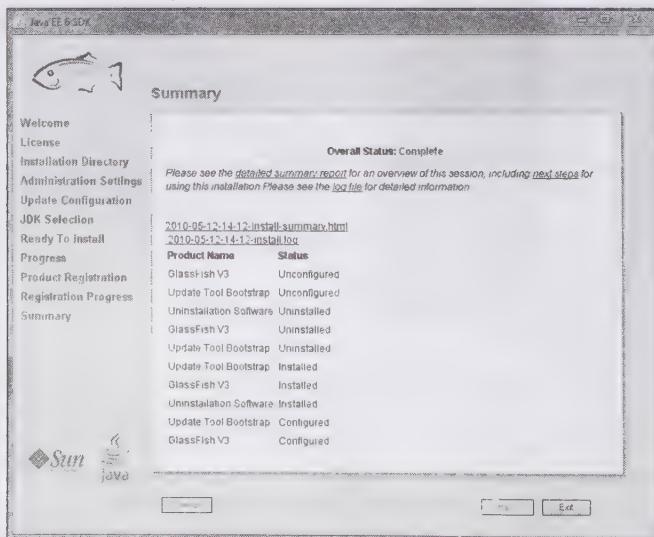


Figure B.9: Displaying the Installation Complete Wizard

10. Click the Exit button to close the Java EE 6 SDK wizard (Figure B.9).

This completes the process of installing the Java EE 6 SDK. You can verify whether or not the Application server is working properly by following these steps:

1. Start the server manually to verify the successful installation by executing the following command from the bin directory located under the installation directory:

```
asadmin start-domain domain1
```

The Application server is started.

2. Open Internet Explorer and navigate to the <http://localhost:8080> URL. The Your server is now running message is displayed, as shown in Figure B.10:

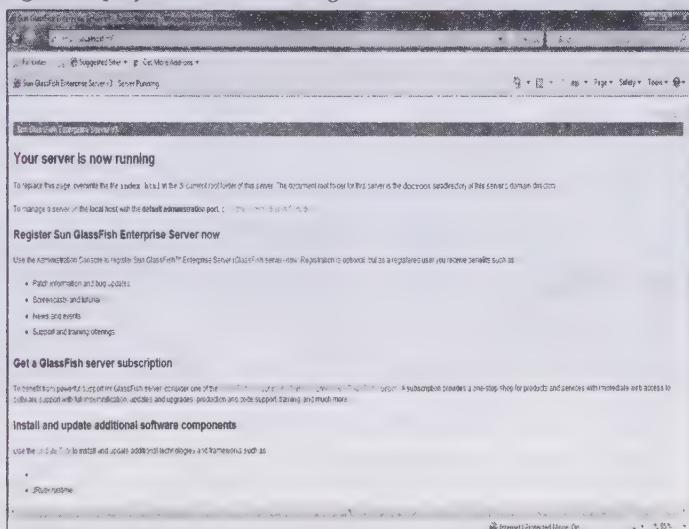


Figure B.10: Displaying the Server Running Window

If the server is running, the browser will be displayed as shown in Figure B.10.

3. Browse <http://localhost:4848> / URL to deploy the Web or enterprise applications. The admin console appears, asking for the login details, as shown in Figure B.11:

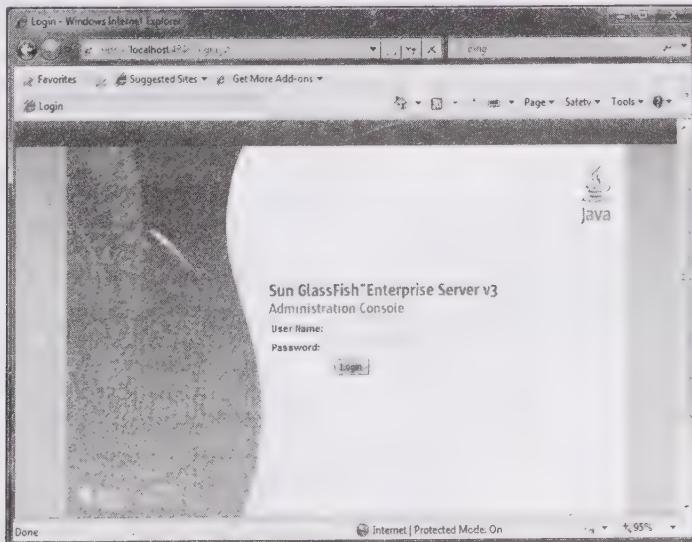


Figure B.11: Displaying the Admin Console

4. Enter admin as the user name and the password that you have entered during installation. After logging in, the index page appears, as shown in Figure B.11:

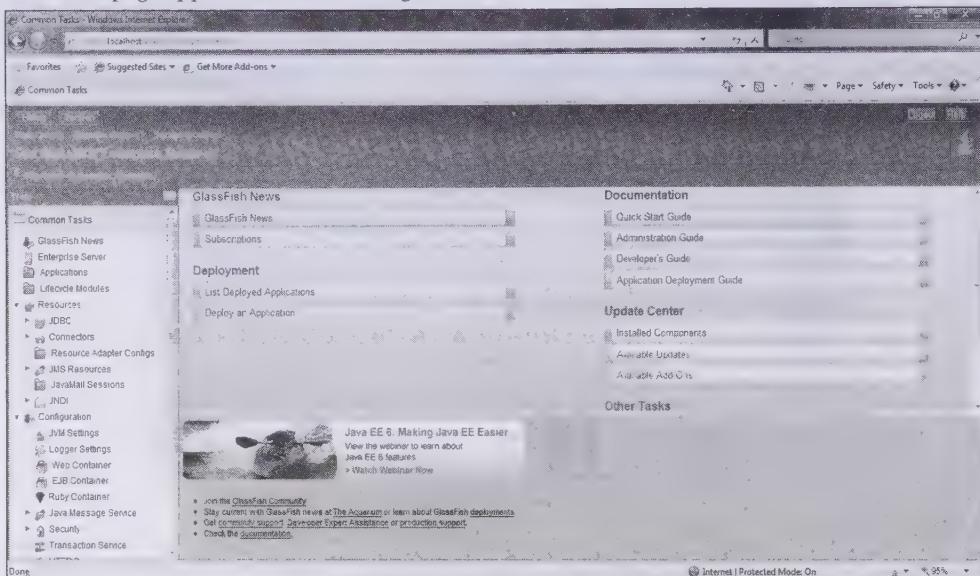


Figure B.12: Displaying the Index Page of Admin Console

Figure B.12 displays the index page of the admin console that is used to deploy Web or enterprise applications. You should note that if you download the SDK bundle without JDK, you need to configure the path of JDK during installation.

C

Working with NetBeans IDE 6.8

In the recent years, there has been a paradigm shift in the complexity and functionality of Java. The primitive approach to develop programs in Java has been to write your Java code in any text editor and then use the compiler in Java Development Kit (JDK) for compilation. This works well in case of simple programs, but would be a tiresome task for complex ones. A good development environment can have a tremendous impact on the creativity and productivity of the programmer. Therefore, this gave rise to the need of developing an environment that could handle the complexities and carry out the execution of all the modules of a Java application under one roof. Sun Microsystems identified the need of developing such an environment in 1999, and acquired the NetBeans Integrated Development Environment (IDE), which was particularly introduced for writing, compiling, and executing Java programs.

NetBeans is a free, open-source, cross-platform IDE with built-in support for Java. If a user uses IDE, it is mandatory for the user to work inside a project. An IDE project contains Java source files and associated information regarding Java ARchive (JAR) files and tools on the classpath, process of building and running the project, and other related information. You should note that the information regarding a project is stored inside a project folder. The project folder contains an Ant build script (build.xml) and the properties file that manages the settings that are required to build and run the project.

In this appendix, you explore the new features of NetBeans IDE 6.8 and the different User Interface (UI) components of NetBeans IDE 6.8, such as Welcome Page, Services window, Files window, Project window, and Toolbars. You also learn how to cerate, build, and run the Web, enterprise, and JavaServer Faces (JSF) applications by using NetBeans IDE 6.8.

New Features of NetBeans IDE 6.8

Different versions of NetBeans IDE have come up since its first release in 1999, the latest version being NetBeans IDE 6.8. The following are the noteworthy new features of NetBeans IDE 6.8:

- New and enhanced editor
- Swing Graphical User Interface (GUI) development
- Support for Ruby and Rails
- Support for Rails code generator graphical wizard

- ❑ Support for visual mobile development
- ❑ Support for profiling

Now, let's learn these features one by one.

New and Enhanced Editor

The new editor of NetBeans IDE 6.8, which is commonly known as Source Editor, allows you to write or edit the code of various files, such as Java source file, Extensible Markup Language (XML) file, and JavaServer Pages (JSP) file. As compared to the earlier versions of NetBeans, the editor of NetBeans IDE 6.8 is more user-friendly because it provides various additional functionalities to the developers, while writing the code. For example, while writing the code of a file in NetBeans IDE 6.8, the new editor automatically provides the suggestions for the completion of the keywords, fields, and variables used in the file. The editor also provides code templates that allow you to quickly enter the code in the blocks defined in a template.

In NetBeans IDE 6.8, if you want to search a piece of code, the searched code is highlighted by the Source Editor. This highlighting feature of the Source Editor is an enhanced substitution for the search function, which was provided in the earlier versions of NetBeans. It is important to note that if an error occurs while writing the code in the editor, the IDE highlights the errors with a background color and puts them into the error stripe. This facilitates the developers to quickly identify or locate the errors in the entire file.

Swing Graphical User Interface (GUI) Development

The Swing GUI development feature of NetBeans IDE 6.8 enhances the development of the Java and Swing desktop applications. Initially, the developers needed to provide the complete code to create a database table in a form (page) of an application. However, in NetBeans IDE 6.8, the introduction of the new Java Desktop Application project template has simplified the process of creating and updating the database table. In NetBeans IDE 6.8, you can add a database table to a form by simply dragging the database table from the Runtime window to the form. The process of dragging the table binds the form to the dragged database table. Apart from this, the NetBeans IDE 6.8 IDE also provides the Beans Binding technology to easily maintain the properties of the various beans that are synchronized with each other.

Another amazing feature of the NetBeans IDE 6.8 is that it provides the Swing Application Framework to easily and quickly develop small and medium-sized Java desktop applications. This framework also provides support to manage the application life cycle, actions, and resources that are used in an application.

Support for Ruby and Rails

In NetBeans IDE 6.8, the introduction of the Ruby and Rails feature allows you to perform the following tasks:

- ❑ Create Ruby projects with logical structure and run Ruby files
- ❑ Configure other Ruby interpreters (such as JRuby or native Ruby)
- ❑ Locate and install Ruby Gems through a graphical wizard
- ❑ Create and execute unit tests
- ❑ Jump between a Ruby file and its corresponding unit test or spec file

The NetBeans IDE 6.8 provides a Ruby Debugger to debug the application that contains the Ruby code. In addition, NetBeans IDE 6.8 provides support to establish breakpoints, view local variables, explore the call stack, switch threads, and calculate expressions by moving the mouse pointer over the variables declared in the code provided in the Source Editor.

Support for Rails Code Generator Graphical Wizard

Initially, third party plugins were used in the NetBeans IDE to create Rails project. The NetBeans IDE 6.8 has introduced the Rails code generator graphical wizard to create Rails project. This IDE also facilitates you to navigate from a Rails action to its associated view in the browser.

Support for Visual Mobile Development

In NetBeans IDE 6.8, you can create Java Micro Edition (ME) based applications that are particularly used for Mobile Information Device Profile (MIDP). In other words, NetBeans IDE 6.8 allows you to create Java ME applications that are supported on Java embedded devices, such as mobile phones and Personal Digital Assistants (PDAs).

NOTE

The MIDP profile allows the development of the Java ME applications that can be used by mobile devices and the devices that are configured by using Connected Limited Device Configuration (CLDC) and Connected Device Configuration (CDC).

In the earlier versions of NetBeans, limited properties, such as configuration support for device fragmentation, support for integrated obfuscation as well as optimization, and multiple deployment options, were available for the CLDC/MIDP projects. Now, NetBeans IDE 6.8 has introduced the NetBeans Mobility Pack, which permits the developers to create, test, and debug CLDC/MIDP projects. The project properties, supported by Mobility pack, are developed using Apache Ant to simplify the coding and management of the project.

In addition, the visual editing feature introduced in NetBeans IDE 6.8 has simplified the task of creating mobile games by using the MIDP 2.0 Game API. Animated sprites and capability to arrange tiled layers into scenes are the features supported by the MIDP 2.0 Game API. To provide enhanced functionality and usability, the Visual Mobile Designer (VMD) has also been redesigned in NetBeans IDE 6.8. Developing and designing of mobile file browsers, Short Message Service (SMS) composers, login screens, and Personal Information Manager (PIM) browsers are now much easier with the introduction of new components in VMD.

Support for Profiling

To understand the profiling feature of NetBeans IDE 6.8, you first need to be well-versed with the term profiling. Profiling refers to the process of monitoring the behavior of an application or a program using the information gathered during its execution. Since NetBeans IDE 6.8 has a built-in profiler to monitor the performance of the Java applications, you do not need to download and install the NetBeans profiler. When you are profiling an application in NetBeans IDE 6.8, you can monitor the activities of JVM and acquire information regarding the performance of the application, including method timing, object allocation, and garbage collection. The resultant data can be used to locate potential areas in the code that can be optimized to improve the performance of your application.

Apart from the preceding features, the NetBeans IDE 6.8 also allows you to add Web services in your application by using the drag and drop functionality. Moreover, it also provides support to add the Ajax-enabled JavaServer Faces components in a Web application. Some of the features of Ajax-enabled component that depict similarity with other component are drag and drop of the component, set properties, and customize server-side event handlers.

In the next section, let's learn about the Welcome page in NetBeans IDE 6.8.

The Welcome Page

When you launch NetBeans IDE 6.8, the first screen that appears is the Welcome Page. You can start a new project by selecting the File→New Project option from the menu bar. You can add tools to the toolbar in the NetBeans IDE by selecting the View→Toolbars option. Several other menu options present in NetBeans IDE 6.8 are similar to the ones that were already available in the earlier versions of NetBeans. Figure C.1 shows the Welcome page of NetBeans IDE 6.8:

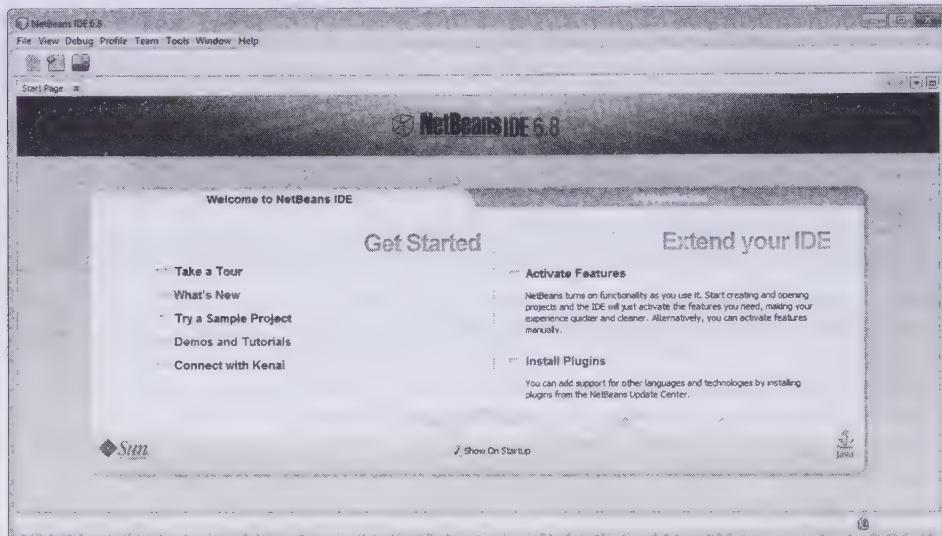


Figure C.1: Showing the NetBeans IDE 6.8 Welcome Page

The menu bar shown in Figure C.1 shows the various menus available in the NetBeans IDE.

Toolbars

Toolbars appear at the top of the NetBeans IDE, which allows you to implement a functionality in an application or a project (Figure C.2). You can add and remove tools to and from the toolbars by clicking the View→Toolbars→Customize menu option in the IDE. The NetBeans IDE also provides the functionality to display a label (tooltip) while pointing the mouse to the menu. Many options in the toolbars get enabled or disabled, depending on the requirements of the project on which you are working. Figure C.2 shows the NetBeans IDE toolbars:

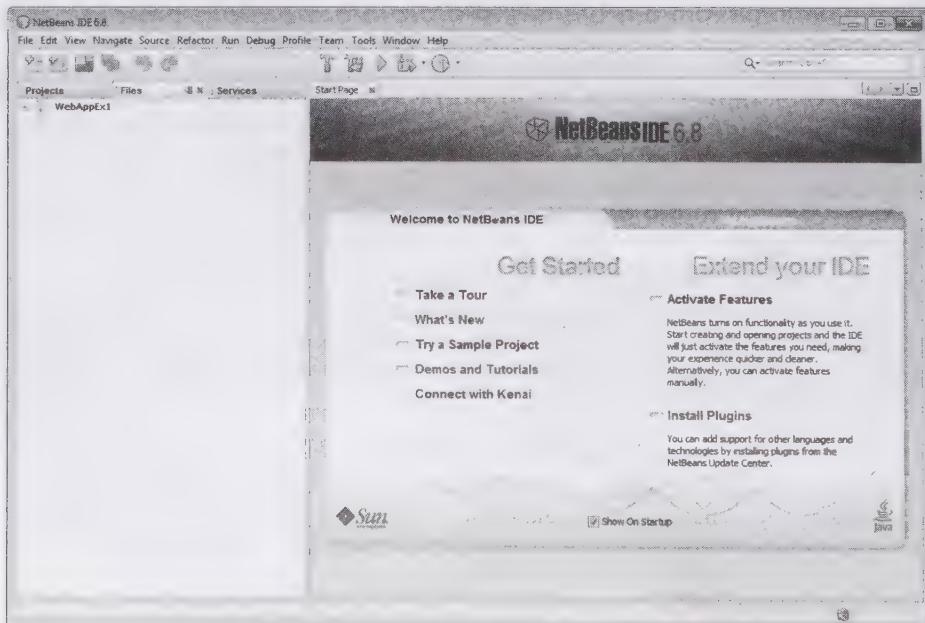


Figure C.2: Showing the NetBeans IDE 6.8 Toolbars

Although it is common to use the Menu system, the toolbar buttons provide quicker access. You can save the file on which you are currently working with the help of the diskette button () on the Standard toolbar. To save all the files at a particular location in the computer, click the stacked diskettes button ().

The Projects Window

The Projects window appears on the upper left corner in the NetBeans IDE 6.8 (Figure C.3). It provides a tree view of all the projects on which you have worked recently. With the help of this window, you can browse through a particular project to access the different packages and libraries present in the project, such as Source Packages, Test Packages, Libraries, and Test Libraries. Figure C.3 shows the Projects window of NetBeans IDE 6.8:

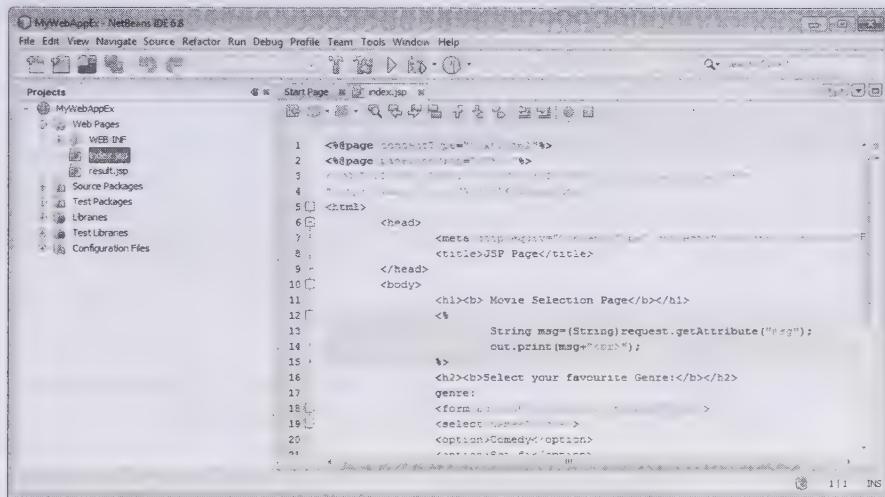


Figure C.3: Showing the Projects Window of NetBeans IDE 6.8

The Projects window is arranged in a parent-child tree manner, in which the parent node is the project and the child nodes are the categories of the project. For example, in Figure C.3, MyWebAppEx is the parent node containing various child nodes, such as Web Pages, Source Packages, and Test Packages. Table C.1 describes the categories of child nodes in the parent node:

Table C.1: Describing the Categories of Various Child Nodes

Node	Description
Web Pages	Contains various Web pages and the WEB-INF folder of an application. In the Web Pages node, you can add HyperText Markup Language (HTML), JSP, Cascading Style Sheets (CSS), and images for your application.
Source Packages	Permits you to access the Java source files of the project by expanding the nodes available beneath the Source Packages node.
Test Packages	Specifies the package structure for an application's test classes and JUnit tests.
Libraries	Specifies the libraries that are used in an application.
Test Libraries	Contains the class files or JAR files that are referred by the debugger (or server) while testing an application.
Configuration Files	Contains web.xml, struts-config.xml, and other project configuration files. The items available beneath the Configuration Files node are usually required to be accessed when configuration changes are made in the application. It also contains deployment descriptor for the Web application.

In the next section, let's learn about the Service window.

The Services Window

The Services window shares its space with the Project window and provides various services, such as accessing different relational databases, configuring the servers, and managing the Web services of an application. The Services window can be opened by selecting the Services option under the Window menu. Figure C.4 displays the Services window of NetBeans IDE 6.8, which contains the list of different servers, databases, and services running under the IDE's control:

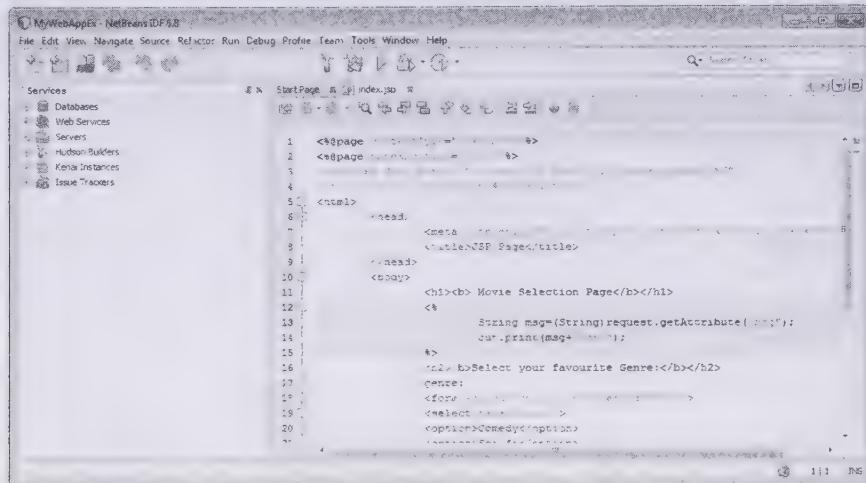


Figure C.4: Showing the Services Window of NetBeans IDE 6.8

The Files Window

The Files window of NetBeans IDE 6.8 displays a directory-based structure of the applications. This window also includes the files that are not displayed by the Projects window, such as the build-impl.xml file. The Files window lets you explore the various packages, files, and objects present in a project. Figure C.5 shows the Files window of NetBeans IDE 6.8:

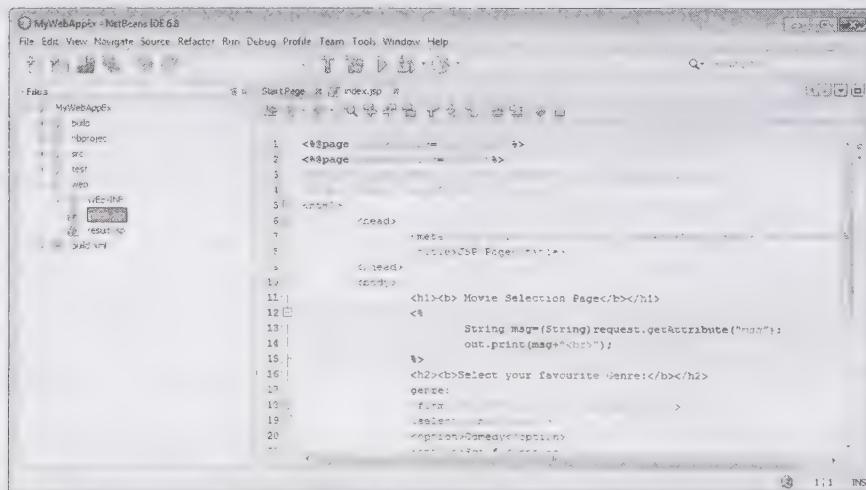


Figure C.5: Showing the Files Window of NetBeans IDE 6.8

You are now aware of the different types of window in NetBeans IDE 6.8. Next, let's learn how to create Web applications in the NetBeans IDE.

Creating Web Applications in the NetBeans IDE

Now, let's proceed to learn how to create a Java program in NetBeans. The NetBeans IDE provides different project templates to develop Java applications, Web applications, and enterprise applications. The two basic categories in which the available project templates can be divided are as follows:

- ❑ **Standard project** – Uses an IDE-generated Ant script to compile, run, and debug an application. An Ant script is an XML build file, which contains tasks that you want the Ant tool to perform. The following standard project templates are available in the IDE:
 - **Java** – Includes the Java Application, Java Class Library, Java Project with Existing Sources, and Java Desktop Application options
 - **Web** – Includes the Web application and Web application with Existing Sources options
 - **Enterprise** – Includes the Enterprise application, Enterprise application with Existing Sources, Enterprise JavaBeans (EJB) Module, EJB Module with Existing Sources, Enterprise Application Client, and Enterprise Application Client with Existing Sources options
 - **NetBeans Modules** – Includes the Module Project, Module Suite Project, and Library Wrapper Module Project options
- ❑ **Free-form project** – Uses an existing Ant script to compile, run, and debug an application. The following free-form project templates are available in the IDE:
 - Java Project with Existing Ant Script
 - Web Application with Existing Ant Script

Let's create a Web application in NetBeans IDE 6.8 by performing the following broad-level steps:

- ❑ Creating a Standard Project in NetBeans IDE 6.8
- ❑ Modifying the Default JavaServer Pages File
- ❑ Developing a Servlet
- ❑ Developing a Java Source File
- ❑ Developing a JavaServer Pages File

Creating a Standard Project in NetBeans IDE 6.8

You can create a standard project in NetBeans IDE 6.8 by performing the following steps:

1. Select the **File→New Project** menu option to open the **New Project** wizard.

There are different categories of projects, such as Java, Java Web, Java EE, Java ME, Java FX, Maven, and many others, available in NetBeans IDE 6.8. To build a Java Web project, select Java Web from the Categories pane and Web Application from the Projects pane. Figure C.6 shows the **New Project** wizard:

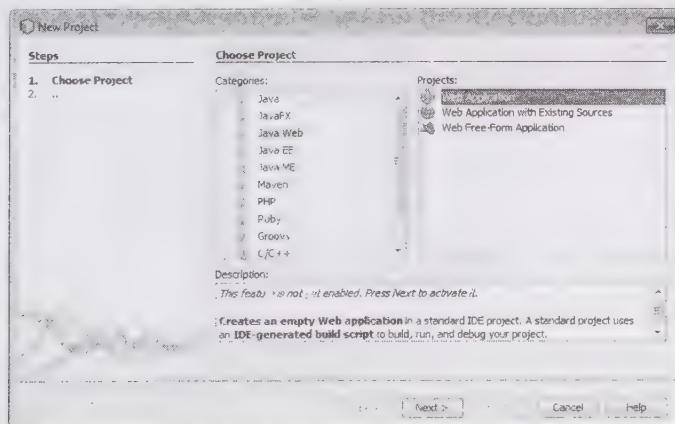


Figure C.6: Showing the New Project Wizard

2. Click the Next button.

The New Web Application dialog box appears, which provides the following options:

- **Project Name** – Allows you to enter the project name, which is used by NetBeans to display the project in its environment
- **Project Location** – Allows you to specify the project location by clicking the Browse button
- **Project Folder** – Contains the location of the project selected by the user or the default value of the project location

Figure C.7 shows the New Web Application dialog box:

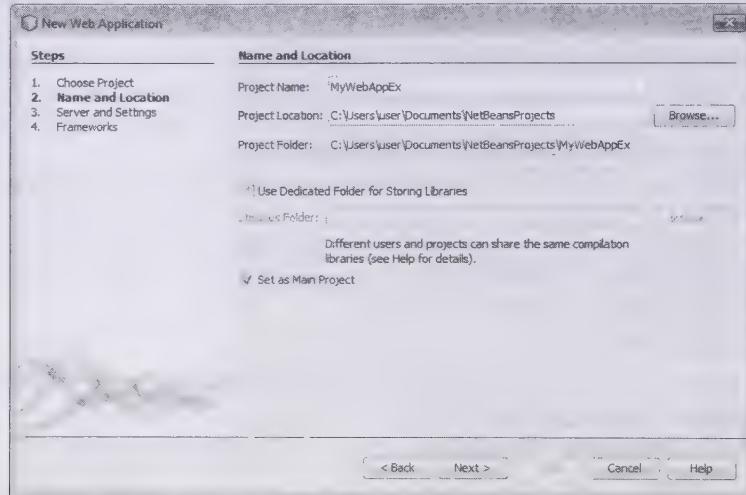


Figure C.7: Showing the New Web Application Dialog Box

3. Click the Next button (Figure C.7).

The Server and Settings page appears, which enables the user to select the desired application server for the Web application. The New Web Application dialog box also enables the user to select the Java EE version and specify the context path of the Web application. Figure C.8 shows the Server and Settings page to select the desired application server, Java EE version, and provide context path:

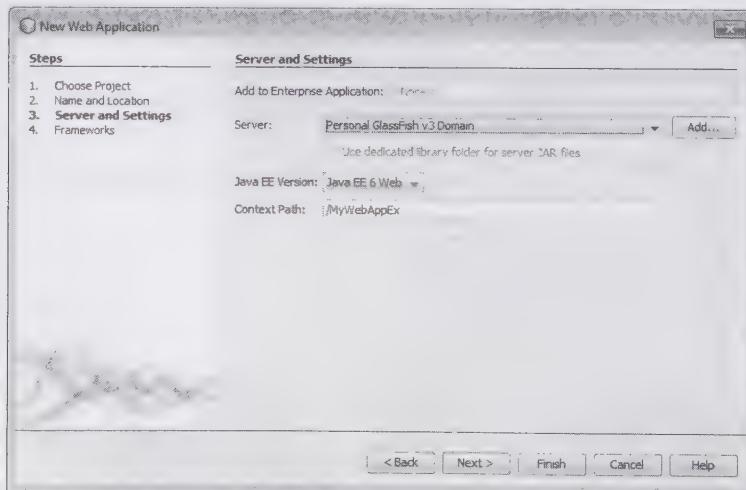


Figure C.8: Showing the Window to Select the Appropriate Application Server

4. Click the Next button (Figure C.8).

The Frameworks page appears that allows you to select the desired framework for your Web application. If you do not want to implement any framework in your application, you can ignore the Frameworks page and click the Finish button. However, to select a framework, say JavaServer Faces, you need to select the checkbox beside the name of the framework. Figure C.9 shows the Frameworks page that displays frameworks to choose for the Web application:

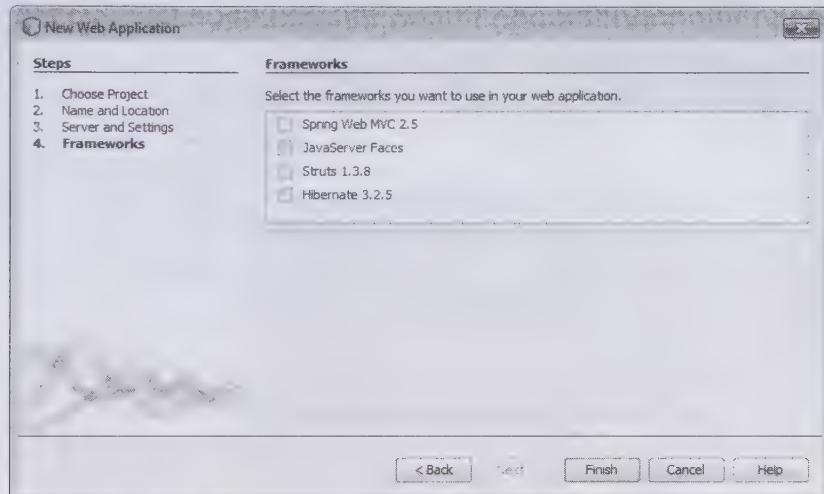


Figure C.9: Showing the Frameworks to Choose for the Web Application Project

5. Click the Finish button (Figure C.9) to proceed.

The MyWebAppEx project opens in the IDE and the default JSP page, i.e., index.jsp, is displayed in the Source Editor. Figure C.10 shows the index.jsp page:

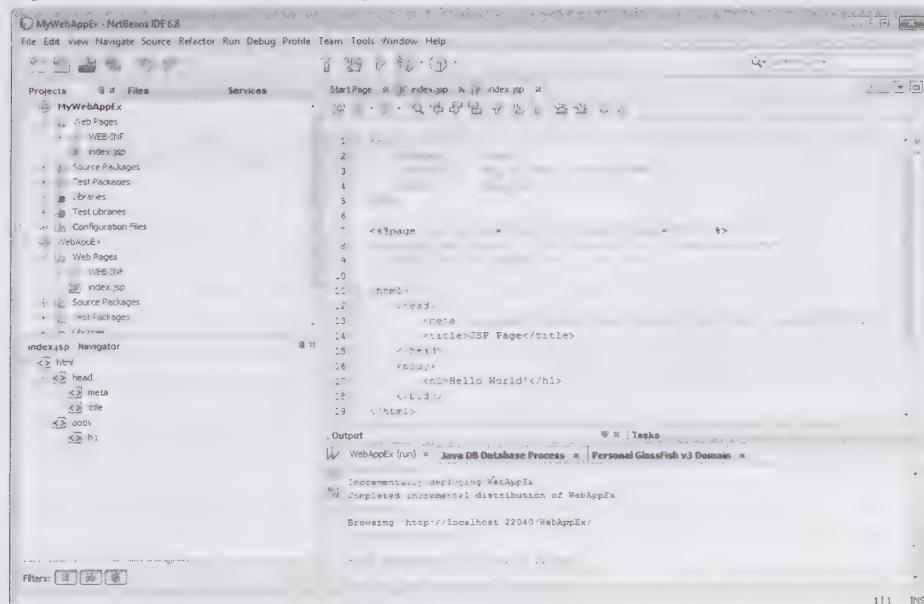


Figure C.10: Displaying the index.jsp Page in the Source Editor

Let's now learn how to edit the index.jsp page.

Editing the Default JavaServer Pages File

Perform the following steps to edit the JSP page:

1. Expand the HTML Forms tab in the Palette window (shortcut to open palette window is Ctrl-Shift-8), which is on the right of the Source Editor.
2. Drag the Form control from the Palette window and drop it just below the <body> tag in the Source Editor. The Insert Form dialog box opens, as shown in Figure C.11:

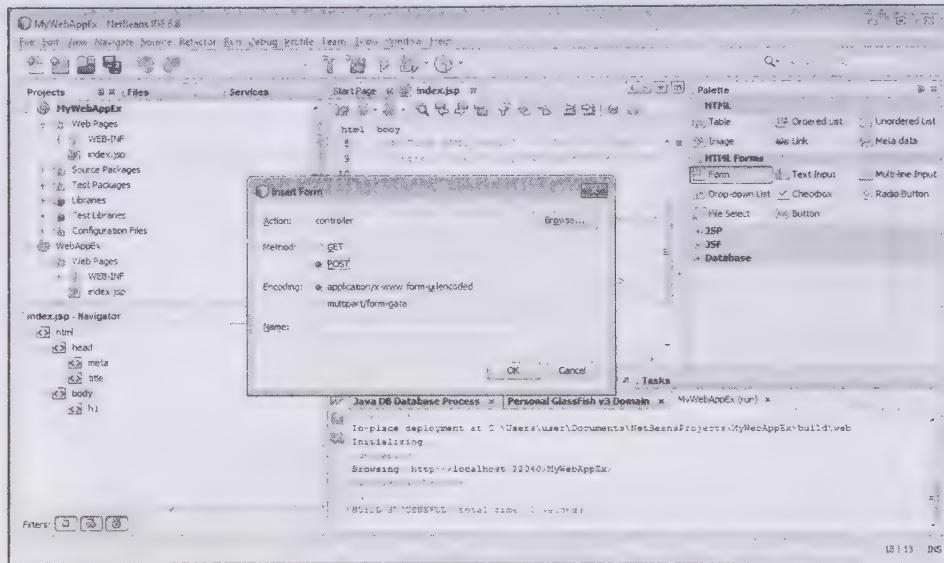


Figure C.11: Showing the Insert Form Dialog Box

3. Specify the values in the following fields of the Insert Form dialog box:
 - **Action** – Accepts the name of the component to be invoked. In our case, we have specified controller.
 - **Method** – Accepts the type of HTTP method, such as GET or POST. In our case, we have selected the POST radio button.
 - **Encoding** – Accepts the form encoding type that can be either application/x-www-form-urlencoded or multipart\form-data. In our case, we have selected the application/x-www-Form-urlencoded radio button.
 - **Name** – Accepts the name of the form, which is optional. In our case, we have not provided any name for the form.
4. Click the OK button (Figure C.11). As a result, an HTML form tag <form>....</form> appears in the index.jsp file.
5. Drag the Drop-down List control from the Palette window to the Source Editor to a point just before the </ form> tag. The Insert Drop-down List dialog box appears, wherein the following values are specified:
 - **Name** – Accepts the name of the component. In our case, we have specified genre as the name of the component.
 - **Options** – Accepts the number of options. In our case, we have specified 3 as the number of options to be provided in the drop-down list.
 - **Visible Options** – Accepts the number of visible options. In our case, we have specified 1 as the number of visible options in the drop-down list.
6. Click the OK button in the Insert Drop-down List dialog box (Figure C.12). An HTML <select> tag is added between the <form> tags.

Figure C.12 shows the Insert Drop-down List dialog box:

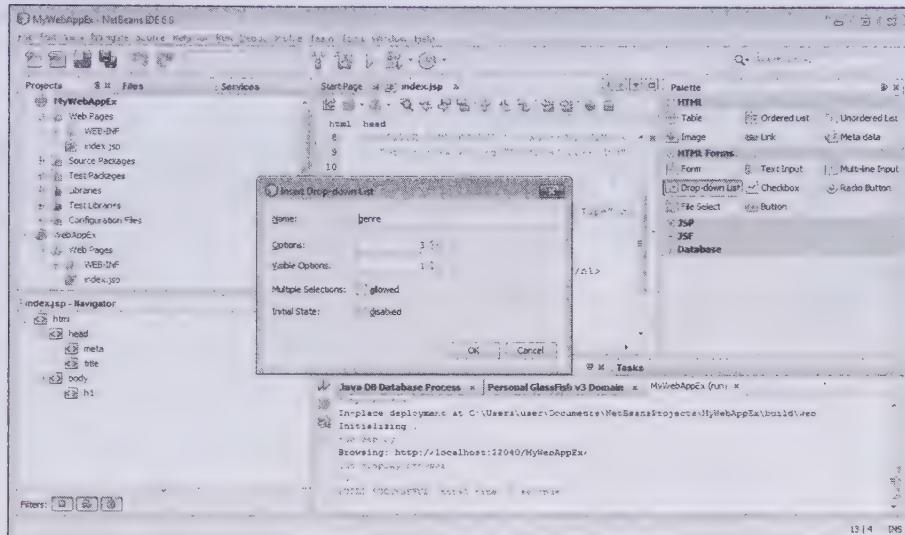


Figure C.12: Showing the Insert Drop-down List Dialog Box

7. Drag the Button control from the Palette window to a point just before the `</form>` tag. The Insert Button dialog box appears, which allows you to specify the following values:
 - **Label** – Allows you to specify the caption of the button. In our case, we have specified submit as the label.
 - **Type** – Allows you to specify the type of the button, such as submit, reset, or standard. In our case, we have specified submit as the type of the button.
 - **Name** – Allows you to specify the name of the button. In our case, we have specified button1 as the name of the button.

Figure C.13 shows the Insert Button dialog box:

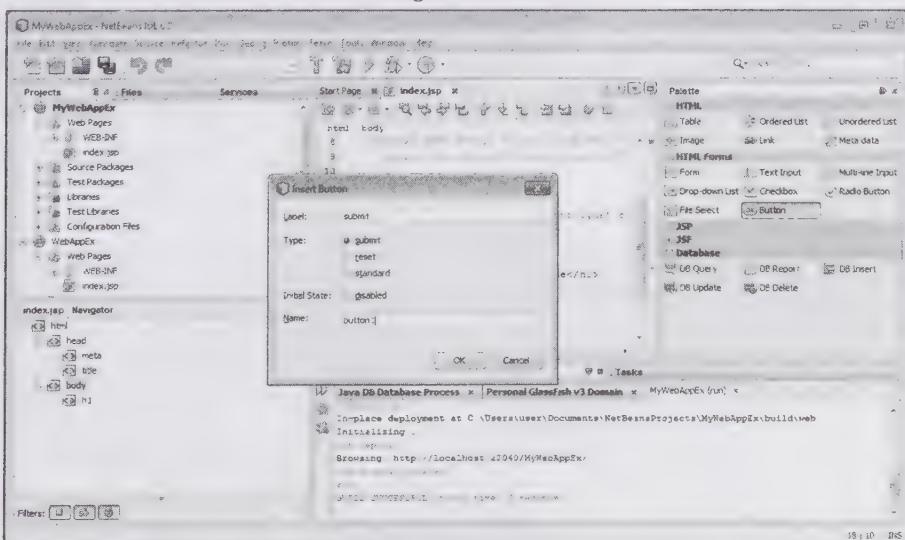


Figure C.13: Showing the Insert Button Dialog Box

8. Click the OK button. The HTML button is appended in the <form> tag.
9. Right click the Source Editor and select the Reformat Code option to edit the code. Listing C.1 shows the code of the index page (you can find this file on CD in the code\JavaEE\AppendixC\MyWebAppEx\web\ folder):

Listing C.1: Showing the Code of the index.jsp File

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1><b> Movie Selection Page</b></h1>
    <%
      String msg=(String)request.getAttribute("msg");
      out.print(msg+"<br>");
    %>
    <h2><b>Select your favourite Genre:</b></h2>
    genre:
    <form action="controller" method="POST">
      <select name="genre">
        <option>Comedy</option>
        <option>Sci-fi</option>
        <option>Action</option>
      </select>
      <input type="submit" value="submit" name="button1"/>
    </form>
  </body>
</html>
```

Creating the Servlet

To create a servlet, right click the Project node of the Web application where the servlet is to be created, and choose the New→Servlet option. The New Servlet wizard opens. You need to provide the name of the file controller and specify com.kogent as the package name in the New Servlet wizard. Finally, click the Finish button. You will notice that a controller.java file node displays in the Project window, and the new file opens in the Source Editor. Listing C.2 shows the code of the controller.java file (you can find this file on CD in the code\JavaEE\AppendixC\MyWebAppEx\src\java\com\kogent\controller folder):

Listing C.2: Showing the Code for the controller.java File

```
package com.kogent.controller;
import com.kogent.model.*;
import java.util.*;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet(name="controller", urlPatterns={"/controller"})
public class controller extends HttpServlet {
  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
```

```

throws ServletException, IOException
{
    String myGenre=request.getParameter("genre");
    movieLists movies=new movieLists();
    List result=movies.getMovies(myGenre);
    RequestDispatcher view=null;
    if(result!=null)
    {
        request.setAttribute("result", result);
        view=request.getRequestDispatcher("result.jsp");
    }
    else
    {
        request.setAttribute("msg","Invalid Option: No Movies for such genre.");
        view=request.getRequestDispatcher("index.jsp");
    }
    view.forward(request, response);
}
// <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">
/**
 * Handles the HTTP <code>GET</code> method.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    processRequest(request, response);
}
/**
 * Handles the HTTP <code>POST</code> method.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    processRequest(request, response);
}
/**
 * Returns a short description of the servlet.
 * @return a String containing servlet description
 */
@Override
public String getServletInfo() {
    return "Short description";
} // </editor-fold>
}

```

Creating the Java Source File

Let's now create the Java source file, movieLists.java, by performing the following steps:

1. Right click the **Source Packages** node in the **Projects** window and select the **New→Java Class** option.
2. Enter **movieLists** in the **Class Name** field and specify **com.kogent.model** in the **Package** field.

3. Click the Finish button to complete the process of creating the Java source file.

The movieLists.java file will be opened in the Source Editor. Listing C.3 shows the code of the movieLists Java source file (you can find this file on CD in the code\JavaEE\AppendixC\MyWebAppEx\src\java\com\kogent\model folder):

Listing C.3: Showing the Code of the movieLists.java File

```
package com.kogent.model;
import java.util.*;
public class movieLists {
    public List getMovies(String genre)
    {
        List movies=new ArrayList();
        if (genre.equals("Comedy"))
        {
            movies.add("I love you to death");
            movies.add("scary movie 1");
            movies.add("scary movie 2");
        }
        else if(genre.equals("Sci-fi"))
        {
            movies.add("matrix");
            movies.add("matrix reloaded");
        }
        else
        {
            return null;
        }
        return (movies);
    }
}
```

The Servlet will then forward the result to another JSP page called result.jsp. Let's learn to create result.jsp file in the next subsection.

Creating the JavaServer Pages File

In the Projects window, right click the project node (in our case, MyWebAppEx) and select the New→JSP option. The New JSP File wizard opens. In the Name field, enter result and click the Finish button to complete the process. Notice that a result.jsp file node displays in the Projects window. Listing C.4 shows the code of the result JSP page (you can find this file on CD in the code\JavaEE\AppendixC\MyWebAppEx\web\ folder):

Listing C.4: Showing the Code of the result.jsp File

```
<%@page contentType="text/html" import="java.util.*"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Check out these cool movies!!!</title>
    </head>
    <body>
        <h1>movies under selected Genres are:</h1>
        <c:if test="${reqattr != null}">
            <table>
                <c:forEach var="movie" items="${reqattr}">
                    <tr>
                        <td>
```

```

        ${movie}
    </td>
    </tr>
</c:forEach>
</table>
</body>
</html>

```

Once all the required files are being coded, it's now time to build and run the application.

Building and Running the Web Application

To build the MyWebAppEx project, select the Run menu → Build Main Project option from the menu bar in the NetBeans IDE 6.8. After the project is built successfully, you can run the MyWebAppEx application by selecting the Run Main Project option from the Run menu.

NOTE

The following are the three ways of running a Web Application in NetBeans IDE 6.8:

- By selecting the Run Main Project option from the Run menu
- By clicking the Run Main Project button (▶)
- By pressing the F6 key

When you run a Web application, the IDE builds the project first and then deploys it to the server that you had specified at the time of project creation. To deploy and run a Web application in the NetBeans IDE, the following steps are performed:

1. Package the application in the Web Archive (WAR) file, which is a Java Archive (JAR) file to package components of a Web application, such as servlets, Java classes, JSPs, and many other.
2. Start the application server and deploy the WAR file. In our case, we have used GlassFish Application Server in the application.
3. Browse the <http://localhost:8080/MyWebAppEx/> URL if the deployment is successful.

Figure C.14 shows the execution of the MyWebAppEx application that displays the Movie Selection Page:

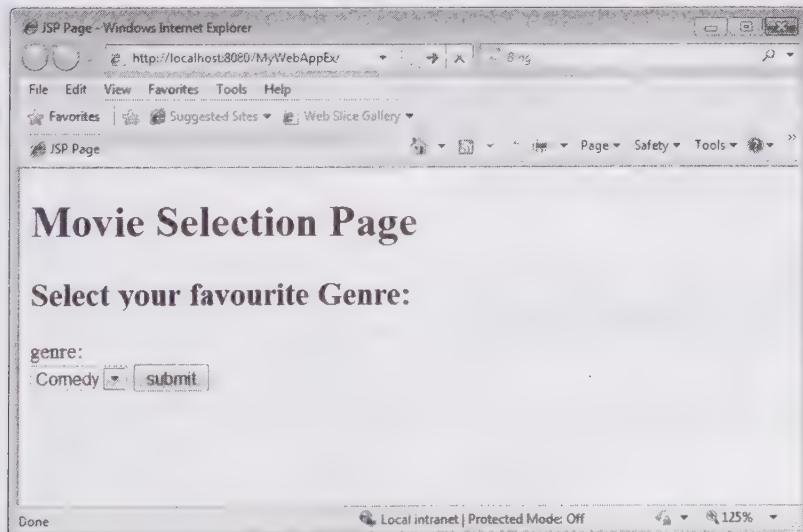


Figure C.14: Showing the Movie Selection Page

4. Select a genre, such as Comedy in our case (Figure C.14), and then click the submit button to check the movies under the selected genres.

Figure C.15 shows a list of movies under the selected genre (Comedy):

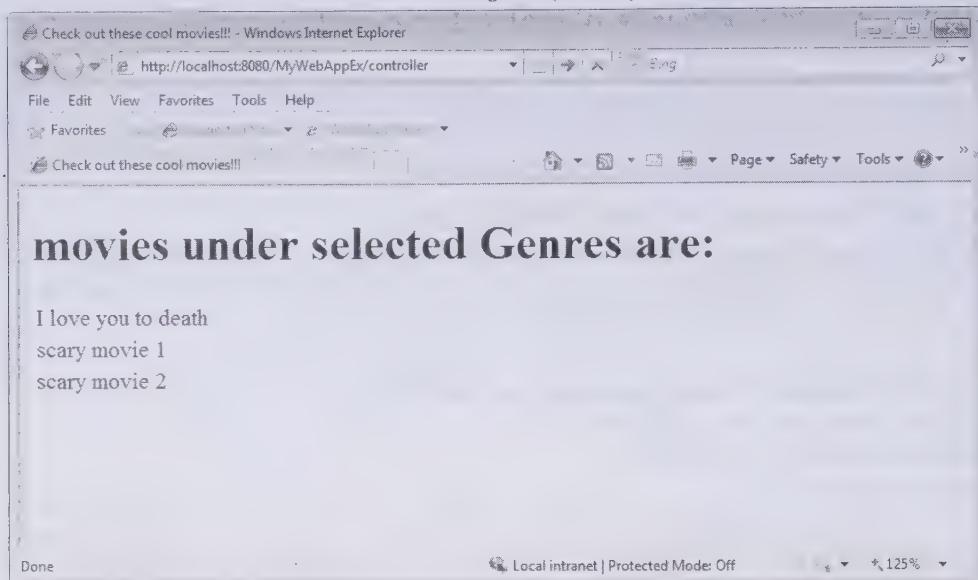


Figure C.15: Showing a Movies List under the Comedy Genre

Similarly, if you select another genre category, you will get another type of movie list specified for that particular genre. Moreover, the Invalid Option: No Movies for such genre message will be displayed if no movie list is available for a particular genre, such as Action. Figure C.16 shows the resultant Web page after selecting the Action category:

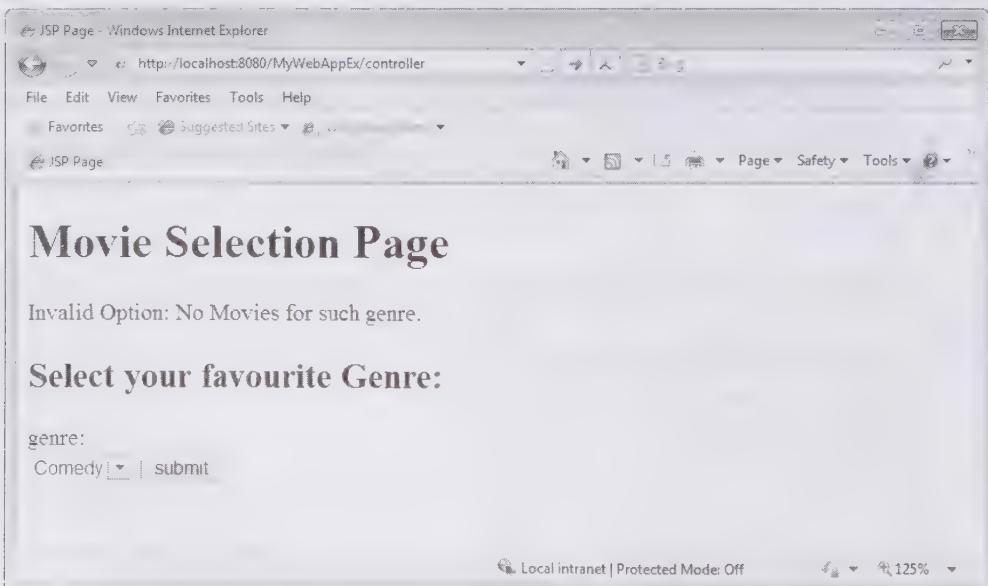


Figure C.16: Showing Invalid Option Message under the Action Genre

This was all about creating a sample Web application using NetBeans IDE 6.8. Let's now learn how to create the enterprise applications in NetBeans IDE 6.8.

Creating Enterprise Applications in NetBeans IDE

The enterprise application project is a collection of Web application and EJB modules that are configured to work together when deployed to the Java EE Application Server.

The enterprise application contains the information regarding integrated working of the modules and how the modules work with the application server. It contains deployment descriptors and other configuration files, but has no source files of its own. In compilation, the JAR files and WAR files for each individual module of the enterprise application are built and assembled into one Enterprise Archive (EAR) file, which is deployed to the application server. An Enterprise Application Project can be created manually or from existing sources.

Creating a Standard Project in NetBeans IDE 6.8

You can create a new Enterprise project by using NetBeans IDE 6.8. The following steps are performed to create the standard project in NetBeans IDE 6.8:

1. Select the File→New Project menu to open the New Project wizard.

You will see a similar window that you saw when you create the MyWebAppEx Web application.

2. Select Java EE from the given category list and Enterprise Application from the given project list. To proceed, click the Next button.

Figure C.17 shows the New Project wizard containing the options for the category list and project list:

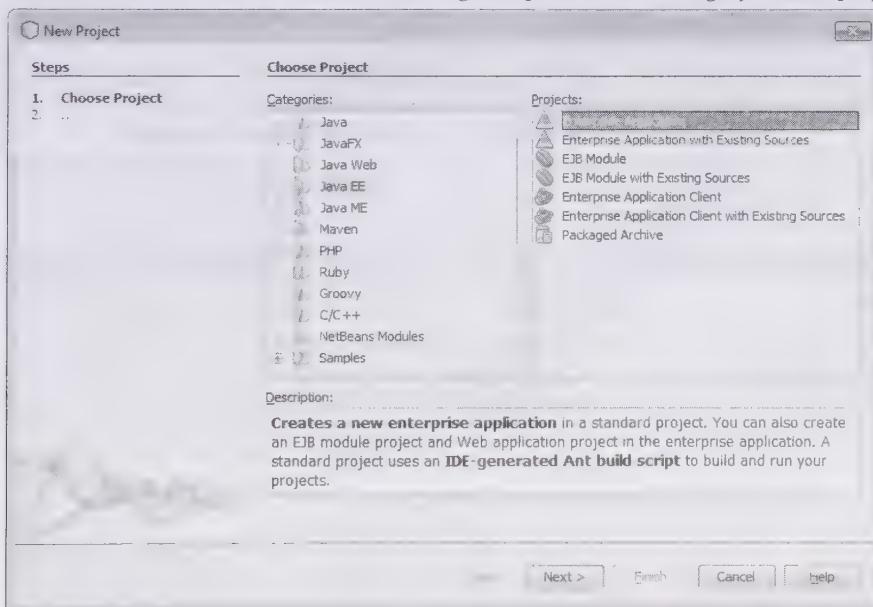


Figure C.17: Showing the New Project Wizard to Choose an Enterprise Application Project

The New Enterprise Application dialog box appears.

3. Enter the details for the following fields in the New Enterprise Application dialog box:

- **Project Name**—Allows the user to enter the project name for the Enterprise Application project, and NetBeans will use this name to display the project in its environment. In our case, we have entered EnterAppEx as the project name.
- **Project Location**—Allows the user to enter the project location by clicking the Browse button. It may also accept the default value if the user do not specify it. In our case, we have continued with the default value.
- **Project Folder**—Contains the location of the application. In our case, we have continued with the EnterApEx project folder located at the default location.

Figure C.18 shows the New Enterprise Application dialog box:

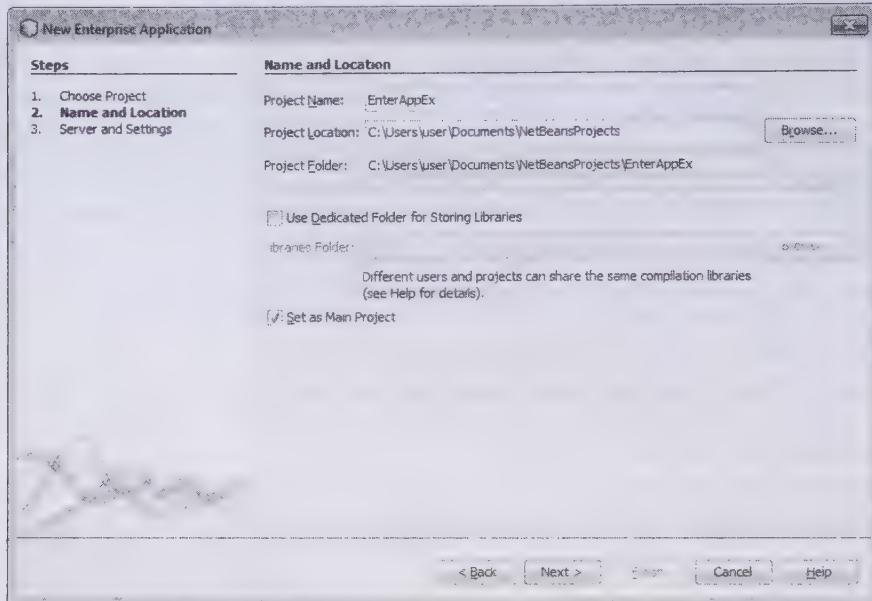


Figure C.18: Showing the New Enterprise Application Dialog Box

4. Click the Next button to proceed. The Server and Settings page appears that enables the user to select the desired application server and Java EE version for the enterprise application.

Figure C.19 shows the Server and Settings page to select the desired application server:

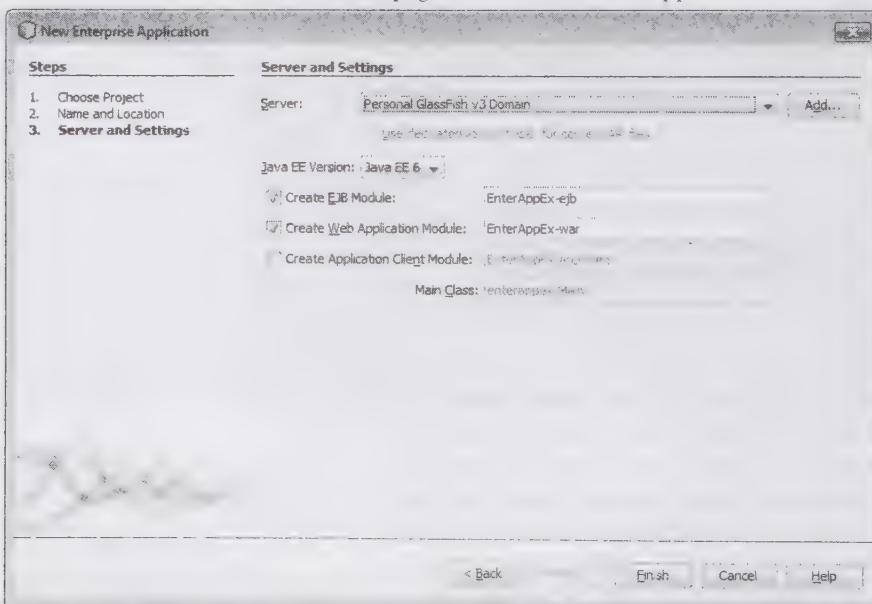


Figure C.19: Showing the Window to Choose Desired Application Server

5. Click the Finish button (Figure C.19) to create the enterprise application.

The newly created enterprise application appears that displays the Web and EJB modules.

Figure C.20 displays the Web and EJB modules of the EnterAppEx application:

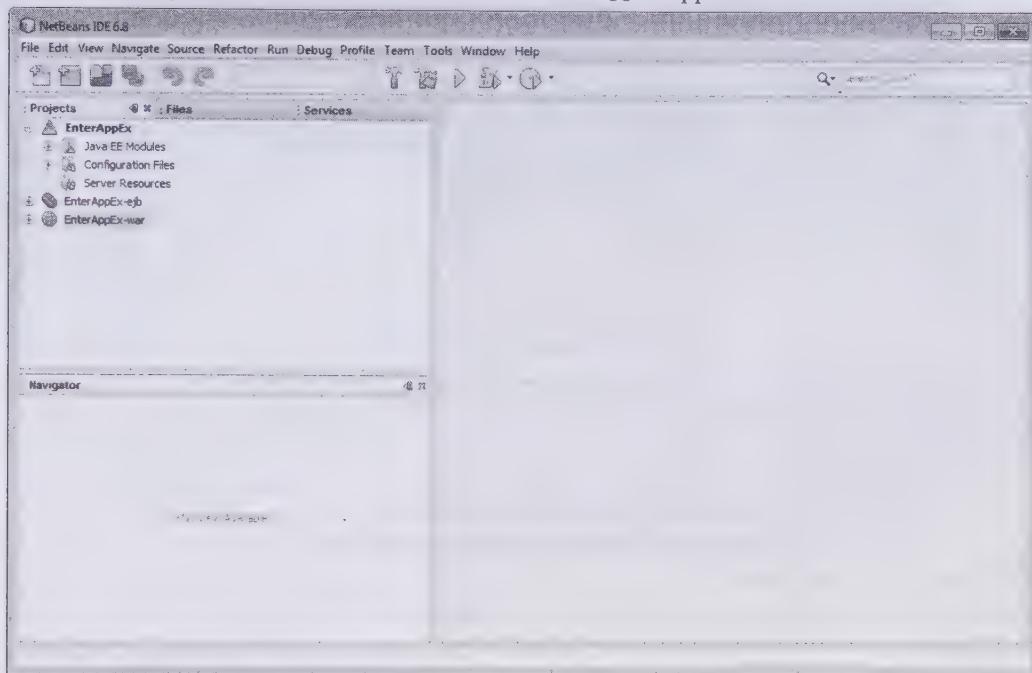


Figure C.20: Showing the Web and EJB Module of the EnterAppEx Application

Now you can provide code for the Web and EJB modules of the EnterAppEx enterprise application.

Creating JSF Applications in NetBeans IDE

The new Java framework, particularly used for creating Web applications, is JSF. JSF framework provides easier application development through component-centric approach for Java Web UI creation. Simplified and robust JSF API provides application developers unparalleled power and programming flexibility. To provide greater maintainability to the application, JSF framework's architecture also accommodates the Model-View-Controller (MVC) design pattern. You need to perform the following steps to create the JSF application in NetBeans IDE 6.8:

1. Create the standard project in NetBeans IDE 6.8
2. Develop the view pages
3. Develop the properties file
4. Develop a Java class
5. Develop the faces-config.xml file

Let's discuss about these steps, which are needed to develop a JSF application, in detail.

Creating a Standard Project in NetBeans IDE 6.8

You can create a new JSF project by using NetBeans IDE 6.8. The following steps are performed to create the standard JSF project in NetBeans IDE 6.8:

1. You can create a new JSF project by selecting the **File→New Project** menu option. The New Project wizard appears that is similar to the wizard displayed during the creation of the MyWebAppEx Web project.
2. For the JSF project, select the **Java Web** option from the given category list and the **Web Application** from the given project list.

Figure C.21 shows the New Project wizard:

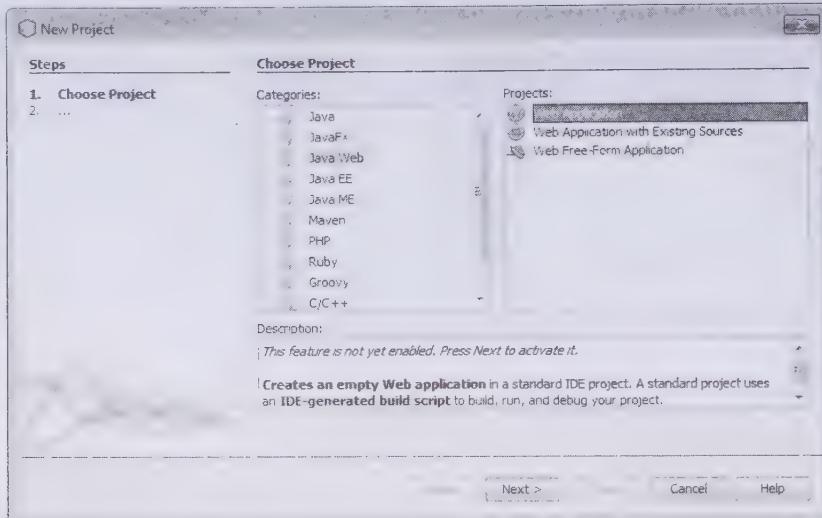


Figure C.21: Showing the New Project Wizard

3. Click the Next button (Figure C.21).

The New Web Application dialog box appears.

Figure C.22 shows the New Web Application dialog box:

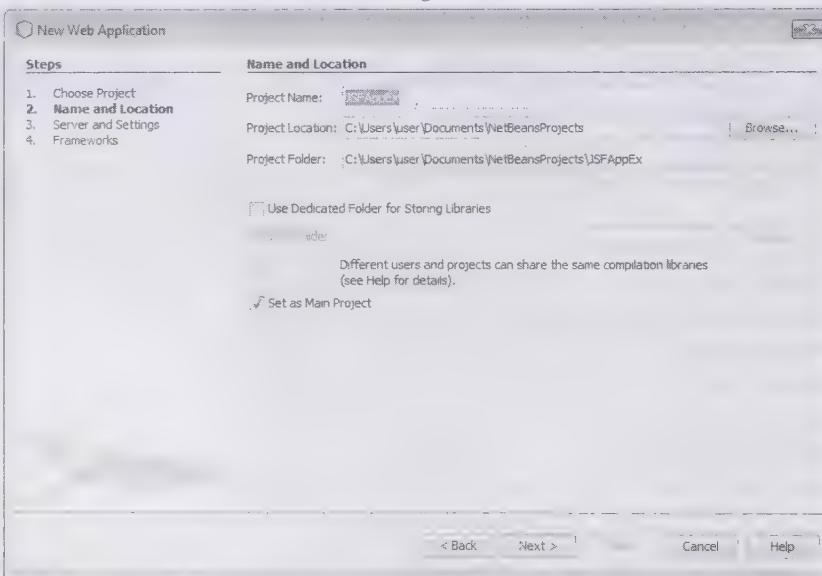


Figure C.22: Showing the New Web Application Dialog Box

The New Web Application wizard provides various options, which allow the user to enter the project name and project location. Provide the value JSFAppEx to the project name option. Leave the Set as Main Project check box selected. Click the Next button to proceed.

Now, the Server and Settings page appears that enables the user to select the desired application server, Java EE version, and specify the context path for the Web application.

Figure C.23 shows the page to select the desired application server:

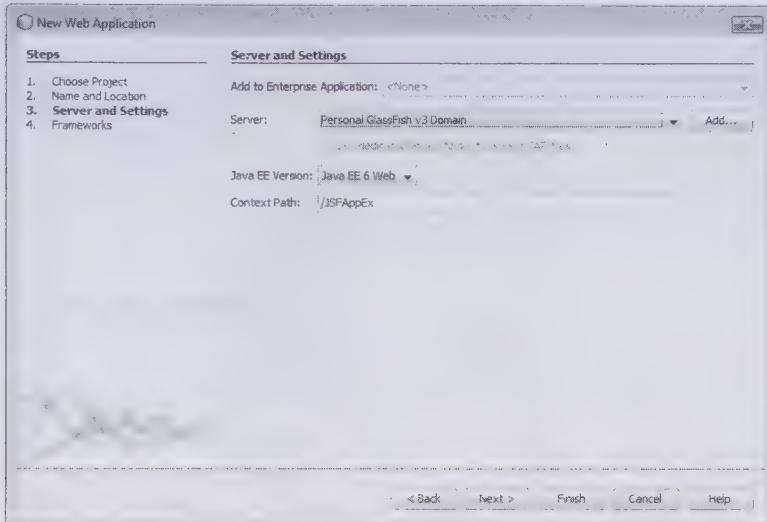


Figure C.23: Showing the Window to Select the Desired Application Server

4. Click the Next button (Figure C.23).

The Frameworks page appears that enables the user to select the appropriate framework. Check the JavaServer Faces check box to include the JSF framework in the application.

Figure C.24 shows the Frameworks page displaying the various frameworks that can be selected:

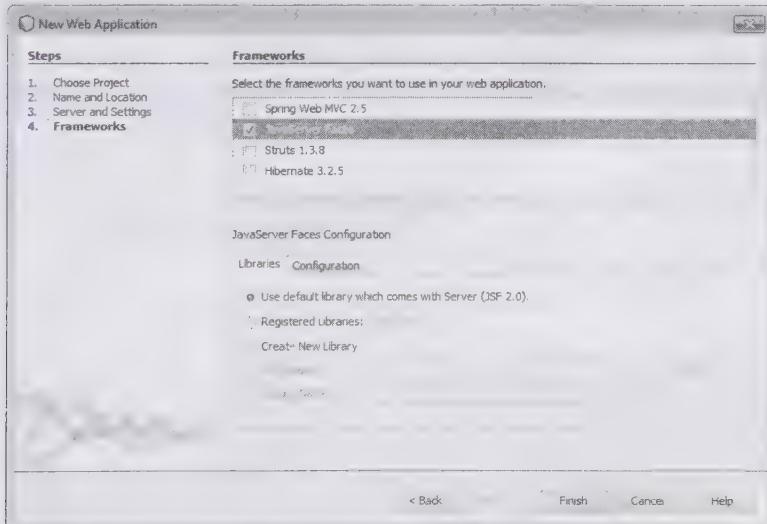


Figure C.24: Showing the Window to Select the Desired Framework

5. Click the Finish button (Figure C.24) to create the JSFAppEx application.

Creating the View Pages

Now it's the time to create view pages for the application. First, the welcome page (refers to the Web page that appears first when the application is executed) is created, which is `DemoMain.xhtml` file. To create an XHTML file, the following steps are performed:

Appendix C

1. Right click the project node, and select the New → Other → Web option. Now, select the XHTML file from the category list.

Figure C.25 shows the New File dialog box that provides the XHTML option to create the XHTML file:

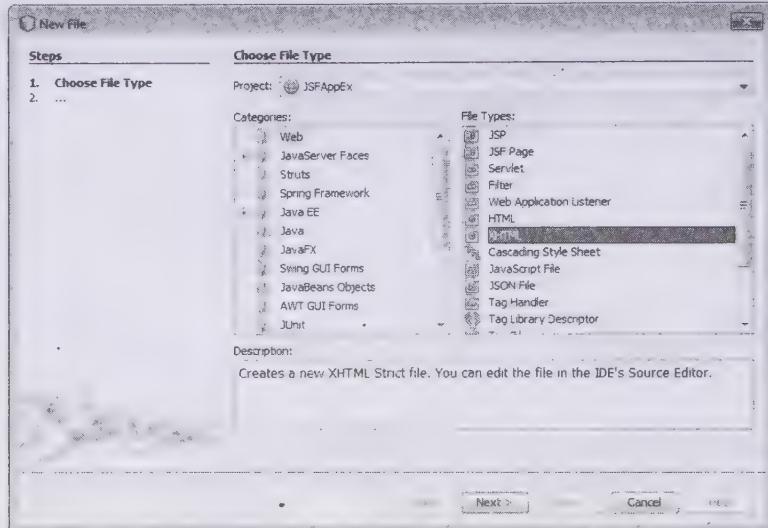


Figure C.25: Showing the Creation of XHTML File

2. Click the Next button (Figure C.25).

The Name and Location page appears that enables the user to enter the name of the XHTML file, which is DemoMain in the application.

Figure C.26 shows the Name and Location page to enter the name of the XHTML file:

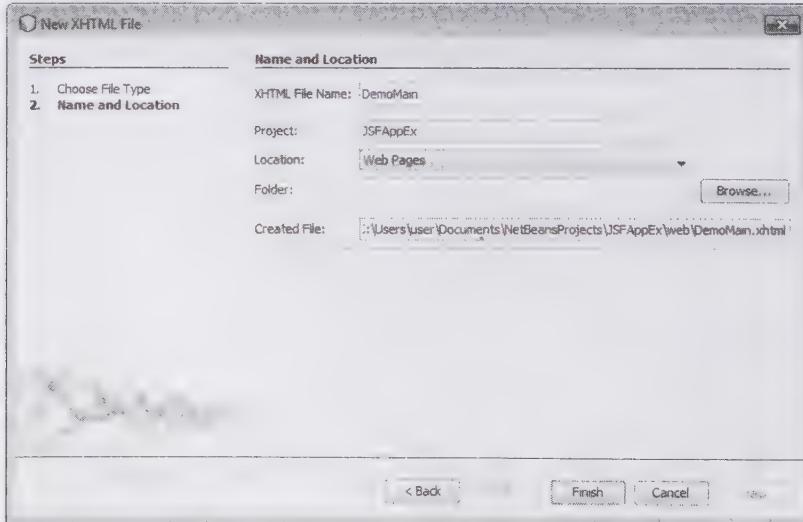


Figure C.26: Showing the Window to Enter the XHTML File Name

3. Click the Finish button (Figure C.26).

The DemoMain.xhtml file appears in the Project window. Edit the code of the DemoMain.xhtml file according to the code provided in Listing C.5 (you can find this file on CD in the code\JavaEE\AppendixC\JSFAppEx\web\ folder):

Listing C.5: Showing the Code of the DemoMain.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:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
<body>
    <ui:composition template="/myTemplate.xhtml">
        <ui:define name="body">
            <h:outputText value="#{msgs.Hello}"/>
            <h:outputText value="#{DemoBean.name}"/>
            &nbsp; <h:outputText value="#{msgs.AppTitle}"/>
        </ui:define>
    </ui:composition>
</body>
</html>
```

When the Web pages are developed in a Web application, there is some repeating content in multiple Web pages. Writing the repeating content again and again in the Web pages reduces the programmer productivity, thereby increasing the project time and cost. The solution to this problem is to use templates, which contain the repeating code and may be reused across multiple Web pages. Listing C.5 uses the `<ui:composition>` tag to invoke a template. The template attribute of the `<ui:composition>` tag specifies the template to invoke, which is `myTemplate.xhtml` in this application. The `<ui:define>` tag is utilized to pass the parameter to the template and is a sub element of the `<ui:composition>` tag. The following code snippet shows the line in Listing C.5, which is to be noticed:

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

In the preceding code snippet, the namespace inclusion permits the page to use the JSF's facelet, html components. Listing C.5 uses the `<ui:define>` tag to invoke the section of template, which is defined with the `<ui:insert>` tag in the template page. The `name` attribute of the `<ui:define>` tag must match with the `name` attribute of the `<ui:insert>` tag in the template page. Listing C.5 uses the `<h:outputText>` tag to output a value. The `value` attribute of the tag signifies the value to print. The `value` `msgs` is the name of the property file, which is `messages.properties`, provided in the `faces-config.xml` JSF configuration file. The property file is discussed under the *Creating the Property File* heading. You should note that `Hello` and `AppTitle` are the properties of the `messages.properties` file. `DemoBean` is the name of Java class in this application, which is discussed under the *Creating a Java Class* heading.

Listing C.6 provides the code of the `myTemplate.xhtml` (you can find this file on CD in the `code\JavaEE\AppendixC\JSFAppEx\web\` folder)::

Listing C.6: Showing the Code of the myTemplate.xhtml File

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">
<head>
</head>
<body>
    <h1>
        <ui:insert name="title">
    </h1>
    <p>
        <ui:insert name="body"/>
    </p>
</body>
</html>
```

When the user submits the name in the `DemoMain.xhtml` file, the `DemoHello.xhtml` page is displayed to the user. The navigation in JSF is fully handled by the Web application, according to the rule specified in the `faces-config.xml` file. The `DemoHello.xhtml` page displays the message `Hi name of the user Welcome to the`

JSF Application. Listing C.7 shows the code of the DemoHello XHTML page (you can find this file on CD in the code\JavaEE\AppendixC\JSFAppEx\web\ folder):

Listing C.7: Showing the Code of the DemoHello.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:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html">
<body>
<ui:composition template="/myTemplate.xhtml">
<ui:define name="body">
<h:outputText value="#{msgs.Hello}"/>
    &nbsp; <h:outputText value="#{DemoBean.name}"/>
    &nbsp; <h:outputText value="#{msgs.AppTitle}"/>
</ui:define>
</ui:composition>
</body>
</html>
```

Creating the Properties File

The properties class is used by the view component to display the text. One of the uses of the property file is to avoid repetition of the same text in various pages within a Web application. The text to be displayed multiple times is written inside the properties file in the form of property with a key and a value. The value of a property in the properties file is the text that is displayed in the Web page. The key of the property is used in the Web pages to display the value of the property. To create a property file, the following steps are performed:

4. Right click the package node and select New→Other→Other→Properties file.

The New Properties File dialog box appears to create a new properties file, enabling the user to enter the name of the properties file and package. Figure C.27 shows the New Properties File dialog box that enables the user to enter the name and location for the properties file that you want to create:

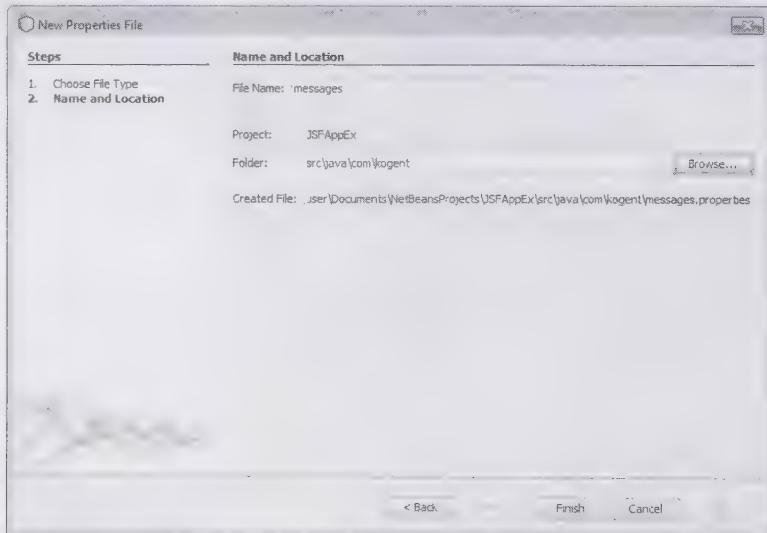


Figure C.27: Showing the Window to Enter Name and Location for a New Properties File

5. Click the Finish button (Figure C.27).

The messages.properties file is created. To add the property to the property file, right-click the properties file node and select Add→Property. The New Property dialog box appears, which enables the user to enter the name and value of the property.

Figure C.28 shows the New Property dialog box to create the Hello property:

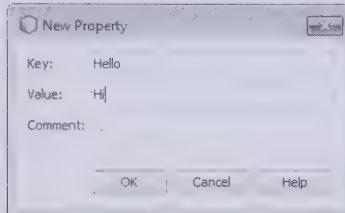


Figure C.28: Showing the New Property Dialog Box

Similarly, two other properties of the messages.properties file can be created, which are YourName and AppTitle.

The code of the messages properties file is shown in Listing C.8:

Listing C.8: Showing the Code of the messages.properties File

```
Hello=Hi
AppTitle=Welcome to the JSF Application
YourName=what is your name?
```

Creating the Java Class

You can create a Java class by performing the following steps:

Right click the project node and select the New→Java Class option.

1. The New Java Class dialog box opens, which enables the user to enter the name of class and package. Provide the value DemoBean in the Class Name text box and com.kogent in the Package text box.

Figure C.29 shows the New Java Class dialog box:

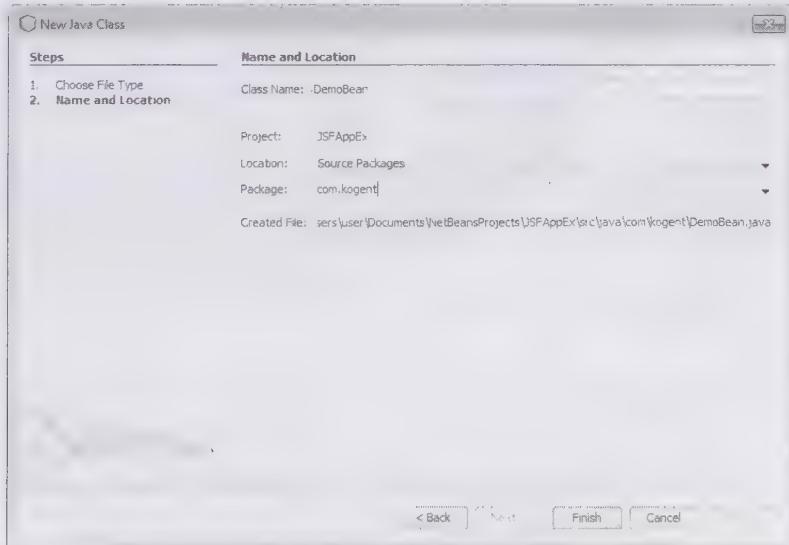


Figure C.29: Showing the New Java Class Dialog Box

2. Click the Finish button (Figure C.29) to create the DemoBean class.

Now, it's the time to write code for the DemoBean.java file. Listing C.9 shows the code of the DemoBean Java class (you can find this file on CD in the code\JavaEE\AppendixC\JSFAppEx\src\java\com\kogent\ folder):

Listing C.9: Showing the Code of the DemoBean.java File

```
package com.kogent;
import javax.faces.bean.ManagedBean;
```

```

import javax.faces.bean.RequestScoped;
import javax.validation.constraints.Pattern;
@ManagedBean(name="DemoBean")
@RequestScoped
public class DemoBean {
    private String name;
    public DemoBean() {
    }
    public String sayDemo() {
        return "hi";
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}

```

In Listing C.8, the @ManagedBean annotation is used to specify that the class is a managed bean and the name attribute specifies the name of the managed bean. The class has the name attribute, whose type is String and the sayDemo() method.

Creating the Faces-Config.xml File

You can create the JSF configuration file, which is a faces-config.xml file, by performing the following steps:

1. Right click the project node and select the New→Other→JavaServer Faces→JSF Faces Configuration option. The New JSF Faces Configuration dialog box appears.

Figure C.30 shows the New JSF Faces Configuration dialog box:

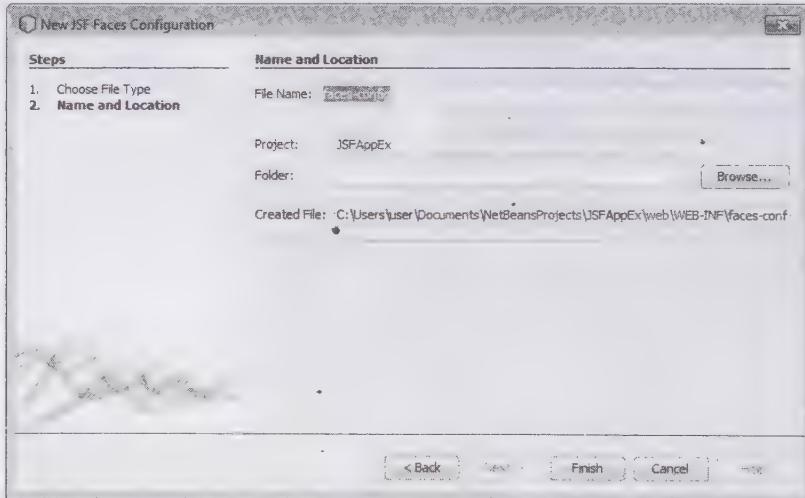


Figure C.30: Showing the New JSF Faces Configuration Dialog Box

2. Click the Finish button (Figure C.30).

The faces-config.xml file appears in the Project window.

Listing C.10 shows the code of the faces-config.xml JSF configuration file (you can find this file on CD in the code\JavaEE\AppendixC\JSFAppEx\web\WEB-INF\ folder)::

Listing C.10: Showing the Code of the Faces-Config.xml File

```

<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.0"

```

```

<xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd">
<application>
  <resource-bundle>
    <base-name>com.kogent.messages</base-name>
    <var>msgs</var>
  </resource-bundle>
</application>
<navigation-rule>
  <from-view-id>/DemoMain.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>hi</from-outcome>
    <to-view-id>/DemoHello.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
</faces-config>

```

In Listing C.10, the `<base-name>` tag, which is a sub-element of the `<resource-bundle>` tag, is used to specify the location of the `messages.properties` file. The navigation rule is specified using the `<navigation-rule>` tag, according to which navigation occurs in the application. The `<from-view-id>` tag specifies the view page from where navigation will start. The `<from-outcome>` tag specifies the String, which determines the view that is presented to the user. The `<to-view-id>` tag specifies the view that is presented to the user.

In the `JSFAppEx` application, when the user enters his/her name in the `DemoMain` XHTML page and submits, the `sayDemo()` method of the `DemoBean` class is invoked. As a result of the `sayDemo()` method invocation, the String value, `hi`, is returned. According to the rule specified in the `faces-config.xml` file, the `DemoHello` page is displayed to the user.

Building and Running the JSF Application

To build the `JSFAppEx` project, select the `Build→Build Main Project` option. After the project is built successfully, you can run the application to view its output. Right click the project node and select the `Run` option to run the JSF application. On execution of the `JSFAppEx` application, the `DemoMain` page is displayed to the user, that is, the Welcome page. Figure C.31 shows the output of the `DemoMain.xhtml` page:

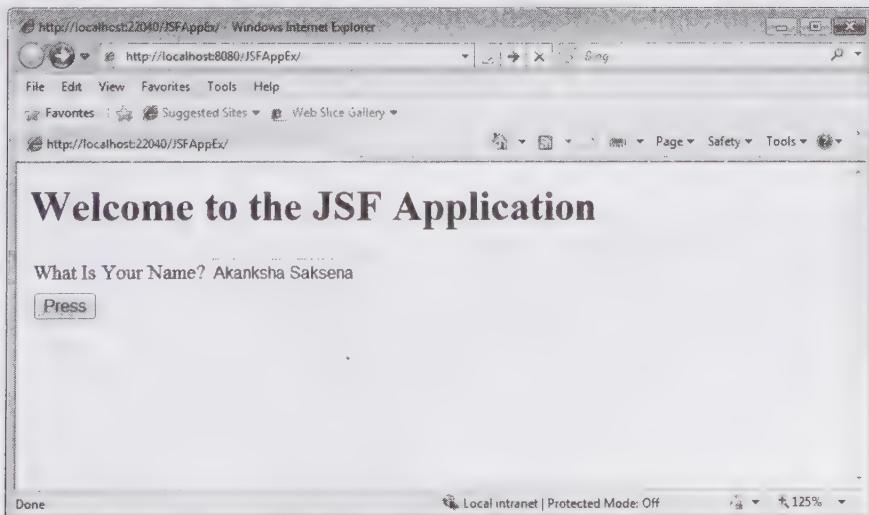


Figure C.31: Showing the Output of the DemoMain.xhtml File

By entering the name and clicking the Press button, the `DemoHello` XHTML page is displayed to the user.

Figure C.32 shows the output of the DemoHello page:

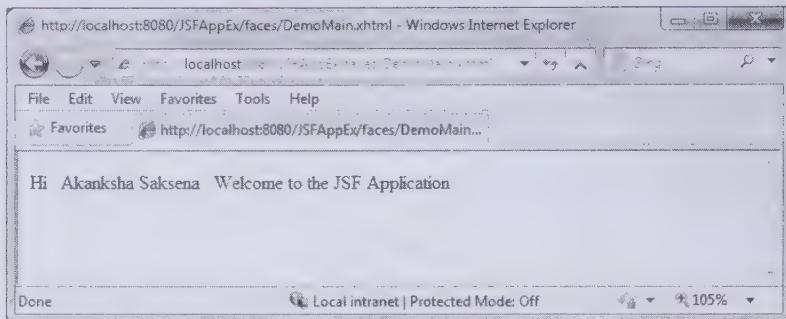


Figure C.32: Showing the Output of the DemoHello.xhtml File

This appendix has helped you to learn about the new features of the NetBeans IDE 6.8. You have also learned to develop Web application, enterprise application, and JSF applications using NetBeans IDE 6.8.



D

Implementing Internationalization

Internationalization can be defined as the process that enables you to use a single Web application in different country and region-specific formats, without making any changes in the source code of the Web application or recompiling the application. Globalization of Web applications can be accomplished through the process of Internationalization and Localization of variables. In today's globalized era, business transactions involve customers across the world. Consequently, with the advent of globalization, the importance of globalizing or internationalizing Web applications has also increased. Globalizing Web applications is important because Web applications are accessed by users from various regions and countries. The customers generally expect data to be presented according to their specific culture and locale, especially when it comes to the language and data formats. Therefore, Web applications, which are accessed by users worldwide, must support different languages, text formats, number and date formats, and currency formats being used in different regions of the world. In other words, Web applications must present information in many languages according to different users from different locales.

The term Internationalization is generally abbreviated as i18n, as there are 18 characters between i and n. The process of customizing software or a Web application for a specific region or language using the locale-specific components is known as Localization. It enables Web applications to adapt message, number, and date formats according to a given locale. Localization can be abbreviated as l10n, as this word contains 10 letters between l and n.

This appendix provides an overview of Java core Internationalization and Localization Application (Programming Interfaces APIs), and explains how to implement Internationalization of Web applications.

Java and Internationalization

Java Standard Edition (Java SE) provides a rich set of APIs to support Internationalization and Localization of an application. This section deals with a brief introduction about how to use these APIs to implement Internationalization and Localization in Java applications. The Internationalization APIs provided with Java SE can easily adapt different messages, number, date, and currency formats according to different country and region conventions. Java also provides support for Unicode character sets and a rich set of APIs to manage locale-specific content. The Internationalization and Localization APIs specified under Java SE platform is based on the Unicode 3.0 character encoding.

Java SE provides two common classes to implement Internationalization, namely, `Locale` and `ResourceBundle`. Let's explore these two classes one by one in the following subsections.

Describing the Locale Class

The `java.util.Locale` class is a part of `java.util` package that encapsulates the country, language, and variant of a specific locale, and is used while creating Java applications for internationalization. In other words, locale is a relatively simple object that defines the specific language and geographic region. These Locale objects affect the language representation, calendar usage, date and time formats, number and currency formats, and many other sensitive data representation. All other locale-oriented classes use the Locale object to adapt to the specific locale and provide the output accordingly.

The Locale class consists of the following three constructors that allow us to construct a Locale object according to the specific requirements:

- ❑ **Locale (String language)**—Constructs a Locale object with the given language code
- ❑ **Locale (String language, String country)**—Constructs a Locale object with the given language and country code
- ❑ **Locale (String language, String country, String variant)**—Constructs a Locale object with the given language, country, and variant

The Locale object depicts the language and cultural preferences of a specific geographical area. Dates, time, numbers, and currency are the examples of data formatted according to the cultural expectations of the customers.

Parameters of the Locale Object

The Locale objects represent a geographic, political, or cultural region. The Locale objects are used for the purpose of implementing Internationalization in a Java application. Three parameters are used to define Locale objects, which are as follows:

- ❑ Language
- ❑ Country
- ❑ Variant

The Language Parameter

The language parameter provides the valid language codes that can be used to construct a Locale object. This parameter is specified by the International Standard Organization 639 (ISO-639) standard. The valid language code must be a two-letter language code in lower case, such as `en` for English, `hi` for Hindi, and `te` for Telugu. For a complete list of valid language codes, refer to the following URL:

<http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt>

The Country Parameter

The country parameter describes valid two-letter codes for a country's name, and is also specified by the ISO 3166 standard. The country code must be in uppercase, as specified by ISO 3166. However, the Locale constructor allows lowercase letters because it converts the lowercase letters into uppercase code to maintain the correct internal representation. Some of the valid country codes are `IN` for India, `US` for United States, and `GB` for United Kingdom. For a complete list of valid country codes, refer to the following URL:

http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

The Variant Parameter

The variant parameter is used as an extension that provides more description about the Locale object. Variants can be used to add additional context with the Locale and make it more descriptive. For example, when a Locale `en_US` is used, it represents English (United States); however, when `en_US_CA` is used, it specifies more information to identify the Locale (California, USA).

Locale-sensitive operations, such as `getDateInstance()`, `getTimeInstance()`, and `getDateTimeInstance()`, use the Locale parameter to retrieve data, time, and date and time instance. These Locale-sensitive operations behave in different ways, depending on the Locale, and they also format the information according to the customs/conventions of the user's native country, region, or culture.

Using the Locale Class

We have aforementioned that the Locale class has three constructors; the following code snippet describes how to use either of the constructors to create a Locale object:

```
Locale myLocale = new Locale("en", "US");
```

In the preceding code snippet, en represents English and US is used for the United States of America. Moreover, the Locale object can also be constructed by using variant. The following code snippet creates a Locale object with the optional variant, which can be used to create a specific Locale, as compared to the Locale created by using just language and country codes:

```
Locale myLocale = new Locale("en", "US", "VENTURA");
```

The Locale class contains the following access methods:

- `getLanguage()`
- `getCountry()`
- `getVariant()`
- `toString()`

The `getLanguage()` method returns the ISO language and the `getCountry()` method returns the country codes, which comprise a Locale object. However, these codes may not be much user friendly. Other methods such as `getDisplayLanguage()` and `getDisplayCountry()` return String objects, which are easily understandable by the users.

Listing D.1 creates the `myLocale` object, which provides an access to the `getDisplayLanguage()` and `getDisplayCountry()` methods. The `getDisplayLanguage()` and `getDisplayCountry()` methods are locale-sensitive methods, which implies that the Locale parameter can be provided to retrieve the language or country string in the target language passed as a parameter. The `Locale.FRENCH` locale gives the reference of the Locale object for FRENCH language (you can find the `Locales.java` file on the CD in the `code\JavaEE\AppendixD\intl` folder), as shown in Listing D.1:

Listing D.1: Showing the Code for the `Locales.java` File

```
package com.kogent.i18n;

import java.util.Locale;
public class Locales
{
    public static void main(String s[]) throws Exception
    {
        Locale myLocale=null;
        String language=null;
        String country=null;
        myLocale = new Locale("en", "US");
        language = myLocale.getDisplayLanguage();
        System.out.println(language);
        country=myLocale.getDisplayCountry();
        System.out.println(country);
        System.out.println(myLocale.getDisplayLanguage(Locale.FRENCH));
        System.out.println(myLocale.getDisplayCountry(Locale.FRENCH));
    }
}
```

Figure D.1 displays the output after compiling Listing D.1:

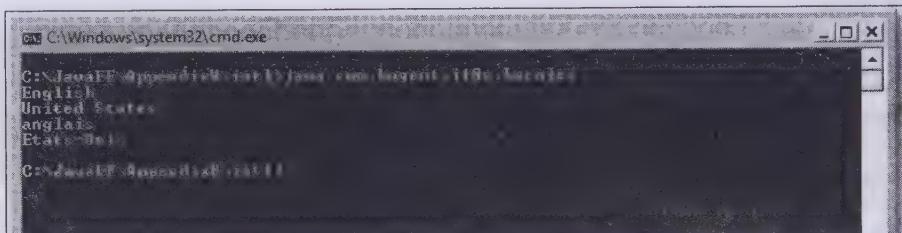


Figure D.1: Displaying the Output of Locales.java File

In Figure D.1, the language displayed is English and the country is United States. Notice that Figure D.1 also displays the language and country name in FRENCH.

Now, after understanding the `Locale` class and learning how to create a `Locale` object, let's discuss about the `ResourceBundle` class.

Describing the ResourceBundle Class

The `ResourceBundle` class helps to separate the User Interface (UI) elements as well as other locale-sensitive data from the application layer. The `ResourceBundle` class is used to segregate localizable elements, such as buttons, labels, error messages, and headings, from the rest of the application. These localizable elements are bundled into a `.properties` file, known as resource bundle, which contains either the resource (UI elements and local-sensitive data) itself or a reference to it. Therefore, all the resources are bundled into different `ResourceBundle` classes, and the Java application only needs to load the appropriate bundle for the locale.

The `ResourceBundle` names have two parts, a base name and a locale suffix. The base name defines the name of the default `ResourceBundle`, and the suffix defines the specific locale. For example, suppose you create a `ResourceBundle` named `myresBundle`. Now, imagine you have translated `myresBundle` for two different locales, namely, `en_US` and `ja_JP`. The `myresBundle` `ResourceBundle` will be the default `ResourceBundle`, and you need to create two more bundles, namely, `myresBundle_en_US` and `myresBundle_ja_JP`, for the two different locales `en_US` and `ja_JP`, respectively. The `ResourceBundle.getBundle()` method searches the required `ResourceBundle` object for an active locale depending on the naming convention for the locale.

The `ResourceBundle` class is a Locale-specific class that is used by a program to retrieve an object for a particular Locale. The `ResourceBundle` object encapsulates locale-specific resources for an application, where the resource can be stored in a list or a properties file. The `ResourceBundle` class is an abstract class in `java.util` package that cannot be used directly for invoking methods.

The `ResourceBundle` class has two direct subclasses:

- ❑ **PropertyResourceBundle** – Provides a mechanism by using which the properties files store the resources. This `PropertyResourceBundle` class is the most widely and commonly used class to work with `ResourceBundle` in Java.
- ❑ **ListResourceBundle** – Uses lists to store `ResourceBundle`.

Let's now discuss these two subclasses next.

The PropertyResourceBundle Class

A `PropertiesResourceBundle` class manages the resources for a `Locale` object by using a set of static strings from a properties file. A properties file is a plain text file containing editable text. The syntax of the name of the properties file is as follows:

`basename_language_country.properties`

The `getBundle()` method of the `ResourceBundle` class automatically looks for the properties file provided as a parameter. Then, a `PropertyResourceBundle` class is created that refers to the specified properties file. A call to the `ResourceBundle.getBundle (ApplicationResources, new Locale (en, US))` method attempts to retrieve the object of the `PropertyResourceBundle` class for the file with the name `ApplicationResources_en_US.properties`.

The `PropertyResourceBundle` class can be used by creating a properties file that contains the key/value pairs in the form of `<key>=<value>`. Each key/value pair is listed in the same line of the `.properties` file, and each pair is

separated with the new-line character. Listing D.2 explains the implementation of the key/value pairs (you can find the Applications_en.properties file on the CD in the code\JavaEE\Appendix H\intl folder):

Listing D.2: Showing the Code of Applications_en.properties File

```
FILE_NOT_FOUND = The file could not be found
HELLO_MESSAGE = Hello, How are you?
WARNING_MESSAGE = There are 0 warnings in this file.
```

Listing D.2 can be saved in the Applications.properties file, and this properties file (or resource bundle) can be loaded by calling the ResourceBundle.getBundle (Applications) method. After loading the ResourceBundle class, individual elements of the properties file can be loaded by using the getString() method.

The ResourceBundle object can be obtained by using any one of its static getBundle () methods according to the requirements. These methods use the PropertyResourceBundle class as a reference to the specified property class. The getBundle() method with different type and number of arguments are given as follows:

- **getBundle(String baseName)**—Returns a ResourceBundle object. This method takes baseName as a String argument and uses default Locale and caller's class loader.
- **getBundle(String baseName, Locale locale)**—Returns a ResourceBundle object. This method takes a baseName and a Locale as string arguments.
- **getBundle (String baseName, Locale locale, ClassLoader loader)**—Returns a ResourceBundle object. This method takes a baseName, a Locale, and a class loader as string arguments.

By default, the getBundle() method searches for the .class file, but uses the .properties file, if it exists, instead of the .class file. The PropertyResourceBundle class has a significant limitation that all values are limited to string objects because the .properties file contains only text. Therefore, only text strings can be placed in the PropertyResourceBundle class. So, for more complex key/value pairs, the ListResourceBundle class may be used. ListResourceBundle class is used instead of the PropertyResourceBundle class because of complex data containing multiple strings that need to be provided to configure certain Classpath components.

After understanding the PropertyResourceBundle class, let us create a simple Java application implementing the PropertyResourceBundle class. The following code snippet describes the PropertyResourceBundle class for the en locale:

```
//ApplicationResources_en.properties
welcome.message=Hello English user, welcome to internationalization
```

In the preceding code snippet, the value specified for the key “welcome.message”, is a welcome message for English users. The following code snippet describes the PropertyResourceBundle class for the en_US locale:

```
//ApplicationResources_en_US.properties
welcome.message = Hello US English user, welcome to internationalization
```

In the preceding code snippet, the value specified for the key “welcome.message”, is a Welcome message for US English users. The following code snippet describes the PropertyResourceBundle class for the it (Italian) locale:

```
//ApplicationResources_it.properties
welcome.message = Hello Italian user, welcome to internationalization
```

In the preceding code snippet, the value specified for the key welcome.message is a Welcome message for Italian users.

Listing D.3 provides the code for the I18NTest.java file, which loads the ResourceBundle class based on the locale provided by the user. The getString() method then retrieves the value of the welcome.message key defined in the ResourceBundle class, which is based on the Locale provided by the programmer. Listing D.3 shows the code for the I18NTest class (you can find the I18NTest.java file on the CD in the code\JavaEE\Appendix D\intl folder):

Listing D.3: Showing the Code of I18NTest.java File

```
package com.kogent.i18n;
import java.util.*;
public class I18NTest
{
```

```

public static void main(String s[])
{
    Locale l=null;
    if (s.length==3)
        l=new Locale(s[0],s[1],s[2]);
    else if (s.length==2)
        l=new Locale(s[0],s[1]);
    else
        l=new Locale(s[0]);
    ResourceBundle
        rb=ResourceBundle.getBundle("ApplicationResources",l);
    System.out.println(rb.getString("welcome.message"));
}//end main
}//end class

```

Execute the I18NTest.java file after compiling it. The output is as displayed in Figure D.2:

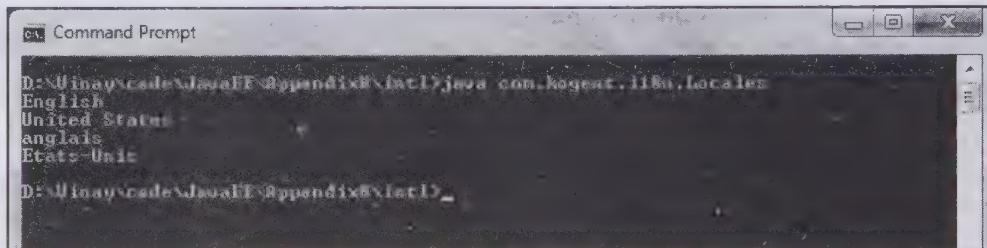


Figure D.2: Displaying the Output of I18NTest.java File

As shown in Figure D.2, when the I18NTest Servlet is executed with different arguments, you can observe the respective locale-specific messages.

After understanding and implementing the `PropertyResourceBundle` class, let's now discuss the `ListResourceBundle` class, which is more appropriate for complex key/value pairs.

The ListResourceBundle Class

The `ListResourceBundle` class is a bit complex as compared to the `PropertyResourceBundle` class because it handles complex data. In addition, the `PropertyResourceBundle` class can store only text, whereas the `ListResourceBundle` class can contain any type of Java objects. Since the `ListResourceBundle` class is an abstract class, you must extend it to create a usable class, as shown in Listing D.4:

Listing D.4: Showing the Code of ListBundle_en_CA.java File

```

import java.util.*;
public class ListBundle_en_CA extends ListResourceBundle
{
    public Object[][] getContents()
    {
        return contents;
    }
    private Object[][] contents =
    {
        { "GDP", new Integer(51300) },
        { "Population", new Integer(125440000) },
        { "Literacy", new Double(0.67) },
    };
}

```

Similar to the `PropertyResourceBundle` class, the `ListResourceBundle` class also contains the list of key/value pairs. However, in the `ListResourceBundle` class, these key/value pairs are arranged as elements in a two-dimensional array of `java.lang.Object`. Moreover, the `ListResourceBundle` class should use the single method `getContents()`, as well as an array of the `Object` type that lists the key/value pairs. In Listing D.4, the

key/value pairs are arranged in the form of elements of a two-dimensional array of the `java.lang.Object` type objects. These key/value pairs are returned by the `getContents()` method to provide the values for the `en_CA` locale.

Listing D.5 shows the code of the `ListBundle_fr_FR.java` file that defines the key/value pairs elements in a two-dimensional array, providing the values for the `fr_FR` locale (you can find the `ListBundle_fr_FR.java` file on the CD in the `code\JavaEE\AppendixD\intl` folder):

Listing D.5: Showing the Code of the `ListBundle_fr_FR.java` File

```
import java.util.*;
public class ListBundle_fr_FR extends ListResourceBundle
{
    public Object[][] getContents()
    {
        return contents;
    }
    private Object[][] contents =
    {
        { "GDP", new Integer(15300) },
        { "Population", new Integer(125447800) },
        { "Literacy", new Double(0.55) },
    };
}
```

Listing D.6 shows the code of the `Listbundle_ja_JP.java` file that defines key/value pairs elements in a two-dimensional array, providing the values for the `ja_JP` locale (you can find the `ListBundle_ja_JP.java` file on the CD in the `code\JavaEE\AppendixD\intl` folder):

Listing D.6: Showing the Code of the `ListBundle_ja_JP.java` File

```
import java.util.*;
public class ListBundle_ja_JP extends ListResourceBundle
{
    public Object[][] getContents()
    {
        return contents;
    }
    private Object[][] contents =
    {
        { "GDP", new Integer(21300) },
        { "Population", new Integer(125449703) },
        { "Literacy", new Double(0.99) },
    };
}
```

Listing D.7 demonstrates how to display the values of the `ListResourceBundle` class, depending on the user-specified locale (you can find the `ListResBundleDemo.java` file on the CD in the `code\JavaEE\AppendixD\intl` folder):

Listing D.7: Showing the Code of `ListResBundleDemo.java` File

```
import java.util.*;
public class ListResBundleDemo {
    static public void main(String[] args) {
        Locale supportedLocales=null;
        if (args.length==3)
            supportedLocales=new Locale(args[0],args[1],args[2]);
        else if (args.length==2)
            supportedLocales=new Locale(args[0],args[1]);
        else
            supportedLocales=new Locale(args[0]);
        ResourceBundle stats=ResourceBundle.getBundle("ListBundle",supportedLocales);
        System.out.println("Locale = "+supportedLocales);
```

```

        Integer gdp = (Integer)stats.getObject("GDP");
        System.out.println("GDP = " + gdp.toString());
        Integer pop = (Integer)stats.getObject("Population");
        System.out.println("Population = " + pop.toString());
        Double lit = (Double)stats.getObject("Literacy");
        System.out.println("Literacy = " + lit.toString());
    } // end main
} // end class

```

In Listing D.7, the ResourceBundle class is loaded based on the locales specified by the user. The `getObject()` method then retrieves the value of the specified parameter and displays it, as shown in Figure D.3:

```

Command Prompt

D:\Minay\code\JavaEE\AppendixD\List1>java com.kogent.i18n.I18NTest en
Hello English user. Welcome to internationalization.

D:\Minay\code\JavaEE\AppendixD\List1>java com.kogent.i18n.I18NTest fr
Hello & S English user. Welcome to internationalization.

D:\Minay\code\JavaEE\AppendixD\List1>java com.kogent.i18n.I18NTest it
Hello Italian user. Welcome to internationalization.

D:\Minay\code\JavaEE\AppendixD\List1>

```

Figure D.3: Displaying the Output of ListResBundleDemo.java File

In Figure D.3, the Gross Domestic Product (GDP), Population, and Literacy data for the en_CA, fr_FR, and ja_JP locales are displayed.

Now, let's learn about internationalizing Web applications.

Internationalizing Web Applications

You can use Internationalization classes available in Java APIs to develop Internationalized Web applications in Java. In addition, Java handles text internally in Unicode, so it can represent almost all the languages as long as the client browser is capable of displaying the character set.

Internationalization is an approach that provides support for more than one language and data format representation. For internationalizing Java Web application, you must first decide how to determine the user's language and locale preferences. A Web application has two ways to find out the user's language preferences. The first way is to use locale preferences that are sent from the client to the server using the Hypertext Transfer Protocol (HTTP) request header field Accept-Language. Browsers usually allow the user to create a list of languages as part of their preferences. Java servlet API provides a utility method to retrieve the objects of different locales using the `getLocale()` and `getLocales()` methods of the `javax.servlet.ServletRequest` interface. The second approach to ascertain user's language preferences is to provide a user view, allowing the user to choose the language from a list of supported languages, and get it as a request parameter. Use of the Accept-Language information during the earlier stage of Internationalization is a good option; however, the user must be given the opportunity to choose a language explicitly from the view page.

Now, after determining the locale for presenting localized content, there are two approaches to internationalize a Web application, which are as follows:

- ❑ The first approach maintains that for each targeted Locales, there should exist a JavaServer Pages (JSP) page, which a Servlet can dispatch as the appropriate response page after processing a user request. This approach is beneficial if large amount of data is available on a page or an entire Web application needs to be internationalized, because the application contains JSP pages for each locale. The disadvantage of this approach is that you need to create separate JSP pages for each locale.
- ❑ The second approach is to isolate the locale-sensitive data on a page into resource bundles (.properties files). The locale-sensitive data can be an error message, string literals, or button labels. The locale-sensitive data can be automatically retrieved from the .properties files and used in the JSP page. This approach does not require specifying the text data into the code; rather, the resource bundle can be created, which contains the locale-sensitive data in the form of key/value pairs. This approach may be preferred if the pages contain mainly text, or if the structures of JSP pages are significantly different for different locales.

After discussing both the approaches used to internationalize Web applications, let's create a simple application named `i18nWebex1`, which demonstrates how to internationalize Web applications. The `i18nWebex1` application consists of the following components:

- ❑ The `index.html` view
- ❑ The `Home.jsp` view
- ❑ The `I18NServlet` servlet class
- ❑ The `ApplicationResourceBundle Properties` (`ApplicationResource_en.properties`, `ApplicationResource_en_US.properties`, and `ApplicationResource_it.properties`) resource bundles

The basic view components designed are `index.html` and `Home.jsp`. The request sent by the user through the `index.html` page invokes the `I18NServlet` servlet class, which sets an attribute and forwards the request to the `Home.jsp`, which displays the final result.

Creating the Views

The view components are designed by using JSP and HyperText Markup Language (HTML). In this application, we design one JSP and one HTML page, which are as follows:

- ❑ `index.html`—Serves as the first view (or page) of the application
- ❑ `Home.jsp`—Refers to the view displaying the output based on the language selected by the user

Now, let's create these views.

Creating the index.html View

The `index.html` view is the first page that appears when the application is accessed. This page provides the link for the English, English (US), and Italian (IT) users. The source code for the `index.html` file is shown in Listing D.8 (you can find this file on the CD in the `code\JavaEE\AppendixD\i18nWebex1` folder):

Listing D.8: Showing the Code for the `index.html` File

```
<html>
<body>
<center>
This is an example to demonstrate the internationalization of web application, click on
the following links to find the output<pre>
    <a href="testser?language=en">English User </a>
    <a href="testser?language=en&country=US">English (US) User</a>
    <a href="testser?language=it">Italian User</a>
</pre>
</center>
</body>
</html>
```

The hyperlinks included in Listing D.8 automatically call the `I18NServlet` servlet class. You can see the relationship between the `index.html` view and the `I18NServlet` servlet class in the Servlet-mapping provided in the `web.xml` file. In Listing D.8, two parameters, `language` and `country`, are passed with the request when the user clicks the reference link.

Creating the Home.jsp Page

The `Home.jsp` page is designed to display the welcome message to the users based on the language preference selected by them in the `index.html` view. Based on the language and country parameters, the resource bundle is loaded in the `Home.jsp` file, as shown in Listing D.9 (you can find this file on the CD in the `code\JavaEE\AppendixD\i18nWebex1` folder):

Listing D.9: Showing the Code for the `Home.jsp` File

```
<%@page import="java.util.*"%>
<jsp:include page="index.html"/>
<hr>
<h2>Output</h2><br/>
```

```
<%
    ResourceBundle rb= (ResourceBundle) request.getAttribute("resource");
%>
<%=rb.getString("welcome.message")%>
```

Listing D.9 also sets the resource bundle value to the resource attribute. In Listing D.9, the value of the resource attribute is retrieved and the welcome message, saved in the ResourceBundle class, is displayed by using the key welcome.message.

After designing the view components, let's now create the I18NServlet servlet class, which loads the ResourceBundle class.

Creating the I18NServlet Servlet Class

I18NServlet is the HTTPServlet class that loads the ResourceBundle class based on the parameters passed by the index.html view. The I18NServlet servlet class retrieves the value of the parameters passed by the index.html page in Listing D.8. Then, based on these parameters, the new Locale instance, l, is initialized and the ResourceBundle class is loaded according to the l Locale instance. Then, the new attribute resource is created and the request is forwarded to the Home.jsp page.

After compiling the Servlet file, put the I18NServlet.class file in the WEB-INF\classes\com\kogent\servlets folder. Listing D.10 provides the code of the I18NServlet.java file (you can find this file on the CD in the code\JavaEE\AppendixD\i18nWebex1 folder):

Listing D.10: Showing the Code for the I18NServlet Servlet Class

```
package com.kogent.servlets;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class I18NServlet extends HttpServlet {
    public void service (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        String cc=req.getParameter("country");
        String ln=req.getParameter("language");
        Locale l=null;
        if (cc==null)
            l=new Locale(ln);
        else
            l=new Locale(ln,cc);
        ResourceBundle rb= ResourceBundle.getBundle("ApplicationResources",l);
        req.setAttribute("resource",rb);
        RequestDispatcher rd=req.getRequestDispatcher("Home.jsp");
        rd.forward(req,res);
    }//end service
}// end class
```

Now, let's us create the various resource bundles for the English, English (US), and Italian (IT) users, required in this application.

Creating the Resource Bundles

The PropertyResourceBundle class allows the application to define resources using the key/value syntax. Let's create three resource bundles (or the .properties files) for three specific locales, which are as follows:

- ❑ ApplicationResource_en.properties(for English)
- ❑ ApplicationResource_en_US.properties (for English (US))
- ❑ ApplicationResource_it.properties (for Italian)

These files are added to the i18nWebex1 application, which prepares the resource bundle with the welcome String message. Listing D.11 shows the code of the ApplicationResources_en.properties file (you can find this file on the CD in the code\JavaEE\AppendixD\i18nWebex1\WEB-INF\classes folder):

Listing D.11: Showing the ApplicationResources_en.properties File

```
welcome.message=Hello English user, Welcome to internationalization
```

In Listing D.11, the key welcome.message is defined for the en Locale. Listing D.12 shows the code of the ApplicationResources_en_US.properties file (you can find this file on the CD in the code\JavaEE\AppendixD\i18nWebex1\WEB-INF\classes folder):

Listing D.12: Showing the ApplicationResources_en_US.properties File

```
welcome.message=Hello US English user, Welcome to internationalization
```

In Listing D.12, the key welcome.message is defined for the en_US Locale. Listing D.13 shows the code of the ApplicationResources_it.properties file (you can find this file on the CD in the code\JavaEE\AppendixD\i18nWebex1\WEB-INF\classes folder):

Listing D.13: Showing the ApplicationResources_it.properties File

```
welcome.message=Hello Italian user, Welcome to internationalization
```

In Listing D.13, the key welcome.message is defined for the it Locale.

Configuring the Application

The deployment descriptor is used to configure and map the servlet used in the i18nWebex1 application. The web.xml configuration file is used to configure the i18nWebex1 application.

The web.xml file defines the servlet name, the servlet class, and the URL pattern for the I18NServlet servlet class. The URL pattern for the I18NServlet servlet class is /testser. Listing D.14 provides the code of the web.xml file (you can find this file on the CD in the code\JavaEE\AppendixD\i18nWebex1\WEB-INF folder):

Listing D.14: Showing the Code for the web.xml File

```
<web-app>
  <servlet>
    <servlet-name>i18n</servlet-name>
    <servlet-class>com.kogent.servlets.I18NServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>i18n</servlet-name>
    <url-pattern>/testser</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Now, after creating all the required files, let's understand how to arrange these files in the directory structure.

Designing the Application Directory Structure

The directory structure of the application is shown in Figure D.4:

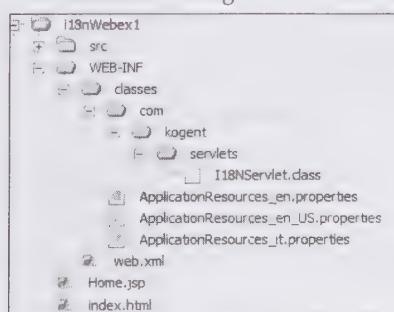


Figure D.4: Displaying the Root Directory Structure for i18nWebex1 Web Application

Figure D.4 shows that all the aforementioned created files are placed in their respective folders, and the contents of the respective folders are described as follows:

- All packages containing class files are placed in the WEB-INF/classes folder
- All message-resource properties files are placed in the WEB-INF/classes folder within the appropriate package
- The web.xml configuration file is placed in the WEB-INF folder
- You can put the src folder containing source files (.java files) for all class files under the application directory folder

Let's now run the i18nWebex1 application.

Running the Application

After developing and configuring various components, deploy the application on the GlassFish V3 server. Now, you can run the application to see the output of various JSP pages and the implementation of Internationalization. Open your browser and access the application using the `http://localhost:8080/i18nWebex1/` URL. The index page appears, as shown in Figure D.5:

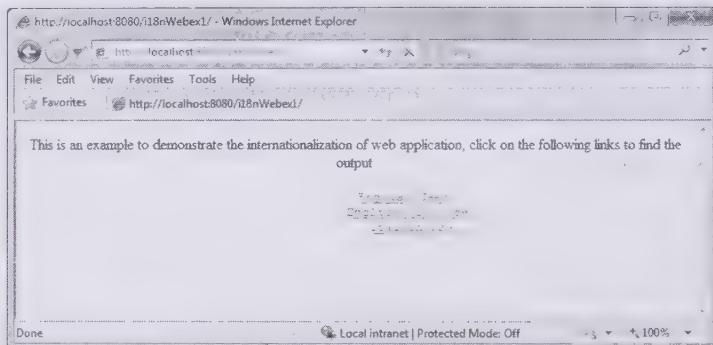


Figure D.5: Displaying the index.jsp Page

Click any of the three hyperlinks to get the message from the respective locale properties file. For example, if you have clicked the English (US) User hyperlink, the output will be as displayed in Figure D.6:

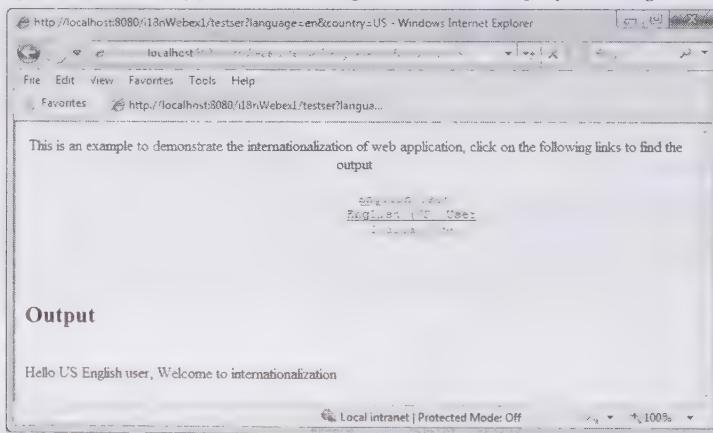


Figure D.6: Displaying the Home.jsp Page

In this example, the user's locale is determined by allowing the user to select his required language using the given links. Instead, you can get the user's locale directly by using the `getLocale()` method of the `ServletRequest` interface.



Working with Facelets

Introducing Facelets

In an enterprise application (3-tier application), there are three layers, namely presentation, business logic, and database. The presentation layer of the enterprise application contains various View components that are designed using different presentation technologies, such as JavaServer Faces (JSF). While using JSF, developers had to create View pages with the help of various Web designing tools, such as Dreamweaver. However, the process of creating JSF View pages with the Web designing tools is very complex; therefore, the developers had to explore some alternatives to easily build the pages. Later, JavaServer Pages (JSP) was used as a presentation technology for JSF. However, with the evolution of JSF 2.0, various new features were introduced which were not supported by JSP. Therefore, Facelets was introduced with Java EE 6 as a presentation technology to build the View pages for JSF 2.0. Facelets also supported all the new features of JSF 2.0.

Facelets is an effective and lightweight presentation technology that allows you to use Hypertext Markup Language (HTML) style templates to develop Views in JSF applications. Moreover, the compilation of the Views developed by using Facelets is much faster than the Views generated using JSP. This is because when a user accesses the View created using Facelets, the bytecode for that View is not generated by the compiler, which is done in case of JSP. Generating and then interpreting the bytecode is a time consuming process; therefore, Facelets is preferred over JSP to create JSF View pages.

Let's learn about the characteristics of Facelets.

Features of Facelets

Facelets consists of various unique features that make it different from the existing presentation technology. Some of these features are as follows:

- **Support for XHTML**—Creates the Web pages using Extensible HTML (XHTML), which conforms to the XHTML Transitional Document Type Definitions (DTD).
- **Support for tag libraries**—Provides support for JSF tag libraries. To include JSF tag libraries, Facelets utilizes XML namespace declaration. Table E.1 describes the tag libraries supported by Facelets:

Table E.1: Describing the Tag Libraries Supported by Facelets

Tag Library	Uniform Resource Identifier (URI) Used	Prefix Used	Explanation
JSF Facelets tag library	http://java.sun.com/jsf/facelets	ui:	Supports tags for templating
JSF HTML tag library	http://java.sun.com/jsf/html	h:	Supports JSF component tags
JSF core tag library	http://java.sun.com/jsf/core	f:	Supports tags for JSF custom actions

Table E.1: Describing the Tag Libraries Supported by Facelets

Tag Library	Uniform Resource Identifier (URI) Used	Prefix Used	Explanation
JSP Standard Tag Library (JSTL) core tag library	http://java.sun.com/jsp/jstl/core	c:	Supports JSTL 1.1 core tags
JSTL functions tag library	http://java.sun.com/jsp/jstl/functions	fn:	Supports JSTL 1.1 function tags

In addition to the tag libraries mentioned in Table E.1, Facelets also supports composite component tags. In a Web page, these composite component tags are used frequently by a programmer; therefore, a custom prefix can be defined to access these tags. To learn more about composite components, refer to the *Exploring Composite Components in Facelets* heading.

- ❑ **Unified expression language support**—Utilizes the unified Expression Language (EL) expressions to access backing bean's methods and properties. EL expression binds the component objects with the backing bean's properties and methods.
- ❑ **Support for templating**—Provides support for templating. You can refer to templating under the *Understanding Templating in Facelets* heading.

Now, let's learn about the advantages of Facelets.

Advantages of Facelets

Facelets has proved to be a boon to the developers, as it allows code reuse and ease of development through the use of templates and composite components. Use of Facelets in Web applications decreases the time and effort spent in development and deployment of the applications. The main advantages of Facelets are as follows:

- ❑ Supports code reusability by using templating and composite components
- ❑ Provides faster compilation of the code provided in a View page
- ❑ Provides compile time EL validation
- ❑ Enhances the performance of the Web application
- ❑ Provides complete support for Expression Language (EL)
- ❑ Makes the use of XML configuration files optional
- ❑ Eradicates the use of the @taglib directive and f:view tag

Now, let's explore templating in Facelets.

Understanding Templating in Facelets

Repeating the same code in similar Web pages in an application is both frustrating and time consuming for the developer. In addition, repetition of same code may result in decline of the programmer's performance, which ultimately leads to hike in project costs as well. Templating is a useful technique, introduced in JSF 2.0, for creating reusable and extensible templates that can be reused in multiple Web pages of an application. Evidently, templating eradicates repetition of code and consequently, enhances the performance of the programmers.

Templating provides a similar look and feel to all the pages of an application and avoids re-creation of similar Web pages. Table E.2 explains the tags used for templating:

Table E.2: Describing the Tags Used for Templating

Tags Used	Explanation
ui:component	Denotes a component, which is created and appended to the component tree.
ui:composition	Denotes a page composition, which may utilize a template page. The Java interpreter does not read the content written outside the ui:composition component.

Table E.2: Describing the Tags Used for Templating

Tags Used	Explanation
ui:debug	Denotes a debug component, which is developed and appended to the component tree.
ui:define	Denotes content in a template page that is added to a Web page.
ui:decorate	Denotes a page composition, which may use a template page. Content written outside the ui:decorate component is not neglected.
ui:fragment	Denotes a component, which is developed and appended to the component tree. Content written outside of ui:fragment component is not neglected.
ui:include	Allows you to add composition in other Web pages.
ui:insert	Adds content in a template page. This tag also provides a structure to the template page.
ui:param	Passes parameters to an included file.
ui:repeat	Provides substitute for loop tags; for example, c:forEach or h:dataTable.
ui:remove	Eradicates the content from a Web page.

Let's learn about the composite components in Facelets.

Exploring Composite Components in Facelets

Composite component is a reusable and user-defined component used to perform a desired task. In other words, a component is a reusable software program that can perform a particular task; for example, the `inputText` component, which can receive user input. A composite component, similar to any JSF component, possesses validators, converters, and listeners attached to it, to carry out certain defined task. A composite component may possess group of markups and other components.

NOTE

The Web page using the composite component is known as a using page.

Table E.3 describes the composite component tags:

Table E.3: Describing the Composite Component Tags

Tag	Description
composite:interface	Defines the composite component's usage contract. The composite component may be utilized as a single component having features defined in the usage contract.
composite:implementation	Defines the composite component's implementation. The <code><composite:interface></code> tag is always used with a corresponding <code><composite:implementation></code> tag.
composite:attribute	Denotes an attribute that is provided to a composite component instance, having the <code><composite:attribute></code> declaration.
composite:insertChildren	Re-parents the child component or template text (that is used within the composite component tag) to the location at which the <code><composite:insertChildren></code> element is used. This re-assignment depends on the place where the <code><composite:insertChildren></code> element is placed in the <code><composite:implementation></code> tag in the composite component.
composite:valueHolder	Holds the value of the component that is declared in the <code><composite:implementation></code> tag. The <code><composite:valueHolder></code> element is placed within the opening and closing tags of the <code><composite:interface></code> element. The <code>valueHolder</code> implementation can be used as target of the attached objects in the page that utilizes the composite component or the using page that uses a composite component.

Table E.3: Describing the Composite Component Tags

Tag	Description
composite:editableValueHolder	Allows you to edit the value of the component declared in the <composite:implementation> tag. Similar to the <composite:valueHolder> element, the <composite:editableValueHolder> element is also placed within the opening and closing of tags of the <composite:interface> element. The editableValueHolder implementation is appropriate to be used as target of the attached objects in the page that utilizes the composite component or the using page.
composite:actionSource	Declares a component to be used as an action listener in a composite component. The <composite:actionSource> element is used within the opening and closing of tags of the <composite:interface> element. The ActionSource2 implementation is appropriate to be used as target of the attached objects in the page that utilizes the composite component or the using page.

The following code snippet shows the code for the composite component, which can accept a name as input:

```
<!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:composite="http://java.sun.com/jsf/composite"
xmlns:h="http://java.sun.com/jsf/html">
<h:head>
<title>This content does not appear to the user
</title>
</h:head>
<h:body>
<composite:interface>
<composite:attribute name="myvalue" required="false"/>
</composite:interface>
<composite:implementation>
    <h:outputLabel value=" Your Name : ">
        </h:outputLabel>
        <h:inputText value="#{cc.attrs.myvalue}">
            </h:inputText>
    </composite:implementation>
</h:body>
</html>
```

The code provided in the preceding code snippet is saved in a file named name.xhtml, in the namecomp folder, which is a subdirectory of the resources folder under the application Web root directory. The namecomp folder is treated as a library by the JSF, and a Ulcomponent is retrieved from this library. Notice the utilization of cc.attrs.myValue, when the value of the inputText component is defined. The word cc is used in JSF for composite components. The #{cc.attrs.ATTRIBUTE_NAME} expression is used to retrieve the composite component's attribute.

A composite component's reference is added in the using page with the help of an xml namespace declaration. Custom prefix is defined for the composite component, which in this case is nm. The following code snippet uses the namecomp library to provide value to its myValue attribute, which was created in the preceding code snippet:

```
<!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"
xmlns:nm="http://java.sun.com/jsf/composite/namecomp">
<h:head>
<title>Web Page Utilizing a composite component</title>
```

```
</h:head>
<body>
<h:form>
<nm:name myValue="Enter your Name " />
</h:form>
</body>
</html>
```

In the preceding code snippet, the xmlns:nm=http://java.sun.com/jsf/composite/namecomp/ declaration is used to declare the local composite component library. The name.xhtml component is retrieved with the help of the nm:name tag.

After exploring templating and composite component of Facelets, it's the time to create a Web application that uses Facelets as its presentation technology.

Implementing Facelets

In this section, we create an application called FaceletsApp in which Facelets is used as the presentation technology. In the FaceletsApp application, two pages are designed, namely the welcome.xhtml and the response.xhtml page. The welcome.xhtml page accepts the name of a user, and the response.xhtml page displays the message Welcome to Facelets and the name entered by the user. Perform the following broad-level steps to create a Web application that uses Facelets:

1. Create a managed bean
2. Develop the views
3. Configure the application
4. Explore the directory structure of the application
5. Compile the managed bean
6. Run the application

Now, let's perform each of these steps one by one.

Creating a Managed Bean

The first step while creating the FaceletsApp application is to create a backing bean, which is a type of managed bean. In the backing bean, methods and properties are defined and are related to the components in a JSF application. Managed bean, named UserBean.java, is created in the package com.kogent.facelets. Listing E.1 shows the code of the UserBean.java file (you can find this file on CD in the code\JavaEE\AppendixE\FaceletsApp\src\com\kogent\facelets folder):

Listing E.1: Showing the UserBean.java Class

```
package com.kogent.facelets;
import java.util.Random;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
@ManagedBean
@SessionScoped
public class UserBean
{
    String userName= null;
    String response = null;
    public UserBean()
    {
        System.out.println("Your name is " + userName);
    }
    public void setUserName(String user_name)
    {
        userName= user_name;
    }
    public String getUserName()
```

```

    {
        return userName;
    }
    public String getResponse()
    {
        if (userName != null && userName == " ")
        {
            return "Welcome to Facelets " + userName;
        }
        else
        {
            return "Please enter the name correctly";
        }
    }
}

```

Note the use of the @ManagedBean and @SessionScoped annotations in listing I.1. The @ManagedBean annotation registers the backing bean as resource with JSF implementation. The @SessionScoped annotation registers the backing bean in Session scope.

Creating the Views

After creating the backing bean, it's time to create view components of the application. Creating view component includes adding components to the Web page and connecting the components with backing bean values and properties. The welcome page of this application is welcome.xhtml, which is presented to the user when the application is started. Listing E.2 provides the code of the welcome.xhtml file (you can find this file on CD in the code\JavaEE\AppendixE\FaceletsApp folder):

Listing E.2: Showing the Code for the welcome.xhtml File

```

<!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"
      xmlns:f="http://java.sun.com/jsf/core">
<h:head>
    <title>Simple Facelets Application</title>
</h:head>
<h:body>
    <h:form>
        <h2>
            Hi,
            <p>Please enter your name</p>
            <h:inputText
                id="user"
                value="#{userBean.userName}">
            </h:inputText>
            <h:commandButton id="submit" value="Submit" action="response.xhtml"/>
        </h2>
        </h:form>
    </h:body>
</html>

```

Notice the binding of value attribute of the inputText component with the userName property of the UserBean.java class in Listing E.2.

When the user submits his/her name on the greeting.xhtml page, the value entered is supplied to the userName property of the UserBean.java class and the response.xhtml page is displayed to the user. Listing E.3 shows the code of the response.xhtml page (you can find this file on CD in the code\JavaEE\AppendixE\FaceletsApp folder):

Listing E.3: Showing the Code for the response.xhtml File

```
<!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>Simple Facelets Application</title>
</h:head>
<h:body>
  <h:form>
    <h2>#{userBean.response}</h2>
    <h:outputText id="result" value="#{userBean.response}" />
    </h2>
    <h:commandButton id="back" value="Back" action="welcome.xhtml"/>
  </h:form>
</h:body>
</html>
```

In the preceding Listing, the line `<h:outputText id="result" value="#{userBean.response}" />` invokes the `response()` method of the `UserBean.java` class and the result of invocation of the method is displayed to the user.

Configuring the Facelets Application

Let's now configure the application or create its Deployment Descriptor. Listing E.4 shows the code of the `web.xml` file of the `FaceletsApp` application (you can find this file on CD in the `code\JavaEE\AppendixE\FaceletsApp\WEB-INF`):

Listing E.4: Showing the Code for the `web.xml` File

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>faces/welcome.xhtml</welcome-file>
  </welcome-file-list>
</web-app>
```

In Listing E.4, FacesServlet servlet class is mapped with the /faces/* url pattern. The welcome.xhtml page is declared as the welcome page. The value of enum javax.faces.application.ProjectStage is configured with the help of context init-parameter in the web.xml file. Values of enum javax.faces.application.ProjectStage can be Development, Production, SystemTest, and UnitTest. The enum value can be retrieved by calling the Application.getProjectStage() method. In addition, the `toString()` method is invoked on the value returned by the Application.getProjectStage() method. The invocation of the `toString()` method returns the value configured in Deployment Descriptor.

Let's now explore the directory structure of the application.

Exploring the Directory structure of the Application

The directory structure of an application depicts the location at which the files required in the application are placed. Figure E.1 shows the directory structure of the FaceletApp application:

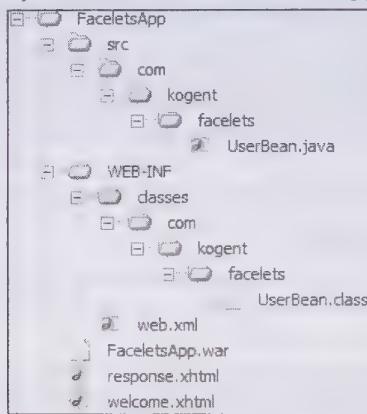


Figure E.1: Showing the Directory Structure of the FaceletApp Application

Let's now compile the UserBean managed bean class of the FaceletApp application.

Compiling the Managed Bean

Now, after creating the Deployment Descriptor of the application, let's compile the UserBean.java class provided in Listing E.1. Figure E.2 shows the compilation of the managed bean class (in our case UserBean.java file):

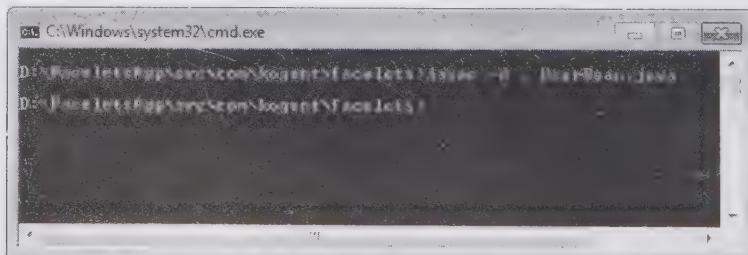


Figure E.2: Showing the Compilation of the UserBean.java Class

Running the Facelets Application

After compiling the backing bean of the FaceletsApp application, we need to run the application. To run the application, you need to start the application server, as shown in the following command:

```
C:\glassfishv3\glassfish\bin\asadmin start-domain domain1
```

Figure E.3 shows the commands to start the application server:

```
C:\Windows\system32\cmd.exe
C:\>glassfish3\glassfish\bin>asadmin start-domain domain1
Waiting for DBS to start
Started domain: domain1
Domain location: C:\glassfish3\glassfish\domains\domain1
Log file: C:\glassfish3\glassfish\domains\domain1\log\server.log
Admin port for the domain: 4849
Command start-domain executed successfully.

Configuring Glassfish server for domain1
```

Figure E.3: Showing the Command to Start the Application Server

Now, deploy the application using Glassfish v3 application server. The welcome.xhtml page is displayed when the application is launched. Figure E.4 shows a user entering his/her name in the text component of the welcome.xhtml page:

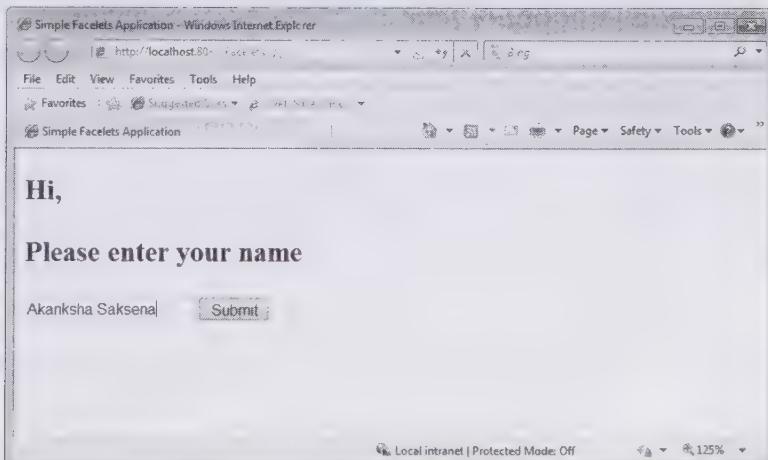


Figure E.4: Showing the welcome.xhtml Page

Figure E.5 shows the Web page, which appears when the user clicks the submit button in Figure E.4:

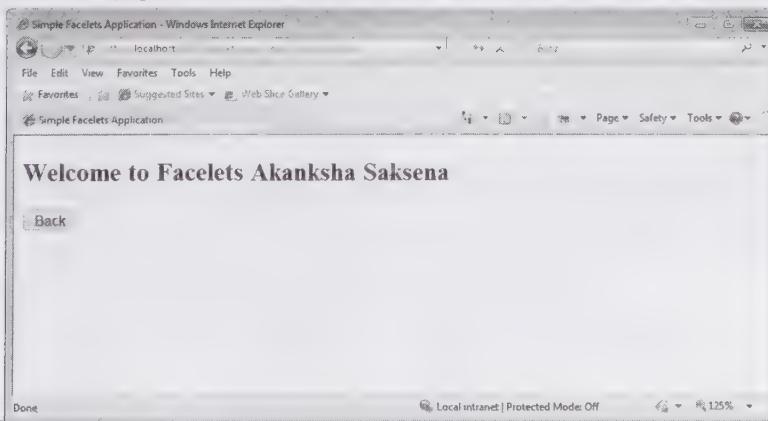


Figure E.5: Showing the Welcome Message Displayed to the User

Figure E.6 shows the Web page that appears when user does not provide the name in Figure E.4 and click the submit button:

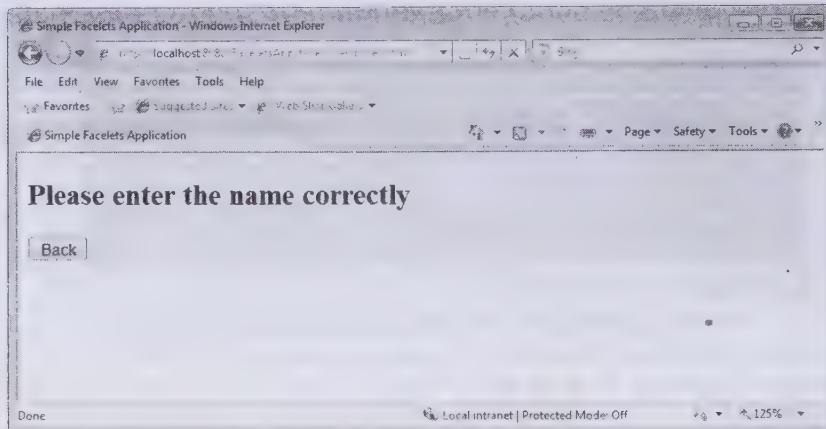


Figure E.6: Showing the Output When User does not Provide Name

Figure E.6 shows the Web page displaying the message Please enter the name correctly to the user. With this, we come to the end of this appendix.



Working with JMS

Java Message Service (JMS) Application Programming Interface (API) is provided by Sun Microsystems and its partner companies to allow Java programs to interact with other messaging implementations. Messaging implementations are applications used for sending and receiving messages over a network. JMS API allows you to establish reliable, asynchronous, and loosely coupled communication among the components of a distributed Java EE application. With the help of this communication, the components can create, read, send, and receive messages by using the JMS API. JMS API contains a set of interfaces, classes, and related semantics that enable components to interact with other messaging systems or implementations.

The javax.jms package provides the classes and interfaces of JMS API. These classes and interfaces are implemented by a JMS provider, which is used to manage the session beans and queues in the Java EE application. Some examples of JMS providers are JBoss Messaging, OpenJMS, and Apache ActiveMQ. It is easier for the programmer to create the JMS application because JMS API is easier to learn and provides sufficient features to communicate with other complicated messaging implementations.

In this appendix, you learn about the need for JMS API; features of JMS API that make it a popular choice to create messaging applications; communication types supported by JMS API; classes, interfaces, and exceptions in JMS API; and JMS API programming model.

Need for JMS API

In a messaging system, a messaging client can deliver and accept messages from any other client. Each client interacts with a messaging agent for creating, sending, receiving, and reading messages. Messaging API, such as JMS API, is needed in contrast to other messaging API, such as remote procedure call (RPC), in the following conditions:

- ❑ When a component provider does not need to rely on other component's interface information for the replacement of components of an application
- ❑ When application provider wants the application to be executed in all situations, irrespective of whether or not all components of the application are being executed simultaneously
- ❑ When the application business model permits a component to deliver the information to another component and continue execution of the messaging service without receiving an immediate response from the receiver

Features of JMS API

With the introduction of JMS API, various vendors have adopted and provided implementation to the JMS API, which can be integrated with the application server using the Java EE Connector architecture (JCA), and accessed using a resource adapter. As a result, JMS API can now offer a complete messaging service for an enterprise. The JMS API in the Java EE platform possesses the following features:

- ❑ Enables the application clients, the Enterprise JavaBeans (EJB) components, and the Web components to deliver or synchronously receive a JMS message. Application clients may receive the JMS messages asynchronously. It is not required by the applets to support the JMS API.
- ❑ Provides the asynchronous consumption of messages by the Message-Driven Bean (MDB), which is a type of enterprise Java Bean.
- ❑ Permits JMS operations and database access operations to occur within a single transaction, because sending and receiving operations can occur in distributed transactions.
- ❑ Supports distributed transactions and permitting the concurrent consumption of messages.

Communication Types Supported by JMS API

As already learned, JMS is a service used to send and receive message through a communication process. JMS API permits the following types of communications:

- ❑ **Asynchronous communication** – Allows JMS provider to send the message to a recipient, as soon as the message arrives. The recipient does not need to make specific requests to receive the messages.
- ❑ **Reliable communication** – Allows JMS API to guarantee that a message is sent exactly once. Lower reliability is meant for applications that can afford to miss messages or accept duplicate messages.
- ❑ **Loosely coupled communication** – Allows a message to be sent to a location by a component; the recipient can then receive the sent message from that location. It is not necessary that for a successful communication, the sender and receiver of the message should be available at the same time. In other words, the sender is not required to have knowledge about the receiver's interface, and vice versa; the only information that a sender or receiver should know is the message format and the location of the message.

Now, let's discuss about the JMS API in detail.

Exploring JMS API

JMS API defines the javax.jms package that contains the classes, interfaces, and exceptions required to create the JMS client application. These JMS client applications are responsible for creating and consuming messages. To understand how an application creates or consumes messages, you need to explore the JMS API.

Let's discuss classes, interfaces, and exceptions in JMS API in detail.

Classes, Interfaces, and Exception in JMS API

The javax.jms package provides a complete set of classes and interfaces, which are used to develop a JMS client application. Some of the important interfaces and classes in the javax.jms package are:

- ❑ The BytesMessage interface
- ❑ The Connection interface
- ❑ The ConnectionFactory interface
- ❑ The MessageConsumer interface
- ❑ The MessageProducer interface
- ❑ The Queue interface
- ❑ The QueueBrowser interface
- ❑ The QueueReceiver interface
- ❑ The QueueSender interface
- ❑ The TopicConnectionFactory interface
- ❑ The XASession interface
- ❑ The QueueRequestor class
- ❑ The TopicRequestor class

Let's discuss these interfaces and classes in detail next.

The BytesMessage Interface

The BytesMessage interface extends the Message interface and sends a message to its receiving point. The message contains a stream of uninterpreted bytes, which are interpreted by the message receiver. The JMS API allows you to use message properties with byte messages; however, the usage of these properties may affect the message format. Therefore, it is advisable not to use the message properties along with the byte message. The methods of the BytesMessage interface depend on the methods of the java.io.DataInputStream class and java.io.DataOutputStream. The important methods of the BytesMessage interface are described in Table F.1:

Table F.1: Explaining the Important Methods of the BytesMessage Interface

Methods	Explanation
getBodyLength()	Returns the number of bytes in a message body during its read only mode.
readBoolean()	Reads a boolean value from the message stream containing bytes.
readChar()	Reads a Unicode character value from a message.
readInt()	Reads a signed 32-bit integer from a message.
readLong()	Reads a signed 64-bit integer from a message.
readUTF()	Reads a modified UTF-8 format encoded string from a message.
readBytes(byte[] bytevalue, int bytelength)	Reads a part of a message. If the value of the bytevalue argument is less than the number of remaining bytes to be read, the byte array is filled with bytes from the message. Moreover, the next invocation of the method reads the next increment and the process continues till the specified number of bytes is read.
writeBoolean(boolean myvalue)	Writes a boolean to a message in the form of 1-byte value.
writeByte(byte myvalue)	Writes a byte to a message in the form of 1-byte value.
writeShort(short myvalue)	Writes a short to a message in the form of 1-byte value.
writeChar(char myvalue)	Writes a char to a message in the form of 2-byte value.
writeInt(int myvalue)	Writes an int to a message in the form of four bytes.
writeLong(long myvalue)	Allows writing a long to a message in the form of eight bytes.
writeFloat(float myvalue)	Changes the float argument to an int with the help of Float class floatToIntBits method and then writing the converted int value to a message as a 4-byte quantity.
writeUTF(String value)	Writes a string to a message using UTF-8 encoding.

The Connection Interface

The Connection interface represents a client's active connection with the JMS provider. Provider resources are allocated externally to the Java virtual machine (JVM) by the instance of the Connection interface. In the JMS application, the instance of the Connection interface performs the following tasks:

- ❑ Encapsulates an open client connection with a JMS provider. The instance of the Connection interface is usually an open TCP/IP socket from the client to the service provider.
- ❑ Defines a distinctive client identifier.
- ❑ Supplies the ConnectionMetaData object.
- ❑ Provides support for the ExceptionListener object.

A connection instance is a heavyweight instance because its creation involves establishing authentication and communication. Single connection is utilized by majority of clients for all their messaging tasks. The important methods of the Connection interface are described in Table F.2:

Table F.2: Describing the Important Methods of the Connection Interface

Methods	Explanation
close()	Closes the connection.
createSession(boolean transactedvalue, int acknowledgeModevalue)	Creates a Session object. The transactedvalue parameter depicts whether the session is transacted or not. The acknowledgeModevalue parameter signifies if the consumer or the client would acknowledge any message on reception of the message. The valid values of the acknowledgeModevalue parameter are: <ul style="list-style-type: none"> • Session.AUTO_ACKNOWLEDGE • Session.CLIENT_ACKNOWLEDGE • Session.DUPS_OK_ACKNOWLEDGE
getClientID()	Returns the value of client identifier for the connection.
setClientID(String uniqueclientID)	Establishes the value of client identifier for the connection.
getMetaData()	Returns the connection's metadata.
getExceptionListener()	Returns the exception listener object related with the connection. It is not necessary for each connection instance to possess an exception listener associated with it.
setExceptionListener(Exception Listener mylistener)	Establishes an exception listener for the connection. If the JMS provider identifies a problem with the connection, it notifies the registered exception listener of the connection regarding the problem.

The ConnectionFactory Interface

The ConnectionFactory interface encapsulates the set of connection configuration parameters, which are configured by an administrator. A client uses the instance of the ConnectionFactory interface to establish a connection with a JMS provider. The ConnectionFactory instance is an administered object that provides support for concurrent use. Administered objects help administer the JMS API in an enterprise. The JMS client accesses administered objects by looking up the administered objects in a JNDI namespace. Looking up of administered objects by the JMS clients in a JNDI namespace provides the following benefits:

- Conceals provider-specific information from JMS clients
- Abstracts administrative information into Java objects that can be organized and administered from a common management console in an efficient manner
- Enables JMS providers to provide one implementation of administered objects that would be executed for every client

The methods of the ConnectionFactory interface are described in Table F.3:

Table F.3: Describing the Methods of the ConnectionFactory Interface

Methods	Explanation
createConnection()	Establishes a connection with the default user identity in the stopped mode. You must explicitly call the start method of the connection object to initiate successful message delivery.
createConnection(String userNamevalue, String passwordvalue)	Establishes a connection with the user identity passed to the method, in the stopped mode.

The MessageConsumer Interface

The MessageConsumer interface allows a client to receive messages from a destination with the help of the message consumer. To create a message consumer, destination object is passed to the createConsumer() method

of the session object. All message consumers inherit the MessageConsumer interface. You can use the MessageConsumer interface to create a message consumer with a message selector. The message selector specifies the criteria based on which the JMS client sends messages to the message consumer. The JMS client can receive the messages asynchronously or synchronously from the message consumer. In case of synchronous receipt, a client may request the next message from a message consumer by invoking the receive method of the message consumer.

In case of an asynchronous delivery, a client registers a message listener object with a message consumer. When messages arrive at the message consumer, they are delivered by the message consumer, which calls the onMessage() method of the message listener.

The methods of the MessageConsumer interface are described in Table F.4:

Table F.4: Describing the Methods of the MessageConsumer Interface

Methods	Explanation
close()	Closes the message consumer
getMessageListener()	Gets the message listener related with the message consumer
setMessageListener(MessageListener listenerValue)	Establishes message listener for a message consumer
receive()	Receives the next message generated for the message consumer
receive(long timeoutvalue)	Receives the next message within the timeout interval limit, which is specified by the timeoutvalue parameter
receiveNoWait()	Receives the next message if the message is instantly available

The MessageProducer Interface

The MessageProducer interface allows the client to deliver messages to a destination. To create a message producer, a destination object is passed to the createProducer method of a session object. All message producers inherit the MessageProducer interface. You can use the MessageProducer interface to create a message producer without providing the destination information. In this case, each send() method operation is provided with the destination. You can also define a default delivery mode, priority, and time to live for messages delivered by a message producer. In addition, you can define the delivery mode, priority, and time to live in case of an individual message.

The methods of the MessageProducer interface are described in Table F.5:

Table F.5: Describing the Methods of the MessageProducer Interface

Methods	Explanation
setDisableMessageID(boolean booleanvalue)	Specifies whether message IDs are disabled or not
getDisableMessageID()	Specifies whether message IDs are disabled or not
setTimeToLive(long timeToLiveValue)	Establishes the default time, in milliseconds, for which a produced message is preserved by the message system
getTimeToLive()	Gets the default time, in milliseconds, that a produced message is preserved by the message system
close()	Closes the message producer
send(Message message, int deliveryModevalue, int priority value, long timeToLivevalue)	Delivers a message to the destination, while providing delivery mode, priority, and time to live for the message

The Queue Interface

A client uses the instance of the Queue interface to provide queue identity to the JMS API. A queue may be utilized to create a MessageConsumer instance or a MessageProducer instance by calling the createProducer() or

`createConsumer()` method of the session object and passing the queue instance as an argument. Queue name, which is provider-specific, is encapsulated by the instance of the Queue interface.

Note that JMS API does not specify the actual time for which messages are retained by a queue.

The methods of the Queue interface are described in Table F.6:

Table F.6: Describing the Methods of the Queue Interface

Methods	Explanation
<code>getQueueName()</code>	Gets the queue name
<code>toString()</code>	Gets the object's string representation

Understanding the QueueBrowser Interface

The QueueBrowser interface helps a client to access messages on a queue without removing the messages. The `getEnumeration` method of the QueueBrowser interface provides an enumeration, which is utilized to scan the queue's messages. The enumeration object returned by the queue browser may consist of the queue's entire content or only the messages matching a message selector. Note that when messages of a queue are scanned using enumeration, some of the queued messages may have already reached their destinations or may have expired. Queue browser can be created by utilizing a session object or a QueueSession object.

The methods of the QueueBrowser interface are described in Table F.7:

Table F.7: Describing the Methods of the QueueBrowser Interface

Methods	Explanation
<code>getQueue()</code>	Retrieves the queue associated with the queue browser
<code>getMessageSelector()</code>	Gets the message selector expression of the queue browser
<code>getEnumeration()</code>	Retrieves an enumeration, which is used for browsing the current queue messages in the order of reception of messages
<code>close()</code>	Closes the queue browser

The QueueReceiver Interface

The QueueReceiver interface receives the messages delivered to a queue. The JMS API does not specify the manner in which messages are distributed between the queue receivers when there are more than one queue receivers for a single queue. A message selector is denoted by a queue receiver and rejected messages by the message selector remain on the queue. Message selector permits a queue receiver to skip messages, which implies that when the skipped messages are eventually read, the order of the reads does not preserve the partial order provided by the message producer. A QueueReceiver object with no message selector would read messages in the same sequence in which the messages have been produced.

The QueueReceiver interface provides a `getQueue()` method that allows you to get the queue associated with the QueueReceiver. The following code snippet shows the syntax of the `getQueue()` method of the QueueReceiver interface:

```
Queue getQueue() throws JMSEException
```

The QueueSender Interface

The QueueSender interface is used by a client to deliver messages to a queue. Usually, a queue is provided when a QueueSender is created. When the `send()` method of the QueueSender instance is executed, the message cannot be modified by other threads within the client. In case a message is modified during the execution of the `send()` method of the QueueSender instance, the result of execution of the `send()` method is uncertain.

The methods of the QueueSender interface are described in Table F.9:

Table F.9: Describing the Methods of the QueueSender Interface

Methods	Explanation
getQueue()	Retrieves the queue related with the QueueSender.
send(Message message, int deliveryModeValue, int priorityValue, long timeToLiveValue)	Delivers a message to the queue. The message parameter signifies the message to send, the deliveryModeValue parameter signifies the delivery mode to be used, the priorityValue parameter signifies the priority for the message, and the timeToLiveValue parameter signifies the message lifetime (in milliseconds).
void send(Queue queue, Message message, int deliveryModeValue, int priorityValue, long timeToLiveValue)	Delivers a message to a queue for an unidentified message producer. The queue parameter signifies the queue object to deliver the message, The message parameter signifies the message to send, the deliveryModeValue parameter signifies the delivery mode to use, the priorityValue parameter specifies the priority for the message, and the timeToLive parameter signifies the message lifetime (in milliseconds).

The TopicConnectionFactory Interface

The TopicConnectionFactory interface is used by the JMS client to create a topic connection with the JMS provider that uses publish/subscribe messaging domain. The TopicConnectionFactory interface generates the topic connection object, using which specific topic-related objects can be created. Table F.10 explains the methods of the TopicConnectionFactory interface:

Table F.10: Describing the Methods of the TopicConnectionFactory Interface

Methods	Explanation
createTopicConnection()	Generates a topic connection in stopped mode, possessing the default user identity. It is necessary to explicitly invoke the start() method of the connection instance to initiate successful message delivery.
createTopicConnection(String userName, String password)	Generates a topic connection in stopped mode, with the user identity passed to the method. It is necessary to explicitly invoke the start() method of the connection object to initiate successful message delivery.

The XASession Interface

The XASession interface provides access to a JMS provider's support for the Java Transaction API (JTA), which is in the form of a javax.transaction.xa.XAResource object. The functionality of the XASession object resembles that of the standard X/Open XA Resource interface. An application server accesses the XA resource of an instance of the XASession interface to control the transactional assignment of a XASession. XAResource supplies some complicated facilities for interleaving multiple transactions, retrieving a list of transactions in progress, and many others. Table F.11 describes the methods of the XASession interface:

Table F.11: Describing the Methods of XASession Interface

Methods	Explanation
getSession()	Gets the session related with the XASession
getXAResource()	Retrieves an XA resource
getTransacted()	Signifies whether the session is in transacted mode or not

Understanding the QueueRequestor Class

The QueueRequestor class generates a TemporaryQueue object, which is a queue generated by the system during connection. The object of the QueueRequestor class provides a request() method that delivers the request message and waits for the reply. The QueueRequestor object is constructed by passing a non-transacted QueueSession object and a destination queue object as parameter to its constructor.

The following code snippet shows the constructor of the QueueRequestor class:

```
QueueRequestor(QueueSession sessionValue, Queue queueValue) throws JMSEException
```

The implementation of the constructor in the preceding code snippet presumes the sessionValue parameter to be non-transacted and the delivery mode of either AUTO_ACKNOWLEDGE or DUPS_OK_ACKNOWLEDGE. The queueValue parameter is the queue object.

Table F.12 describes the methods of the QueueRequester Class:

Table F.12: Explaining the Methods of the QueueRequester Class	
Methods	Explanation
request(Message messageValue)	Delivers a request. This method waits for the reply.
close()	Closes the QueueRequestor object and its session.

The TopicRequestor Class

The TopicRequestor helper class makes the work of service requests easier. The constructor of the TopicRequestor class is provided with a non-transacted TopicSession object and a topic. An object of the TopicRequestor class generates a TemporaryTopic object, which is a topic created by the system during connection. This object provides a request() method that delivers the request message and waits for the reply. The following code snippet shows the constructor of the TopicRequestor class:

```
TopicRequestor(TopicSession sessionValue, Topic topic) throws JMSEException
```

This implementation presumes the sessionValue parameter to be non-transacted and the delivery mode to be either AUTO_ACKNOWLEDGE or DUPS_OK_ACKNOWLEDGE. The sessionValue parameter passed to the constructor represents the TopicSession object, to which the topic belongs, and topic parameter represents the topic on which the request/reply calls are performed.

Table F.13 describes the methods of the TopicRequestor class:

Table F.13: Describing the Methods of the TopicRequestor Class	
Methods	Explanation
close()	Closes the TopicRequestor and its session. The close() method also closes the TopicSession object, which is passed as a parameter to the constructor of the TopicRequestor class.
request(Message messageValue)	Delivers a request and waits for the reply.

Let's learn about messaging domains in JMS.

Understanding Messaging Domains in JMS

A majority of messaging products support either the point-to-point or the publish/subscribe approach for messaging. JMS supports both types of messaging approach. The JMS specification ensures that a separate domain is provided for both the messaging domains, and defines compliance for both the domain. Implementation of either one or both messaging domains can be provided by a stand-alone JMS provider. A Java EE provider implements both messaging domains.

A majority of JMS API implementations support both types of messaging domains. Few JMS client applications utilize both type of messaging domains in a single application. Therefore, it can be interpreted that JMS API has improved the power and flexibility of messaging products.

Let's discuss both types of messaging domains.

The Point-to-Point Messaging Domain

In the point-to-point (PTP) messaging domain, a PTP application is based on the message queues, senders, and receivers. Every message is addressed to a particular queue, and JMS clients receive messages from the specific queue, which is set up to store their messages. All the messages delivered to queues are preserved in queues till the messages are consumed or expired. The main characteristics of PTP are:

- ❑ Provides only one consumer for every message.
- ❑ Enables the receiver of the message to receive the message, irrespective of whether the client application at the receiver is being executed or not. Message sender and message receiver, both do not possess timing dependencies.
- ❑ Enables the receiver to acknowledge successful message processing.

Figure F.2 shows the PTP messaging domain:

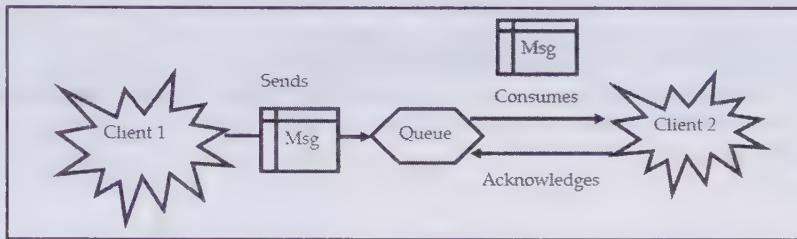


Figure F.2: Showing the Point-to-Point Messaging Domain.

In Figure F.2, Client 1 sends a message to the queue, which is consumed by Client 2.

The Publish/Subscribe Messaging Domain

Publish/subscribe (pub/sub) applications consist of a client that addresses the messages to a topic, which behaves similar to a bulletin board. Publishers and subscribers are usually anonymous and can publish or subscribe the hierarchical content dynamically. Distribution of the messages coming from a topic's multiple publishers to that topic's multiple subscribers is performed by JMS. A topic preserves the messages till the messages are distributed to the current subscribers.

Publish/subscribe messaging domain provides various consumers for every message. Publish/subscribe messaging domain maintains timing dependency between the publishers and the subscribers. A client, which has subscribed to a topic, can receive only the messages from a topic that are published after the client has generated a subscription. The application at a subscriber's end must be executed to consume messages.

JMS API provides solution to the problem of timing dependency in the publish/subscribe messaging domain by permitting subscribers to generate durable subscriptions. Subscribers generating durable subscription can receive the messages even when they are not active. Clients may send messages to multiple recipients using durable subscription. Figure F.3 shows the publish/subscribe messaging domain:

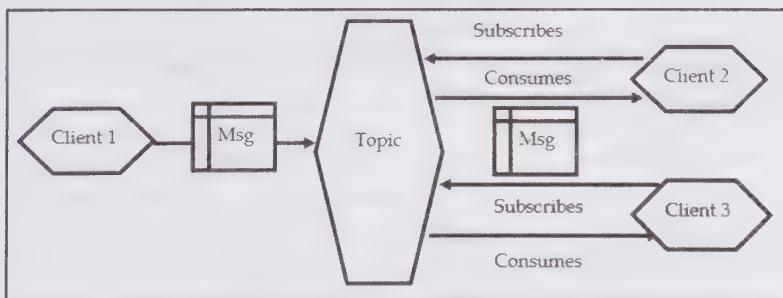


Figure F.3: Showing the Publish/Subscribe Messaging Domain

In Figure F.3, Client 1 publishes a message to the topic, which is consumed by Client 2 and Client 3.

Let's learn about message consumption in JMS.

Understanding Message Consumption in JMS

In general situations, no timing dependency exists between message production and message consumption. JMS specification supports message consumption in the following two ways:

- ❑ **Synchronous Consumption**—Enables explicit fetching of the message from the destination by the receiver or the subscriber by invoking the receive method.
- ❑ **Asynchronous Consumption**—Enables the JMS client to register a message listener (similar to event listener) with a consumer. When a message arrives at the destination, the JMS provider delivers the message by invoking the message listener's onMessage method. The onMessage method of the message listener processes the message content.

Now, let's explore the JMS API programming model in detail.

Understanding JMS API Programming Model

To understand the JMS API programming model, you must be acquainted with the main components of a JMS application. The main components of a JMS application are as follows:

- ❑ Administered objects
- ❑ Connections
- ❑ Sessions
- ❑ Message producers
- ❑ Message consumers
- ❑ Message listeners
- ❑ Message selectors
- ❑ Messages

Figure F.4 shows the main components of the JMS API programming model:

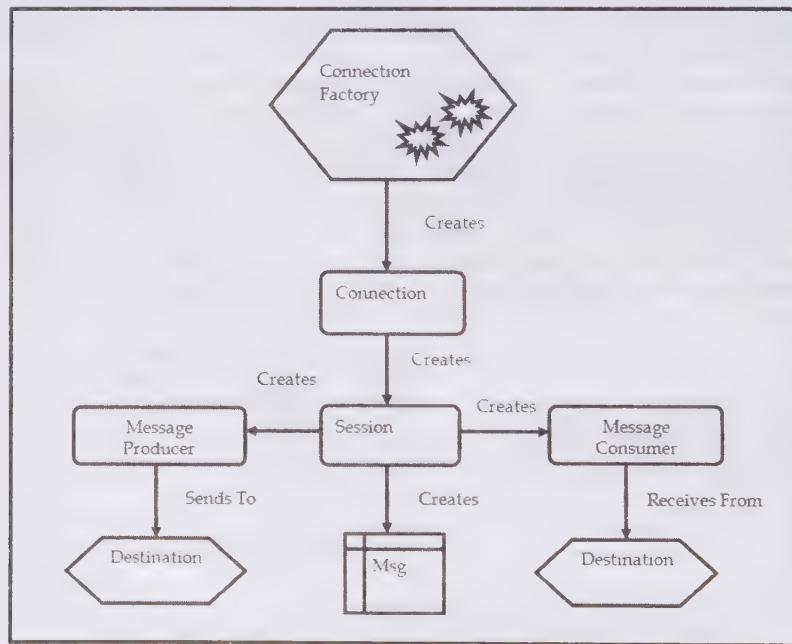


Figure F.4: Showing the JMS API Programming Model

In Figure F.4, a **ConnectionFactory** instance creates **connection**, which in turn creates a **session**. A **session** creates **message**, **message producer**, and **message consumer**. The **message producer** then sends the **message** to the **destination** using the **send()** method, and the **message consumer** receives the **message** from the **destination** using its **receive** method.

Let's discuss about the various components of the JMS application in detail.

Understanding Administered Objects

JMS Administered Objects are the configurable resources introduced by a JMS Provider that are to be consumed by a JMS client. An administered object enables JMS clients to be executed on multiple JMS API implementations. Administered objects are configured in a JNDI namespace by an administrator. A JMS client can access the administered objects using resource injection. There are two types of administered objects, which are as follows:

- ❑ **Connection factory**—Enables the JMS client to create a connection to a JMS provider. Connection configuration parameters, defined by an administrator, are encapsulated by a connection factory instance. Connection factory object is an instance of any of the ConnectionFactory, QueueConnectionFactory, or TopicConnectionFactory interface.
- ❑ **Destination**—Enables JMS client to define the targets of messages produced and the source of messages received. Destinations are known as queues in PTP messaging domain and topics in pub/sub messaging domain.

Understanding Connections

A connection object represents a virtual connection of a client with a JMS provider. Session objects can be created using the connection object. Connection object implements the Connection interface. The following code snippet shows the use of the connectionFactory object to create a connection object:

```
Connection con = connectionFactory.createConnection();
```

Any connection created in an application must be closed before the execution of the application is complete. In case you do not close a connection, resources may not be released by the JMS provider. When a connection is closed, all the associated sessions, message producers and message consumers are also closed. The following code snippet shows how to close a connection:

```
con.close();
```

Understanding Sessions

A session object, which is a single-threaded context, is used particularly for creating and consuming messages. Session objects are used for creating message producers, message consumers, messages, queue browsers, temporary queues, and topics.

A session object implements the Session interface and can be created with the help of connection object. The following code snippet shows the creation of the session object by using the createSession method of the connection object:

```
Session sess = connection.createSession(false,  
Session.AUTO_ACKNOWLEDGE);
```

In the preceding code snippet, the first argument passed to the createSession method is the Boolean value, false, which indicates that the session is not transacted. The second argument passed to the createSession method indicates that the session implicitly acknowledges messages after receiving them.

Understanding Message Producers

A message producer object implements the MessageProducer interface and is created using the session object. Message producers are used particularly for sending messages to a destination. The following code snippet demonstrates the creation of a message producer for a destination object, a queue object, or a topic object:

```
MessageProducer proddest = session.createProducer(dest);  
MessageProducer prodqueue = session.createProducer(queue);  
MessageProducer protopic = session.createProducer(topic);
```

In the preceding code snippet, the proddest message producer object is created by passing the destination object, dest, to the createProducer method of the session object. The prodqueue message producer object is created by passing the queue object, queue, to the createProducer method of the session object. The protopic message producer object is created by passing the topic object, topic, to the createProducer method of the session object.

Understanding Message Consumers

A message consumer object implements the MessageConsumer interface and is created using a session object. Message consumers are utilized for receiving messages, which are delivered to a destination. To register a JMS client's interest in a destination with a JMS provider, a message consumer is used. Message sending from a destination to the registered destination's consumer is managed by JMS provider. The following code snippet shows the creation of a message consumer using a destination object, a queue object, or a topic object:

```
MessageConsumer contest = session.createConsumer(dest);
MessageConsumer conqueue = session.createConsumer(queue);
MessageConsumer contopic = session.createConsumer(topic);
```

In the preceding code snippet, the contest message consumer object is created by passing the destination object, dest, to the createConsumer method of the session object. The conqueue message consumer object is created by passing the queue object, queue, to the createConsumer method of the session object. The contopic message consumer object is created by passing the topic object, topic, to createConsumer method of session object.

A message consumer object becomes active on creation and can be used to receive messages. The close method of a message consumer object makes the message consumer inactive. To start the message delivery, the connection object is started by invoking its start method.

Understanding Message Listeners

A message listener object is an asynchronous event handler for messages. The message listener object implements the MessageListener interface and possesses one method, onMessage, which defines the actions to be taken on the arrival of a message.

The setMessageListener method registers a message listener with a particular message consumer. The following code snippet demonstrates the registration of a message listener:

```
Listener myListener = new ListenerObject();
consumer.setMessageListener(myListener);
```

In the preceding code snippet, the myListener object of a class that implements the MessageListener interface is created and registered with the consumer consumer object by using the setMessageListener method. After a message listener is registered, message delivery can be initiated by calling the start method of the connection object.

Understanding Message Selectors

JMS API message selector is used to filter the messages that a messaging application receives. Message selector permits a message consumer to define the messages of its interest. Filtering of messages is assigned to the JMS provider by a message selector. It consists of a string that expresses an expression. The syntax of the string expression, defined by a message selector, depends on subset of the SQL92 conditional expression syntax.

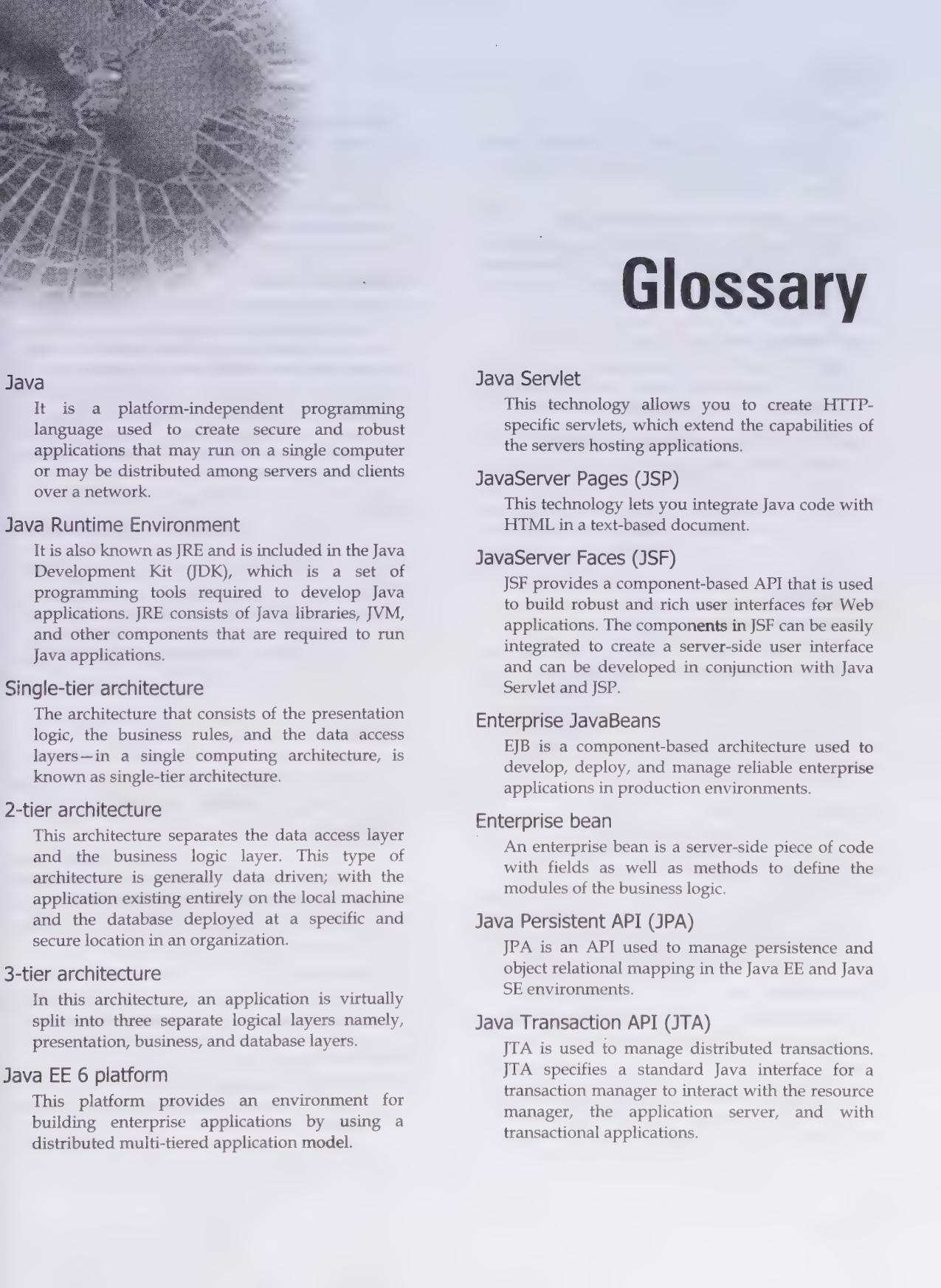
Understanding Messages

JMS message objects possess a simple, basic, and highly flexible format. Highly flexible format of JMS messages permits the messages to match formats of non-JMS applications on heterogeneous platforms. The three parts of a JMS message are as follows:

- ❑ **Message Header**—Consists of predefined fields utilized by both the client and provider to recognize and route messages.
- ❑ **Message Properties**—Enables compliance of JMS messaging system with other messaging systems. Message selectors can be created using the message properties.
- ❑ **Message Bodies**—Represents the message body. There are five message body formats, known as message types, defined by JMS API, which are used to send and receive data in multiple forms.

For creating a JMS client application, you can refer the MDB application provided in *Chapter 13, Working with EJB 3.1*.

With this, we come to the end of the appendix.



Glossary

Java

It is a platform-independent programming language used to create secure and robust applications that may run on a single computer or may be distributed among servers and clients over a network.

Java Runtime Environment

It is also known as JRE and is included in the Java Development Kit (JDK), which is a set of programming tools required to develop Java applications. JRE consists of Java libraries, JVM, and other components that are required to run Java applications.

Single-tier architecture

The architecture that consists of the presentation logic, the business rules, and the data access layers—in a single computing architecture, is known as single-tier architecture.

2-tier architecture

This architecture separates the data access layer and the business logic layer. This type of architecture is generally data driven; with the application existing entirely on the local machine and the database deployed at a specific and secure location in an organization.

3-tier architecture

In this architecture, an application is virtually split into three separate logical layers namely, presentation, business, and database layers.

Java EE 6 platform

This platform provides an environment for building enterprise applications by using a distributed multi-tiered application model.

Java Servlet

This technology allows you to create HTTP-specific servlets, which extend the capabilities of the servers hosting applications.

JavaServer Pages (JSP)

This technology lets you integrate Java code with HTML in a text-based document.

JavaServer Faces (JSF)

JSF provides a component-based API that is used to build robust and rich user interfaces for Web applications. The components in JSF can be easily integrated to create a server-side user interface and can be developed in conjunction with Java Servlet and JSP.

Enterprise JavaBeans

EJB is a component-based architecture used to develop, deploy, and manage reliable enterprise applications in production environments.

Enterprise bean

An enterprise bean is a server-side piece of code with fields as well as methods to define the modules of the business logic.

Java Persistent API (JPA)

JPA is an API used to manage persistence and object relational mapping in the Java EE and Java SE environments.

Java Transaction API (JTA)

JTA is used to manage distributed transactions. JTA specifies a standard Java interface for a transaction manager to interact with the resource manager, the application server, and with transactional applications.

Glossary

JavaMail

The JavaMail API provides a JavaMail service provider, which allows application components to send e-mail messages.

Java EE Connector Architecture (JCA)

JCA is used to create resource adapters that help Java EE application components to interact with the resource manager of Enterprise Information Systems (EIS).

Java Naming Directory Interface (JNDI)

JNDI provides the functionality naming and directory services for various objects or resources available in the Java EE container.

JavaBeans Activation Framework (JAF)

JAF is a standard extension to the Java platform providing standard services to identify the type of any arbitrary information, encapsulate access to the information, identify the operations available on it, and instantiate appropriate JavaBean components to perform these operations.

Component contract

Component contracts are rules, stored in the form of APIs, which application components must extend or implement. A Java EE container acts as a runtime environment to manage application components. At runtime, the instances of these application components are invoked within the JVM of the container.

Declarative services

The Java EE architecture allows you to use Deployment Descriptors to declare application components. These Deployment Descriptors provide services to application components and simplify application programming by allowing components and applications to be customized at packaging and deployment time.

EJB module

An EJB module contains one or more EJBs, and other helper classes and resources.

Client module

A client module is a group of Java client classes that can be directly accessed by a client. These classes are packaged into JAR files.

Web server

A Web server is a computer or virtual machine used to run Web applications. The main purpose of the Web server is to provide Web pages to clients. In the case of Web servers, a client uses a Web browser to make an HTTP request for a specific resource.

Application server

An application server is a server program that provides the business logic for an application. In other words, the application server provides a GUI to run three-tier applications. The application server acts similar to an extended virtual machine used to run applications and handle database transactions. In Java Platform, the term application server sometimes refers to the Java EE platform.

Seam

Seam is a framework that integrates the JSF and EJB technologies. JSF helps to create UI components of an enterprise application. However, creating UI components by using JSF requires large volumes of coding. Due to this, programmers have to focus more on the presentation layer of an enterprise application, when they should be focusing on the business logic.

Hibernate

Hibernate is an open source framework that enables a developer to work easily with relational databases in an enterprise application.

HTTP protocol

HTTP is a stateless, application-level communication protocol used to transfer information on the Internet. The main aim of HTTP is to send and receive user information over a network.

GET method

The GET method contains a Request-URI, which is used to retrieve information from the request.

HEAD method

The HEAD request method can be used to retrieve the meta information of the entity requested by the user.

POST method

The POST method is generally used in cases where a large amount of information needs to be sent to a Web server.

PUT method

The PUT method allows you to store an entity in the specified Request-URI.

View

A View provides the Graphical User Interface (GUI) for Model. A user interacts with the application through View, which displays the information based on Model and allows the user to alter data.

Controller

The Controller component controls all the Views associated with a Model. When a user interacts with the View component and tries to update the Model, the Controller component invokes various methods to update the Model.

Model

The Model component displays the data on which an application is based. In a Web application, JavaBeans hold the data needed by a Web application to process user queries.

Type-1 driver

The Type-1 driver acts as a bridge between JDBC and other database connectivity mechanisms, such as ODBC.

Type-2 driver

The JDBC call can be converted into the database vendor specific native call with the help of the Type-2 driver. In other words, this type of driver makes Java Native Interface (JNI) calls on database specific native client API. These database specific native client APIs are usually written in C and C++.

Type-3 driver

The Type-3 driver translates the JDBC calls into a database server independent and middleware server-specific calls. With the help of the middleware server, the translated JDBC calls are further translated into database server specific calls.

Type-4 driver

The Type-4 driver is a pure Java driver, which implements the database protocol to interact

directly with a database. This type of driver does not require any native database library to retrieve the records from the database. In addition, the Type-4 driver translates JDBC calls into database specific network calls.

java.sql package

The java.sql package is also known as the JDBC core API. This package includes the interfaces and methods to perform JDBC core operations, such as creating and executing SQL queries. The java.sql package consists of the interfaces and classes that need to be implemented in an application to access a database. The developer uses these operations to access the database in an application.

javax.sql package

The javax.sql package is also called as the JDBC extension API, and provides classes and interfaces to access server-side data sources and process Java programs.

PreparedStatement

The PreparedStatement interface, is subclass of the Statement interface, can be used to represent a precompiled query, which can be executed multiple times.

CallableStatement

In Java, the CallableStatement interface is used to call the stored procedures and functions. Therefore, the stored procedure can be called by using an object of the CallableStatement interface.

ResultSet

A ResultSet is an interface provided in the java.sql package, and is used to represent data retrieved from a database in a tabular format. It implies that a ResultSet object is a table of data returned by executing a SQL query. A ResultSet object encapsulates the resultant tabular data obtained when a query is executed. A ResultSet object holds zero or more objects, where each of the objects represents one row that may span over one or more table columns.

Batch update

The batch update option allows you to submit multiple DDL/DML operations to a data source to process data simultaneously. Submitting multiple DDL/DML queries together, rather than

submitting them individually, improves the performance of the query execution time.

Transactions

Transactions are used to ensure data integrity when multiple users access and modify data in a DBMS. A database transaction includes the interaction between the databases and users.

ServletContext

ServletContext objects help to provide context information in a Servlet container and communicate with the Servlet container.

ServletConfig

ServletConfig is a servlet configuration object passed to the servlet by the container when the servlet is initialized. A ServletConfig object contains a ServletContext object, which specifies the parameters for a particular servlet while the ServletContext object specifies the parameters for an entire Web application.

HttpServletRequest

An HttpServletRequest object always represents a client's HTTP request. HttpServletRequest is an interface and a subtype of the ServletRequest interface.

HttpServletResponse

The HttpServletResponse object helps to set an HTTP response header, set the content type of the response, or redirect an HTTP request to another URL. Let's discuss the implementation of the HttpServletResponse interface.

Request delegation

Request delegation refers to the request of a single client passing through many servlets or other resources in a Web application. The entire process is performed entirely on the server-side, unlike response redirection. Request delegation does not require any action from a client or extra information sent between the client and the server.

Session

A session can be defined as a collection of HTTP requests shared between a client and Web server over a period of time. While creating a session, you require setting its lifetime, which is set to thirty minutes by default. After the set lifetime expires, the session is destroyed and all its resources are returned back to the servlet engine.

Session tracking

Session tracking is a process of gathering the user information from Web pages, which can be used in an application.

Hidden form fields

The hidden form fields are the fields in a HTML or JSP form that are not shown to the user and used to store information about a session.

URL Rewriting

The URL rewriting mechanism uses the encodeURL() method of the response object to encode the session ID into the URL path of a request.

Secure socket layer (SSL)

SSL is used to protect the data during transmission that covers all network services. This layer uses TCP/IP to support typical application tasks that require communication between clients and servers.

Event

An event refers to a set of actions that may occur while an application is running. It could also be defined as a set of actions, such as clicking a button or pressing a key, performed by a user.

Event handlers

In Java, events are generated by objects. When an event is fired, a specific method of an object (to be notified) is called. This specific method is then notified about the event and the event object is passed as a parameter to that method. These methods are known as event handlers.

Wrappers

Wrappers are sandwiched between servlet classes and servlet containers and can easily handle the servlet environment by wrapping the servlet's APIs.

Scripting tags

JSP scripting tags, also called JSP scripting elements, allow you to add Java coding statements into a JSP page. The Java code incorporated by using scripting elements is translated and generated by the JSP translator while translating the page into a servlet. In most cases, Java is the scripting language used to build a JSP page; however, other supported languages can also be used, depending on the language supported by a Web container.

Implicit objects

JSP implicit objects are used in a JSP page to make the page dynamic. Java objects within scripting elements can be used to create and access the dynamic content. JSP implicit objects are predefined objects that are accessible to all JSP pages.

Action tags

Action tags were first introduced in JSP 1.1, and additional tags were added in the JSP 1.2 and 2.0 specifications. Action tags allow Java programmers to include some basic actions, such as inserting the resources of other pages, forwarding a request to another page, creating or locating JavaBean instances, and setting and retrieving JavaBean properties, in JSP pages.

Expression language (EL)

EL is a language that allows JSP programmers to fetch application data stored in JavaBeans components.

Tag Library Descriptor (TLD) file

The TLD file is an Extensible Markup Language (XML) document describing a tag library, which serves as a container of custom tags. These custom tags are used to encapsulate the functionalities to be provided within a JSP page. TLD is used by a JSP container to interpret the pages that include the taglib directives.

taglib directive

The taglib directive is used in a JSP page to implement the functionality of a custom tag. To set a custom tag in a JSP page, you need to provide the Universal Resource Identifier (URI) of the tag library and a prefix for the tag library.

Tag handler

A tag handler is a Java class where a custom tag is defined. The JSP container invokes the tag handler object to evaluate a custom tag during the execution of a JSP page.

Tag extension API

The tag extension API provides the `javax.servlet.jsp.tagext` package containing various classes and interfaces used to handle custom tags in a JSP page.

Classic tag handler

A classic tag handler is a Java class that implements the Tag, IterationTag, or BodyTag

interface. The implementation class instantiates a tag handler object or reuses an existing tag handler object for each action provided in the JSP page.

Simple tag handler

A simple tag handler is a Java class that implements the SimpleTag interface and has a no-argument constructor. This class is mainly used by authors to make the use of tags flexible in tag handlers. The `javax.servlet.jsp.tagext.SimpleTagSupport` class provides a default implementation of all the methods in simple tags.

JSP fragments

JSP fragments are portions of the JSP code in the `JspFragment` object, which can be invoked multiple times in an application. These fragments are configured either by specifying the `<jsp:attribute>` standard action in a tag file as a fragment type or as a body content of a simple tag.

Filter

A filter is a Java class that is called for responding to the requests for resources, such as Java Servlet and JavaServer Pages (JSP). Filters dynamically change the behavior of a resource when a client requests the resource. In other words, the filter mapped to a resource, such as a Uniform Resource Locator (URL) or a servlet, is invoked when the resource is accessed.

UI components

UI components are the basic reusable components for developing UIs using the JSF framework. The developer can define UI components as stateful objects maintained on server side. The server communicates with the client through these UI components.

Backing Beans

In a JSF application, there are some JavaBean objects, which handle or store data between the business model and UI component at intermediate stage. These JavaBean objects are known as Backing Beans.

Data model events

The data model events are associated with data-aware UI components. A data-aware component is a component that gives a list of items to be selected.

E-mail protocols

E-mail protocols can be defined as a set of some rules, formats, and functions used to exchange messages between the components of a messaging system.

Annotations

The annotation feature is introduced in the Java Platform, Standard Edition 5 (Java SE 5). These annotations are inspected at compile time or runtime by different tools to generate additional constructs, such as Deployment Descriptors, and to customize the component's runtime behavior. You can annotate class fields, methods, and classes.

Message-Driven Bean (MDB)

The term MDB itself suggests that it is associated with some sort of messaging. To communicate among software components or applications, messaging is used. Messaging is a facility by using which a client can send/receive messages to/from other clients. In case of MDBs, each client connects to a message agent that facilitates creating, sending, receiving, and reading messages.

Java Messaging Service (JMS)

JMS is a core service provided by Java EE application servers. JMS allows asynchronous invocation of different services via messages. JMS clients send messages to the server maintained message queues.

Container-managed transactions (CMT)

In case of CMT, the EJB container is responsible for managing the transaction demarcation for every method of the bean.

Bean-managed transactions (BMT)

BMT is implemented in those enterprise beans that manage their own transactions. BMT can be used only for a session bean and not for an entity bean.

Interceptors

Interceptors are the methods that are used to implement business logic in a business method or in a life cycle callback method of an enterprise bean.

Entities

Entities are POJOs that are used to store data in a database. Entities hold the data of an application

in the form of fields and the methods associated with the details of an entity. All the information regarding an entity is always available in ORM. Entities support relational and object-oriented capabilities, such as entities, inheritance, and polymorphism.

EntityListener

The EntityListener class is a stateless bean class containing a non-argument constructor. A class can be made as the EntityListener class by using the @EntityListener annotation. The entity listeners are applied to entity classes.

Callbacks

Callbacks are the methods of entities that are used to receive notifications about a specific entity. These methods are represented by the callback annotations. Life cycle callbacks are used for receiving callbacks, validating data, auditing, sending notifications regarding the changes made in a database, and generating data after an entity is loaded.

JPQL

JPQL is the extended version of the EJB Query Language (EJB-QL). Although referred as a query language, JPQL is different from SQL. JPQL operates on classes and objects (entities) available in the Java workspace, while SQL operates on table properties, such as rows and columns in the database space.

Hibernate

The Hibernate framework is a lightweight ORM, which is a technique for mapping an object model to a relational model. This framework handles mapping from Java classes to database tables and provides Application Programming Interface (API) for querying and retrieving data from a database.

HQL

HQL is an easy-to-learn and powerful query language designed as an object-oriented extension to SQL that bridges the gap between the object-oriented systems and relational databases.

ORM

ORM is a technique to map object-oriented data with the relational data. In other words, it is used to convert the datatype supported in object-

oriented programming language to a datatype supported by a database.

Contexts

A context is a set of namespaces and data items that are associated with the Seam components. Seam contexts are used to maintain the state of sessions in a Web application. These contexts are generated and destroyed by the Seam framework.

Seam components

The Seam components are similar to Plain Old Java Object (POJO) components that integrate JavaBeans or EJBs and JSF to implement the business and presentation logic.

Dependency Injection (DI)

DI allows you to separate the construction and implementation of an object. It allows a component to obtain a reference of another component by injecting a setter method or an instance variable in a class. The injection mechanism occurs only when the component is created and the reference of the component is changed during the lifetime of the component instance.

Stateless Navigation Model

The stateless navigation model defines a mapping of a set of logical outcome of events with the resources of a Seam application.

Stateful navigation model

The stateful navigation model works on the principle of jPDL. This model defines the flow of the pages of a Seam application by using a set of defined elements, such as <start-page> and <transition>.

Managed environment

In a managed environment, a resource adapter is deployed inside an application server. The methods of the resource adapter are called from inside an EJB container.

Non-managed environment

In a non-managed environment, a resource adapter is separated from an application server and is used outside the application server as a library. In a non-managed environment, with the help of JCA, clients such as applets or Java client applications can access an EIS system.

System contracts

System contracts enable a resource adapter to link with the services of an application server to manage connections, transactions, and security.

Common client interface

JCA provides a common interface in the form of CCI for clients to access EIS. CCI provides standard client APIs to solve the problems faced while integrating an application server with heterogeneous EIS systems.

Lifecycle management contract

The Lifecycle Management contract plays an important role in managing the life cycle of a resource adapter. It provides an environment to connect and integrate an application server with EIS and vice versa.

Workflow management

Workflow management refers to the process of monitoring the Work instances submitted by a resource adapter to an application server for their execution.

ExecutionContext

The ExecutionContext class allows a resource adapter to specify an execution context, such as a transaction context in which the Work instance must be executed.

Synchronous communication

The synchronous communication refers to a direct communication process in which all parties engaged in the process are available at a single point of time.

Asynchronous communication

The asynchronous communication refers to a process in which all parties may not be available at a single point of time.

Design pattern

Design patterns refer to language-independent solutions provided for solving commonly recurring design problems. A design pattern first describes the problem and then the solution that can be applied to the problem.

Front Controller pattern

The Front Controller pattern provides a centralized control to the application request processing. It introduces a central controller

component that acts as a common entry point for all application requests.

Composite View pattern

The Composite View pattern allows the management of layout and presentation of the data on a Web page in a more effective manner by providing a composite view which consists of multiple sub-views.

Composite Entity pattern

The Composite Entity pattern helps in effectively modeling a set of dependent interrelated objects when they are mapped to an Enterprise JavaBean object model.

Intercepting Filter pattern

The Intercepting Filter pattern introduces a filter that intercepts and intermediates the request to be received and the responses to be transmitted. The filter allows pre-processing and post-processing of the request and response, respectively.

Transfer Object pattern

The Transfer Object pattern, also termed as Value Object pattern, implements a Transfer Object component, which is a serializable class that aggregates related attributes together to form a composite value that can be returned as a return type by the methods.

Session Façade pattern

The Session Facade pattern uses a central high level component, which is implemented as a session bean and contains the functionality of the complex interactions that occur between various lower-level business components.

Service Locator pattern

The Service Locator pattern uses a service locator object to centralize the look up process for distributed service components.

Data Access pattern

The Data Access Pattern separates the data processing logic from the data access logic. It abstracts the code for data retrieval and encapsulates it separately from the rest of the data manipulation code using a Data Access Object (DAO).

View Helper pattern

The View Helper pattern enforces separation of presentation and business logic. In this pattern, the presentation and formatting logic is handled by the view and the processing and data retrieval logic is handled by the View Helper class.

Dispatcher View pattern

The Dispatcher View pattern considers the problems solved by the Front Controller and View Helper patterns, and combines the controller and the dispatcher components with the views and helpers for handling client requests and providing a dynamic response.

Service to Worked pattern

The Service to Worker pattern shows various similarities with the Dispatcher View pattern. This pattern also deals with the problems considered in the Dispatcher View pattern; however, the solution is obtained and implemented slightly differently.

Service Oriented Architecture (SOA)

SOA defines how different components of a software application interact to execute the business logic of an enterprise application.

JAXB portability

It simply means that a runtime marshaller of any JAXB implementation can serialize a JAXB annotated class to an instance of the XML schema defining a Web service, or deserialize a schema instance to the JAXB annotated class's instance.

WebParam annotation

It customizes the mapping for an individual parameter of an SEI method to a single WSDL message part or element.

WebResult annotation

It is used to customize the mapping of the return value of an SEI method to a WSDL part or XML element.

RequestWrapper annotation

It is used to annotate methods in an SEI with the request wrapper bean used at runtime.

ResponseWrapper annotation

It is used to annotate methods in an SEI with the response wrapper bean used at runtime.

ActionContext

The ActionContext class provides objects required to execute an action class. The ActionContext class provides objects such as request, response, session, parameters locale, and so on.

Bundled interceptors

Bundled interceptors are a set of pre-defined interceptors, which are used in a Web application to provide required processing before and after an action class is executed.

OGNL

OGNL is an expression language used to manipulate and retrieve different properties of Java objects. OGNL has its own syntax, which is very simple in structure; therefore, it is easy to learn and use and also makes the code more readable. OGNL acts as an expression language for the GUI elements to model objects.

Localization

It is a process of adding locale-specific components, such as property files, in Web applications to customize them according to specific languages.

Resource bundle

A resource bundle is a file that contains the key-value pairs for a particular language. Different resource bundles are required for different languages.

Plugin

A plugin is a computer program that communicates with a main or host application.

Inversion of Control (IoC)

IoC is a principle of software construction, where the developers no longer need to create objects from classes.

Aspect Oriented Programming (AOP)

AOP is a complement of OOP, which is defined as a programming technique.

Aspects

Aspects enable the modularization of concerns, such as transaction management, which crosscut multiple types and objects.

Advice

It defines both what and when of an aspect.

Joinpoint

It refers to a point where an aspect can be plugged in.

Pointcut

It refers to many joinpoints that are grouped together to perform an advice, which makes a pointcut.

Around advice

It specifies whether to proceed to the joinpoint or to make a cross-cutting concern by returning its own return value or throwing an exception.

Before advice

It fires an advice before the execution of a joinpoint.

After throws advice

It fires an advice when a method throws an exception.

After returning advice

It fires an advice when a method invocation or joinpoint is completed.

After finally advice

It fires an advice regardless of whether a joinpoint exits with a normal or exceptional return.

Declarative security

It refers to the type of security provided to the Web components by using the Deployment Descriptor, which is an Extensible Markup Language (XML) file that explains the deployment of Web and enterprise applications.

Programmatic security

It refers to the type of security that is embedded in an application and is used to make security decisions, such as determining whether or not a user is authorized to access a resource. Programmatic security defines the security model of the application.

Authentication

It is a security mechanism in which callers and service providers validate each other on behalf of specific users or systems in a distributed computing environment.

Protection domain

A collection of entities that are expected or known to trust each other is called a protection domain.

Realm

In terms of Java EE 6 application security, a realm is a database of users and groups that is used to identify valid users of a Web application.

User

A user is an individual identity defined by the application server. It can have multiple roles corresponding with that identity.

Group

A group is a collection of authenticated users, which have common traits specified in an application server.

Role

To access a specific collection of resources in an application, an abstract name called role is used.

HTTP basic authentication

It refers to the authentication mechanism used to protect Web resources through specified username and password. This is the simplest authentication method supported by a Web application.

HTTP digest authentication

It refers to the authentication mechanism in which the information is transmitted from the Web browser to the server in HTTP basic authentication containing a password. However, HTTP digest authentication is rarely used because only few Web browsers support this type of authentication.

Form-based authentication

It provides a visual effect by using HTML in a Web application. It is implemented by using server-side session tracking, so the session can be invalidated when the user logs out.

Declarative authorization

It allows you to implement access control rules enforced by a container in a Java EE application.

Programmatic authorization

It requires the implementation of authorization rules within the Java code of a Web application, but it uses the servlet security framework to authenticate the users.

Client-certificate authentication

It is the advanced authentication method which is more secure than the basic and form-based authentication mechanisms.

Index

@

@ActivationConfigProperty, 576
@Columns, 619
@ConversionErrorFieldValidator, 926, 972
@CustomValidator, 979
@DateRangeFieldValidator, 926, 972
@DoubleRangeFieldValidator, 926, 973
@EJB, 554, 670
@EmailValidator, 926, 973, 978
@Entity, 616, 617, 619, 623, 630, 631, 635, 639, 640, 643, 645, 647, 651, 652, 653, 664, 680, 740
@EntityListeners, 623, 680
@Enumerated, 612, 620
@ExcludeDefaultListeners, 623
@ExcludeSuperclassListeners, 623
@ExpressionValidator, 973
@FieldExpressionValidator, 974
@IntRangeFieldValidator, 926, 974, 977
@Lob, 612, 621
@ManyToMany, 628, 643, 644, 647
@ManyToOne, 628, 639, 640
@MessageDriven, 573, 574, 576, 577, 607
@OneToMany, 553, 634, 635, 644
@OneToOne, 553, 628, 630, 631, 635
@Out, 723, 724, 731, 732, 742, 760
@PersistenceContext, 613, 626, 627, 632, 633, 637, 641, 646, 653, 663, 1049
@PostActivate, 553, 604, 605, 606, 609
@PostConstruct, 553, 568, 572, 573, 605, 606, 607, 608, 609, 732

@PreDestroy, 553, 572, 574, 605, 606, 607, 608, 609, 732
@PrePassivate, 553, 604, 605, 606, 609
@RegexFieldValidator, 975
@RequiredFieldValidator, 926, 975, 977, 978
@RequiredStringValidator, 926, 975, 977, 978
@Stateful, 552, 553, 555, 568, 603, 608
@Stateless, 552, 560, 561, 568, 590, 591, 596, 626, 627, 633, 636, 641, 646, 725, 731, 732, 884
@StringLengthFieldValidator, 926, 976
@StringRegexValidator, 976
@Table, 553, 612, 616, 619, 653, 740
@Temporal, 612, 621
@TransactionManagement, 590, 591
@UrlValidator, 977
@Validation, 977, 978
@Validations, 978
@VisitorFieldValidator, 979
@WebFilter, 161, 407, 414
@WebInitParam, 160, 174
@WebListener, 160
@WebServlet, 160, 161, 174, 1090, 1099, 1104, 1108, 1128, 1131, 1149, 1151, 1168, 1171, 1182, 1190, 1204, 1211
@XmlAnyAttribute, 868
@XmlAnyElement, 867, 868
@XmlAttachmentRef, 868
@XmlAttribute, 868, 874, 880, 881
@XmlElement, 845, 867, 868, 871, 872, 873, 874, 879, 880
@XmlElementRef, 867, 868

@XmlElementRefs, 867, 868
@XmlElements, 867
@XmlElementWrapper, 867, 879
@XmlAttribute, 867
@XmlAttributeValue, 867
@XmlID, 868
@XmlIDREF, 868
@XmlJavaTypeAdapter, 869
@XmlJavaTypeAdapters, 869
@XmlList, 868
@XmlMimeType, 868
@XmlMixed, 868
@XmlRootElement, 866, 872, 873, 878, 879
@XmlTransient, 868
@XmlType, 866, 871, 872, 873, 874, 879, 880, 881
@XmlValue, 868

A
abstract, 10, 22, 43, 55, 82, 313, 364, 592, 593, 594, 595,
AbstractCommandController, 1025, 1027, 1028
AbstractController, 1025, 1027, 1028, 1031
AbstractFormController, 1028
AbstractWizardFormController, 1009, 1029
Action Classes, 922, 926
Action Events, 435
Action Interface, 927, 928
Action tags, 297
ActionContext Class, 933, 934, 935
ActionMapping Class, 930, 931
ActionServlet, 922, 1002
ActionSupport Class, 928
Aggregate Function, 691
Applet container, 20
Application client container, 20
Application servers, 28
ApplicationAware interface, 941, 944, 952
Application-managed EntityManager, 613
Apply Request Values Phase, 439, 440
Arithmetic operators, 315
ArrayELResolver, 314
Associations, 690

Asynchronous JavaScript and XML (AJAX), 13, 918
Authenticator, 521, 523, 524, 542, 543, 544, 545, 546,
740, 741, 742

B

Backing Beans, 427, 432, 434, 474, 477, 515
BeanELResolver, 314
BigDecimalConverter, 481
BigIntegerConverter, 481
Bijection, 722, 731
Binary Large Object (BLOB), 55, 621
BodyContent, 332, 334, 335, 336, 339, 340
BodyTagSupport, 334, 335, 336, 344, 345, 346
booleanConverter, 481
Business Delegate, 799, 819
Business interface, 560
Business process management (BPM), 734
BusinessLifeCycleManager, 856, 902, 903, 906
BusinessQueryManager, 856, 901, 902, 904, 916
ByteArrayDataSource, 522, 533
bytecodes, 5
ByteConverter, 481

C

Cache Provider, 686, 687
Call Level Interface (CLI), 64
CallableStatement, 55, 61, 65, 70, 71, 72, 75, 77, 78, 88,
89, 90, 91, 92, 93, 94, 95, 106, 111, 112, 114, 117
Callback Methods, 547, 553, 603, 605, 732
Character Large Object (CLOB), 55, 150, 621
CharacterConverter, 481
Checkbox Component, 473
checkbox tag, 1019, 1020, 1021
checkboxes tag, 1019, 1021, 1022
ClassNotFoundException, 79, 101, 102, 103, 104, 105
Common Gateway Interface (CGI), 43, 152, 208
Component Interface, 552
Composite Entity, 799, 804, 805, 806, 823
Configuring Results, 965
Connection pooling, 62, 127, 128
ConnectionPoolDataSource, 61, 67, 128, 129, 130, 131
ConnectionSpec, 767, 786, 787

Container-managed EntityManager, 613
 Container-Managed Persistence (CMP) entity beans, 551
 Conversational State, 556
 ConversionErrorFieldValidator, 926, 968, 969, 972, 973, 978
 Cookie, 164, 208, 209, 210, 211, 212, 213, 220, 224, 233, 234, 268, 318, 322
 Core tag library, 359
 Cursors, 94
 Custom Validators, 480, 970
 CustomValidator, 978, 979

D

Data Access Object, 799, 815, 816, 817, 823, 824, 1007
 Data Component, 472
 Data Model Events, 437
 Data Tags, 962
 Database Management System (DBMS), 54, 393, 585
 DatabaseMetaData, 55, 66, 70, 146
 DataHandler, 527, 528, 854, 855, 868, 896
 DATALINK, 61, 66, 101
 dataSource, 150, 389, 390, 391, 392, 393, 394, 395, 1014, 1015
 DateRangeFieldValidator, 926, 968, 969, 972, 978
 DateTimeConverter, 462, 481, 493
 DDL, 73, 106, 146, 392, 697
 DELETE, 38, 40, 51, 73, 77, 166, 167, 391, 392, 637, 655, 657, 658, 659, 664, 665, 666
 Dependency Injection, 23, 549, 554, 722, 841, 917, 927, 951, 1004, 1005, 1006, 1009, 1035, 1040
 Deployment Descriptors, 13, 14, 16, 20, 22, 27, 36, 549, 552, 554, 602, 622, 722, 791, 792, 849, 882, 883, 884, 916, 1089
 Design Pattern, 430, 795, 797, 798, 799, 801, 803, 805, 807, 809, 811, 813, 815, 817, 819, 821, 823, 918
 Directive, 280, 291, 292, 293, 298, 326, 329
 Discovery, 826
 DispatcherServlet, 1009, 1025, 1026, 1027, 1029, 1032, 1040
 Distributed Computing Environment (DCE) RPC, 158
 DML, 93, 106, 146
 DNS server, 36

DoubleConverter, 481
 DoubleRangeFieldValidator, 926, 968, 969, 973
 DoubleRangeValidator, 466, 467, 479, 486
 DriverManager, 54, 55, 63, 64, 68, 69, 75, 76, 77, 79, 86, 87, 90, 92, 102, 104, 127, 128, 132, 149
 DynamicAttributes, 330, 332, 340, 356

E

ECHCache, 687
 EJB 3 Interceptors, 547, 599
 EJB client, 548, 549, 550, 551, 554, 784
 EJB container, 20, 33, 34, 47, 548, 549, 550, 551, 553, 554, 556, 557, 559, 589, 590, 591, 593, 594, 596, 600, 601, 603, 604, 605, 606, 610
 EJB module, 26, 27, 34, 563, 564, 567, 624
 EJB server, 47, 549, 550, 553, 573
 ejb-jar.xml, 590, 604, 725, 739, 782, 884, 1013
 ELContext, 313, 314
 ELResolver, 313, 314
 EmailValidator Class, 427, 506
 EmailValidator, 427, 468, 489, 493, 506, 507, 926, 968, 969, 972, 973, 978
 empty operator, 315, 316, 317
 Enterprise Application Integration (EAI), 762
 Enterprise Information Systems (EIS), 17, 762
 Enterprise JavaBeans (EJB), 13, 15, 44, 302, 548, 612, 682, 762, 1045
 Entity beans, 555, 613, 621
 Entity Listeners, 623
 EntityManager API, 554, 611, 612, 613, 622, 626, 742
 EVAL_BODY_INCLUDE, 330, 331, 332, 344, 345, 356
 EVAL_PAGE, 331, 334, 336, 344, 346, 356
 EXECUTE_FAILED, 74, 106
 Expression Language (EL), 276, 358, 429, 737
 ExpressionValidator, 968, 969, 973, 974, 978
 Extensible Markup Language (XML), 13, 36, 278, 328, 358, 430, 549, 622, 682, 722, 827, 1007, 1042

F

FacesServlet, 430, 431, 440, 487, 489, 507, 737, 750
 Fast Lane Reader, 799
 FieldExpressionValidator, 968, 969, 974, 978

FilterChain, 162, 403, 404, 405, 408, 409, 414, 417, 422, 424, 425

FilterConfig, 162, 403, 404, 405, 406, 408, 417, 425

First-level cache, 685

FloatConverter, 481

Folder, 518, 522, 523, 534, 535, 536, 537, 538, 539, 544, 545, 546

Form Component, 472

form tag, 371, 459, 919, 963, 964, 1018, 1019, 1025, 1026, 1100

Form Tags, 963

FunctionInfo, 334, 336, 356

Functions tag library, 359

G

Generic tags, 961, 963

GenericServlet, 152, 161, 163, 175, 205, 269, 270

GET, 37, 38, 39, 40, 51, 152, 156, 166, 167, 175, 176, 206, 214, 1027, 1056, 1061, 1063

Glassfish Application server, 29

H

HEAD, 38, 39, 41, 156, 167, 197, 198, 200, 203, 223, 229, 288, 319, 320, 322, 418, 1096

Hibernate Query Language (HQL), 32, 682

Hibernate, 681, 682, 683, 684, 685, 686, 687, 688, 689, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 707, 708, 709, 710, 711, 712, 713, 714, 715, 717, 718, 719, 720

hibernate.cfg.xml, 693, 699, 700, 702, 704, 705, 706, 707, 1000, 1001, 1002

hidden tag, 1019, 1023

HOLD_CURSOR_OVER_COMMIT, 62

Home Interface, 552

HttpServlet, 28, 42, 151, 152, 160, 161, 162, 163, 164, 165, 166, 169, 170, 174, 175, 176, 177, 180, 182, 185, 186, 187, 188, 190, 191, 192, 197, 198, 199, 200, 203, 204, 205, 206

HttpServletRequest, 28, 42, 151, 160, 163, 164, 166, 169, 174, 175, 176, 177, 180, 182, 185, 187, 188, 190, 191, 192, 197, 198, 200, 203, 204

HttpServletRequestWrapper, 164, 267, 268

HttpServletResponse, 42, 151, 164, 166, 169, 174, 175, 177, 180, 182, 185, 186, 187, 188, 192, 197, 198, 200, 203, 204, 205

HttpServletResponseWrapper, 164, 267, 268, 420, 421, 422, 425

HttpSession, 164, 205, 208, 217, 218, 219, 221, 222, 224, 226, 229, 230, 233, 234

HttpSessionActivationListener, 164, 237, 246, 247, 274

HttpSessionAttributeListener, 164, 237, 246, 247, 248, 249, 250, 274

HttpSessionBindingEvent, 164, 247, 248, 249, 274

HttpSessionBindingListener, 164, 237, 247, 248

HttpSessionContext, 164

HttpSessionEvent, 164, 247, 248, 249, 274

HttpSessionListener, 164, 237, 246, 247, 249, 250, 274

Hyper Text Transfer Protocol (HTTP), 36, 826

Hypertext Markup Language (HTML), 16, 36, 152, 213, 402, 520, 563

Hypertext Preprocessor (PHP), 152

I

Identifier, 38, 156, 268, 287, 316, 329, 359, 683, 692, 695, 792, 922, 931

IDS Driver, 60

Image Component, 473

IMAP, 518, 519, 520, 521, 531, 538, 545

ImplicitObjectResolver, 314

Inheritance, 133, 202, 611, 648, 651, 653

initialPoolSize, 62, 149

Input Component, 473

input tag, 1019, 1020, 1021, 1023

IntegerConverter, 481

Interceptors, 547, 552, 555, 599, 600, 602, 603, 605, 607, 608, 610, 724, 725, 917, 920, 927, 938, 939, 954, 955, 956, 957, 958, 964, 966, 1003

Internationalization Components, 726

Internationalization tag library, 359

Internet Message Access Protocol (IMAP), 518

Internet Protocol (IP), 217, 519

InternetHeader, 521, 531

IntRangeFieldValidator, 926, 968, 969, 974, 977, 978

Inversion of Control (IoC), 722, 918, 1004, 1006, 1040

Invoke Application Phase, 439, 441

IoC container, 1006, 1008, 1009, 1010, 1011, 1012, 1013, 1026, 1037, 1040

IterationTag, 329, 330, 331, 332, 334, 344, 356

J

Java API for RESTful Web services (JAX-RS), 15
 Java API for XML Binding (JAXB), 828
 Java API for XML Registries (JAXR), 15, 22, 915
 Java API for XML Web Services (JAX-WS), 16
 Java applet, 5, 12, 15, 21, 23, 24, 158, 302
 Java Architecture for XML Binding (JAXB), 16, 22
 Java ARchive (JAR), 15, 155, 294, 412
 Java Authentication and Authorization Services (JAAS), 16
 Java Database Connectivity (JDBC), 16, 43, 54, 158, 390, 550, 612, 762, 799, 1006
 Java Development Kit (JDK), 4
 Java EE Connector Architecture (JCA), 15, 762
 Java EE6, 646
 Java Message Service (JMS), 16, 722, 766, 779, 813
 Java Messaging, 15, 571, 588, 725
 Java Naming Directory Interface (JNDI), 15, 16
 Java Native Interface (JNI), 57, 58
 Java Persistence Query Language (JPQL), 32, 612
 Java Persistent API (JPA), 15
 Java Runtime Environment (JRE), 4, 303
 Java SE, 5, 15, 16, 17, 18, 23, 54, 63, 64, 552, 844, 849, 859, 915
 Java Servlet, 1, 15, 16, 22, 30, 31, 32, 34, 36, 42, 43, 46, 48, 151, 152, 155, 157, 158, 159, 161, 163, 168, 191, 204, 205, 207, 210, 217, 236, 266, 267, 273, 275, 276, 277, 280, 318, 325, 402, 425, 431, 672, 703, 1006, 1089, 1104
 Java Transaction API (JTA), 15, 588, 624, 683, 727, 777, 1006
 Java Virtual Machine (JVM), 4, 43, 152, 158, 278, 523, 687
 java.beans.PropertyDescriptor, 780
 java.sql, 53, 54, 55, 61, 64, 65, 66, 67, 68, 75, 76, 77, 78, 79, 85, 87, 90, 92, 93, 94, 95, 96, 97, 98, 101, 103, 106, 107, 108, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 125, 126, 127, 130, 138, 147, 148, 149
 java.sql.Array, 65, 98, 101, 122, 123, 125, 126
 java.sql.BatchUpdateException, 66, 106
 java.sql.Blob, 65, 101, 116, 621
 java.sql.CallableStatement, 65, 117
 java.sql.Clob, 65, 101, 114, 116, 621

java.sql.Connection, 64, 66, 112, 115, 125, 130, 149, 393, 503
 java.sql.DatabaseMetaData, 66
 java.sql.DataTruncation, 66
 java.SQL.Date, 65
 java.sql.Driver, 64, 76, 130, 503
 java.sql.DriverManager, 64, 76, 503
 java.sql.DriverPropertyInfo, 64
 java.sql.Nclob, 65
 java.sql.ParameterMetaData, 66
 java.sql.PreparedStatement, 65, 117, 503
 java.sql.Ref, 65, 101, 126
 java.sql.ResultSet, 65, 66, 96, 97, 98, 117, 138, 503
 java.sql.ResultSetMetaData, 66
 java.sql.RowId, 65, 101
 java.sql.Savepoint, 65
 java.sql.SQLData, 66, 101, 117
 java.sql.SQLException, 66, 77, 106, 130, 503
 java.sql.SQLInput, 66
 java.sql.SQLOutput, 66
 java.sql.SQLPermission, 64
 java.sql.SQLWarning, 66
 java.sql.SQLXML, 65
 java.sql.Statement, 65, 66, 503
 java.sql.Struct, 65, 101, 117, 118, 121
 java.sql.Time, 65, 85, 96, 97, 98, 101, 395, 575, 576, 577
 java.sql.Timestamp, 65, 97, 98, 101, 395, 575, 576, 577
 java.sql.Types, 61, 65
 java.util.concurrent.Executor, 842
 java.util.Random, 210, 214
 JavaBeans Activation Framework (JAF), 15, 23, 518
 JavaMail Architecture, 521
 JavaMail, 1, 15, 17, 23, 30, 31, 514, 517, 518, 519, 520, 521, 522, 523, 525, 527, 528, 529, 531, 533, 535, 537, 538, 539, 541, 542, 543, 544, 545, 546, 727
 JavaScript, 13, 33, 152, 429, 433, 722, 918, 1096
 JavaServer Faces (JSF), 14, 15, 276, 358, 425, 428, 722, 1007
 javax.annotation, 568, 590, 591, 603, 605, 606, 891, 892, 901
 javax.ejb, 549, 552, 560, 561, 567, 568, 573, 574, 576, 590, 591, 594, 595, 597, 603, 604, 606, 607, 627, 632, 636, 641, 645, 646, 670, 784, 1049
 javax.ejb.ActivationConfigProperty, 576

javax.ejb.MessageDriven, 574, 576, 607, 784
javax.ejb.MessageDrivenBean, 574, 784
javax.faces, 428, 437, 439, 442, 460, 461, 462, 463, 466, 468, 469, 471, 472, 473, 474, 475, 479, 480, 481, 482, 486, 487, 488, 489, 499, 506, 507, 737, 738, 750
javax.faces.application.FacesMessage, 480, 499, 506
javax.faces.component.html.HtmlMessages, 473
javax.faces.component.html.HtmlDataTable, 472
javax.faces.component.html.HtmlForm, 472, 499
javax.faces.component.html.HtmlGraphicImage, 473
javax.faces.component.html.HtmlMessage, 473
javax.faces.component.UIComponentBase, 471
javax.faces.component.UIOutput, 473
javax.faces.component.UIParameter, 473
javax.faces.validator.Validator, 468, 479, 480, 506
javax.faces.validator.ValidatorException, 480, 506
javax.jms.Message, 574, 576, 577
javax.jms.MessageListener, 574, 576, 577
javax.jms.TextMessage, 576
javax.jws, 843, 850, 851, 884, 888, 892
javax.jws.soap.SOAPMessageHandlers, 851
javax.jws.SOAPBinding, 843
javax.mail.Address, 525, 526, 533, 545
javax.mail.Authenticator, 524, 543, 544, 545, 546
javax.mail.Folder, 535, 544, 545, 546
javax.mail.internet.InternetAddress, 533, 534, 542
javax.mail.internet.InternetHeaders.InternetHeader, 531
javax.mail.internet.MimeBodyPart, 529
javax.mail.internet.MimeMessage, 524, 542
javax.mail.internet.MimeUtility, 530
javax.mail.internet.NewsAddress, 533, 534
javax.mail.internet.ParameterList, 531
javax.mail.internet.PreencodedMimeBodyPart, 529
javax.mail.Message, 524, 542, 544, 545
javax.mail.Multipart, 528
javax.mail.Quota, 531, 532
javax.mail.Quota.Resource, 532
javax.mail.QuotaAwareStore, 531
javax.mail.search, 538, 539, 540, 541, 545
javax.mail.Store, 534, 544, 546
javax.mail.Transport, 541, 542, 544, 546
javax.mail.util.ByteArrayDataSource, 533
javax.mail.util.SharedByteArrayInputStream, 533
javax.mail.util.SharedFileInputStream, 532
javax.naming.InitialContext, 646, 784
javax.persistence, 614, 617, 627, 630, 631, 632, 635, 636, 639, 640, 641, 643, 645, 646, 647, 664, 665, 670, 740, 741, 742
javax.persistence.Entity, 627, 630, 631, 632, 635, 636, 639, 640, 641, 643, 645, 646, 740, 741, 742
javax.persistence.Id, 630, 631, 635, 639, 640, 643, 645, 740
javax.persistence.Table, 740
javax.resource.cci, 766, 767, 786, 787, 788, 793
javax.resource.cci.Connection, 786, 787
javax.resource.cci.ConnectionFactory, 786
javax.resource.cci.ConnectionMetaData, 786
javax.resource.cci.ConnectionSpec, 786, 787
javax.resource.cci.IndexedRecord, 786
javax.resource.cci.Interaction, 786, 788
javax.resource.cci.InteractionSpec, 786, 788
javax.resource.cci.LocalTransaction, 786, 788
javax.resource.cci.MappedRecord, 786
javax.resource.cci.Record, 786
javax.resource.cci.RecordFactory, 786
javax.resource.cci.ResourceAdapterMetaData, 786
javax.resource.cci.ResultSet, 786
javax.resource.cci.ResultSetInfo, 786
javax.resource.cci.Streamable, 786
javax.resource.NotSupportedException, 780
javax.resource.Referenceable, 786
javax.resource.ResourceException, 781, 786, 789
javax.resource.spi, 768, 769, 770, 771, 775, 776, 780, 781, 783, 788, 789, 790, 793
javax.resource.spi.endpoint, 780, 781, 793
javax.resource.spi.endpoint.MessageEndpointFactory, 780
javax.resource.spi.work, 769, 775, 776, 793
javax.servlet, 152, 161, 163, 164, 165, 167, 169, 174, 177, 180, 182, 187, 191, 192, 197, 198, 199, 203, 205, 208, 209, 210
javax.servlet.annotation.WebServlet, 174, 1104, 1128, 1149, 1168, 1182, 1204
javax.servlet.Filter, 403, 417
javax.servlet.FilterChain, 403, 417
javax.servlet.FilterConfig, 417

- javax.servlet.http, 152, 161, 163, 164, 165, 169, 174, 177, 180, 182, 187, 192, 197, 198, 199, 203, 205, 208, 209, 210
 javax.servlet.http.annotation.Jaxrs, 174
 javax.servlet.http.Cookie, 209
 javax.servlet.http.HttpServlet, 165, 205, 210, 267, 285, 708, 709, 710, 933, 944, 953, 954, 1031
 javax.servlet.http.HttpServletRequestWrapper, 267
 javax.servlet.http.HttpServletResponseWrapper, 267
 javax.servlet.http.HttpSession, 208, 218, 222, 233, 249, 708, 709, 710, 933, 953
 javax.servlet.http.HttpSessionAttributeListener, 249
 javax.servlet.http.HttpSessionBindingEvent, 249
 javax.servlet.jsp.tagext, 329, 330, 334, 345, 349, 350
 javax.servlet.RequestDispatcher, 191, 708, 709, 710
 javax.servlet.ServletContext, 239, 241, 286
 javax.servlet.ServletContextAttributeEvent, 241
 javax.servlet.ServletContextAttributeListener, 241
 javax.servlet.ServletContextEvent, 239, 241
 javax.servlet.ServletContextListener, 239, 241
 javax.servlet.ServletRequest, 165, 191, 267, 345, 403, 417
 javax.servlet.ServletRequestWrapper, 267
 javax.servlet.ServletResponse, 165, 191, 267, 403, 417
 javax.servlet.ServletResponseWrapper, 267
 javax.servlet.http.annotation, 174
 javax.sql, 53, 54, 55, 64, 66, 67, 68, 127, 129, 130, 131, 134, 142, 148, 149, 150
 javax.sql.CommonDataSource, 67
 javax.sql.ConnectionEvent, 67, 130, 131, 150
 javax.sql.ConnectionEventListener, 67, 130, 131, 150
 javax.sql.ConnectionPoolDataSource, 67, 130
 javax.sql.DataSource, 67, 130, 554, 677
 javax.sql.PooledConnection, 67, 130, 150
 javax.sql.RowSet, 64, 68, 131, 142
 javax.sql.RowSetEvent, 68
 javax.sql.RowSetListener, 68
 javax.sql.RowSetMetaData, 68
 javax.sql.RowSetReader, 68
 javax.sql.RowSetWriter, 68
 javax.sql.StatementEvent, 67
 javax.sql.StatementEventListener, 67
 javax.sql.XAConnection, 67
 javax.sql.XADataSource, 67
 javax.transaction.UserTransaction, 590, 591
 javax.transaction.xa.XAResource, 590, 781
 javax.xml.bind.Binder, 847
 javax.xml.registry, 856, 900, 901
 javax.xml.registry.infomodel, 856
 javax.xml.transform.Source, 841, 854, 876
 javax.xml.ws.BindingProvider, 841
 javax.xml.ws.EndPoint, 844, 915
 javax.xml.ws.handler.MessageContext, 840, 891
 javax.xml.ws.Service, 842, 843, 860, 864, 886, 889, 894
 JAXB 2.2, 825, 839, 844, 845, 859, 864, 869, 874, 878, 914, 915, 916
 JAXR 1.0, 839, 859
 JAXR Architecture, 825, 855, 856
 JAXR Pluggable Provider, 856, 857
 JAXRBridge Provider, 856, 857
 JAXRclient, 855
 JAXRprovider, 855
 JAX-WS 2.2, 825, 839, 841, 842, 843, 851, 859, 914
 jax-ws-catalog.xml, 884
 JBoss application server, 29, 30, 752, 754, 755
 JBoss Cache, 687
 jboss-web.xml, 749, 752, 755
 JDBC-ODBC bridge, 55, 56, 57, 149
 Joins, 688, 690
 JSF Request Processing Life Cycle, 427, 438, 439
 JSP Standard Tag Library (JSTL), 14, 429
 JspFragment, 333, 334, 336, 337, 341, 349, 352, 352, 353, 355
 JspIdConsumer, 330, 333
 JspTag, 330, 331, 333, 337, 356
 Just In Time (JIT) compiler, 5
- ## K
- KEEP_CURRENT_RESULT, 62, 74
- ## L
- LengthValidator, 467, 479, 486
 ListELResolver, 314
 LongConverter, 481
 LongRangeValidator, 467, 479, 486, 493

M

Mail User Agent (MUA or UA), 518

mail.debug, 522, 543

mail.from, 522

mail.host, 522

mail.mime.decodeparameters, 531

mail.mime.encodeparameters, 531

mail.protocol.host, 522

mail.protocol.user, 522

mail.store.protocol, 522, 544

mail.transport.protocol, 522, 523, 535

mail.user, 522

MapELResolver, 314

maxIdleTime, 62, 131, 149

maxPoolSize, 62, 131, 149

maxStatements, 62, 149

MDB, 547, 548, 555, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 598, 599, 600, 605, 606, 607, 610

Message Delivery Agent (MDA), 519

Message store (MS), 519

Message Transfer Agent, 518

MessageListener, 573, 574, 576, 577, 607, 782, 785

Messages Component, 473

META-INF, 27, 76, 562, 563, 564, 569, 579, 625, 728, 739, 792, 859, 884

MethodExpression, 313, 314

MimeBodyPart, 521, 525, 528, 529, 530

MimeMessage, 521, 524, 525, 528, 541, 542, 543

MimeUtility, 521, 525, 530

minPoolSize, 62, 131, 149

Model-View-Controller (MVC), 10

MS SQL, 30, 61, 695

MultiActionController, 1027, 1028

MultiPart, 521, 527, 529

N

Navigation, 427, 432, 438, 482, 507, 509, 515, 660, 735, 918, 959, 993, 1004

NNTP, 518, 519, 520

NO_GENERATED_KEYS, 61, 74

Non-Form Tags, 963

n-tier architecture, 6, 9, 10, 11, 33

NumberConverter, 463, 481

O

Object Oriented Programming (OOP), 2

Object Relational Mapping (ORM), 32, 549, 612, 682, 739

Object-Graph Navigation Language (OGNL), 918

OCI (Oracle Call Interface) Driver, 58

Open Database Connectivity (ODBC), 54

Open Network Computing (ONC) RPC, 158

OpenSymphony cache, 687

option tag, 1019, 1022, 1023

OPTIONS, 38, 40, 41, 167

org.apache.struts2.dispatcher.FilterDispatcher, 920, 923, 948, 995

org.apache.struts2.dispatcher.mapper.ActionMapper, 930, 938

org.apache.struts2.dispatcher.mapper.ActionMapper, 938

org.apache.struts2.interceptor.ApplicationAware, 939, 941, 944, 952, 979

org.apache.struts2.interceptor.ParameterAware, 952, 953

org.apache.struts2.interceptor.ServletRequestAware, 944, 953

org.apache.struts2.interceptor.ServletResponseAware, 954

org.apache.struts2.interceptor.SessionAware, 954

org.apache.struts2.ServletActionContext, 933

org.hibernate.cfg.Configuration, 685, 705

org.hibernate.Session, 685, 705

org.hibernate.SessionFactory, 685, 705

org.hibernate.Transaction, 685, 705

org.hibernate.validator.Length, 740

org.hibernate.validator.NotNull, 740

org.hibernate.validator.Pattern, 740

org.jboss.seam.annotations.Name, 740, 742

org.jboss.seam.annotations.Scope, 740, 742

org.jboss.seam.core.init.clientSideConversations, 727

org.jboss.seam.core.init.debug, 727

org.jboss.seam.core.init.jndiPattern, 727

org.jboss.seam.core.init.userTransactionName, 727

org.jboss.seam.core.manager.conversationIsLongRunningParameter, 727
 org.jboss.seam.core.manager.conversationTimeout, 727, 728
 org.jboss.seam.core.resourceBundle, 726
 org.jboss.seam.international.locale, 726
 org.jboss.seam.international.timezone, 726
 org.jboss.seam.mail.mailSession, 726, 727
 org.jboss.seam.mail.mailSession.debug, 727
 org.jboss.seam.mail.mailSession.host, 726
 org.jboss.seam.mail.mailSession.password, 727
 org.jboss.seam.mail.mailSession.port, 727
 org.jboss.seam.mail.mailSession.sessionJndiName, 727
 org.jboss.seam.mail.mailSession.ssl, 727
 org.jboss.seam.mail.mailSession.username, 727
 org.jboss.seam.ScopeType.SESSION, 740, 741
 org.springframework.beans, 1006, 1007, 1008, 1009, 1033, 1040
 org.springframework.context, 1006, 1007, 1009, 1033
 org.springframework.context.ApplicationContext, 1009
 org.springframework.jdbc.datasource.DataSourceUtils, 1015
 org.w3c.dom.Element, 847, 848, 851
 OUT, 72, 88, 89, 91, 92, 143, 863, 881, 891, 1084
 Output Component, 473

P

PageData, 334, 338, 342, 343
 pages.xml, 736, 737, 748, 749, 750, 751, 755
 Parameter Component, 473
 ParameterAware interface, 952, 953
 ParameterList, 521, 531
 ParameterMetaData, 61, 66
 password tag, 1019, 1022
 persistence.xml, 624, 625, 628, 668, 680, 749, 752, 755
 Phase Events, 437
 Plain Old Java Interfaces (POJI), 549
 Plain Old Java Objects (POJOs), 13, 682, 926
 POP3, 518, 519, 520, 521, 522, 544, 545, 546
 POST, 38, 39, 40, 41, 51, 152, 156, 166, 167, 175, 176, 206, 214, 229, 242, 422, 562, 578, 1019, 1024, 1025, 1027, 1056, 1061, 1063, 1065, 1068

PreencodedMimeBodyPart, 521, 529, 530
 PreparedStatement, 55, 61, 62, 65, 71, 72, 75, 77, 78, 82, 83, 84, 85, 86, 87, 88, 106, 108, 109, 111, 112, 114, 115, 117, 119, 120, 123, 124, 125, 126, 133, 147
 Process Validations Phase, 439, 440
 Property Settings, 728
 propertyCycle, 62, 149
 Pure Java driver, 56, 149
 PUT, 38, 40, 51, 166, 167, 213, 229, 578, 881, 927, 966

Q

Query Cache, 686, 687
 Quota, 518, 521, 531, 532
 QuotaAwareStore, 521, 531, 532

R

radiobutton tag, 1019, 1021
 radiobuttons tag, 1019, 1022
 RegexFieldValidator, 968, 969, 974, 975, 978
 RegistryService, 856, 901
 Registry-Specific JAXR Provider, 856, 857
 Relational and logical operators, 315
 Relational DataBase Management System (RDBMS), 682
 Remote Procedure Call (RPC), 158
 Render Response Phase, 439, 441
 Renderer, 432, 433, 515
 RequestDispatcher, 160, 162, 191, 192, 204, 205, 206
 RequestWrapper, 163, 164, 267, 268, 269, 270, 271, 862, 863, 864
 RequiredStringValidator, 926, 968, 969, 975, 977, 978
 Resource adapter, 762, 777, 790
 Resource manager, 762
 ResourceBundleELResolver, 314
 Restore View Phase, 439
 ResultSet, 54, 55, 61, 62, 65, 66, 70, 71, 72, 73, 74, 75, 77, 82, 84, 88, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 113, 116, 117, 120, 122, 123, 125, 126, 127, 131, 132, 133, 134, 135, 136, 137, 138, 140, 142, 143, 144, 294, 437, 472, 503, 767, 786, 787, 1085, 1094, 1109, 1112, 1134, 1135, 1138, 1174, 1187, 1190, 1212, 1213, 1216, 1228
 ResultSetMetaData, 55, 66, 135
 RETURN_GENERATED_KEYS, 61, 74

S

SAAJ 1.3, 825, 839, 851, 859
 SAAJ, 18, 825, 839, 845, 846, 851, 852, 853, 859, 896, 897, 898, 914, 915
 Savepoints, 62, 146, 147, 148
 ScopedAttributeResolver, 314
 Scriptlet, 280, 281
 Seam Components, 724, 725, 728, 733
 Seam framework, 718, 722, 723, 724, 725, 726, 728, 730, 731, 732, 733, 734, 737, 738, 739, 749, 759, 760
 Seam Resource Servlet, 738, 750
 Seam Servlet Filters, 738, 739
 Second-level cache, 685
 Secure Socket Layer (SSL), 209, 234, 727
 SEI, 840, 843, 847, 849, 851, 859, 860, 861, 862, 863, 864, 884, 887, 916
 SelectItem and SelectItems Component, 473
 SelectMany and SelectOne Component, 473
 Service Activator, 799
 Service Locator, 799, 813, 814, 815, 823, 824
 Service Oriented Architecture (SOA), 823, 826
 ServletConfig, 151, 161, 162, 166, 167, 168, 169, 170, 174, 175, 200, 205, 206
 ServletContext, 151, 153, 161, 162, 163, 169, 170, 174, 175, 202, 205, 206
 ServletContextAttributeListener, 162, 237, 238, 239, 241, 246, 248, 273, 274
 ServletContextListener, 162, 237, 238, 239, 241, 258, 259, 273, 274
 ServletInputStream, 153, 163, 182, 205
 ServletOutputStream, 153, 163, 243, 420, 421, 422
 ServletRequest, 28, 42, 151, 152, 160, 161, 162, 163, 164, 165, 166, 169, 174, 175, 176, 177, 180, 182, 185, 187, 188, 190, 191, 192, 197, 198, 200, 203, 204, 205, 206
 ServletRequestAttributeEvent, 163
 ServletRequestAttributeListener, 162, 237
 ServletRequestAware interface, 944, 952, 953, 954
 ServletRequestEvent, 163
 ServletRequestListener, 161, 162, 237
 ServletRequestWrapper, 163, 164, 267, 268, 270, 271
 ServletResponse, 42, 151, 152, 162, 163, 164, 165, 166, 169, 174, 175, 177, 180, 182, 185, 186, 187, 188, 191, 192, 197, 198, 200, 203, 204, 205

ServletResponseAware interface, 952, 954
 ServletResponseWrapper, 163, 164, 267, 268, 271, 420, 421, 422, 425
 Session beans, 555, 610, 621
 SessionAware interface, 952, 954
 SessionFactory, 683, 685, 705, 706, 707, 1002, 1014, 1015, 1036, 1037
 SharedByteArrayInputStream, 522, 533
 SharedFileInputStream, 522, 532, 533
 ShortConverter, 481
 SIB, 849, 851, 883, 884, 885, 887, 888, 891, 892, 916
 Simple Mail Transfer Protocol (SMTP), 161, 518, 726
 Simple Object Access Protocol (SOAP), 15, 826, 839, 1046
 SimpleFormController, 1009, 1025, 1027, 1028, 1029
 SimpleTag, 329, 330, 331, 332, 333, 334, 337, 338, 349, 350
 SimpleTagSupport, 334, 337, 349, 350
 SingleThreadModel, 162, 163, 165
 single-tier architecture, 6
 SKIP_BODY, 330, 331, 332, 334, 336, 344, 345, 346, 356
 SKIP_PAGE, 331, 337, 345, 353, 356
 SqlData, 55
 Stateful Navigation Model, 735
 stateful session bean, 551, 556, 558, 559, 560, 567, 568, 600, 603, 604, 606, 609, 610
 Stateless Navigation Model, 735
 stateless session bean, 552, 556, 557, 558, 559, 560, 561, 567, 568, 572, 573, 598, 610
 StAX 1.0, 825, 839, 857, 859, 915
 StAX APIs, 857
 Streaming API for XML (StAX), 23, 915
 StringLengthFieldValidator, 926, 968, 969, 970, 971, 976, 978
 StringRegexValidator, 976, 978
 Struts 2 Annotations, 926

T

2-tier architecture, 6, 7, 8, 9, 19, 58, 60
 3-tier architecture, 6, 8, 9, 19, 34, 59, 796
 Tag extensions, 328, 329
 Tag handler, 313, 332, 344, 349, 356
 Tag Libraries, 51, 357, 359, 427, 441

Tag Library Descriptor (TLD), 328, 738, 1018

TagAdapter, 334, 337, 338

TagAttributeInfo, 334, 338, 341

TagData, 334, 340, 341, 342, 343, 356

TagFileInfo, 334, 338, 339, 340, 341

TagInfo, 334, 338, 339, 340, 341

TagLibraryInfo, 334, 338, 339, 340

TagLibraryValidator, 329, 334, 342, 343, 344, 356

TagSupport, 334, 335, 336, 337, 344, 345, 346, 349, 350

TagVariableInfo, 334, 340, 356

Tiles Plugin, 987, 989

TLD file, 323, 324, 328, 329, 336, 339, 341, 345, 346, 349, 350, 428

TRACE, 38, 41, 167

Transaction Processing (TP), 762

TransactionAwareDataSourceProxy, 1015, 1016

Transfer Object, 799, 806, 808, 809, 810, 823, 1128

Transmission Control Protocol (TCP), 217, 519

Transport, 518, 522, 524, 541, 542, 543, 544, 546, 1045, 1078, 1204

TryCatchFinally, 330, 333

Type-1 Driver, 56, 57, 149

Type-2 Driver, 56, 58, 59, 149

Type-3 Driver, 56, 59, 60, 149

Type-4 Driver, 56, 60, 61, 149

U

UDDI, 826, 827, 828, 857, 900

UI framework, 444, 470, 478, 484, 514

UI tags, 961, 963

UnavailableException, 163, 780, 781, 794

Unit Testing, 1034

Universal Description, 826

Update Model Values Phase, 439, 440

UrlValidator, 976, 977

User Interface (UI), 44, 311, 428, 430, 1085

V

ValidationMessage, 334, 340, 342, 343, 344

validators.xml, 967, 969, 971

Value List Handler, 799

Value-change Events, 436

ValueExpression, 313

VariableInfo, 334, 340, 341, 342, 356

ViewResolver, 1009, 1029, 1030, 1031, 1032

ViewResolvers, 1030

ViewRoot Component, 474

VisitorFieldValidator, 968, 969, 978, 979

W

Web ARchive (WAR), 15, 171, 244, 308, 348, 715

Web containers, 36, 45, 47, 51, 685, 1042, 1055

Web Service Distributed Management (WSDM), 829

Web Services Description Language (WSDL), 826, 914

WebLogic application server, 29

Weblogic RMI Driver, 60

webservices.xml, 849, 882, 884, 890, 891, 893, 916

WebSphere application server, 29, 47

WebWork 2, 920, 921

Wireless Markup Language (WML), 16, 49, 431

WorkEvent, 775, 776, 794

Workflow Management, 761, 773, 774

WorkListener, 774, 775, 776, 793

WSDL, 825, 826, 827, 828, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 842, 843, 844, 846, 848, 849, 850, 851, 857, 859, 860, 861, 862, 863, 865, 882, 884, 885, 886, 887, 889, 890, 894, 895, 903, 914, 915, 916

WSEE 1.3, 825, 839, 848, 849, 859, 914, 916

WS-Metadata 2.0, 839

X

XADatasource, 61, 67

XML namespaces, 323, 728, 730, 841

XML Path (XPath), 366

XML tag library, 358, 359, 366, 367, 369, 400

What's on the CD-ROM

The CD-ROM included in the *Java Server Programming Java EE 6 Black Book, Platinum Edition* contains elements specifically selected to enhance the usefulness of this book, including:

- Code from the book** – All executable applications provided in the book are included in the CD as well, so that you can run them without entering them line by line.
- Appendices** – The CD contains the appendices on the following technologies:
 - **RMI-IIOP** – Refers to a technology used in a distributed system to remotely invoke Java methods of the objects stored in a network. You can learn more about RMI-IIOP in *Appendix A, RMI-IIOP*, which is provided in CD.
 - **JNDI** – Refers to an API that enables Java programs to interact with any naming and directory service. You can learn more about JNDI in *Appendix B, Understanding Directory Services and JNDI*, which is provided in CD.
 - **XML** – Refers to a markup language based on simple, platform-independent rules for processing and displaying textual information in a structured way. You can learn more about XML in *Appendix C, XML*, which is provided in CD.
 - **UML** – Refers to a language used to model different types of applications, such as Web applications and enterprise applications. You can learn more about UML in *Appendix D, UML*, which is provided in CD.
- JBoss 4 Application Server (AS)** – The CD contains JBoss AS to run the Seam applications.
- Glassfish V3** – The CD contains Glassfish V3 application server used to deploy and run Web and enterprise applications provided in this book.

Hardware Requirements

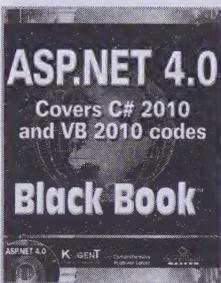
- PC – P4/AMD 64**
- Processor** – 1 GHz CPU (recommended)
- Hard disk** – 5 GB of disk space
- Memory** – 1 GB of RAM minimum (2 GB recommended)

Software Requirements

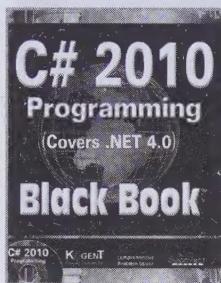
- Windows XP/Vista/2000/2003/7
- JDK 1.6
- Sun Application Server (Glassfish V3)
- NetBeans 6.8
- Oracle 10g
- Jboss application server

A wide range of Black Books

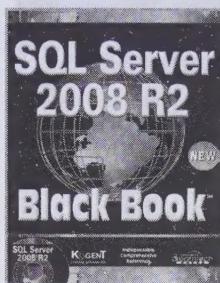
New Releases



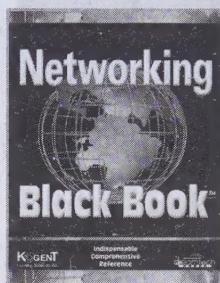
AUTHOR: KOGENT
LEARNING SOLUTIONS INC.
PAGES: 1704
PRICE: Rs. 599/- w/CD



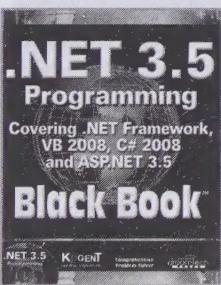
AUTHOR: KOGENT
LEARNING SOLUTIONS INC.
PAGES: 1800
PRICE: TBA



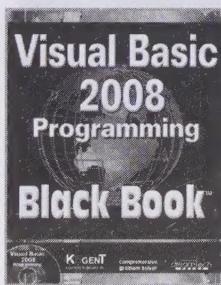
AUTHOR: KOGENT
LEARNING SOLUTIONS INC.
PAGES: 1170
PRICE: Rs. 599/- w/CD



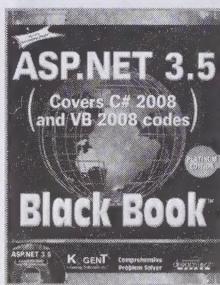
AUTHOR: KOGENT
LEARNING SOLUTIONS INC.
PAGES: 900
PRICE: TBA



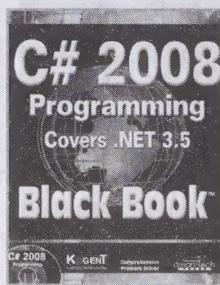
ISBN: 978-81-7722-834-2
AUTHOR: KOGENT LEARNING SOL. INC.
PAGES: 1764
PRICE: Rs. 599/- w/CD



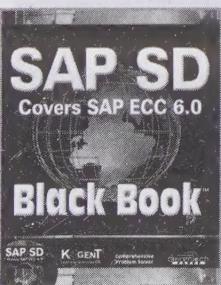
ISBN: 978-81-7722-833-5
AUTHOR: KOGENT SOLUTIONS INC.
PAGES: 1792
PRICE: Rs. 599/- w/CD



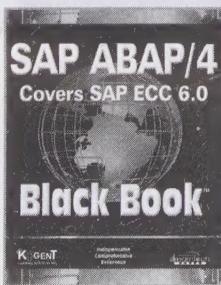
ISBN: 978-81-7722-831-1
AUTHOR: KOGENT SOLUTIONS INC.
PAGES: 1792
PRICE: Rs. 599/- w/CD



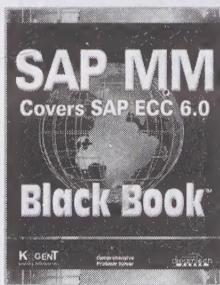
ISBN: 978-81-7722-832-8
AUTHOR: KOGENT SOLUTIONS INC.
PAGES: 1808
PRICE: Rs. 599/- w/CD



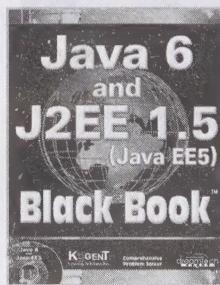
ISBN: 978-81-7722-379-8
AUTHOR: KOGENT LEARNING SOL. INC.



ISBN: 978-81-7722-429-0
AUTHOR: KOGENT LEARNING SOL. INC.



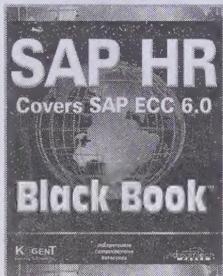
ISBN: 978-81-7722-380-4
AUTHOR: KOGENT LEARNING SOL. INC.
PAGES: 550
PRICE: Rs. 499/-



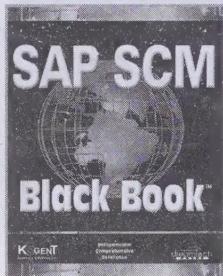
ISBN: 978-93-5004-009-6
AUTHOR: KOGENT LEARNING SOL. INC.
PAGES: 850
PRICE: Rs. 599/- w/CD

A wide range of Black Books

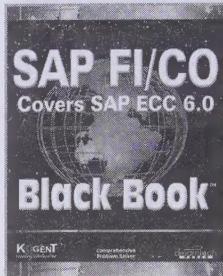
Forthcoming



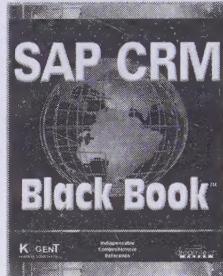
AUTHOR: KOGENT
LEARNING SOLUTIONS INC.
PAGES: 550
PRICE: Rs. 499/-



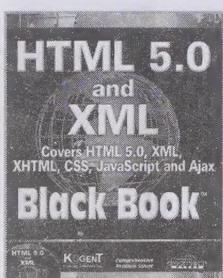
AUTHOR: KOGENT
LEARNING SOLUTIONS INC.
PAGES: 550
PRICE: Rs. 499/-



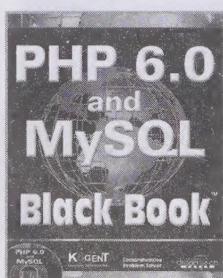
AUTHOR: KOGENT
LEARNING SOLUTIONS INC.
PAGES: 550
PRICE: Rs. 499/-



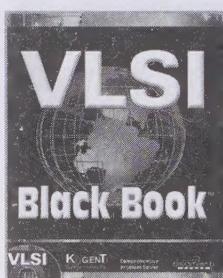
AUTHOR: KOGENT
LEARNING SOLUTIONS INC.
PAGES: 550
PRICE: Rs. 499/-



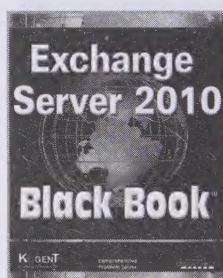
AUTHOR: KOGENT
LEARNING SOLUTIONS INC.
PAGES: 1200
PRICE: TBA



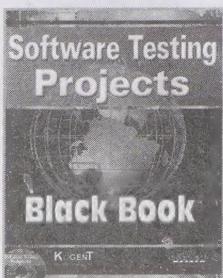
AUTHOR: KOGENT
LEARNING SOLUTIONS INC.
PAGES: 800
PRICE: TBA



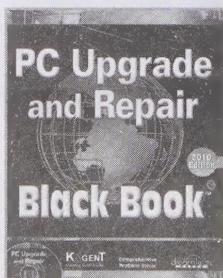
ISBN: 978-81-7722-744-4
AUTHOR: KATTULA SHYAMALA
PAGES: 776
PRICE: Rs. 379/-



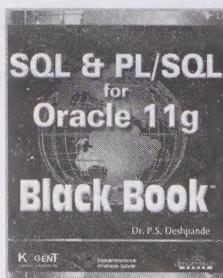
AUTHOR: KOGENT
LEARNING SOL. INC.
PAGES: 1000
PRICE: TBA



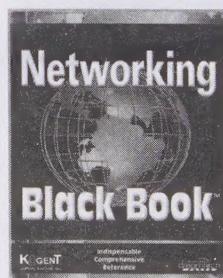
AUTHOR:
NAGARESWARA RAO PUSULARI
PAGES: 400
PRICE: TBA



ISBN: 978-81-7722-743-7
AUTHOR: KOGENT SOLUTIONS INC.
PAGES: 740
PRICE: Rs. 479/-



ISBN: 978-81-7722-940-0
AUTHOR: DR. DESHPANDE
PAGES: 776
PRICE: Rs. 399/- w/CD



AUTHOR: KOGENT
LEARNING SOLUTIONS INC.
PAGES: 900
PRICE: TBA



Java Server Programming Black Book

Many bookstores offer numerous choices of books on Java Server Programming; however, most of these books are intricate and complex to grasp. So, what are your chances of picking up the right one? If this question has been troubling you, be rest assured now! This book, Java Server Programming: Java EE 6 (J2EE 1.6) Black Book, Platinum Edition, is a one-time reference book that covers all aspects of Java EE in an easy-to-understand approach—for example, running of an application server; deploying a Java application on the GlassFish V3 application server; how GlassFish V3 Application server deploys a Java application; exploring design patterns, best practices, and design strategies; working with Java related technologies, such as NetBeans IDE 6.8, Hibernate 3.5, Spring 3.0, and Seam frameworks; and proven solutions using the key Java EE 6 technologies, such as JDBC 4.0, Servlets 3.0, JSP 2.1, JSTL 1.2, RMI, JNDI, JavaMail, Web services, JCA 1.6, Struts 2.1, JSF 2.0, UML, and much more. The book explores all these concepts with appropriate examples and executable applications—no doubt, every aspect of the book is worth its price.

This book will help you to:

- Explore the world of Java EE Connector Architecture
- Employ the right tools, design patterns, and frameworks effectively and appropriately for developing Web and enterprise applications
- Discover the concepts of Web containers and Servlets 3.0
- Get in-depth knowledge about JNDI, RMI, CORBA, and JDBC
- Appreciate JSP including JSTL, Struts, JSF syntax, and directives
- Create enterprise applications using EJB 3.1
- Learn the new features of EJB, such as the Persistence API
- Get familiar with Web Services
- Deploy Web and enterprise applications
- Handle Listeners and filters in Web applications
- Work with Hibernate 3.5, Seam 2.0, and Spring 3.0
- Deploy Seam applications on GlassFish V3 and JBoss Application servers
- Implement the UML diagrams

Technologies covered are:

- | | |
|----------------|---------------------------|
| • JDBC 4.0 | • JPA 2.0 |
| • Servlets 3.0 | • Hibernate 3.5 |
| • JSP 2.1 | • JBoss Seam 2.0 |
| • JSTL 1.2 | • JCA 1.6 |
| • JSF 2.0 | • SOA (Java Web Services) |
| • JavaMail 1.4 | • Struts 2.1 |
| • EJB 3.1 | • Spring 3.0 |
- AJAX
 - Facelets
 - JMS
 - RMI-IOP
 - JNDI
 - XML
 - UML

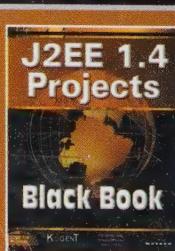
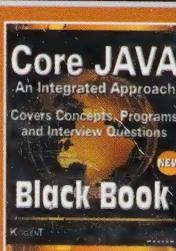
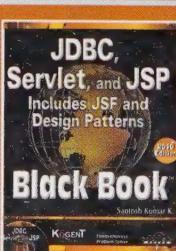
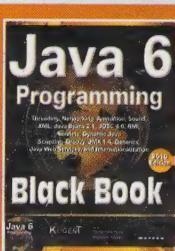
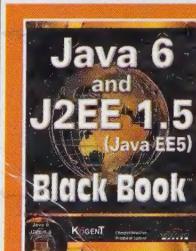


The CD-ROM with this book is packed with source code files from the chapters.

About the Author

The proficient team at Kogent Learning Solutions Inc. and Dreamtech Press has seized the market of computer books bringing excellent content in software development to the fore. The team is committed to excellence—excellence in quality of content, excellence in the dedication of its authors and editors, excellence in the attention to detail, and excellence in understanding the needs of their readers.

Also Available



CATEGORY:
Programming/J2EE

USER LEVEL:
Intermediate to Advanced

INDIAN PRICE: Rs. 599/- with CD

International Edition: US \$ 67

978-81-7722-936-3



9 788177 229363

Published by:



19-A, Ansari Road, Daryaganj, New Delhi-110002
Tel.: 91-11-23284212, 23243075
Fax: 91-11-23243078
Email: feedback@dreamtechpress.com
www.dreamtechpress.com
Blog: http://dreamtechpress.wordpress.com