Capitulo 4. Desarrollando la Interface Web

Esta es la Parte 4 del tutorial paso a paso para desarrollar una aplicacion web desde cero usando Spring Framework. En la Parte 1 hemos configurado el entorno y montado la aplicacion basica. En la Parte 2 hemos mejorado la aplicacion que habiamos construido hasta entonces. La Parte 3 añade toda la logica de negocio y los tests de unidad. Ahora es el momento de construir la interface web para la aplicacion.

4.1. Añadir una referencia a la logica de negocio en el controlador

Para empezar, renombramos `HelloController` a algo mas descriptivo, como por ejemplo `InventoryController`, puesto que estamos construyendo un sistema de inventario. Aqui es donde un IDE con opcion de refactorizar es de valor incalculable. Renombramos `HelloController` a `InventoryController` asi como `HelloControllerTests` a `InventoryControllerTests`. A continuacion, ,modificamos `InventoryController` para almacenar una referencia a la clase `ProductManager`. Tambien añadimos codigo para permitir al controlador pasar algo de informacion sobre un producto a la vista. El metodo `getModelAndView()` ahora devuelve tanto un Map con la fecha y hora como una lista de productos.

"springapp/src/springapp/web/InventoryController.java":

```
package springapp.web;

import org.springframework.web.servlet.mvc.Controller;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.util.Map;
import java.util.HashMap;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import springapp.service.ProductManager;

public class InventoryController implements Controller {

    protected final Log logger = LogFactory.getLog(getClass());

    private ProductManager productManager;

    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        String now = (new java.util.Date()).toString();
        logger.info("returning hello view with " + now);

        Map<String, Object> myModel = new HashMap<String, Object>();
        myModel.put("now", now);
        myModel.put("products", this.productManager.getProducts());

        return new ModelAndView("hello", "model", myModel);
    }


    public void setProductManager(ProductManager productManager) {
        this.productManager = productManager;
    }

}
```

Tambien necesitaremos modificar InventoryControllerTest para proporcionar un ProductManager y extraer el valor para 'now' desde el modelo Map antes de que los test sean pasados de nuevo.

"springapp/test/springapp/web/InventoryControllerTests.java":

```
package springapp.web;

import java.util.Map;

import org.springframework.web.servlet.ModelAndView;

import springapp.service.SimpleProductManager;
import springapp.web.InventoryController;

import junit.framework.TestCase;

public class InventoryControllerTests extends TestCase {

    public void testHandleRequestView() throws Exception{
        InventoryController controller = new InventoryController();
        controller.setProductManager(new SimpleProductManager());
        ModelAndView modelAndView = controller.handleRequest(null, null);
        assertEquals("hello", modelAndView.getViewName());
        assertNotNull(modelAndView.getModel());
        Map modelMap = (Map) modelAndView.getModel().get("model");
        String nowValue = (String) modelMap.get("now");
        assertNotNull(nowValue);
    }
```

```
}
```

4.2. Modificar la vista para mostrar datos de negocio y añadir soporte para archivos de mensajes

Usando la etiqueta JSTL `<c:forEach/>`, añadimos una seccion que muestra informacion de cada producto. Tambien vamos a reemplazar el titulo, la cabecera y el texto de bienvenida con una etiqueta JSTL `<fmt:message/>` que extrae el texto a mostrar desde una ubicacion `'message'` – veremos esta ubicacion un poco mas adelante.

"springapp/war/WEB-INF/jsp/hello.jsp":

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>

<html>
  <head><title><fmt:message key="title"/></title></head>
  <body>
    <h1><fmt:message key="heading"/></h1>
    <p><fmt:message key="greeting"/> <c:out value="${model.now}"/></p>
    <h3>Products</h3>
    <c:forEach items="${model.products}" var="prod">
      <c:out value="${prod.description}"/> <i>$<c:out value="${prod.price}"/></i><br><br>
    </c:forEach>
  </body>
</html>
```

4.3. Añadir datos de prueba para rellenar algunos objetos de negocio

Es el momento de añadir `SimpleProductManager` a nuestro archivo de configuracion y de pasarlo a traves del setter de `InventoryController`. Todavia no vamos a añadir ningun codigo para cargar los objetos de negocio desde una base de datos. En su lugar, podemos reemplazarlos con unas cuantas instancias de la clase `Product` usando beans Spring y el soporte de la aplicacion. Simplemente pondremos los datos que necesitamos en un puñado de entradas bean en el archivo **"springapp-servlet.xml"**. Tambien añadiremos la entrada bean para `'messageSource'` que nos permitira recuperar mensajes desde la ubicacion (**"messages.properties"**) que crearemos en el proximo paso. Ademas, debes renombrar la referencia a `HelloController` por `InventoryController` puesto que le hemos cambiado el nombre.

"springapp/war/WEB-INF/springapp-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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <!-- the application context definition for the springapp DispatcherServlet -->

    <bean id="productManager" class="springapp.service.SimpleProductManager">
        <property name="products">
            <list>
                <ref bean="product1"/>
                <ref bean="product2"/>
                <ref bean="product3"/>
            </list>
        </property>
    </bean>

    <bean id="product1" class="springapp.domain.Product">
        <property name="description" value="Lamp"/>
        <property name="price" value="5.75"/>
    </bean>

    <bean id="product2" class="springapp.domain.Product">
        <property name="description" value="Table"/>
        <property name="price" value="75.25"/>
    </bean>

    <bean id="product3" class="springapp.domain.Product">
        <property name="description" value="Chair"/>
        <property name="price" value="22.79"/>
    </bean>

    <bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
        <property name="basename" value="messages"/>
    </bean>

    <bean name="/hello.htm" class="springapp.web.InventoryController">
        <property name="productManager" ref="productManager"/>
    </bean>

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

</beans>
```

4.4. Añadir una ubicacion para los mensajes y la tarea "clean" a "build.xml"

Creamos un archivo llamado **"messages.properties"** en el directorio **"war/WEB-INF/classes"**. Este archivo de propiedades contiene tres entradas que coinciden con las claves especificadas en las etiquetas `<fmt:message/>` que hemos añadido a **"hello.jsp"**.

"springapp/war/WEB-INF/classes/messages.properties":

```
title=SpringApp
heading=Hello :: SpringApp
greeting=Greetings, it is now
```

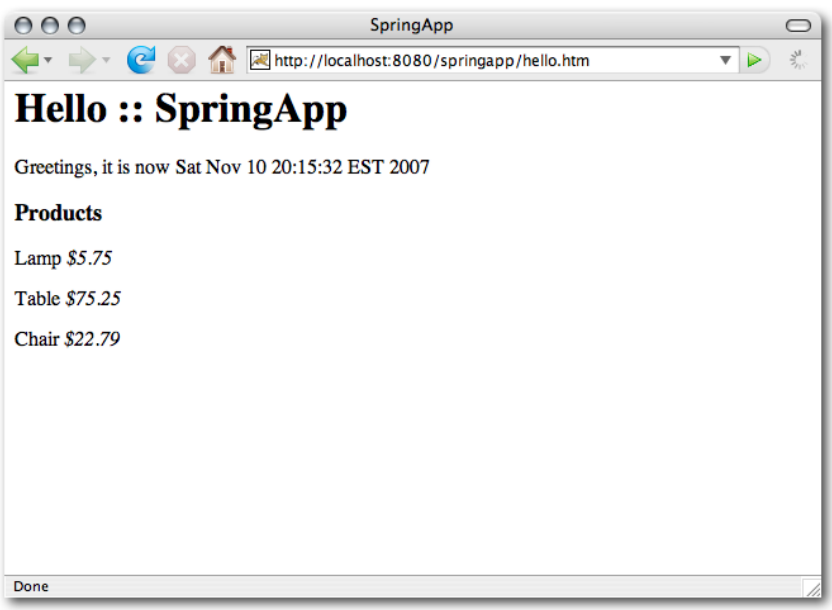
Puesto que hemos movido algunos archivos de codigo fuente, tiene sentido añadir los comandos 'clean' y 'undeploy' al script de Ant. Añadimos las siguientes entradas en el archivo "build.xml".

"build.xml":

```
    <target name="clean" description="Clean output directories">
        <delete>
            <fileset dir="${build.dir}">
                <include name="**/*.class"/>
            </fileset>
        </delete>
    </target>

    <target name="undeploy" description="Un-Deploy application">
        <delete>
            <fileset dir="${deploy.path}/${name}">
                <include name="**/*.*"/>
            </fileset>
        </delete>
    </target>
```

Ahora deten el servidor Tomcat y ejecuta Ant con las opciones 'clean', 'undeploy' y 'deploy'. Esto eliminara todos los archivos de clases, reconstruira la aplicacion y la desplegara de nuevo. Arranca Tomcat de nuevo y deberias ver lo siguiente al cargar la aplicacion desde tu navegador:



La aplicacion actualizada

4.5. Añadir un formulario

Para proveer de una interface a la aplicacion web que muestre la funcionalidad para incrementar los precios, vamos a añadir un formulario que permitira al usuario introducir un valor de porcentaje. Este formulario usa una libreria de etiquetas llamado 'spring-form.tld' que es suministrada con Spring Framework. Tenemos que copiar este archivo desde nuestra distribucion de Spring ("spring-framework-2.5/dist/resources/spring-form.tld") al directorio "springapp/war/WEB-INF/tld" que ademas debemos crear. A continuacion debemos añadir tambien una entrada <taglib/> en el archivo "web.xml".

"springapp/war/WEB-INF/web.xml":

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

<web-app version="2.4"
         xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" >

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

  <servlet-mapping>
    <servlet-name>springapp</servlet-name>
    <url-pattern>*.htm</url-pattern>
```

```
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>
      index.jsp
    </welcome-file>
  </welcome-file-list>

  <jsp-config>
    <taglib>
      <taglib-uri>/spring</taglib-uri>
      <taglib-location>/WEB-INF/tld/spring-form.tld</taglib-location>
    </taglib>
  </jsp-config>

</web-app>
```

Tambien tenemos que declarar este taglib en una directiva page en el siguiente archivo JSP, y ya podremos comenzar a utilizar las etiquetas que habremos asi importado. Añade el archivo JSP **"priceincrease.jsp"** al directorio **"war/WEB-INF/jsp"**.

"springapp/war/WEB-INF/jsp/priceincrease.jsp":

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<html>
<head>
  <title><fmt:message key="title"/></title>
  <style>
    .error { color: red; }
  </style>
</head>
<body>
<h1><fmt:message key="priceincrease.heading"/></h1>
<form:form method="post" commandName="priceIncrease">
  <table width="95%" bgcolor="f8f8ff" border="0" cellspacing="0" cellpadding="5">
    <tr>
      <td align="right" width="20%">Increase (%):</td>
        <td width="20%">
          <form:input path="percentage"/>
        </td>
        <td width="60%">
          <form:errors path="percentage" cssClass="error"/>
        </td>
    </tr>
  </table>
  <br>
  <input type="submit" align="center" value="Execute">
</form:form>
<a href="<c:url value="hello.htm"/>">Home</a>
</body>
</html>
```

La siguiente clase es un JavaBean muy sencilla que solamente contiene una propiedad con su correspondientes metodos getter y setter. Este es el objeto que el formulario rellenara y desde el que nuestra logica de negocio extraera el procentaje de incremento que queremos aplicar a los precios.

"springapp/src/springapp/service/PriceIncrease.java":

```
package springapp.service;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class PriceIncrease {

    /** Logger for this class and subclasses */
    protected final Log logger = LogFactory.getLog(getClass());

    private int percentage;

    public void setPercentage(int i) {
        percentage = i;
        logger.info("Percentage set to " + i);
    }

    public int getPercentage() {
        return percentage;
    }

}
```

La siguiente clase de validacion toma el control despues de que el usuario pulse el boton submit. Los valores introducidos en el formulario seran guardados en el objeto de comando por el framework. El metodo `validate(..)` es llamado en el objeto de comando `PriceIncrease`. Ademas un objeto que contiene cualquier error que se haya producido al completar el formulario es pasado tambien.

"springapp/src/springapp/service/PriceIncreaseValidator.java":

```
package springapp.service;

import org.springframework.validation.Validator;
import org.springframework.validation.Errors;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class PriceIncreaseValidator implements Validator {
```

```
    private int DEFAULT_MIN_PERCENTAGE = 0;
    private int DEFAULT_MAX_PERCENTAGE = 50;
    private int minPercentage = DEFAULT_MIN_PERCENTAGE;
    private int maxPercentage = DEFAULT_MAX_PERCENTAGE;

    /** Logger for this class and subclasses */
    protected final Log logger = LogFactory.getLog(getClass());

    public boolean supports(Class clazz) {
        return PriceIncrease.class.equals(clazz);
    }

    public void validate(Object obj, Errors errors) {
        PriceIncrease pi = (PriceIncrease) obj;
        if (pi == null) {
            errors.rejectValue("percentage", "error.not-specified", null, "Value required.");
        }
        else {
            logger.info("Validating with " + pi + ": " + pi.getPercentage());
            if (pi.getPercentage() > maxPercentage) {
                errors.rejectValue("percentage", "error.too-high",
                    new Object[] {new Integer(maxPercentage)}, "Value too high.");
            }
            if (pi.getPercentage() <= minPercentage) {
                errors.rejectValue("percentage", "error.too-low",
                    new Object[] {new Integer(minPercentage)}, "Value too low.");
            }
        }
    }

    public void setMinPercentage(int i) {
        minPercentage = i;
    }

    public int getMinPercentage() {
        return minPercentage;
    }

    public void setMaxPercentage(int i) {
        maxPercentage = i;
    }

    public int getMaxPercentage() {
        return maxPercentage;
    }

}
```

4.6. Añadir un controlador de formulario

Ahora tenemos que añadir una entrada en el archivo "`springapp-servlet.xml`" para definir el nuevo formulario y su controlador. Definimos los objetos a inyectar en el controlador del formulario mediante las propiedades `commandClass` y `validator`. Tambien especificamos dos vistas, `formView` que es usada por el formulario, y `successView`, a la que iremos despues de procesar satisfactoriamente el formulario. La ultima vista puede ser de dos tipos. Puede ser una referencia a una vista normal, que es enviada a uno de nuestras paginas JSP. Una desventaja de esta aproximacion es que si el usuario recarga la pagina, los datos del formulario se enviaran de nuevo, y podrias terminar incrementando varias el porcentaje. Una manera alternativa es usar redirect, donde la respuesta se envia de vuelta al navegador del usuario informandole que debe redireccionar a una nueva URL. Esta URL no puede ser una de nuestras paginas JSP, puesto que estan ocultas (en WEB-INF) y no es posible su acceso directo. Tiene que ser una URL alcanzable desde el exterior. Hemos elegido usar "`hello.htm`" como URL para redirect. Esta URL esta mapeada a la pagina "`hello.jsp`", que funcionara correctamente.

"`springapp/war/WEB-INF/springapp-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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<!-- the application context definition for the springapp DispatcherServlet -->

<beans>

    <bean id="productManager" class="springapp.service.SimpleProductManager">
        <property name="products">
            <list>
                <ref bean="product1"/>
                <ref bean="product2"/>
                <ref bean="product3"/>
            </list>
        </property>
    </bean>

    <bean id="product1" class="springapp.domain.Product">
        <property name="description" value="Lamp"/>
        <property name="price" value="5.75"/>
    </bean>

    <bean id="product2" class="springapp.domain.Product">
        <property name="description" value="Table"/>
        <property name="price" value="75.25"/>
    </bean>

    <bean id="product3" class="springapp.domain.Product">
        <property name="description" value="Chair"/>
        <property name="price" value="22.79"/>
```

```
    </bean>

    <bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
        <property name="basename" value="messages"/>
    </bean>

    <bean name="/hello.htm" class="springapp.web.InventoryController">
        <property name="productManager" ref="productManager"/>
    </bean>

    <bean name="/priceincrease.htm" class="springapp.web.PriceIncreaseFormController">
        <property name="sessionForm" value="true"/>
        <property name="commandName" value="priceIncrease"/>
        <property name="commandClass" value="springapp.service.PriceIncrease"/>
        <property name="validator">
            <bean class="springapp.service.PriceIncreaseValidator"/>
        </property>
        <property name="formView" value="priceincrease"/>
        <property name="successView" value="hello.htm"/>
        <property name="productManager" ref="productManager"/>
    </bean>

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

</beans>
```

A continuacion, echemos un vistazo al controlador de este formulario. El metodo `onSubmit(..)` toma el control y hace algo de logging antes de llamar al metodo `increasePrice(..)` en el objeto `ProductManager`. Entondes devuelve un objeto `ModelAndView` pasando en el una nueva instancia de `RedirectView` que es creada usando la URL de la vista que se mostrara si no hay ningun error en el formulario.

"springapp/src/web/PriceIncreaseFormController.java":

```
package springapp.web;

import org.springframework.web.servlet.mvc.SimpleFormController;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.view.RedirectView;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import springapp.service.ProductManager;
import springapp.service.PriceIncrease;

public class PriceIncreaseFormController extends SimpleFormController {

    /** Logger for this class and subclasses */
    protected final Log logger = LogFactory.getLog(getClass());

    private ProductManager productManager;

    public ModelAndView onSubmit(Object command)
            throws ServletException {

        int increase = ((PriceIncrease) command).getPercentage();
        logger.info("Increasing prices by " + increase + "%.");

        productManager.increasePrice(increase);

        logger.info("returning from PriceIncreaseForm view to " + getSuccessView());

        return new ModelAndView(new RedirectView(getSuccessView()));
    }

    protected Object formBackingObject(HttpServletRequest request) throws ServletException {
        PriceIncrease priceIncrease = new PriceIncrease();
        priceIncrease.setPercentage(20);
        return priceIncrease;
    }

    public void setProductManager(ProductManager productManager) {
        this.productManager = productManager;
    }

    public ProductManager getProductManager() {
        return productManager;
    }

}
```

Vamos a añadir tambien algunos mensajes al archivo de mensajes **"messages.properties"**.

"springapp/war/WEB-INF/classes/messages.properties":

```
title=SpringApp
heading=Hello :: SpringApp
greeting=Greetings, it is now
priceincrease.heading=Price Increase :: SpringApp
error.not-specified=Percentage not specified!!!
error.too-low=You have to specify a percentage higher than {0}!
error.too-high=Don""t be greedy - you can""t raise prices by more than {0}%!
required=Entry required.
typeMismatch=Invalid data.
```

```
typeMismatch.percentage=That is not a number!!!
```

Compila y despliega, y despues de recargar la aplicacion podemos probarla. Ahora el formulario puede mostrar los errores.

Finalmente, vamos a añadir un enlace a la pagina de incremento de precio desde `"hello.jsp"`.

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>

<html>
  <head><title><fmt:message key="title"/></title></head>
  <body>
    <h1><fmt:message key="heading"/></h1>
    <p><fmt:message key="greeting"/> <c:out value="${model.now}"/></p>
    <h3>Products</h3>
    <c:forEach items="${model.products}" var="prod">
      <c:out value="${prod.description}"/> <i>$<c:out value="${prod.price}"/></i><br><br>
    </c:forEach>
    <br>
    <a href="<c:url value="priceincrease.htm"/>">Increase Prices</a>
    <br>
  </body>
</html>
```

Ahora, ejecuta Ant con los comandos 'deploy' y 'reload' y prueba la nueva funcionalidad de incremento de precio.

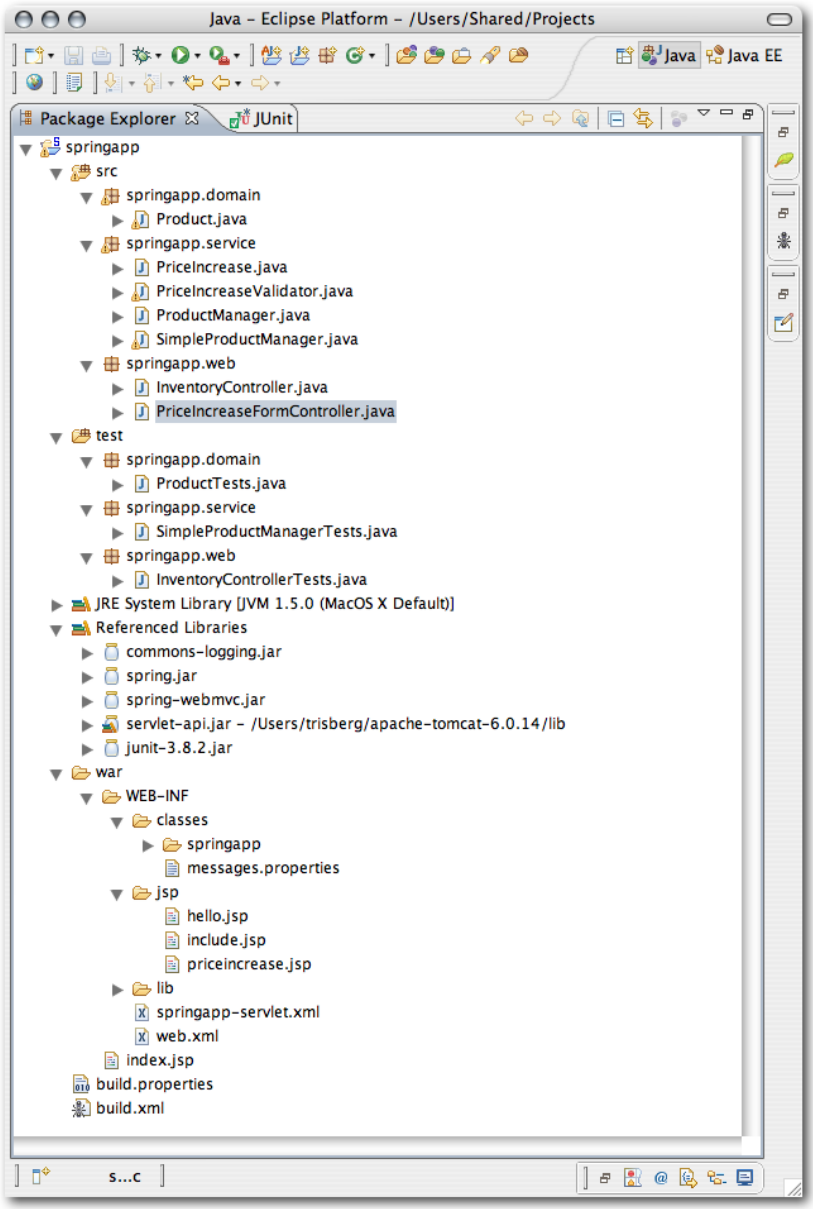
La aplicacion actualizada

4.7. Resumen

Vamos a ver lo que hemos hecho en la Parte 4.

- Hemos renombrado nuestro controlador a `InventoryController` y le hemos dado una referencia a `ProductManager` por lo que ahora podemos recupear una lista de productos para mostrar.
- Entonces hemos definido algunos datos de prueba para rellenar objetos de negocio.
- A continuacion hemos modificado la pagina JSP para usar una ubicacion de mensajes y hemos añadido un loop forEach para mostrar una lista dinamica de productos.
- Despues hemos creado un formulario para disponer de la capacidad de incrementar los precios.
- Finalmente hemos creado un controlador de formulario y un validador, y hemos desplegado y probado las nuevas caracteristicas.

Debajo puedes ver una captura de pantalla mostrando como debe aparecer tu estructura de directorios despues de seguir todas las instrucciones anteriores.



La estructura de directorios del proyecto al final de la parte 4