Capítulo 5
Diseño y desarrollo de aplicaciones web con patrón MVC

Aspectos docentes

Este capítulo es muy importante ya que es la pieza clave para adoptar el patrón MVC, concepto fundamental entorno al cual gira la estructura de una aplicación web. Es preciso que los alumnos sean capaces de "ver" los JavaBeans en las aplicaciones, lo que obliga a tener muy claro el conjunto de elementos funcionales de la aplicación. Aquí entran en juego los diagramas UML vistos en los capítulos anteriores. Será necesario recurrir a los diagramas de casos de uso y de clases para definir los aspectos funcionales de la aplicación (diagrama de casos de uso) y aproximar el modelo (diagrama de clases). El patrón vista vendrá perfilado por ambos diagramas.

El proyecto El Fotograma Perdido ampliará su enunciado en este capítulo, incorporando nuevas funcionalidades, tanto a nivel de servicios ofrecidos como en cuanto al aspecto visual del mismo. Se empleará MySQL para almacenar toda la información que procesará la aplicación.

Herramientas necesarias

Eclipse Helios o Indigo
MySQL 5
Conector J
Servidor de aplicaciones JBoss 6 o 7.

Bibliografía

Bases de datos con Java	Kevin Mukhar y otros	Anaya Multimedia	84-415-1362-
Desarrollo Web con JSP	Varios autores	Anaya Multimedia	84-415-1352-X
J2EE Manual de referencia	Jim Keogh	McGraw Hill	84-481-3980-1
JSP. Ejemplos prácticos	Andrew Patzer	Anaya Multimedia	84-415-1466-6
Manual de referencia de JSP	Phil Hanna	McGraw-Hill	84-481-3264-5
MySQL	Paul DuBois	Prentice Hall	84-205-2956-7
Programación Java Server con J2EE Edición 1.3	Varios autores	Anaya Multimedia	84-415-1358-9
Programación de bases de datos con JDBC y Java	George Reese	Anaya Multimedia	84-415-1183-7

Contenido

Cómo funciona un servlet

Dado que el patrón MVC se basa en el uso de un servlet, se hace necesario proceder con una breve introducción a la estructura y ciclo de vida de un servlet.

Se puede ejecutar directamente un servlet desde un contenedor Web mediante una solicitud HTTP, de forma que en la URL quede claro para el contenedor web que lo que se solicita es precisamente un servlet. La manera en que esto puede hacerse la veremos más adelante, pero por ahora bastará con saber que en la estructura de la URL puede aparecer el nombre de directorio *servlet*. Un ejemplo lo tenemos en la siguiente URL:

http://localhost:8080/nombreaplicacionweb/servlet/miservlet[59]

Conviene recordar que también puede enmascararse el uso del directorio *servlet*.

Una vez que el navegador ha enviado la solicitud HTTP, el contenedor Web localizará la clase del servlet, y comenzará su procesamiento.

Para ver cómo funciona un servlet lo haremos con un ejemplo simple.

El siguiente código se corresponde con un servlet que devuelve un saludo bastante escueto.

Proyecto: PruebaServlet

59 Con algunos servidores de aplicaciones esto no funciona salvo que se configuren los servlets en el descriptor de despliegue de manera que puedan ser invocados directamente desde la url */servlet*. Este es el caso de Jboss.

Contexto aplicación: pruebaservlet
Archivo Servlet: SaludoSimple.java
URL: /pruebaservlet/SaludoSimple?nombre=loquesea

```
1.  package servlets;
2.
3.  import java.io.*;
4.  import javax.servlet.*;
5.  import javax.servlet.http.*;
6.
7.  /**
8.   * Servlet implementation class SaludoSimple
9.   */
10. public class SaludoSimple extends HttpServlet {
11.     private static final long serialVersionUID = 1L;
12.
13.     /**
14.     * Establece el tipo de contenido devuelto por el servlet.
15.     */
16.     private static final String CONTENT_TYPE = "text/html; charset=UTF-8";
17.
18.     /**
19.      * Constructor por defecto.
20.      * @see HttpServlet#HttpServlet()
21.      */
22.     public SaludoSimple() {
23.         super();
24.     }
25.
26. /**
27. * Inicialización del servlet.
28. * @see Servlet#init(ServletConfig)
29. */
30. public void init(ServletConfig config) throws ServletException
31. {
32.     super.init(config);
33. }
34.
35. /**
36. * Método para las peticiones del tipo GET.
37. * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
38. */
39. protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
40.                         doPost(request, response);
41. }
42.
43. /**
44. * Método para las peticiones del tipo POST.
45. * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
46. */
47. protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
48.             String nombre = request.getParameter("nombre");
49.             response.setContentType(CONTENT_TYPE);
50.             PrintWriter out = response.getWriter();
51.             out.println("<html>");
52.             out.println("<head><title>SaludoSimple</title></head>");
53.             out.println("<body>");
54.             out.println("<p>Saludos, "+nombre+"</p>");
55.             out.println("</body></html>");
56.             out.close();
57.  }
58.
59. }
```

Como se aprecia en primer lugar, el servlet hereda de la clase *HttpServlet*[60]. Esto será así la mayoría de las veces en que programemos servlets. La clase *HttpServlet* hereda a su vez de *GenericServlet*[61] e

60 javax.servlet.http.HttpServlet.
61 javax.servlet.http.GenericServlet.

implementa las interfaces *Serializable, Servlet* y *ServletConfig*. Se trata de una clase abstracta, es decir, no va a permitir crear instancias directamente de ella (se tendrán que crear desde clases inferiores).

Entre los diferentes métodos que posee *HttpServlet*, nos van a interesar los siguientes:

public void service (ServletRequest request, ServletResponse response)
protected void service (HttpServletRequest request, HttpServletResponse response)
protected void doGet (HttpServletRequest request, HttpServletResponse response)
protected void doPost (HttpServletRequest request, HttpServletResponse response)

El método público *service()* es una implementación del método homónimo de la clase *GenericServlet* (cuyo método *service()* es abstracto). Este método recibe los objetos de solicitud y respuesta *HttpServletRequest* y *HttpServletResponse* proporcionados por el contenedor JSP/Servlet, e invoca al siguiente método sobrecargado protegido *service()* pasándole ambos objetos, que procesará como objetos asociados a peticiones y respuestas en el ámbito de HTTP.

El método protegido *service()* toma objetos solicitud y respuesta específicos de HTTP, y es invocado por el método público anterior. *HttpServlet* implementa este método para ser un lanzador de solicitudes HTTP.

El método protegido *service()* determinará el tipo de solicitud HTTP recibida (GET, POST, HEAD, DELETE, OPTIONS, PUT o TRACE) y llamará al correspondiente método *doXXX*, donde XXX se corresponde con uno de los posibles tipos de solicitud HTTP.

Por tanto, y resumiendo, la secuencia de llamadas de método cuando el contenedor recibe una solicitud para un servlet es la siguiente:

- El contenedor invoca el método público *service()* del servlet, que es heredado de *HttpServlet*.
- El método público *service()* invoca el método protegido *service()* después de crear los objetos *HttpServletRequest* y *HttpServletResponse*, y pasárselos como argumentos.
- El método protegido *service()* invoca uno de los métodos *doXXX()*, dependiendo del tipo de solicitud HTTP.

Cuando el contenedor recibe una petición de servlet, éste crea una instancia del servlet correspondiente. Después de la instanciación, el contenedor inicializa el servlet invocando al método *init()*. Después de la inicialización, la instancia está preparada para servir solicitudes entrantes. El propósito de este proceso de inicialización es cargar cualquier parámetro de inicio requerido para el servidor.

El contenedor garantiza que antes de que se invoque el método público *service()* de la instancia, el método *init()* podrá ser completado y que, antes de que el servlet sea destruido, su método *destroy()* será ejecutado (en nuestro caso no se dispone de método *destroy()*). El método *init()* no vuelve a ejecutarse en cada petición del usuario. El servlet puede crearse cuando un usuario invoca inicialmente una URL que corresponda con el servlet o cuando el servidor arranca.

Existen dos versiones de *init()*. Una que no tiene argumentos y otra que toma un objeto *ServletConfig* como argumento. La primera de ellas se utiliza cuando el servlet no requiere leer ninguna configuración asociada al servlet y establecida a nivel de servidor (por ejemplo, dentro del descriptor de despliegue *web.xml*).

El método *init()* con argumento, se utiliza cuando el servlet necesita leer una configuración específica del servidor antes de que pueda completar la carga inicial. Por ejemplo, el servlet podría necesitar conocer la configuración de la base de datos, los archivos de contraseña, parámetros del propio servlet establecidos en el descriptor de despliegue.

Si apreciamos con detalle, veremos que en el cuerpo del método *init()* con argumento hay una instrucción *super.init(config)*. Esta llamada es clave, pues permitirá hacer accesible el objeto *ServletConfig* desde cualquier parte del servlet, por lo que siempre es obligatoria.

Una vez ejecutado el método *init()*, se ejecutan, por este orden, el método público *service()*, el método protegido *service()* y el correspondiente método *doXXX()*. El método *doXXX()* que se ejecute se encargará

de devolver la respuesta al peticionador, siendo el contenedor web quien determinará el destino de la respuesta. Esta respuesta se emite en formato HTML. El objeto *request* encapsula la petición, mientras que el objeto *response* se corresponde con la respuesta.

En teoría, no hay nada que impida que un contenedor Web ejecute el ciclo de vida completo del servlet (desde que se crea la instancia hasta que se destruye) cada vez que un servlet es solicitado. En la práctica, los contenedores Web cargan e inicializan los servlets durante el arranque del contenedor o cuando el servlet es invocado por primera vez y mantienen esa instancia de servlet en memoria para atender todas las solicitudes que se reciban. El contenedor puede decidir en cualquier momento liberar la referencia de servlet, finalizando así el ciclo de vida del servlet. Esto podría ocurrir, por ejemplo, si el servlet no ha sido invocado durante un tiempo o si el contenedor se está cerrando. Cuando esto sucede, el contenedor invoca el método *destroy()*.

En el modelo habitual de ciclo de vida de servlet, el contenedor Web crea una única instancia de cada servlet. Pero ¿qué ocurre si el método *service()* del servlet está todavía ejecutándose cuando el contenedor Web recibe otra solicitud? Para los servlets que no implementan la interfaz j*avax.servlet.SingleThreadModel*, el contenedor invoca la misma instancia de servlet en cada hilo de solicitud. Por lo tanto, siempre es posible que el método *service()* sea ejecutado en más de un hilo, siendo necesario que el método *service()* esté a salvo de hilos. Aparte de no acceder a recursos a salvo de hilos (como escribir en archivos), debemos también considerar mantener nuestros servlets sin estado (es decir, no definir ningún atributo en sus clases de servlet). Al definir propiedades de instancias en nuestros servlets, debemos asegurarnos de que tales propiedades son manipuladas de modo seguro.

Seguidamente comento las líneas de código más importantes del ejemplo anterior.

```
public class SaludoSimple extends HttpServlet
```

Declaración de la clase, estableciendo que hereda de HttpServlet, imprescindible para ejecutar los servlets en el ámbito del servidor de aplicaciones.

```
private static final String CONTENT_TYPE = "text/html; charset=UTF-8";
```

Se establece mediante una cadena (*String*) constante (*final*) de clase (*static*) el tipo de formato de documento que se devuelve al usuario. En este caso, texto HTML con codificación UTF-8.

```
  public SaludoSimple() {
     super();
  }
```

Constructor de la clase, que requiere llamada al método *super()* con objeto de que se cree la instancia correspondiente a la superclase (en otras palabras, que puedan usarse coherentemente los método heredados).

```
public void init(ServletConfig config) throws ServletException
{
   super.init(config);
}
```

El método *init()* recibe como argumento un objeto de tipo *ServletConfig*, proporcionado por el contenedor JSP/Servlet, y que contiene los parámetros de inicio del mismo, establecidos en web.xml. La clase *ServletConfig* proporciona los métodos[62] adecuados para acceder a estos parámetros. En el caso que se muestra, se limita a llamar al método *init()* de la superclase proporcionando el objeto *ServletConfig*, con el propósito de que los métodos de la superclase puedan emplearse de forma coherente.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                    doPost(request, response);
}
```

62 Los métodos son *getInitParameter()* y *getInitParameterNames()*.

Este método lo ejecuta el método protegido *service()* de la superclase (*HttpServlet*) si la petición recibida es de tipo *GET*. Puede lanzar *ServletException* o *IOException*. Aunque puede contener el código correspondiente, se opta por ejecutar *doPost()* con el propósito de que, si el procesamiento es el mismo independientemente del tipo de petición, no se duplique código.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
```

Este método será el que contenga el código correspondiente al procesamiento de las peticiones tanto de tipo *GET* como de tipo *POST*.

```
String nombre = request.getParameter("nombre");
```

Se recupera el valor del parámetro *nombre* de la petición, almacenándola en la variable correspondiente. Se almacenará un *null* si no existe el parámetro.

```
response.setContentType(CONTENT_TYPE);
```

Se establece el tipo de formato de documento devuelto mediante el método *setContentType()*.

```
PrintWriter out = response.getWriter();
```

Se recupera mediante el método getWriter() el flujo empleado para responder al usuario y que viene referenciado en response. Es de tipo *PrintWriter*.

```
out.println("<p>Saludos, "+nombre+"</p>");
```

Se envía al usuario una cadena que contiene un saludo en el que se ha incluido el valor del parámetro *nombre*.

```
out.close();
```

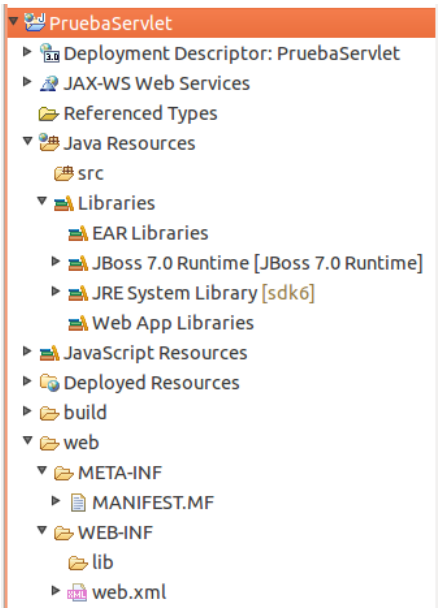
Se cierra el flujo asociado al usuario.

Veamos cómo crear el servlet en Eclipse. Para ello se comienza por crear un nuevo proyecto web dinámico, *File → New → Dinamic Web Project*. Se le da el nombre de *PruebaServlet*, y se establece como *target runtime* el servidor *JBoss 7*.

Se pulsa el botón *Next*. Seguidamente solicita las carpetas donde se almacenarán las fuentes del proyecto y el código de las clases compiladas. Se dejan ambas carpetas por defecto.

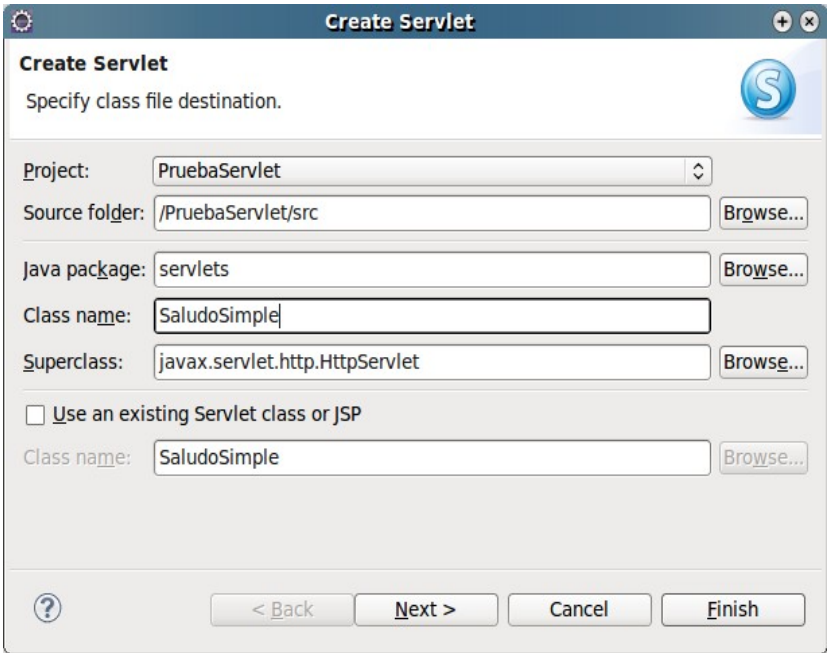
Se pulsa *Next*, y se solicitará el nombre del contexto raíz y el nombre de la carpeta que almacenará el contenido web. Como contexto raíz de la aplicación se deja *pruebaservlet* y *web* como carpeta de contenido web. Se deja marcada (y se marca si no lo está) la casilla que posibilita que se genere de forma automática el archivo *web.xml*.

Se pulsa el botón *Finish*. Llegados a este punto, se habrá creado un nuevo proyecto.

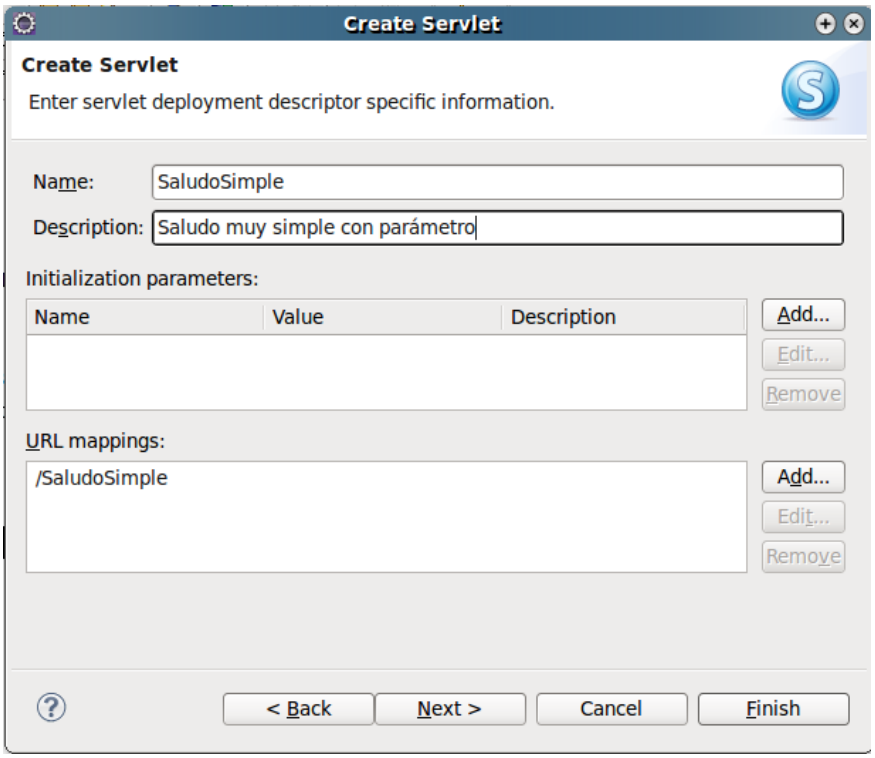


El siguiente paso es crear un paquete en el que se registrará la clase de tipo servlet que se va a diseñar. Para ello se selecciona el proyecto y accediendo al menú contextual se escoge la opción *New → Package*. Se proporciona *servlets* como nombre del paquete.

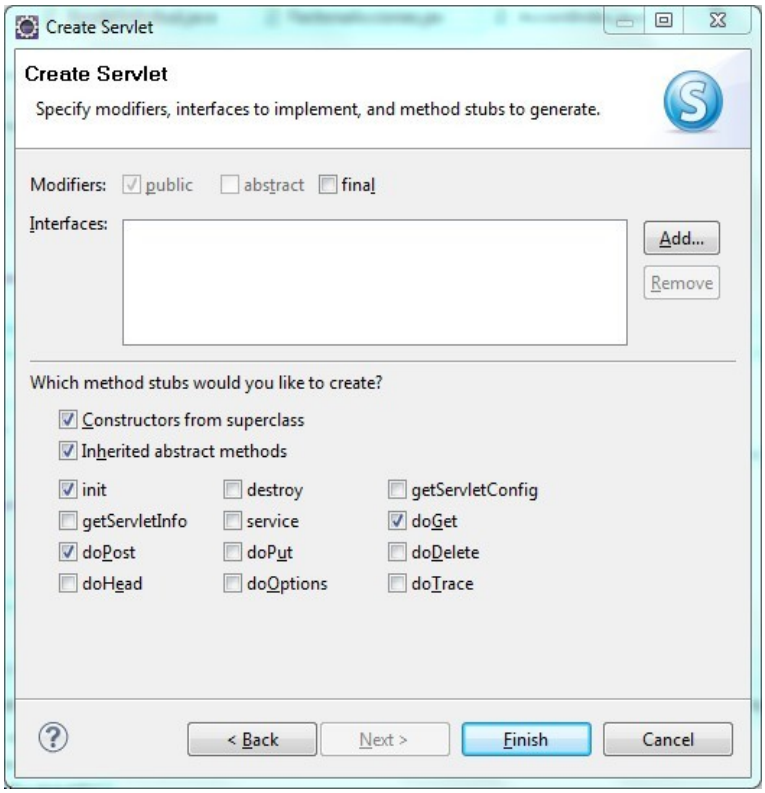
Se pulsa *Finish* y ya se tendrá creado el paquete. Ahora se procede a crear el servlet. Se escoge, también desde el menú contextual asociado al proyecto, *New → Servlet*. Aparece un cuadro de diálogo solicitando los datos básicos para crear el servlet. Se le dará a la clase el nombre de *SaludoSimple*, comprobaremos que pertenece al paquete servlets, que hereda de la clase *javax.servlet.http.HttpServlet*.



Se pulsa el botón *Next*, y aparece un nuevo cuadro de diálogo en el que añadiremos una descripción del servlet que se va a diseñar, así como verificar el nombre lógico que se asociará a esa clase de servlet, que en nuestro caso será el propuesto por el asistente, */SaludoSimple* (aunque podía haberse escogido cualquier otro).



Se pulsa el botón *Next*, y el asistente solicita que se establezcan qué métodos se van a sobreescribir. Marcamos *init()*, *doGet()* y *doPost()*.



Se pulsa el botón *Finish* y el código generado tras el proceso es el siguiente:

```
package servlets;

import java.io.IOException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class SaludoSimple
 */
public class SaludoSimple extends HttpServlet {
	private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public SaludoSimple() {
        super();
        // TODO Auto-generated constructor stub
    }

	/**
	 * @see Servlet#init(ServletConfig)
	 */
	public void init(ServletConfig config) throws ServletException {
		// TODO Auto-generated method stub
	}

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub
	}

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub
	}

}
```

Ahora será necesario darle contenido, por ejemplo estableciendo el código que debe ejecutarse cuando se reciban peticiones al servlet. Para empezar modificamos las instrucciones *import* para que hagan directamente referencia a paquetes completos y no a clases individualmente. Después se añade la constante de clase *CONTENT_TYPE* para que establezca el formato de documento generado como respuesta y el tipo de codificación de caracteres que se aplicará. Después se añadirá la llamada al método *init()* de la superclase en el método *init()* de este servlet, proporcionando como argumento el objeto *ServletConfig*. En el método *doGet()* se incluirá como código la llamada al método *doPost(request,response)*. Y finalmente, en el método *doPost()*, se añaden las líneas que posibilitan mostrar al usuario un saludo incluyendo la cadena correspondiente al valor del parámetro *nombre*, que debe proporcionarse de forma explícita en la petición al servlet. Se comprueba la correcta elaboración del servlet desplegando el proyecto en el servidor y realizando la petición *http://localhost:8080/pruebaservlet/SaludoSimple?nombre=loquesea*. El código resultante debe coincidir con el comentado al principio de este apartado.