```
"http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">

  <display-name>Lesson35 Sample</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <servlet>
        <servlet-name>springServletDispatcher</servlet-name>
        <servlet-class>
          org.springframework.web.servlet.DispatcherServlet
       </servlet-class>
        <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
        <servlet-name>springServletDispatcher</servlet-name>
        <url-pattern>*.html</url-pattern>
  </servlet-mapping>
</web-app>
```

In our example the MVC pattern will be represented by the following: for the model a Java class called `ContactUs`; for the view two JSP pages, `hello` and `contactUs`; and for the classes two Java controller classes, `HelloWorldController` and `ContactUsController`. Figure 35-2 shows a snapshot of my Eclipse project to give you an idea of where the files are located.

`HelloWorldController` is shown in Listing 35-2. It includes the `@RequestMapping` annotation that instructs the controller to call the method `sayHello()` for any requests with the `/hello` pattern.

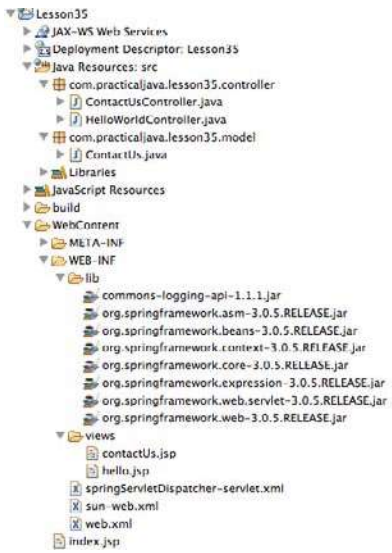


FIGURE 35-2



LISTING 35-2: HelloWorldController

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class HelloWorldController {

    @RequestMapping("/hello")
    public ModelAndView sayHello(){

        String greetings = "Hello people";
        return new ModelAndView("hello", "greetings", greetings);
    }
}
```

The view resolver of our application is an XML configuration file named `springServletDispatcher-servlet.xml`, shown in Listing 35-3. Note that the prefix of this file name matches the value in the tag `<servlet-name>` in Listing 35-1. This application uses one of the view resolvers provided by Spring, `UrlBasedViewResolver`, which creates the appropriate view based on the URL.



LISTING 35-3: springServletDispatcher-servlet.xml

```
<?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:p="http://www.springframework.org/schema/p"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-3.0.xsd">

      <context:component-scan
            base-package="com.practicaljava.lesson35.controller" />

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

The preceding file is configured to add the prefix `WEB-INF/views` and the suffix `.jsp` to the value specified in the `@RequestMapping` of the controller. As an example, for `HelloWorldController` the URL of the view given to the response object becomes `WEB-INF/views/hello.jsp`. The source code for `hello.jsp` is shown in Listing 35-4.



LISTING 35-4: hello.jsp

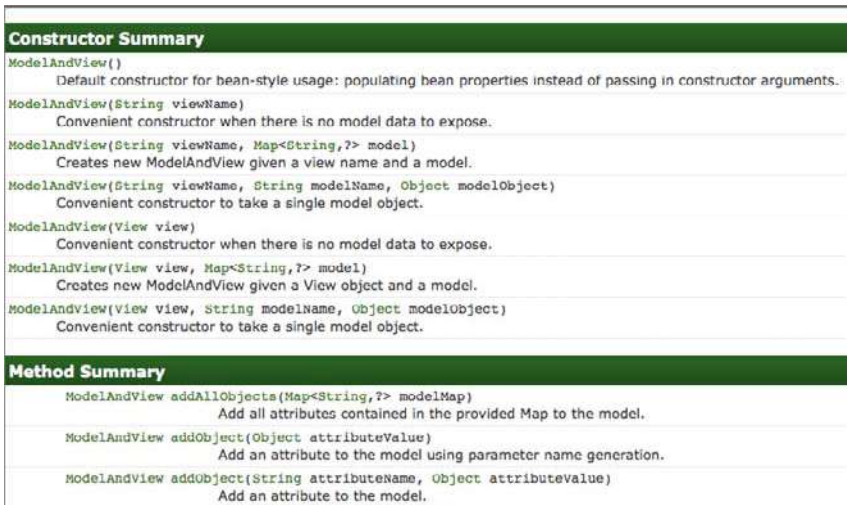
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
 ${greetings}
</body>
</html>
```

What does `${greetings}` mean? Look at the code of `HelloWorldController` in Listing 35-2. It contains the following lines:

```
String greetings = "Hello people";
return new ModelAndView("hello", "greetings", greetings);
```

This is how the Spring controller tells the view resolver what to do with the view before it is returned to the client. The first argument of the `ModelAndView` constructor contains the name of the view, and the second one contains the name of the JSP element to be replaced with the content of the third argument. Basically, the name of the class reflects the fact that it contains elements of both the model and the view.

The class `ModelAndView` has a number of overloaded constructors (see Figure 35-3) that enable you to pass multiple values to the view object. You can also populate `ModelAndView` with data by using the method `addObject()`.



Constructor Summary

`ModelAndView()`
Default constructor for bean-style usage: populating bean properties instead of passing in constructor arguments.

`ModelAndView(String viewName)`
Convenient constructor when there is no model data to expose.

`ModelAndView(String viewName, Map<String,?> model)`
Creates new ModelAndView given a view name and a model.

`ModelAndView(String viewName, String modelName, Object modelObject)`
Convenient constructor to take a single model object.

`ModelAndView(View view)`
Convenient constructor when there is no model data to expose.

`ModelAndView(View view, Map<String,?> model)`
Creates new ModelAndView given a View object and a model.

`ModelAndView(View view, String modelName, Object modelObject)`
Convenient constructor to take a single model object.

Method Summary

`ModelAndView addAllObjects(Map<String,?> modelMap)`
Add all attributes contained in the provided Map to the model.

`ModelAndView addObject(Object attributeValue)`
Add an attribute to the model using parameter name generation.

`ModelAndView addObject(String attributeName, Object attributeValue)`
Add an attribute to the model.

FIGURE 35-3

Add the project Lesson35 to GlassFish in the Eclipse Servers view and start the server. Direct your web browser to `http://localhost:8080/Lesson35/hello.html` and you'll see something like what's shown in Figure 35-4. Just to recap, because of the `*.html` mapping in `web.xml`, Spring's `ServletDispatcher` was invoked. The `.html` part was stripped off, the word `hello` was identified, and the `HelloWorldController` was instantiated and its method `sayHello()` invoked. The view resolver found `hello.jsp`, and its variable `greetings` was replaced with the text "Hello People" as per the content of the `ModelAndView` instance.



FIGURE 35-4

Processing HTML Forms

Processing HTML forms with Spring is just a little bit more complex than the "Hello People" example. The Spring Framework comes with integrated tag libraries with a tag corresponding to each HTML element. In particular, there is the `form` tag library for processing HTML forms. Listing 35-5 shows `ContactUs.jsp`, which includes the appropriate `<%taglib>` directive.



LISTING 35-5: ContactUs.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
       <%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<!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=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h2>Contact Manager</h2>

<form:form method="post" action="addQuestion.html">

      <table>
      <tr>
            <td><form:label path="author">Who are you?</form:label></td>
            <td><form:input path="author" /></td>
      </tr>
      <tr>
            <td><form:label path="subject">Subject</form:label></td>
            <td><form:input path="subject" /></td>
      </tr>
      <tr>
```

```
            <td><form:label path="message">You Question</form:label></td>
            <td><form:textarea path="message" /></td>
      </tr>
      <tr>
            <td colspan="2">
                  <input type="submit" value="Submit your question"/>
            </td>
      </tr>
</table>

</form:form>
</body>
</html>
```

When the user clicks Submit, the request is posted to the server with `addQuestion.html` in the URL. Then `DispatcherServlet` maps `addQuestion` with the method `addUserQuestion()` in the controller shown in Listing 35-6.



LISTING 35-6: ContactUsController.java

```
package com.practicaljava.lesson35.controller;

import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.SessionAttributes;
import org.springframework.web.servlet.ModelAndView;

import com.practicaljava.lesson35.model.ContactUs;

@Controller
@SessionAttributes
public class ContactUsController {

    @RequestMapping(value="/addQuestion", method=RequestMethod.POST)
    public String addUserQuestion(@ModelAttribute("contactUs") ContactUs
                                        contactUs, BindingResult result){

       System.out.println("New message:\nSubject:"
                                            +contactUs.getSubject());
       System.out.println("Author " + contactUs.getAuthor());
       System.out.println("Message "+ contactUs.getMessage());

       return "redirect:contactUs.html";
    }

    @RequestMapping("/contactUs")
    public ModelAndView contactUs(){
        return new ModelAndView("contactUs", "command", new ContactUs());
    }
}
```

`@ModelAttribute` binds the method parameter to the named model attribute (see the model class in Listing 35-7). The Spring Framework maps the attributes from the model to corresponding attributes in `ContactUs.jsp`, shown in Listing 35-5.



LISTING 35-7: ContactUs.java

```
public class ContactUs {
    private String subject;
    private String message;
    private String author;

    public String getSubject() {
        return subject;
    }
    public void setSubject(String subject) {
        this.subject = subject;
    }
   public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
}
```

Open your web browser and enter the following URL: `http://localhost:8080/Lesson35/contactUs.html`. The `@RequestMapping("/contactUs")` annotation in Listing 35-6 responds to the client with `ContactUs.jsp`, which looks as in Figure 35-5.

Enter some values in the form and click Submit. This time the `/addQuestion` action will be matched with the method `addUserQuestion()` of `ContactUsController`. Received values from the client will be printed in the log file of GlassFish Server, located in `logs/server.log` under your domain directory.

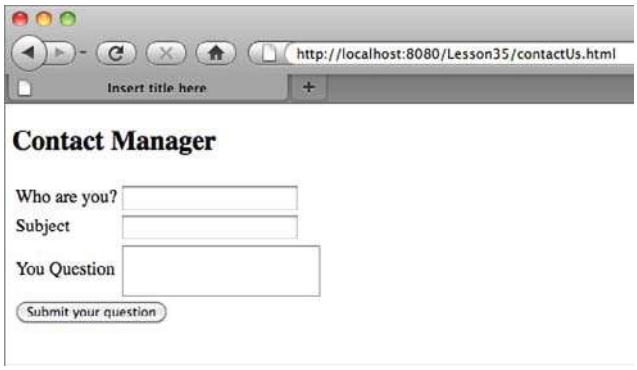


FIGURE 35-5

TRY IT

Modify the code of `hello.jsp` to accept more than one variable. Change the code of `ModelAndView` so that it carries a `Map` of values.

Lesson Requirements

You should have Java and GlassFish v3 installed.



You can download the code and resources for this Try It from the book's web page at `www.wrox.com`*. You can find them in the Lesson35 folder in the download.*

Step-by-Step

1. Modify `hello.jsp` to add one more variable in addition to `$greetings`:
2. Modify `HelloWorldController` to create a `Map` with two Strings:

```
Map helloModel = new HashMap();
helloModel.put("Hello Mary");
helloModel.put("My name is John");
```

3. Use the two-argument constructor of `ModelAndView` and pass the `Map` as the second argument.
4. Direct your web browser to `http://localhost:8080/Lesson35/hello.html` and observe that `hello.jsp` shows the values from the `Map`.



Please select Lesson 35 on the DVD with the print book, or watch online at `www.wrox.com/go/fainjava` *to view the video that accompanies this lesson.*

36

Introduction to Hibernate Framework

In Lesson 33 you got familiar with the Java Persistence API, which is part of Java EE 6. But JPA wasn't created from scratch. It took many ideas from Hibernate — an open-source object-relational framework started by Gavin King in 2001. Hibernate version 3.5 and above support the JPA 2.0 specification.

The official website of Hibernate is `www.hibernate.org`, where you can find documentation, code samples, and tools. Plenty of books and online articles are published about this popular framework. I prefer using SQL and JDBC for all data-related processing, but these days knowledge of Hibernate is required in many job postings. That's why I decided to include this brief overview of Hibernate even though it's not a part of the Java specification but a third-party framework.

If you are going to use Hibernate for work, consider using the Eclipse plug-ins available at `www.hibernate.org/subprojects/tools.html`.

THE BIG PICTURE

The main idea behind any ORM framework is to replace writing SQL with manipulating the objects. In the Java world, this means putting a layer on top of JDBC that will enable you to simply create a POJO, assign values to its fields, and tell it "Persist yourself in the database." This is what Hibernate is for.

Proponents of this approach argue that removing the dependency on vendor-specific SQL syntax provides easy switching from one DBMS to another without the need to modify the Java code responsible for querying and persisting data. To me this doesn't sound like a compelling reason to move away from SQL, because it's very seldom that you need to switch from one

DBMS to another in real-world projects. Besides, if you'll be working on projects that require swift query execution, the ability to tune SQL manually and use tricks specific to DBMSes can be helpful.

On the other hand, Hibernate offers such goodies as *lazy loading* and the ability to easily configure the in-memory cache to greatly minimize disk access, which is a lot slower than memory access.

Lazy loading is easy to explain with an example. Suppose the user is presented with a list of customers, each of which may have multiple orders. With lazy loading only the list of customers is initially loaded, and only when the user selects a customer will the query to retrieve the corresponding orders be sent to data storage.

The process of retrieving data with Hibernate is very similar to what you saw in Lesson 33. You create the mapping of entities to database tables, then instantiate the `Configuration` object needed to create a `SessionFactory` and `Session`. After this is done you execute queries on these entities using Hibernate Query Language (HQL), Criteria queries, or native SQL.

The rest of this lesson is a step-by-step walkthrough that demonstrates how to write a simple application that will get the data from a DBMS using Hibernate.

INSTALLING AND CONFIGURING HIBERNATE

In this section you'll download required jars and create the configuration file `hibernate.cfg.xml`. I'll continue using Derby, which is included with GlassFish, as our sample DBMS, but the application server GlassFish is not needed here. The idea is to look at Hibernate as an alternative persistence solution that doesn't require the Java EE application server.

Hibernate is an open-source framework hosted on the web at SourceForge (`http://sourceforge.net/projects/hibernate/files/hibernate3/`). At the time of this writing, the latest release of Hibernate is 3.6; you need to download the file `hibernate-distribution-3.6.0.Final-dist.zip`. This distribution includes required jar files, a well-written reference manual, and a folder with libraries that might be used to support memory caching.

Adding Hibernate Jars to an Eclipse Project

Now create a new Eclipse project called Lesson36 by selecting File ➪ New ➪ Other ➪ Java Project. Note that this is a Java SE project — I'm not going to use any Java EE application server here. Make sure your Eclipse Java project includes `hibernate3.jar`, all the jars from the `required` folder, and the jar from the `jpa` folder. There is one more logging-related file, `slf4j-1.6.1.zip`, that for some licensing reason has to be downloaded from `www.slf4j.org/download.html`. You need to add to the build path just one file from there: `slf4j-simple-1.6.1.jar`.

You can add these external jars in the tab Libraries of the Java Setting window while you're creating the project, or you can add them after the project is created via Properties ➪ Java Build Path ➪ Libraries ➪ Add External Jars. Figure 36-1 shows which libraries are included in the build path of my Eclipse project.

Page 448 of 506 (IPR2025-00350)
Stingray Group Inc. et al v. Hernandez-Mondragon

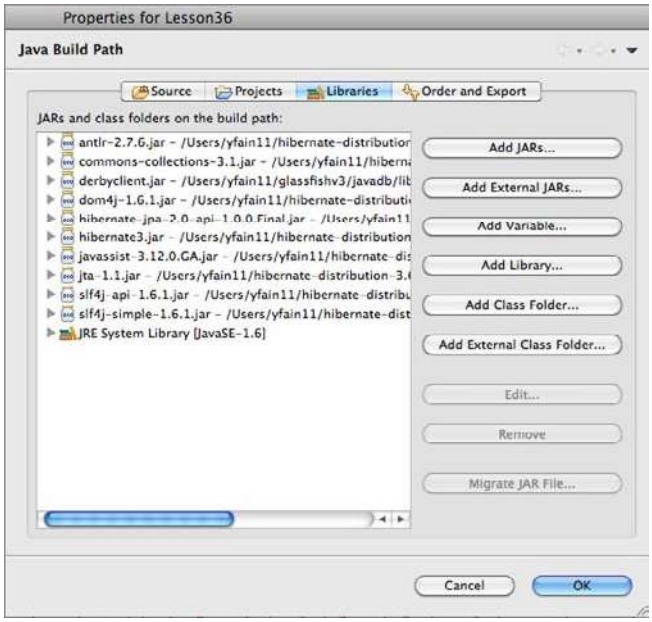


FIGURE 36-1

Testing the Database Connection

In this lesson I'll keep working with the database that was initially created in Lesson 22 and then reused in Lesson 33. It contains a table called Employee and the connection is already configured (see Figure 33-4). This connection uses the Derby JDBC driver located in `derbyclient.jar`. Add this jar, located in `glassfish/javadb/lib`, to the build path of the new Eclipse project.

Now right-click the database connection Lesson22 in Eclipse and select Properties ➪ JDBC Connection Properties. Keep the user ID as `user` and change the password to `password123`. Now copy the connection URL (refer to Figure 33-3) to a text editor — you'll need the connection parameters in the next section. You can also check the properties of the selected DerbyDB driver by clicking the small button with an asterisk on it — copy the name of the JDBC driver class too.

Configuring Hibernate with a DBMS

Now it's time to add the DBMS-related entries to the file `hibernate.cfg.xml`. You don't need to create this file from scratch; copy to the `src` folder of your Eclipse project one of the files that come with Hibernate, such as the one located in the folder `documentation/quickstart/tutorials/basic/src/test/resources`. Modify the driver class and connection URL with the values from the previous section. Make sure that this file doesn't have a setting to drop and re-create the database schema on startup. You won't need any mapping files because our entity will use annotations.

Every DBMS implements standard SQL plus some additional features meant to give it a competitive edge. To deal with these extras Hibernate includes configurable `dialects` that hide the differences

from the application developer. That's why the configuration file has to specify the dialect that the current application has to support. Each dialect is a class located in the package `org.hibernate.dialect` inside `hibernate3.jar`, and you can find the name of the dialect that matches our database, which is `DerbyDialect`.

You might remember that the table Employee was originally created in the database schema APP. Because I specified that the connection ID is `user`, by default the queries will try to find the referenced database objects in the schema USER, which doesn't exist. To fix this situation let's specify in the configuration file that the default schema is APP.

Finally, we want Hibernate to print in the Eclipse console the underlying SQL statements that it'll use in communication with the database.

After you make all these changes your `hibernate.cfg.xml` file should look like Listing 36-1.



LISTING 36-1: Hibernate.cfg.xml

```
<?xml version='1.0' ?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <!-- Database connection settings -->
        <property name="connection.driver_class">
            org.apache.derby.jdbc.ClientDriver
        </property>

        <property name="connection.url">
            jdbc:derby://localhost:1527/Lesson22;create=true
        </property>

        <property name="connection.username">user</property>
        <property name="connection.password">password123</property>

        <!-- Set the default schema to be APP, where table Employee was
                                                              created-->
        <property name="hibernate.default_schema">APP</property>

        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>

        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.DerbyDialect</property>

        <!--  Enable Hibernate's current session context -->
        <property name="current_session_context_class">thread</property>

        <!-- Disable the second-level cache  -->
        <property name="cache.provider_class">
```

```
                org.hibernate.cache.NoCacheProvider
        </property>

        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>

    </session-factory>

</hibernate-configuration>
```

RETRIEVING DATA WITH HIBERNATE

Now we need to create a Java `Employee` entity, which in this simple example corresponds to the database table Employee. The good news is that because Hibernate supports JPA 2, we can simply reuse the `Employee` entity from Lesson 33. Listing 36-2 shows this entity after we copy it into the package `com.practicaljava.lesson36`.



LISTING 36-3: Employee entity

```
package com.practicaljava.lesson36;

import java.io.Serializable;
import javax.persistence.*;

@Entity
public class Employee implements Serializable {

  private static final long serialVersionUID = 1L;

  @Id
  private int empno;

  private String ename;

  @Column(name="JOB_TITLE")
  private String jobTitle;

  public Employee() {}

  public int getEmpno() {
    return this.empno;
  }

  public void setEmpno(int empno) {
    this.empno = empno;
  }

  public String getEname() {
   return this.ename;
```

continues

LISTING 36-2 *(continued)*

```
    }

    public void setEname(String ename) {
     this.ename = ename;
    }

    public String getJobTitle() {
     return this.jobTitle;
    }

    public void setJobTitle(String jobTitle) {
     this.jobTitle = jobTitle;
    }
}
```

The code to retrieve employees comes next. The program `HRManager` shown in Listing 36-3 will do this job. It starts by loading the configuration file `hibernate.cfg.xml` and creating a Hibernate session. We didn't provide any arguments to the method `configure()` — by default this method assumes that the configuration file is called `hibernate.cfg.xml`.

Then `SessionFactory` creates an instance of the `Session`. The transaction is opened, and `createQuery("from Employee")` means that we're interested in `Employee` entities. In this sample, the name of the entity and the name of the database table are the same. If this weren't the case we'd have had to specify the table name in the `@Entity` annotation in Listing 36-2.



LISTING 36-3: Retrieving Employee entities

```
import java.util.List;
import org.hibernate.*;
import org.hibernate.cfg.Configuration;
import com.practicaljava.lesson36.Employee;

public class HRManager {

public static void main(String[] args){

  Configuration config = new Configuration().configure();
  config.addAnnotatedClass(Employee.class);

  SessionFactory sf = config.buildSessionFactory();
  Session session = sf.getCurrentSession();

  session.beginTransaction();

  List<Employee> employee = session.createQuery("from Employee").list();

  session.getTransaction().commit();

  for (Employee emp: employee){
```

```
            System.out.println("Got " + emp.getEmpno() + ", "
                               + emp.getEname() + ", "+ emp.getJobTitle());
        }
       }
      }
```

The method `addAnnotatedClass(Employee.class)` tells Hibernate that there is an entity called `Employee` configured using annotations. After running, `HRManager` prints the following in Eclipse's Console view:

```
Hibernate: select employee0_.empno as empno0_, employee0_.ename as ename0_,
 employee0_.JOB_TITLE as JOB3_0_ from APP.Employee employee0_

Got 7369, John Smith, Clerk
Got 7499, Joe Allen, Salesman
Got 7521, Mary Lou, Director
```

Retrieving data with Hibernate is pretty trivial. Persisting the data in the database is not rocket science either. You'll have a chance to see such persisting for yourself when you work on the assignment in the "Try It" section.

TRY IT

This lesson shows you how Hibernate creates database objects from Java entities. You will have to declare the Java entity `Address` and have Hibernate create and populate the corresponding table in the database.

Lesson Requirements

You should have Java and GlassFish v3 installed.



You can download the code and resources for this Try It from the book's web page at `www.wrox.com`*. You can find them in the Lesson36 folder in the download.*

Hints

In the toolbar of the Data Source Explorer view is an icon that opens SQL Scrapbook. It enables you to select the DBMS and the database and to execute SQL statements. For example, enter **`Select * from APP.Employee`**, then right-click and select Execute All. The SQL statement will be executed and you'll see the content of the table Employee.

Step-by-Step

1. Create the following Java entity, `Address`:

```
package com.practicaljava.lesson36;

import javax.persistence.*;

@Entity
public class Address {

   private long addressID;
   private String streetAddress;
   private String city;
   private String zip;

   @Id
   public long getAddressID() {
      return addressID;
   }

   public void setAddressID(long addressID) {
      this.addressID = addressID;
   }

   public String getStreetAddress() {
      return streetAddress;
   }

   public void setStreetAddress(String streetAddress) {
      this.streetAddress = streetAddress;
   }

   public String getCity() {
      return city;
   }

   public void setCity(String city) {
      this.city = city;
   }

   public String getZip() {
      return zip;
   }

   public void setZip(String zip) {
      this.zip = zip;
   }
}
```

2. Create a class called `AddressCreator`. It should look like this:

```
import java.util.List;

import org.hibernate.*;
```

```
import org.hibernate.cfg.Configuration;
import org.hibernate.tool.hbm2ddl.SchemaExport;

import com.practicaljava.lesson36.Address;

public class AddressCreator {

      public static void main(String[] args){
         Configuration config = new Configuration().configure();
         config.addAnnotatedClass(Address.class);

         //Create the Address table
         new SchemaExport(config).create(true,true);

         SessionFactory sf = config.buildSessionFactory();
         Session session = sf.getCurrentSession();

         session.beginTransaction();

         // Instantiate and Populate the entity Address
         Address addr1 = new Address();
         addr1.setStreetAddress("123 Main St.");
         addr1.setAddressID(1);
         addr1.setCity("New York");
         addr1.setZip("10001");

       session.save(addr1);

       // Save the new entity in the database
         session.getTransaction().commit();

      }
}
```

3. Run the program AddressCreator. Observe the output in the Eclipse console. You should see the following SQL statements generated by Hibernate:

```
drop table APP.Address
create table APP.Address (addressID bigint not null, city varchar(255),
 streetAddress varchar(255),
zip varchar(255), primary key (addressID))

Hibernate: insert into APP.Address (city, streetAddress, zip, addressID)
values (?, ?, ?, ?)
```

4. Go to the Data Source Explorer view, reconnect to the Lesson22 database, and observe the table Address in the schema APP. Right-click the Address table and select Data ➪ Sample Contents — you'll see one row with the New York address specified in `AddressCreator`.



Please select Lesson 36 on the DVD with the print book, or watch online at `www.wrox.com/go/fainjava` *to view the video that accompanies this lesson.*

37

Bringing JavaFX to the Mix

JavaFX was designed to serve as a tool for creating rich Internet applications (RIAs). It is meant to compete with such RIA tools as Adobe Flex and Microsoft Silverlight. The competing technologies use two languages for creating RIAs — one for declarative definition of the UI components, and another for the coding of processing logic. Adobe Flex uses MXML for UI components and ActionScript for processing logic. Microsoft Silverlight uses XAML for UI components and one of the .NET languages (such as C#) for processing.

JavaFX was originally designed to use JavaFX Script for UI components and Java for processing logic. But recently things changed, and up to the last minute I wasn't sure if this lesson should be included in the book. Oracle has announced that JavaFX will be undergoing a major redesign in 2011. JavaFX 1.3 includes the JavaFX Script language, but JavaFX 2.0 will be completely Java-based.

Currently, no pre-release version of JavaFX 2.0 is available, and there is a chance that whatever you read about the client portion of this lesson's application will be outdated. But no matter what changes in JavaFX 2.0, the deployed application will run in JVM with the installed JavaFX run time. There is also hope that the Java community will continue supporting the open-source version of JavaFX Script.

This lesson is not an introduction to JavaFX programming, but rather an overview of an application that uses multiple technologies you are already familiar with, plus a client created with JavaFX 1.3.

CONSUMING STOCK QUOTES WITH JAVAFX

In this lesson you'll see how a web client written in JavaFX can subscribe to price quotes published from the server side via messaging. Multiple technologies used in this version of the Stock Server example are depicted in Figure 37-1.

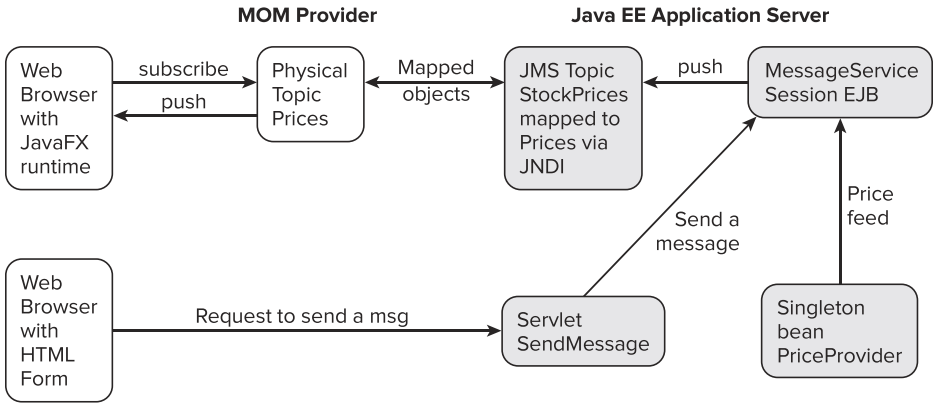


FIGURE 37-1

These are the players that provide the generation and processing of the price quote feed:

1. The singleton session bean `PriceProvider` (see the source code in Listing 37-1) knows how to generate random price quotes. It gets instantiated on startup of the GlassFish application server.
2. A stateless session bean called `MessageService` (Listing 37-2) gets the references to JMS-administered objects via resource injection and publishes the price quote to the JMS topic known as `jms/StockPrices`, which is mapped to the physical topic `Prices` in the Open MQ messaging server.
3. The Java client subscribes to the topic `Prices`. Listing 37-3 shows the Java classes `Messages` and `JMSListener`, and the interface `MessageHandler`. These client classes do not have any presentation logic.
4. The presentation client `Main.fx` (Listing 37-4) is written in JavaFX Script. It wraps the Java client messaging classes and gets the price quotes from them. As soon as the new quote comes in, the old one is pushed up and off the screen with a special animation effect.
5. To illustrate another way of sending the message, this application has a simple HTTP client (Listing 37-5) that allows the user to enter any text, which is then routed via the servlet `SendMessage` (Listing 37-6) to the `MessageService` EJB. The rest of the data flow is as described in Steps 2 through 4.

CODE WALKTHROUGH

By now you should be familiar with each of the Java technologies used in this application, except JavaFX, so I'll provide only very short explanations for the code listings.

The `PriceProvider` singleton bean shown in Listing 37-1 serves as an emulator of the data feed with stock quotes. Replace the random-number generator with a real-time Wall Street stock market feed, and this sample application will show the real data. I'd like to point out two

Page 458 of 506 (IPR2025-00350)
Stingray Group Inc. et al v. Hernandez-Mondragon

annotations: `@Startup` and `@Schedule`. The first ensures that the Java EE application server automatically instantiates the `PriceProvider` bean whenever the server starts. The second is a calendar-based timer that will call the method `getPriceQuote()` every five seconds.



LISTING 37-1: PriceProvider singleton bean

```
@Singleton
@Startup
public class PriceProvider {

    @EJB
    private MessageService messageService;

    // In this example the method getQuotes() simply generates
    // prices, but in the real world it will get the quote
    // from some external data feed

    @Schedule(second = "*/5", minute="*", hour="*")
    public void getPriceQuote() {

        String price = Double.toString(generator.nextDouble() * 100.0);
        String symbol = symbols[generator.nextInt(symbols.length)];

        messageService.sendMessage(symbol + " " + price);
    }

    private static final Random generator = new Random();

    private static final String[] symbols =
                    {"AAPL", "MSFT", "YHOO", "AMZN", "MOT"};
}
```

The stateless session bean `MessageService` publishes the text message given as an argument to `sendMessage()` to the JMS topic that has been bound to the JNDI tree of GlassFish under the name `jms/StockPrices`. Both the topic connection factory and the topic objects are injected into the bean by the EJB container. The JMS topic will be mapped to the physical topic `Prices` of the MOM provider Open MQ.



LISTING 37-2: Stateless bean MessageService

```
@Stateless
public class MessageService {

    @Resource(mappedName = "jms/StockPrices")
    private Topic topic;

    @Resource(mappedName = "TopicConnectionFactory")
    private TopicConnectionFactory topicConnectionFactory;

    public void sendMessage(String text) {
```

continues

LISTING 37-2 *(continued)*

```
            try {
                Connection connection = topicConnectionFactory.createConnection();
                Session session = connection.createSession(true, 0);
                MessageProducer producer = session.createProducer(topic);

                TextMessage textMessage = session.createTextMessage(text);
                producer.send(textMessage);

                connection.close();
            }
            catch (JMSException e) {
                throw new RuntimeException(e);
            }
        }
    }
```

The web clients of this application are written in Java, JavaFX Script, and HTML. The Java class `Messages` is a part of the JavaFX applet that directly subscribes to the Open MQ topic `Prices`. Originally, I wanted the applet to use a JNDI call to GlassFish to get the topic `jms/StockPrices` from there. This would be a cleaner way of subscribing to the topic — no dependency on the particular MOM provider, and no client code changes needed should we decide to change the MOM provider or the application server.

Unfortunately GlassFish v3 doesn't offer a lightweight means of adding JNDI-related classes to the applet. It comes with the utility `appclient`, which simply packages 40 MB worth of Java classes into the one jar. Sending such a heavy extra weight over the network with the applet is not right, and that's why I decided to use a lighter solution: the applet's class `Messages` (Listing 37-3) doesn't perform JNDI lookup but directly connects to Open MQ. The Java class `Messages` uses an inner class called `JMSListener` and the interface `MessageHandler`.



LISTING 37-3: The Java client subscriber

```
public class Messages {

    public static void startListening(MessageHandler messageHandler)
                                                          throws Exception {
       JMSListener jmsListener = new JMSListener(messageHandler);

        ConnectionFactory connectionFactory = new ConnectionFactory();
        connectionFactory.setProperty(ConnectionConfiguration.imqAddressList,
                                                   "mq://localhost:7676");

        connection = connectionFactory.createQueueConnection("admin","admin");

        Session session = connection.createSession(false,
                                                 Session.AUTO_ACKNOWLEDGE);

        Topic topic = session.createTopic("Prices");
```

```
        MessageConsumer consumer = session.createConsumer(topic);

        consumer.setMessageListener(jmsListener);
        connection.start();
    }

    public static void stopListening() {
        try {
            connection.stop();
        }
        catch (JMSException e) {
            throw new RuntimeException(e);
        }
    }

    private static Connection connection;

    private static class JMSListener implements MessageListener {
        private MessageHandler messageHandler;

        private JMSListener(MessageHandler messageHandler) {
            this.messageHandler = messageHandler;
        }

        @Override
        public void onMessage(Message message) {

            TextMessage textMessage = (TextMessage) message;

            try {
                System.out.println("Got message: " + textMessage.getText());

                messageHandler.handle(textMessage.getText());
            }
            catch (JMSException e) {
                throw new RuntimeException(e);
            }
        }
    }
}

public interface MessageHandler {
    void handle(String message);
}
```

Reading JavaFX Code

The JavaFX client shown in Listing 37-4 is responsible for the UI. It's deployed as an applet and is played in the client by the Java plug-in. Starting from Java 6 Update 10, this plug-in runs not as a part of the web browser, but as an independent JVM.

This applet is deployed on the server and is described in a special text file with the extension `.jnlp` (see Listing 37-7). The client's machine has to have the JavaFX run time installed to play this applet. Because you have the latest Java SE installed on your computer, you won't have any problem playing JavaFX applets. But users with older versions of Java are not guaranteed to have the same results. This is one of the reasons JavaFX has difficulties competing with Adobe Flex or Microsoft Silverlight in a non-corporate uncontrolled environment.

JavaFX Script can access the Java API and in this sample the JavaFX code communicates with the preceding Java message listener. JavaFX Script files are compiled into Java classes and then run on JVM. JavaFX developers need to have JavaFX SDK installed in addition to the run time.

The code in Listing 37-4 shows you the syntax of JavaFX Script. Variable declarations start with the keyword `var`, and specifying the data type is optional — the compiler will try to figure out what type to use based on assigned values.

The keyword `bind` represents binding — the value of the variable on the right is assigned to the variable on the left, and all the changes to the right-side values will be immediately reflected in the variable on the left.

Functions in JavaFX are similar to methods in Java, but they don't have to be defined inside a class and can just appear anywhere in the script. Functions can be defined dynamically, which makes them much like Java anonymous inner classes, but simpler. To call a function asynchronously, much as you would call Java's `invokeLater()`, just give this function `FX.deferAction()` as an argument, where `FX` is a class with system utilities. This is done at the beginning of Listing 37-4.

The instance of `JavaFXMessageHandler` is given to the class `Messages` (Listing 37-3), and every time the new message is published to the topic the `addNewText()` method is invoked.



LISTING 37-4: The JavaFX client Main.fx

```
class JavaFXMessageHandler extends MessageHandler {
    override public function handle(message:String):Void {
        FX.deferAction( function() { addNewText(message) } );
    }
}

Messages.startListening(new JavaFXMessageHandler());

var oldText = "";
var newText = "";

var oldTextOpacity = 0.0;
var oldTextYShift = -40.0;

var font = Font {
    name: "Verdana"
    size: 16
}

var oldTextLabel:Label = Label {
```

```
    font: font
    text: bind oldText
    textFill: Color.web("#808020")
    translateY: bind oldTextYShift
    opacity: bind oldTextOpacity
}

var newTextOpacity = 0.0;
var newTextYShift = 40.0;

var newTextLabel:Label = Label {
    textAlignment: TextAlignment.CENTER
    font: font
    textFill: Color.web("#808020")
    text: bind newText
    translateY: bind newTextYShift
    opacity: bind newTextOpacity;
}

var oldTextAnimation = Timeline {
    repeatCount: 1
    keyFrames: [
        at (0s) {
            oldTextOpacity => 1.0 tween Interpolator.LINEAR;
            oldTextYShift => 0.0 tween Interpolator.LINEAR;
        }

        at (1s) {
            oldTextOpacity => 0.0 tween Interpolator.LINEAR;
            oldTextYShift => -40.0 tween Interpolator.LINEAR;
        }
    ];
}

var newTextAnimation = Timeline {
    repeatCount: 1
    keyFrames: [
        at (0s) {
            newTextOpacity => 0.0 tween Interpolator.LINEAR;
            newTextYShift => 40.0 tween Interpolator.LINEAR;
        }

        at (.5s) {
            newTextOpacity => 1.0 tween Interpolator.LINEAR;
            newTextYShift => 0.0 tween Interpolator.LINEAR;
        }
    ];
}

function addNewText(text: String): Void {
    oldText = newText;
    newText = text;

    oldTextAnimation.playFromStart();
```

continues

LISTING 37-4 *(continued)*

```
        newTextAnimation.playFromStart();
    }


    Stage {
        title: "Application title"

        scene: Scene {
            width: 600
            height: 100

            content: VBox {
                content: [
                    Stack {
                        nodeHPos: HPos.CENTER
                        content: [
                            Rectangle {
                                width: 600
                                height: 80
                                opacity: 0.3
                                fill: LinearGradient {
                                    startX: 0, startY: 0.0, endX: 0, endY: 80.0
                                    proportional: false
                                    stops: [
                                        Stop { offset: 0.0 color:
                                                       Color.web("#303030") }
                                        Stop { offset: 1.0 color:
                                                       Color.web("#FFFFFF") }
                                    ]
                                }
                            },
                            oldTextLabel,
                            newTextLabel ]
                    }]
            }
        }

        onClose : function() { Messages.stopListening(); }
    }
```

The price quotes are displayed on a gray gradient background (see Figure 37-2) and each new price quote pushes the previous one from the screen. Both old and new message texts have animation effects (`oldTextAnimation` and `newTextAnimation`) that make the view more engaging.

The JavaFX UI components are not created to live inside a specific top-level container. You add a group of components to a *scene*, which is added to the *stage*. At run time the stage will be linked to a top-level container according to the platform on which the application has to work. A mobile device and a desktop computer will use different top-level containers to display your stage. Each of the components that you add to the scene is an instance of a `Node` class.

The content of a `Scene` is listed in its `content` property. The code in Listing 37-4 creates the `Stage` object and adds an instance of a `Scene`, which contains a `VBox` — a container with a vertical layout. Inside the `VBox` are three nodes: `Rectangle` with `LinearGradient` (a color effect changing from dark to light gray) and two `Label` nodes: `oldTextLabel` and `newTextLabel`.

Another interesting JavaFX component is `Timeline` — think of a reel of film in a projector. Spooling through frames rapidly produces the illusion of movement. If you need to repeat some actions in JavaFX you can create a `Timeline` with `keyFrames` that will occur at specific intervals.

The last line in the code of the `Stage` instance handles the `onClose` event. This event instructs the `Messages` class to stop listening to the price quotes.



FIGURE 37-2

The HTML Client

To provide another door to the messaging system you can use an HTML client that can send any text you want. This may be useful for managing the application. For example, an administrator may want to arrange it so that as soon as the client application receives the word `STOP` as a text message, it will stop listening to price quotes.

The simple HTML form in Listing 37-5 has just one input field, and when the user presses the Enter key the text is posted to the server by the action `send`.



LISTING 37-5: Index.html — the HTML client

```
<html>
<head>
    <title>JMS Demo</title>
</head>
<body>
    Click this link to open <a href="prices.html">
                    JavaFX client consuming price quotes</a>
    <p>Enter any text in the input field below and watch how JavaFx
client will receive it</p>
    <form action="send" method="POST">
        <label title="Text to send">
            <input value="" name="text"/>
        </label>
    </form>
</body>
</html>
```

The Java EE application server will route all HTTP requests with the URL pattern `send` to the servlet `SendMessage` from Listing 37-6, which will get the text entered by the user and invoke `sendMessage()` on the session bean `MessageService`. This text will be published to the topic `jms/StockPrices` as will any other messages that originate from the singleton `PriceProvider`.



LISTING 37-6: SendMessage servlet

```
@WebServlet("/send")
public class SendMessage extends HttpServlet {

 @EJB
 private MessageService messageService;

 protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws IOException, ServletException {
  String text = request.getParameter("text");
  messageService.sendMessage(text);

  response.sendRedirect("index.html");
 }
}
```

When this JavaFX applet is ready for deployment on the server, you need to add resources that are needed on the user's machine. Our `Main.fx` client from Listing 37-4 uses the JMS API to subscribe to the messages from the Open MQ MOM provider. Therefore the classes implementing the JMS API have to be downloaded to the user's computer along with `Main.fx`.

The JNLP file shown in Listing 37-7 has to be deployed on the server. It includes the `resources` section that lists what has to be downloaded to the client: the run time for JavaFX and the `javafx-ui.jar` file that includes our sample code and all related JMS supporting classes. The `<applet-desc>` tag includes the `main-class` parameter with a fully qualified name of the JavaFX applet itself with the name of the main class.



LISTING 37-7: JNLP deployment file

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.0+">
    <information>
        <title>jms-send-receive</title>
        <vendor>practicaljava</vendor>
        <homepage href=""/>
        <description>jms send and receive</description>
        <offline-allowed/>
    </information>
    <resources>
        <j2se version="1.5+"/>
        <extension name="JavaFX Runtime"
 href="http://dl.javafx.com/1.3.1/javafx-rt.jnlp"/>

        <jar href="javafx-ui.jar" main="true"/>
```

```
    </resources>
    <applet-desc name="javafx-ui"
    main-class="com.sun.javafx.runtime.adapter.Applet"
width="200" height="80">
        <param name="MainJavaFXScript" value="com.practicaljava.lesson37.ui.Main"/>
    </applet-desc>
    <update check="always"/>
</jnlp>
```

The hyperlink in the main HTML page (Listing 37-5) downloads the HTML file `Prices.html` (Listing 37-8), which includes a special JavaScript function called `javafx` (its source code is available at `http://dl.javafx.com/1.3/dtfx.js`). This function is a part of the JavaFX deployment toolkit. It ensures that the client's computer has the right version of JRE and automatically generates the reference to the JNLP file shown in Listing 37-7.



LISTING 37-8: Prices.html

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
    <head>
        <title>Prices</title>
    </head>
    <body>
        <p>Real time price updates</p>

        <script src="http://dl.javafx.com/1.3/dtfx.js"
type="text/javascript" language="JavaScript"></script>
        <script type="text/javascript" language="JavaScript">
            javafx(
            {
                archive: "javafx/javafx-ui.jar",
                draggable: true,
                width: 600,
                height: 80,
                code: "com.practicaljava.lesson37.ui.Main",
                name: "Messages"
            }
        );
       A </script>
    </body>
</html>
```

TRY IT

This is the final hands-on exercise in this tutorial. The goal is to build the application and make it work on your computer. This time you'll be building and deploying the application and starting the servers from the command window without any help from Eclipse IDE.

Lesson Requirements

You should have Java and GlassFish v3 installed. You'll need to install Mercurial version control software to get the code for this example from the Google source control repository where all code samples used in this book are stored. You'll also need to install the build tool Maven.



You can download the code and resources for this Try It from the book's web page at `www.wrox.com`. *You can find them in the Lesson37 folder in the download.*

Hint

After this application is deployed from the command line, download the JavaFX plug-in for Eclipse IDE from `http://download.oracle.com/javafx/1.2/gettingstarted/eclipse-plugin` and try to create, deploy, and test this project inside Eclipse.

Step-by-Step

1. Download and install the distributed source control management tool Mercurial from `http://mercurial.selenic.com`. In the Terminal window enter **`hg -version`** and you should see the version of Mercurial installed.
2. Check out the code samples for the book into a folder called `java24hourtrainer` from `http://code.google.com/p/practicaljava`. You can do this by entering the following command in the Terminal (command) window:

   ```
   hg clone https://practicaljava.googlecode.com/hg/ java24hourtrainier
   ```

3. Download and install JavaFX 1.3 SDK, available at `http://javafx.com/downloads/all.jsp?bundle=javafx_sdk`. In Windows, JavaFX SDK will be installed in the default location `c:\Program Files\JavaFX`, and in Mac OS in `/Library/Frameworks/JavaFX.framework`.
4. Check to see if you have the Maven build tool already installed on your computer by entering **`mvn -v`** at the command prompt. If you don't have it, download and install it from `http://maven.apache.org/download.html`. All you have to do is unzip the downloaded file into a directory and add Maven's `bin` directory to the system `PATH` variable. In Windows you do this via Control Panel, and on Mac OS by entering the following command in the Terminal window (enter the exact name of your maven directory):

   ```
   export PATH=~/development/apache-maven-3.0/bin/:$PATH
   ```

   You'll be using the Maven build scripts that were created specifically for this lesson. Maven is out of the scope of this tutorial, but if you are curious to see where all the build logic is located, open up the three files called `pom.xml` in subdirectories `lesson37`, *`webdemo`*, and `javafx-ui` under `java24hourtrainer`.

5. To build the project for this lesson, in the Terminal or command window go to `java24hourtrainer/java-ee/lesson37` and run `mvn install`.

6. As a part of Maven's install, it'll create a directory called.*m2*. Create a plain text file in this directory, named `settings.xml`, with the following content:

```
<settings>
  <pluginGroups>
     <pluginGroup>com.practicaljava.maven.plugin</pluginGroup>
  </pluginGroups>
</settings>
```

7. Change to the directory `java24hourtrainer/maven-plugins` and run `mvn install`.

   In several seconds Maven will have built two plug-ins — one for GlassFish and another for JavaFX.

8. Now go to the `/java-ee/lesson37/web-demo` directory and run the script to create a new domain in GlassFish; start it and deploy the web application there. In this example you are not going to run the instance of GlassFish that you've been using in previous lessons.

```
mvn gf:create-domain
mvn gf:start-domain
mvn gf:deploy
```

9. After the instance of GlassFish is running start Open MQ on port 7676 and its admin console, as described in the section "Administering Objects of Open MQ" in Lesson 30. You need to create a topic called `Prices` there.

10. Now start the admin console of GlassFish by entering the following URL in your web browser: `http://localhost:5000/`. Create there a `TopicConnectionFactory` and a topic `jms/StockPrices` mapped to `Prices`. This process was described in Lesson 31.

11. Finally, test this application from end to end. Make sure that the Open MQ and GlassFish servers are up and running, and point your web browser to `http://localhost:8080/javafx-web-demo/`. First you'll see a scary-looking message that an applet from `dl.javafx.com` is requesting access to your computer. Accept it and you'll see a simple web page with a hyperlink and an input field, as per Listing 37-5. Click the hyperlink and you should see price quotes being pushed to your JavaFX client every five seconds, as shown in Figure 37-2.

12. Test the second way of publishing messages through the HTTP request to the servlet. To do this, open another web browser page and direct it to the same URL: `http://localhost:8080/javafx-web-demo/`. Arrange both browsers on your computer monitor so you can see them at the same time. This time do not click the link, but type any text in the input field and hit the Enter key on the keyboard. The text you entered should be received and displayed by the JavaFX client in the second web browser.



Please select Lesson 37 on the DVD with the print book, or watch online at `www.wrox.com/go/fainjava` *to view the video that accompanies this lesson.*

38

Java Technical Interviews

Technical job interviewing is a game with well-defined rules. I've worn the hats of both interviewer and interviewee many times over the last 12 years of my Java career. In this lesson I'll share with you my views on hiring Java developers and on preparing for technical interviews.

Regardless of whether the IT job market is hot or not, there are some rules and techniques that can increase your interview success rate. The process of getting a job consists of three separate tiers:

1. Getting the interview
2. Interviewing successfully
3. Considering the offer

I can't stress enough how important it is to work on achieving each of these goals separately, one step at a time! Your résumé is the most important thing in the first step. Adjust it for each position you are applying for (no, I'm not asking you to lie). Make sure it's short and to the point (not more than two pages long).

GETTING THE INTERVIEW

If you are applying for a Java developer's position, nobody needs to know the details of that Visual Basic project from 10 years ago. Always update your résumé based on the feedback you receive from recruiters or more experienced programmers.

There is a summary section on each résumé and many people just waste this space with some junk like "I'm looking for a challenging position that will let me improve my talents." What a waste! Use this summary line to show your relevant skills.

Say you've been specializing in Java messaging during the last two years, but this job posting requires web developers. Chances are that you've developed web applications before; highlight your web experience in the summary section of your résumé. The same day you may be

responding to another ad, this one for people who know Java messaging. Modify your summary section accordingly and send a version of your résumé that emphasizes your experience with messaging.

Job requirements are more involved these days, and recruiting companies don't even want to submit your résumé to the client if you have "only" 8 out of 10 required skills. Read the requirements and highlight all the relevant skills you have. Do not be lazy; work with your résumé.

Networking is another great way to get an interview. This time I'm not talking about Java networking — I'm talking about meeting other IT people who may be hiring or know people who are hiring. Attend your local Java Users Group meeting, follow Java developers on Twitter, go to professional conferences and seminars.

DOING WELL AT THE INTERVIEW

OK. The interview is scheduled, and now your main goal is to ace it. Remember, your interviewer has a difficult task: He needs to assess your technical skills within 30 to 60 minutes, so help him! Try to get as many technical details about the job as possible from your recruiter. If the position you are applying for requires knowledge of Java sockets, research working with non-blocking sockets. If they're building Java EE applications, find out which application server they use and read up on it.

Do your homework and prepare a short talk about some interesting and challenging technical problems you might have experienced in one of your college or real-world projects. If you didn't have any super-complex projects, just pick up a topic from one of the multiple online programmers' forums and research it. Another great source of Java tips and tricks is Josh Bloch's book *Effective Java*, Second Edition, Addison Wesley, ISBN: 0-321-35668-3.

For example, if you have prepared a talk on concurrent collections, you're not allowed to leave the interview without talking about this. But what if the interviewer won't ask you about Java concurrency? It doesn't really matter. Find a way to switch the conversation to the prepared topic and do your best. The interviewer will be happy because he doesn't need to think about what to ask next, and you're happy because you've had a chance to talk about a well-prepared subject.

Will this technique work all the time? No. But it'll work most of the time. Obviously you need to research the selected subject well, or you'll get fried. You must talk about technical challenges you've resolved. You need to remember that your interviewers have a difficult time trying to figure out your technical abilities in just 30 minutes. So, you need to help them out. Be in charge.

If you're a junior developer, spend some time answering the multiple-choice computer tests that are usually required for certification exams. You don't need to get certified, but all these books and online mock tests will help you pass similar tests offered by some job agencies. Find some sample interview questions online.

A technical interview is a game with known rules, but in many cases the interviewers are not prepared to run the interviews. Sometimes they just go by a prepared list of questions. Some interviewees take advantage of this and just spend some time studying introductory courses, and then memorize questions and answers for technical interviews. Believe it or not, in many cases this works.

What does a good enterprise Java developer have to know in addition to understanding the difference between abstract classes and interfaces? Usually employers are looking for people with knowledge of the following: Java Servlets, JSP, Spring framework, EJB, JMS, any commercial message-oriented middleware, JDBC, JNDI, HTML, XML, Hibernate, Ant or Maven build tools, SQL, one of the major application servers, and a couple of relational database management systems.

Understanding why a particular Java EE component is being used in your current project is equally important. If the interviewer asks you, "Why did you use EJB in this project?" please do not answer, "That decision was made before I joined the project." Have your own opinion and explain why you think it was a good or bad choice for this particular project.

Here's another tip: Don't critique the application architecture of your potential employer. You'll have plenty of chances to provide technical advice if you're hired, so just focus on getting an offer.

Be energetic during the interview and show your interest in the job. Even if you are a technical guru, don't behave as if you're doing the interviewer a favor just by showing up. Personality matters. People don't like prima donnas.

Be prepared to write code on a whiteboard and practice that skill. The interviewer may not ask you to do this, but it may be a good idea to start illustrating your thoughts on the board if there is one in the room. If there is no board, use your own notepad. Often interviewers are trying to judge how well you think and approach a problem more than how effectively you've memorized an algorithm, class name, or method arguments.

No need to rush. If you start writing a code fragment on a board, don't be afraid of making mistakes. Think aloud. It may be even beneficial for the interviewer to see how you can identify the wrong approach and then pick the right one. Practice explaining little code fragments to your friends or relatives; they don't need to know Java to participate in these role-playing games.

After the interview, as soon as you leave the building, take notes about what just happened. Don't postpone it until you get home; you may forget important details. Make a note to work on the questions you might have not answered correctly. These questions require your attention and research. Improve your technical skills after each interview.

CONSIDERING THE OFFER

You've got an offer! Now think hard about whether you want to accept it. Have I mentioned that you should look for a new job not when your employer decides to let you go or your contract ends, but when you have a stable job, the sky is blue, and the grass is green? This gives you a tremendous advantage: You can consider the offer without being under pressure from unpaid bills.

Don't accept an offer just because the new job pays an extra $5,000 a year, which comes to less than $300 a month after taxes. But do accept the offer that will give you a chance to work with interesting technologies or business applications even if it won't pay you an extra dime.

Other nonmonetary factors of an offer are health benefits, flexible work hours, ease of commute, or simply a kinship that you feel with the people working for the potential employer. Believe it or not,

some people won't accept offers from employers who require you to wear suit and tie. Just make a decision about what's more important for you — having an interesting job or being able to show off your latest tattoos.

No matter what your preferences are, take charge of your career and actively build it the way you want.

INTERVIEWING ENTERPRISE DEVELOPERS

When the job market is healthy, major online job search engines show thousands of openings, and people are competing for these jobs. These days a seasoned developer has to know about 10 different tools or technologies to find a good job and feel relatively secure for a couple of years. Over the last several years I've interviewed lots of Java developers, and this is what I've noticed:

- People do not call themselves Java developers or programmer-analysts anymore — most of them prefer the title *Java architect*. Unfortunately, only some of them really understand how Java EE components operate and can suggest design solutions. Of course, knowledge of Java EE is not all that a Java architect should know.
- Job applicants are more senior, and I barely see any college graduates or junior programmers. Many of the junior positions are being outsourced and the number of graduates with computer science degrees has declined over the past several years.
- Having software certification does not make a résumé stand out. Actually, if a résumé starts with a list of certifications, most likely it's a beginner's. I'm not against certifications, as they help you learn a language or a tool, and show that you are willing to and can study. But a certificate doesn't make someone a skilled professional.
- With the introduction of middle-tier object-relational mapping frameworks such as Hibernate, many people don't even bother learning how database management systems work or how to write a SQL query that performs well — they just map Java classes to database tables.
- In a slow economy, be prepared to pass at least four interviews to get hired. Back in 1999 two good interviews would be enough; in 2001 it was very difficult to even get an interview, let alone a job!
- In 2010 a large portion of the development is done in India. Even though the total cost of development is not low despite the lower hourly rates of offshore developers, Western hiring managers unfortunately have an impression that local candidates must be flexible in their rates, too. Being a native English speaker is not enough for you to charge top dollar. Always be technically current, otherwise the more motivated guys from overseas will leave you in the dust.

Good knowledge of the business terminology of your potential employer is also important. I'm not sure about the Silicon Valley or Europe, but here in New York just being a techie may not be good enough to get you a senior job. Of course, interviewing with Google or Microsoft is an exception.

For example, if you're applying for a Java position in a financial brokerage company and don't know what a short sale is, this may be a deal breaker. If you are a senior developer you should be able to hit the ground running: Try to find out from your recruiter as many details as possible about the business. Do your homework, and you'll get the job! Recruiters are desperately looking for good programmers and you can be one of them.

Page 474 of 506 (IPR2025-00350)
Stingray Group Inc. et al v. Hernandez-Mondragon

TO GET OR NOT TO GET CERTIFIED?

Any certification program is a business for the vendor that wants to sell training to award certificates. Oracle is no exception: you can find various certification programs to rate your Java skills. The following website offers various certification programs in Java SE and EE technologies: `http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=140#13`.

Although most certification programs require you to pass a multiple-choice computer test, some of them (such as those for Java EE architects) will give you a challenging task to design and program.

Overall I think it's a good idea to prepare yourself and go through the certification process, because it will definitely improve your understanding of the Java language or a specific enterprise technology or framework. In some areas it may also improve the performance of your résumé. It'll also help you slip through computer screening tests that are used by job placement agencies. Just check job postings from your potential employers to see if they insist on having certificates.

But I also want to caution you against overestimating the importance of getting Java certification. When I see a résumé that starts with a list of certificates, it tells me that this candidate is good… at passing multiple-choice tests. Having practical hands-on experience working on open-source or enterprise projects is a lot more valuable than any certificate you might have earned. If you have certificates — fine, but keep them somewhere at the end of your résumé rather than trying to present them as a major achievement.

TECHNICAL QUESTIONS AND ANSWERS

Suggesting a list of technical questions for an interview is a risky business. Seasoned Java developers have different views on what's fair and what's not fair to ask. But junior developers who just completed a Java tutorial can definitely benefit from some guidance in preparing for Java job interviews.

The following are suggested technical interview questions and expected brief answers on various Java-related topics. The questions included in this section are those that I had to answer while working on various projects. In no way is this a complete list of possible questions, but it will definitely give you an idea of what to expect at an interview. Most of the answers to these questions can be found in this book, but some will require additional research.

Q: What's the difference between an interface and an abstract class?

A: An abstract class may contain code in method bodies, which is not allowed in an interface. With abstract classes you have to inherit your class from the abstract one because Java does not allow multiple inheritance. On the other hand, you can implement multiple interfaces in your class.

Q: How do you deploy a web application in the application server that you currently use?

A: In most Java EE application servers you create a web archive (war file) and copy it to the assigned directory per the application server documentation. (You should be able to explain the directory structure in a war file.)

Q: What's the usage of the keyword `static`?

A: It's used in declarations of methods and variables to make them available without creating an instance of the class. For example, the `main()` method is a `static` one. If a variable is `static`, its value is shared by all instances of the class.

Q: How can you force garbage collection?

A: You can't force garbage collection, but you can request it, because JVM does not guarantee that it'll be started immediately. Invoking `System.gc()` requests Java garbage collection.

Q: Explain the usage of event adapters.

A: Some `Swing` event listener interfaces declare multiple methods (for example, `WindowListener` has seven methods) and a class must implement all of them. Adapters already have all these methods predefined with empty bodies. So if a class needs to process just one of the `WindowListener` events, it has to override only the corresponding method of `WindowAdapter`.

Q: How do you decide if explicit casting is needed?

A: If you assign a superclass object to a variable of a subclass's data type, you need to use explicit casting. For example:

```
Object a;
Customer b;
b = (Customer) a;
```

Java generics eliminate the need for explicit casting. For subclass-to-superclass assignments the casting is performed automatically. You can also cast to interfaces that a class implements.

Q: Can you perform casting between objects of different types?

A: No, you can't. The objects can have a superclass-subclass relationship, or you can cast to interfaces implemented by a class.

Q: Can a Java class be inherited from two classes?

A: No, Java does not allow multiple inheritance, but a class can implement multiple interfaces, which, to some extent, can be used as a workaround.

Q: Both applets and Servlets are deployed on the server. What's the difference between the runtime environments of applets and Servlets?

A: Applets run in a JVM on the user's machine under control of a web browser, and they have security restrictions limiting access to a computer's devices. Servlets run on the server side under control of the Servlet container or Java EE Application Server.

Q: What's the difference between constructors and regular methods?

A: Constructors must have the same name as the class and cannot return a value. They are invoked only once, while regular methods can be invoked many times.

Q: What's the difference between the HTML methods `Get` and `Post`?

A: `Get` appends the parameters to the URL, and the resulting URL is bookmarkable. `Post` allows the sending of objects as well as text data, while `Get` works with text only.

Q: What's a cookie? Which Java components create them?

A: A cookie is an object that represents a name/value pair. Servlets, JSP, and JSF can create and send cookies to a web browser that saves them on the user's disk in a special directory. Cookies help a Servlet identify a user. For example, a bank can store your account number in a cookie file on your machine, so you do not need to enter it on a log-on screen.

Q: Can a non-abstract class have both abstract and concrete methods?

A: No, but only abstract classes can have abstract methods.

Q: Explain the use of Java packages.

A: Packages offer a way to organize multi-file projects. They also help in resolving naming conflicts when different packages have classes with the same names. Package access level also enables you to protect data from being used by unauthorized classes, permitting only the classes from the same package to see each other's member variables.

Q: Explain the usage of the keyword `transient`.

A: The `transient` keyword indicates that the value of this member variable does not have to be serialized with the object. When the class gets de-serialized, `transient` variables will be initialized with the default values of the variable data types (such as `0` for integers).

Q: What do you know about thread synchronization? Explain the difference between

```
public void synchronized myMethod() { ... }
```

and

```
public void myMethod() {
      ...
    synchronized (some_object) {... }
}
```

A: The keyword `synchronized` is used to prevent race conditions when more than one thread tries to update some values. Synchronized blocks are preferable to synchronized methods because they place locks for shorter periods. But before manually synchronizing blocks of code you should see if you can use one of the collection classes from the package `java.util.concurrent` as a better solution for dealing with race conditions.

Q: What's the difference between the methods `sleep()` and `wait()`?

A: The code `sleep(1000)` puts the thread aside for exactly one second. The code `wait(1000)` makes the thread wait for up to one second. A thread can stop waiting earlier if it receives the `notify()` or `notifyAll()` call. The method `wait()` is defined in the class `Object`, but `sleep()` is defined in the class `Thread`.

Q: What do you know about the method `invokeLater()`?

A: This method is defined in the class `SwingUtilities` and is used to ensure that `Swing` windows will be updated through the event-dispatching thread. You can also do this with the `SwingWorker` class.

Q: What's the difference between creating threads subclassing the class `Thread` and implementing the `Runnable` interface?

A: Thread creation with the class `Thread` requires that your class be inherited from it. Then you need to create an instance of your class and call its `start()` method. If a class implements the `Runnable` interface the procedure is different — you have to create the instance of your class, and the instance of the `Thread` object, passing the `Runnable` class to the latter. Using `Runnable` enables you to create threads from classes that have to be inherited from classes other than `Thread` classes.

Q: How can you create a thread that can return a value from its method `run()`?

A: You need to implement `Callable` interface.

Q: What would you use to compare two `String` variables — the `equals()` method, or the `==` operator?

A: I'd use `equals()` to compare the values of the `String`s, and the `==` operator to check if two variables point at the same `String` object in memory.

Q: What do you know about the MVC design pattern?

A: MVC stands for the model-view-controller design pattern, which is used to separate presentation modules from business logic and data ones. The model part represents the data and the business logic of the application, the view is a visual representation (for example, a UI), and the controller accepts the data from the view and passes it to the model and vice versa. For example, a JSP page is a view, a Servlet is a controller, and a Java bean represents a model.

Q: How can you reduce the time spent by JVM on creating frequently used instances of some objects?

A: I'd create my own or configure existing object pools (e.g., DataSource or EJB objects can be configured to use object pools of a certain size).

Q: Will the following statement create the file `xyz.txt` on disk?

```
File a = new File("xyz.txt");
```

A: No, it just creates an object pointing to this file.

Q: How can a subclass invoke a method defined in a superclass?

A: Java has the keyword `super`. If a method has been overridden, but you'd like to invoke its version defined in the superclass, use the following syntax: `super.myMethod();`. To call a constructor of the superclass, just write `super();` in the first line of the subclass's constructor.

Q: What access level do you need to specify to ensure that only classes from the same directory can access the directory?

A: You do not need to specify any access level. In this case, Java will use the default package access level, which will guarantee visibility only to the classes located in the same directory (package).

Q: What's the use of JNDI?

A: JNDI is a Java API for naming and directory servers. It is used for finding Java objects by name in a distributed application. For example, you can use JNDI to get a reference to the `DataSource` or `JMS` objects.

Q: Do you have to invoke a method that reads a file inside the `try/catch` block?

A: You have to either put it in the `try/catch` block or declare that the calling method may throw an exception, for example:

```
void myMethod() throws IOException{...}
```

Q: If you need to catch more than one exception (such as `FileNotFoundException` and `IOException`), does the order of the `catch` statements matter?

A: Yes, it does. `FileNotFoundException` is inherited from `IOException`. The exception subclasses have to be caught first.

Q: Explain in detail the data workflow between a web page and a Servlet called `FindBooks` after the user presses Submit. (In this question interviewers can replace the web page and Servlet with any other client-server components. The main point is to find out if the job applicant understands the entire end-to-end data flow.)

A: The web browser connects to the server located at the specified URL, and if `FindBooks` has not been started yet, the Servlet container will start it, invoking the servlet's `init()` method followed by the `service()` method, which in turn calls `doGet()` or `doPost()`, depending on the method of HTTP request. The objects `HTTPServletRequest` and `HTTPServletResponse` are used for the interaction between the client and `FindBooks`. The Servlet's output can be sent back to the user by means of one of the methods of `HTTPServletResponse`, for example `println()`.

Q: Explain the process of getting the data from a database table in a Java SE program using JDBC. Which classes and methods have to be used?

A: First you load the appropriate JDBC driver using `Class.forName()`. After that you get the `Connection` object using `DriverManager.getConnection()`. Then you create a `Statement` object and invoke one of its methods, like `executeQuery()` or `executeUpdate()`. Process the `ResultSet`, if any, and close the `Connection`, `Statement`, and `ResultSet` objects.

Q: What are the differences among the Java keywords `final`, `finalize`, and `finally`?

A: Depending on its position, the keyword `final` means either that the variable can be initialized only once, or that you cannot override a method, or that you cannot subclass a class.

The method `finalize()`, if defined in your class, is invoked by the garbage collector when it's ready to release a memory used by the instance of your class.

The keyword `finally` is used in a `try/catch` block to place code, such as code to close a stream, that must be executed whether the code in the `try` block succeeds or fails.

Q: Can you declare fields in an interface?

A: Yes you can, but they should be `final` and `static`.

Q: Can an inner class, declared inside a method, access local variables of this method?

A: It's possible only if the variables are `final`.

Q: Give an example of an anonymous inner class.

A: An event listener class can be created on the fly as an anonymous inner class, for example:

```
addActionListener (new ActionListener() {
          public void actionPerformed (ActionEvent e) {
                   doSomething();}
                  }
 );
```

Q: What could be used to keep track of sessions in web applications?

A: You can use cookies, URL rewriting, and the `HTTPSession` object.

Q: Will session management with cookies always work?

A: No, if a user disables cookies in the web browser, it won't work. In such cases application servers usually automatically switch to URL rewriting.

Q: How can you stop a long-running thread?

A: The class `Thread` has a deprecated method called `stop()`, but this method does not guarantee that the thread will be stopped. Depending on the process that is run by this thread, you could try to close connections or open streams, if any, or use the method `interrupt()`.

Q: When could a Java class be called a bean?

A: You call the class a bean if it has a nonargument constructor, implements the `Serializable` interface, and has public setter/getter methods for its private properties.

Q: What are the advantages of using JSP rather than Servlets?

A: JSP enables you to separate presentation from business logic and the resulting web page can be modified by people who do not know Java.

Q: Give an example that shows the difference between the use of the operators `&&` and `&`.

A. In the following code snippet the second expression in the `if` statement will not even be evaluated if variable `a` is `null`. If a single ampersand were used here we'd get a `NullPointerException`.

```
String a=null;
if (a==null && a.length()>10) {...}
```

Q: Name some predefined JSP variables.

A: `request`, `response`, `out`, `session`...

Q: How do you deploy a JSP?

A: You can place `.jsp` files in a document root directory of the Servlet container or application server, or create a war file.

Q: Why is JSF preferable to JSP from the MVC design pattern perspective?

A: JSF enforces separation of the presentation and the processing logic, while JSP doesn't.

Q: What's the default port number that web servers use for HTTP-based communications?

A: HTTP requests go through port 80, HTTPS ones through port 443.

Q: What method of the Servlet class is an equivalent of a constructor in a regular Java class?

A: The Servlet's method `init()` plays a similar role, but because all clients use the same instance of the Servlet, you should initialize only those variables that are allowed to have the same value for each user's request, such as the name of the database server.

Q: Which is faster, array or `ArrayList`?

A: Most likely array is faster. It can be used if you know in advance the number of elements in the array. `ArrayList` is an API on top of an array. You do not need to know its size in advance — new elements can be added as needed. Arrays work faster because JVM allocates memory only once for all elements. But to give you a better answer I'd have to write an appropriate benchmark in context to find out.

Q: What do you know about Java reflection?

A: Reflection is a way of finding information about a Java class during run time. For example, you can find out the constructors and the method signatures of a particular class. The class `Class` has such methods as `getConstructor()`, `getFields()`, `getMethods()`, and others.

Q: What would you do if your Java program crashed during the run time with an "out of memory" error?

A: I'd try to increase the size of the JVM's dynamic memory (heap) on program start-up using command-line parameters. For example, you can request the minimum heap size of 512 MB and the maximum heap size of 1024 MB as follows:

```
java -Xms64 -Xmx512 MyProgram
```

Q: How can you ensure that only one instance of some class can be created in your application?

A: You need to implement the singleton design pattern. Create a class with a private constructor and provide a public `Get` method that will return the only instance of this class, for example `MyClass.getInstance()`.

Q: What's the major difference between a `Hashtable` and a `HashMap`?

A: The `Hashtable` class is internally synchronized, while the `HashMap` is not.

Q: Name some design patterns.

A: Singleton, MVC, data transfer object, facade…

Q: The word stateless is used to describe a session EJB as well as HTTP protocol. What's the meaning of the word stateless in these two cases?

A: HTTP protocol is page-based, meaning that the web browser does not hold the connection between requests, but in the case of an EJB, *stateless* means that the session bean cannot be used to store the state of a particular client.

Q: Are Java objects passed by value or by reference?

A: Objects are passed by reference, but their reference variables are passed by value.

Q: How can a Java client access a message-driven bean?

A: Java clients don't access MDBs directly — they just place messages into a queue or publish them to topics, and the MDB retrieves them from there.

EPILOGUE

The book is over. Even though it's rather small in size for the number of the covered subjects, I tried to touch on a wide spectrum of topics that most Java practitioners have to know. You may still need to do additional research on certain subjects, depending on your project needs, but at least now you know where to dig.

I really hope that you'll keep this book handy as a quick reference or for a refresher when it's time to hit the job market. No matter what your motivation is, have fun reading — I certainly had fun writing it. You can send thank-you notes and suggestions via e-mail to `yakovfain@gmail.com`. Thank you for reading my book!

APPENDIX

What's on the DVD?

This appendix provides you with information on the contents of the DVD that accompanies this book. For the most up-to-date information please refer to the ReadMe file located at the root of the DVD. Here is what you will find in this appendix:

- Using the DVD
- What's on the DVD
- Troubleshooting

USING THE DVD ON A PC

To access the content from the DVD, follow these steps:

1. Insert the DVD into your computer's DVD-ROM drive. The license agreement will appear.



The interface won't launch if you have autorun disabled. In that case click Start ➪ Run (For Windows Vista, Start ➪ All Programs ➪ Accessories ➪ Run). In the dialog box that appears, type `D:\Start.exe`. *(Replace D with the proper letter if your DVD drive uses a different letter. If you don't know the letter, see how your CD drive is listed under My Computer.) Click OK.*

2. Read through the license agreement, and click the Accept button if you want to use the DVD.
3. The DVD interface appears. Simply select the lesson video you want to view.

USING THE DVD ON A MAC

To install the items from the DVD to your hard drive, follow these steps:

1. Insert the DVD into your computer's DVD-ROM drive.
2. The DVD icon will appear on your desktop; double-click to open.
3. Double-click the Start button.
4. Read the license agreement and click the Accept button to use the DVD.
5. The DVD interface will appear. Here you can install the programs and run the demos.

WHAT'S ON THE DVD?

Most lessons in the book have a corresponding screencast that illustrates examples in the lesson and provides content beyond what is covered in print.

I recommend using the following steps when reading a lesson:

1. Read the lesson's text.
2. Watch the DVD and try to repeat all the instructor's actions on your computer.
3. Read the step-by-step instructions in the lesson's "Try It" section and complete the assignment on your own.

You can also download all the solutions to the "Try It" sections at the book's website. If you get stuck and don't know what to do next, visit the p2p forums (`p2p.wrox.com`), locate the forum for the book, and leave a post.

TROUBLESHOOTING

If you have difficulty installing or using any of the materials on the companion DVD, try the following solutions:

- **Turn off any antivirus software that you may have running:** Installers sometimes mimic virus activity and can make your computer incorrectly believe that it is being attacked by a virus. (Be sure to turn the antivirus software back on later.)
- **Close all running programs:** The more programs you're running, the less memory is available to other programs. Installers also typically update files and programs; if you keep other programs running, installation may not work properly.
- **Reference the ReadMe:** Please refer to the ReadMe file located at the root of the CD-ROM for the latest product information as of publication time.
- **Reboot if necessary:** If all else fails, rebooting your machine can often clear any conflicts in the system.

Page 484 of 506 (IPR2025-00350)
Stingray Group Inc. et al v. Hernandez-Mondragon

CUSTOMER CARE

If you have trouble with the CD-ROM please call the Wiley Product Technical Support phone number, (800) 762-2974. Outside the United States call 1 (317) 572-3994. You can also contact Wiley Product Technical Support at `http://support.wiley.com`. John Wiley & Sons will provide technical support only for installation and other general quality-control issues. For technical support on the applications themselves, consult the program's vendor or author.

To place additional orders or to request information about other Wiley products, please call (877) 762-2974.

INDEX

Symbols

. (dot), dot notation, 33
& (ampersand), NullPointerException, 444
&& (ampersand-double), AND operation, 49
*/ (asterisk/slash), comments, 22
@ (at sign), metadata, 262
{} (curly brackets, if statements, 49–50
== (equal sign-double), String, 441
! (exclamation mark)
    boolean, 50
    NOT operation, 50
|| (pipe sign-double), OR operation, 49
? (question mark), conditional operator, 50
"" (quotes-double), key/value pairs, 147
// (slash-double), comments, 18, 22
/* (slash/asterisk), comments, 22
/** (slash/asterisk-double), comments, 22
[] (square brackets), arrays, 41
<%-- , JSP comments, 313
<%= , JSP expressions, 313
<> (angled brackets), JSP tags, 310
() parentheses
    comparison expression, 50
    constructors, 33
    data types, 61
    operation sequence, 70

A

about:plugins, 104
absolute layout, 86
abstract classes, 65–68
    interfaces, 70–71, 439
Abstract Windowing Toolkit (AWT), 75
AbstractTableModel, 252
access levels, 5, 57, 95, 442
ActionEvent, 92
ActionListener, 89–91
actionPerformed(), 91–93, 230
actions, JSP, 313–314
activation, 176, 363
ActiveMQ, 336
adapters, 97–98, 440
add()
    ArrayList, 144–145
    LinkedList, 151
addNewText(), 426
addObject(), 405
addPaneltoFrame(), 126
addQuestion.html, 407
addUserQuestion(), 407–408
Adobe Flex, 426
ancestors, 26
    class instances, 60
    interfaces, 70–71
AND operation, 49
Annotation Processing Tool (APT), 267
annotations, 261–272
    POJO, 363
    reflection, 267–270
    run-time, 270–271
ANNOTATION_TYPE, 265
anonymous inner classes, 99–100
Ant, 297
Apache Commons, 205–206
Apache Taglibs, 320
Apache Tomcat, 287, 296
Apache Velocity, 206

Page 487 of 506 (IPR2025-00350)
Stingray Group Inc. et al v. Hernandez-Mondragon

ApacheDS, 357
API. *See* application program interface
`appclient`, 424
`Applet`, 71
`<applet>`, 105, 106
`</applet>`, 106
`<applet-desc>`, 430
applets, 103–109
  certificates, 104–105
  Eclipse, 105
  HTML, 105–107
  inheritance, 71
  JavaFX, 424, 430
  JVM, 103
  `main()`, 107
  plug-ins, 103–104
  RMI, 279
  servlets, 440
  Swing, 75, 107–108, 111–122
  unsigned, 104–105
appletviewer, 105, 108
`application`, 312, 317
application program interface (API), 46, 337–338.
  See also specific APIs
  JavaMail, 199
  method overloads, 32
`ApplicationContext`, 400
APT. *See* Annotation Processing Tool
`archive`, 106
arguments, 31, 51–52, 92
`ArithmeticException`, 134
arrays, 41–42
  collections, 143–144
  data types, 60
  JVM command-line arguments, 51
  upcasting, 60
`ArrayBlockingQueue`, 229
`ArrayIndexOutOfBoundsException`, 144
`ArrayList`, 41, 144–145, 445
  collections, 144–146
  `Collections`, 149
  `main()`, 201
`@Asynchronous`, 367
asynchronous communication, 335, 336
asynchronous method invocation, 362, 367–368
asynchronous servlets, 306–307
`attribute`, 312
authentication, 201
`AUTO_ACKNOWLEDGE`, 342
autoboxing, 21, 255
AWT. *See* Abstract Windowing Toolkit

B

Bean Validation, 382
bean-managed currency, 370
behaviors, 58
`bind()`, 277
birthday mailer, 200–204
`BitSet`, 151–152
Bloch, Josh, 436
`BLOCKED`, 219
blocking calls, 336
`BlockingQueue`, 229
`<bodycontent>`, 318
`boolean`, 20, 49–50, 215
`BorderLayout`, 81
bounded wild cards, 159–160
`BoxLayout`, 84
`break`, 26, 44
breakpoints, 47
browsers, 300–301
buffered streams, 165–166
`BufferedInputStream`, 165–166
`BufferedOutputStream`, 166
`BufferedReader`, 188
`BufferReader`, 166–167
`byte`, 20
`byte[]`, 182–183
byte arrays, 182–183
byte streams, 164–165
`BytesMessage`, 338

C

`Calculator`, 90–92, 94–96
`CalculatorEngine`, 90–91

Page 488 of 506 (IPR2025-00350)
Stingray Group Inc. et al v. Hernandez-Mondragon

`call()`, 228
`Callable`, 210, 226, 441
`CallableStatement`, 243–244
callback methods, 107, 253–257, 399
`CardLayout`, 86
`case`, 26
case-sensitivity, 5, 378
casting, 60–62, 92, 440
    generics, 156, 158–159
    run-time errors, 70
`catch`, 135, 136, 138, 164
`CATEGORY_`, 218
CDB. *See* Clear Data Builder
CDI. *See* Context and Dependency Injection
cell editor, 257
cell renderers, 257–259
`cellRenderer`, 258
certificates, 104–105
certification programs, 439
`char`, 20
character streams, 166–170
`CharSet`, 166
checked exceptions, 135
`checkForWinner()`, 115, 117
`CLASS`, 265
classes, 5
    behaviors, 58
    compiler, 12
    constants, 57
    declarations
        `import`, 56
        properties, 22–23
    fields, 17–18
    `final`, 57
    generics, 155–157
    inheritance, 26–27, 60, 69, 440
    inner, 98–100
    instances, 18
        ancestors, 60
        interfaces, 61
        `static`, 21
    interfaces, 58–60
    libraries, 58
    `main`, 51
    methods, 17, 23–24
        declarations, 31–39, 58
        instances, 100
    multi-threading, 210–211
    OO, 17–18
    outer, 99
    packages, 56
    properties, 18
    Swing, 76, 97–98
    variables, 18, 21, 33
        serialization, 179
        `static`, 37–38
    versioning, 182
`Class.forName()`, 443
`ClassNotFoundException`, 179
Clear Data Builder (CDB), 267
`CLIENT_ACKNOWLEDGE`, 342
clients, RMI, 273, 277–278
cloning objects, 183
`close()`, 138, 164
closures, 100
`code`, 106
`codebase`, 106
`Collection`, 146
collections, 143–153
    `ArrayList`, 144–146
    arrays, 143–144
    `BitSet`, 151–152
    `Enumeration`, 149–150
    generics, 156
    interfaces, 146
    `Iterator`, 149–150
    `LinkedList`, 150–151
    multi-threading, 230
    `Properties`, 148–149
    `Vector`, 144–146
`Collections`, 149
`Collections.syncrhonizedList()`, 144
`@Column`, 377
`commandButton`, 330, 332
command-line arguments, 51–52
comma-separated values (CSV), 191
comments, 18, 21–27, 313
`commit()`, 247, 342

Page 489 of 506 (IPR2025-00350)
Stingray Group Inc. et al v. Hernandez-Mondragon

Commons Logging API, 402
Comparable, 219
compiler
  classes, 12
  errors, 6, 31
    Eclipse, 15
  Hello World, 6–7
  multi-threading, 215
composition pages, 325
CONCUR_READ_ONLY, 246
concurrent execution, 223–233
ConcurrentHashMap, 230
ConcurrentSkipListMap, 230
ConcurrentSkipListSet, 230
CONCUR_UPDATABLE, 246
Condition, 225
conditional operator, 50
conditional statements, 25, 49, 131
config, 312
Configuration, 412
Connection, 337, 339, 443
  start(), 338, 340
connection pools, 247–248
ConnectionFactory, 338, 353
connection.setClientID(), 343
constants, 57
  declaration, 19
  interfaces, 60
  ping-pong, 127–128
CONSTRUCTOR, 265
constructors, 33
  EJB, 370
  JFrame, 76
  ModelAndView, 405
  overloads, this, 35
  regular methods, 440
  TableModelEvent, 257
container-managed concurrency, 370
containers
  absolute layout, 86
  EJB, 361–362
  Java EE, 287–288
  Spring Framework, 401
containsKey(), 147
containsValue(), 147
contexts, 349, 399
Context and Dependency Injection (CDI), 288
context root, 296
continue, 44, 45
converters, 333
cookies, 302–303, 441, 444
CopyOnWriteArrayList, 230
CopyOnWriteArraySet, 230
core module, Spring Framework, 400
create, retrieve, update, and delete (CRUD), 375
createCalendarTimer(), 372
createDurableSubscriber(), 343
createIntervalTimer(), 372
createNewFile(), 171
createSession(), 342
createSingleActionTimer(), 372
createSubscriber(), 343
createTimer(), 372
Criteria API, 375, 380–381
CriteriaBuilder, 381
CRUD. *See* create, retrieve, update, and delete
CSV. *See* comma-separated values
currentButton.SetText(), 116
cursive, 45
custom cell renderers, 257

D

DAS. *See* Domain Administration Server
data access module, Spring Framework, 400
data streams, 171
Data Transfer Objects (DTOs), 241–242, 315
data types, 19–20, 61
  arrays, 60
  method signatures, 31
database management system (DBMS), 4
  Derby, 236–238
  Hibernate, 413–515
  MOM, 337
  streams, 163
databases
  JDBC, 235–249
  JPA, 376–378

Page 490 of 506 (IPR2025-00350)
Stingray Group Inc. et al v. Hernandez-Mondragon

DataInputStream, 171
DataOutputStream, 171
DataSource, 247–248, 355–357
DBMS. *See* database management system
debugging, 46–48
declarations
    annotations, 265
    arrays, 41
    classes
        import, 56
        methods, 31–39, 58
        properties, 22–23
    constants, 19
    event listeners, 89
    interfaces, 58
    JSP, 312–313
    methods, classes, 31–39, 58
    variables, 19, 20
deep copy, 183
DefaultTableCellRenderer, 258
DefaultTableModel, 252
DelayQueue, 229
DELETE, 388
delete(), 171
dependency injection, 289, 399
@deprecated, 264
Deque, 229
Derby, 236–238
descendent, 26
deserialization, 176
destroy(), 107–108, 301, 304–305
detail, 141
directives, 312
directory service, 279, 349
DispatcherServlet, 401, 407
distributed Java applications, 273
, 327
divideByZero(), 134
DNS. *See* domain name system
document root, 296
@documented, 264
doFilter(), 304
doGet(), 299, 303
doInBackground(), 231–232
Domain Administration Server (DAS), 350
domain name system (DNS), 279, 349
doPost(), 299, 301
doSomething(), 160, 332
doStartTag(), 319
dot-notation, 33
double, 20
downcasting, 61
downloads, 189–191
DriverManager.getConnection(), 443
DTOs. *See* Data Transfer Objects
DUP_OK_ACKNOWLEDGE, 342
durable subscribers, 343
dynamic IP address, 185

E

<E>, 156, 157
eager initialization, 369
.ear, 370
Eclipse, 9–11
    applets, 105
    appletviewer, 108
    compilation errors, 15
    debugging, 46–48
    GlassFish, 297
    Hello World, 11–15
    Hibernate, 412–413
    IDEs, 9–15
        stack trace, 134
    JRE, 12
    for loop, 158
    servlets, 298–300
    Spring Framework, 402–406
    templates, 328
Eclipse Foundation, 75, 287
Effective Java (Bloch), 436
EJB. *See* Enterprise JavaBeans
@EJB, 364
ejb-jar.xml, 363
EL. *See* Expression Language
ElementType, 265

Page 491 of 506 (IPR2025-00350)
Stingray Group Inc. et al v. Hernandez-Mondragon

else, 25, 49
    if, 49
e-mail, 199–207
    authentication, 201
    birthday mailer, 200–204
    Java EE, 199–200
    retrieval, 204–205
<embed>, 106
EMPNO, 237
EMS, 335
ENAME, 237
encapsulation, 56–57, 95
Enterprise Archive, 370
Enterprise JavaBeans (EJB), 284, 286, 361–374
    asynchronous method invocation, 367–368
    GlassFish, 366–367
    JMS, 362
    JNDI, 364
    JSF, 324
    MOM, 362
    multi-threading, 362
    passivation, 363
    POJO, 361–362
    servlets, 353
entities, 376
@Entity, 376
entity beans, 362
EntityManager, 376, 379–381
enum, 218–219
Enumeration, 149–150
EOFException, 136
equals(), 441
Error, 135
errors, 133–142
    compiler, 6, 31
        Eclipse, 15
    I/O, 134–135
    JSP, 314
    JVM, 135
    run-time, 70, 135
event dispatch threads, 230
event handling, 89–102
event listeners, 89–90
    JDK, 96–97
    methods, 98
    servlets, 305–306
    Swing, 440
EventQueue, 231
Exception, 135, 136
exception, 311
exceptions, 133–142
    checked, 135
    creating, 139–141
    hierarchy, 135
    I/O, 186
    objects, throw, 139
    throwing, 135
executeQuery(), 443
executeUpdate(), 443
executor framework, 225–229
Executors, 226
ExecutorService, 226
exists(), 172
Expert One-on-One J2EE Design and Development (Johson), 399
explicit casting, 440
Expression Language (EL), 313
expressions, 313
extends, 160
Externalizable, 179–181

F

<f: validator>, 333
fantasy, 45
fat clients, 285
FIELD, 265
fields, 17–18
FIFO. *See* first-in-first-out
File, 171–172
FileDownload, 189–191
FileInputStream, 164–166
FileNotFoundException, 442
FileReader, 166–170
FileWriter, 166–170
fillRect(), 127
filters
    destroy(), 304–305
    init(), 304–305

Page 492 of 506 (IPR2025-00350)
Stingray Group Inc. et al v. Hernandez-Mondragon