Objetivos

- Aprender a utilizar la arquitectura MVC en proyectos JSP.

Conceptos

1. La arquitectura MVC

La arquitectura MVC (Modelo – Vista – Controlador) es un patrón de arquitectura de software que propone una división de las aplicaciones en tres partes:

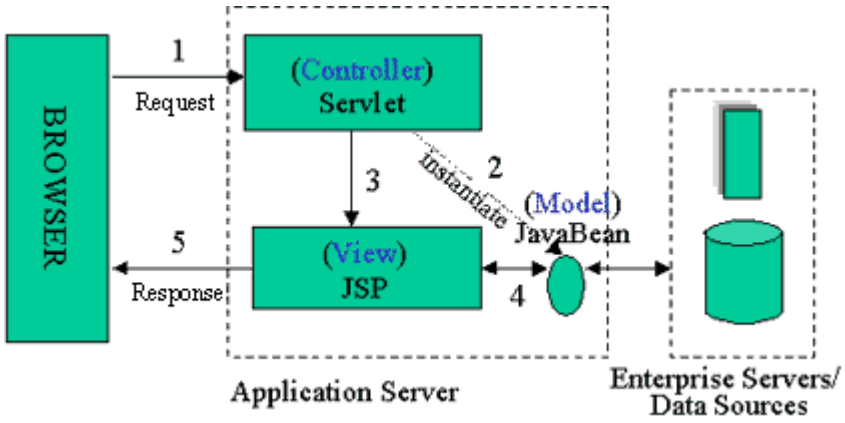
- Modelo. Representación específica de la información, que encapsula también la lógica que se ocupa de procesar los datos. Esto incluye procesar la lógica de negocio, guardar datos y desempeñar diversas funciones relativas al procesamiento de estos. En esta capa es donde se hace la mayor parte del trabajo.
- Vista. Presentación de los datos del modelo en un formato adecuado para interactuar con el usuario.
- Controlador. Encargado de controlar el flujo de datos, y determinar a dónde dirigir las peticiones desde y por el sistema.

En JSP, hay tres tipos de arquitecturas de aplicaciones web que *mapean* cada una de las partes anteriores en distintos componentes:

1. 1 capa, denominada también página-céntrica. En esta arquitectura se implementan el modelo, la vista y el controlador en la misma página JSP.

2. 2 capas. En esta arquitectura el modelo se implementa con un *JavaBean* mientras que la vista y el controlador son implementados por una página JSP.

3. 3 capas. En esta arquitectura el modelo es implementado por un *JavaBean*, el controlador por un *servlet* y la vista por una página JSP. En la Figura 4 se da un esquema general de la arquitectura MVC a tres capas.



Arquitectura MVC de tres capas [1].

Para más detalles consultar [1, 2].

2. Creación de servlets en NetBeans

Los pasos que se detallan en este apartado se llevan a cabo dentro de un proyecto "*Web Application*" de la categoría "*Java Web*".

Para poder crear un *servlet* en *NetBeans*, seleccionamos "*New File*" y dentro de la categoría "*Web*" seleccionamos *servlet*, como en la Figura 1.

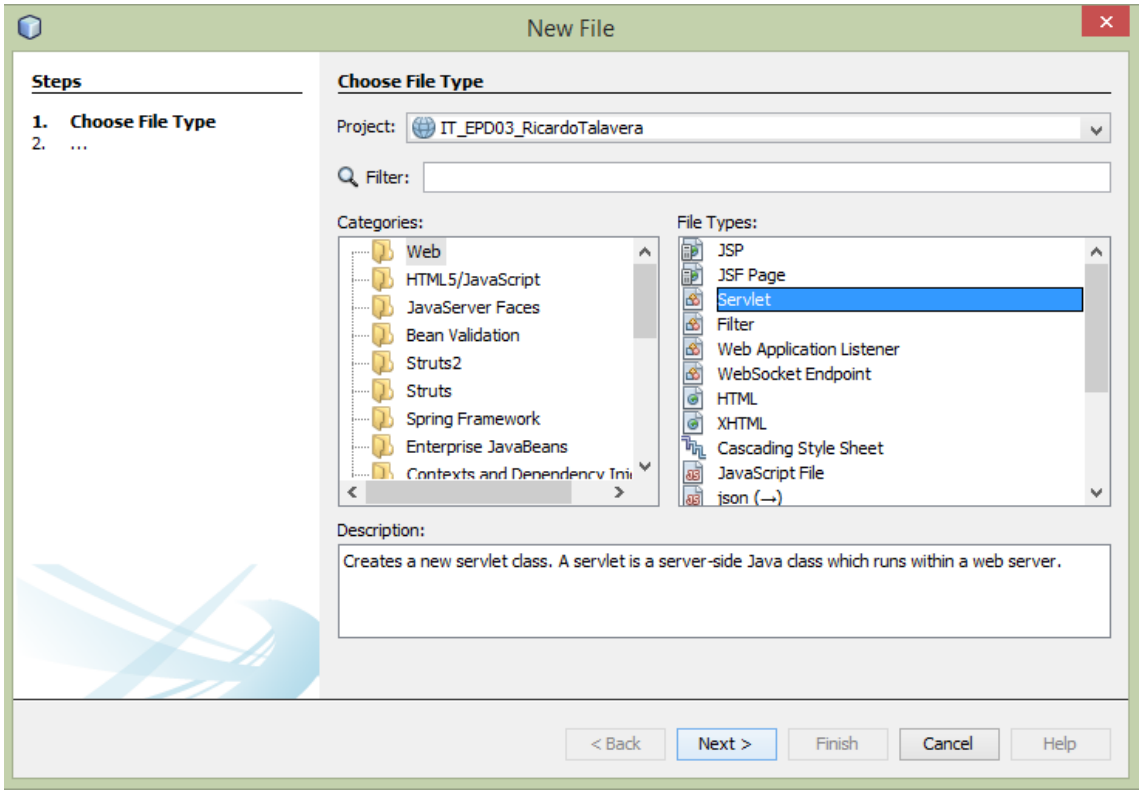


Figura 1: Creación de un servlet en NetBeans.

Pulsamos *Next* y nos aparecerá la ventana mostrada en la Figura 2, donde se podrá nombrar el *servlet* y seleccionar el paquete donde se ubicará el *servlet* (**es obligatorio especificar el paquete donde estará el *servlet***).

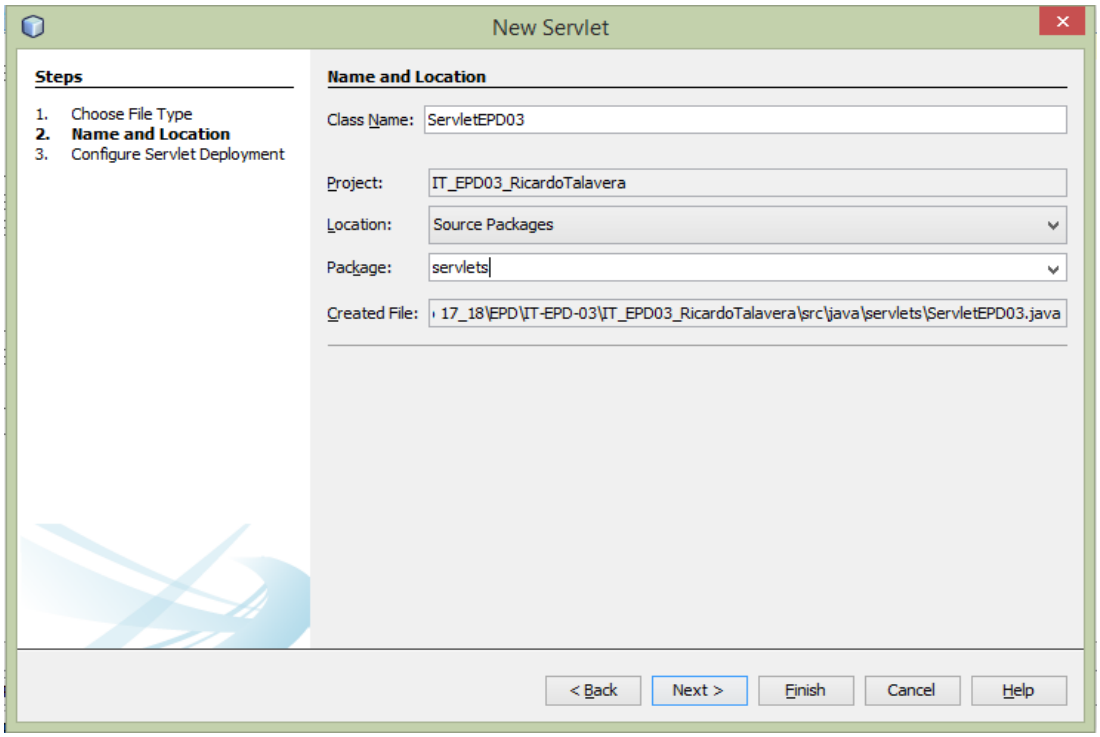


Figura 2: Selección del nombre y localización del servlet.

Pulsamos *Next* y nos aparecerá la ventana de configuración para el despliegue del *servlet*. En esta ventana es muy importante marcar la casilla "*Add information to deployment decriptor (web.xml)*" para que se añada al archivo de configuración todo lo relacionado con el servlet. Pulsando *Next* se creerá el esqueleto del nuevo *servlet*.

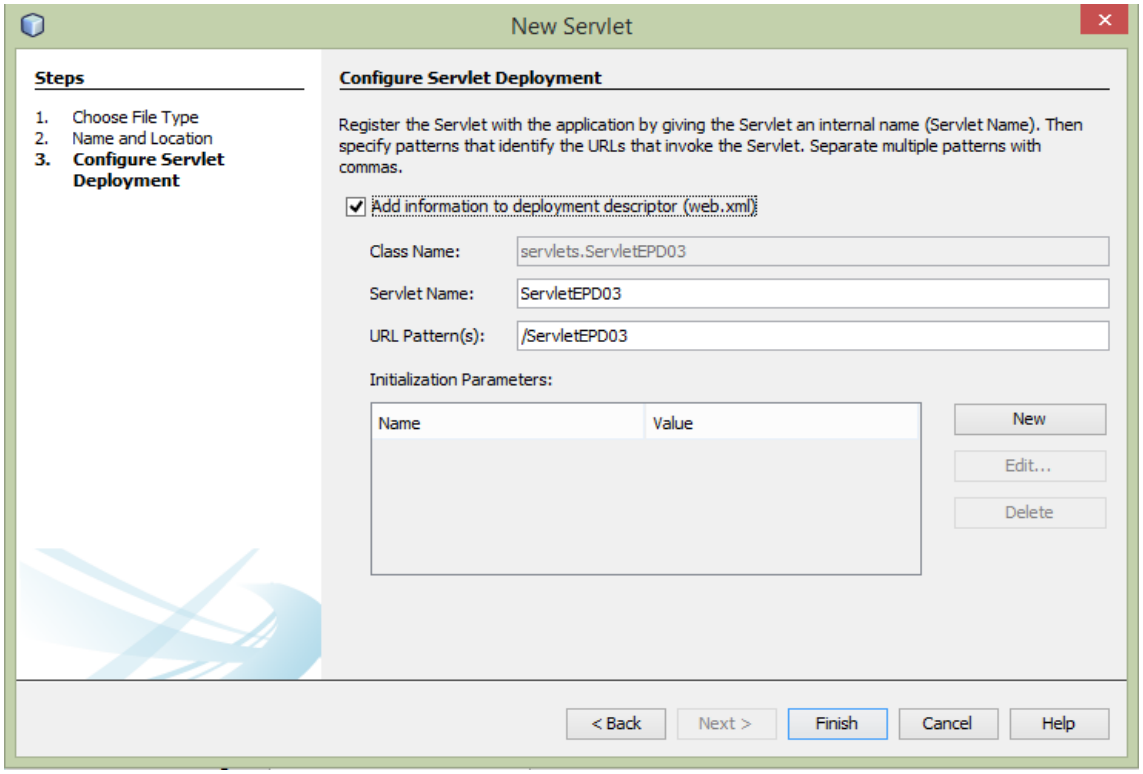


Figura 3: Configuración del uso del Servlet.

3. Edición y configuración del servlet.

Una vez creado el servlet que actuará como controlador de nuestra aplicación, tendremos que configurar su funcionamiento.

Lo primero que debemos observar es que el servlet se crea con 4 métodos. Estos son:

- doPost y doGet: métodos para la recepción de peticiones desde un formulario mediante GET o POST (se procesa de la misma forma)
- getServletInfo: devuelve una breve descripción del servlet.
- processRequest: este es el método principal ya que procesará las peticiones que lleguen por GET o POST y emitirá una respuesta. Es por ello por lo que recibe dos parámetros: request y response.

Nos centraremos en la edición y configuración de este último método, processRequest. Es recomendable que lo primero que hagamos sea comprobar si existe una sesión ya creada, lo cual indicará si es la primera vez que el usuario esta accediendo a la aplicación, o si ya ha interactuado con esta. Para ello utilizaremos el objeto session:

```
HttpSession session=request.getSession(false);
```

Se pone a false para que en caso de que no exista, se indica que no la cree en y en cuyo caso devolvería null. Una vez comprobada que la sesión existe, se recogerían por ejemplo los parámetros de un formulario y se prepararían atributos a enviar en la respuesta de la petición.

A continuación, para poder emitir la respuesta necesitamos recoger el contexto del servlet, dentro del contenedor, ya que en una JVM habrá varios servlet. Necesitaremos para ello un objeto del tipo ServletContext:

```
//recogemos el contexto (ubicación) de nuestro servlet
ServletContext sc=getServletContext();
```

Después, haremos uso de un objeto de tipo RequestDispatcher. Es un objeto que actúa como wrapper (envoltorio), recibiendo peticiones del cliente y reenviándolas a cualquier otro recurso de la aplicación (encapsula para poder enviar objetos de tipo request y response). Necesita un argumento, la url, que es donde queremos dirigirnos. Esta url tiene que venir precedido de "/" para indicarle que es una ruta relativa:

```
RequestDispatcher rd=sc.getRequestDispatcher("/Ej01/pagina.jsp");
```

Finalmente, a través del objeto requestDispatcher, realizaremos el reenvió a la página deseada pasándole los objetos request y response:

```
rd.forward(request, response);
```

A continuación, se muestra un ejemplo de cómo quedaría el método completo:

```
protected void processRequest(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        HttpSession session=request.getSession(false);
        if(session!=null){
            //recogida de parámetros
            String usuario=request.getParameter("nombre");
            //a donde queremos ir
            String url="/Ej01/pagina.jsp";
            //creamos los atributos que necesitemos
            request.setAttribute("usuario", usuario);
            //recoger el contexto del servlet
            ServletContext sc=getServletContext();
            //encapsulación para poder enviar request, response y reenvio
            RequestDispatcher rd=sc.getRequestDispatcher(url);
            rd.forward(request, response);
        }
    }
```

Experimentos

Ex1 (10 mins): A partir del material del experimento 2 de la anterior práctica, simule una arquitectura de 3 capas incluyendo un *servlet* que actúe como controlador. Este será el que recoja los datos y redirija a la página que el usuario haya seleccionado. Utilice los objetos explicados anteriormente e intente comprender su funcionamiento.

Ex2 (50 mins). En este experimento implementaremos una librería electrónica haciendo uso de una arquitectura de tres capas MVC. El usuario podrá elegir una serie de libros (de cada uno se pueden solicitar más de un ejemplar) que serán incorporados al carro de la compra. Una vez haya finalizado la selección de libros, se procederá a solicitar su compra, y se mostrará una tabla con los libros solicitados, el total asociado a cada título y, finalmente, el total de la compra.

Los componentes de la tienda correspondientes a la capa de la vista son las páginas JSP *Tienda.jsp*, *CarroCompra.jsp* y *Compra.jsp*. El controlador es implementado por el *servlet* llamado *LibreriaServlet* y finalmente el modelo de datos vendrá representado por el *JavaBean Libro.java*. El acceso a los datos se llevará a cabo simulando consultas a la base de datos mediante la clase *Almacen.java*.

El código de cada uno de los ficheros es el siguiente:

1. *Tienda.jsp*

```
<%@page contentType="text/html" pageEncoding="ISO-8859-1"%>
<%@page import="java.util.List" %>
<%@page import="java.util.ArrayList" %>
<%@page import="libreria.Almacen" %>
<%@page import="libreria.Libro" %>

<%!
    List<Libro> listaLibros = Almacen.consultaLibrosDisponibles();
%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Tienda Electrónica de Libros</h1>
        <form name="formularioTienda" action="LibreriaServlet" method="post">
            <b>Libros</b>
            <select name="seleccionLibros">
              <% for (int i = 0; i < listaLibros.size(); i++) {
                     Libro libro = listaLibros.get(i);
                     String txtLibro = libro.getTitulo() + "; "
                             + libro.getAutor() + "; " + libro.getPrecio();
              %>
             <option value="<%=String.valueOf(libro.getIsbn())%>"><%=txtLibro%></option>
             <% } %>
            </select>
            <input type="hidden" name="Accion" value="agregar">
            <input type="submit" name="Agregar" value="Agregar Libro">
        </form>
        <hr>
            <jsp:include page="CarroCompra.jsp" flush="true"/>
    </body>
</html>
```

2. *CarroCompra.jsp*

```
<%@ page session="true"%>
<%@page import="java.util.*" %>
<%@page import="libreria.*" %>
<%
        List<String> listaIsbns =
                (List<String>) session.getAttribute("tienda.carro");
        List<Libro> listaCompra =
                Almacen.consultaListaLibrosSolicitados(listaIsbns);
        if (listaCompra != null && (listaCompra.size() > 0)) {
%>
<center>
    <table border=1 cellspacing=1 cellpadding=2 width="100%" bgcolor="#FFFFFF">
        <tr>
            <td><b>Titulo</b></td>
            <td><b>Autor</b></td>
            <td><b>Precio</b></td>
            <td><b>Cantidad</b></td>
            <td></td>
        </tr>
        <%
    for (int idx = 0; idx < listaCompra.size(); idx++) {
        Libro libro = listaCompra.get(idx);
        %>
        <tr>
            <td><b><%= libro.getTitulo()%></b></td>
            <td><b><%= libro.getAutor()%></b></td>
            <td><b><%= libro.getPrecio()%></b></td>
            <td><b><%= libro.getCantidad()%></b></td>
            <td></td>
        </tr>
        <%
    }%>
    </table>
</center>
<p>

<form name="formularioComprar" action="LibreriaServlet" method="post">
    <input type="hidden" name="Accion" value="comprar">
    <input type="submit" name="Comprar" value="Comprar Libros">
</form>
<%
    }
%>
```

3. *Compra.jsp*

```
<%@page contentType="text/html" pageEncoding="ISO-8859-1"%>
<%@ page session="true" import="java.util.*, libreria.*" %>
<%
        List<String> listaIsbns =
                (List<String>) session.getAttribute("tienda.carro");
        List<Libro> listaCompra =
                Almacen.consultaListaLibrosSolicitados(listaIsbns);
%>

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>LIBROS COMPRADOS:</h1>
        <center>
            <table border=1 cellspacing=1 cellpadding=2 width="100%" bgcolor="#FFFFFF">
                <tr>
                    <td><b>Titulo</b></td>
                    <td><b>Autor</b></td>
                    <td><b>Precio</b></td>
                    <td><b>Cantidad</b></td>
                    <td><b>Subtotal</b></td>
                    <td></td>
                </tr>
                <%
        if (listaCompra != null) {
            double suma = 0.;
            for (int idx = 0; idx < listaCompra.size(); idx++) {
                Libro libro = listaCompra.get(idx);
                double subtotal = libro.getPrecio() * libro.getCantidad();
                suma += subtotal;
                %>
                <tr>
                    <td><b><%= libro.getTitulo()%></b></td>
```

```
                    <td><b><%= libro.getAutor()%></b></td>
                    <td><b><%= libro.getPrecio()%></b></td>
                    <td><b><%= libro.getCantidad()%></b></td>
                    <td><b><%= subtotal%></b></td>
                </tr>
                <%
                    }
                %>
                <br>
                <tr>
                    <td><b>TOTAL</b></td>
                    <td><b></b></td>
                    <td><b></b></td>
                    <td><b></b></td>
                    <td><b><%= suma%></b></td>
                </tr>
                <%
                } else {%>
                <tr>
                    <td><b>TOTAL</b></td>
                    <td><b></b></td>
                    <td><b></b></td>
                    <td><b>0</b></td>
                    <td></td>
                </tr>
                <%        }
                %>
            </table>
        </center>
    </body>
</html>
```

4. LibreriaServlet.java

```
package libreria.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import libreria.Almacen;

public class LibreriaServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        HttpSession session = request.getSession(false);
        if (session != null) {
            String accion = request.getParameter("Accion");
            if (accion.equals("agregar")) {
                List<String> listaIsbns = (List<String>)
                        session.getAttribute("tienda.carro");
                String isbn = request.getParameter("seleccionLibros");
                if (Almacen.consultaDisponibilidadLibro(Integer.parseInt(isbn))) {
                    if (listaIsbns == null) {
                        listaIsbns = new ArrayList<String>(10);
                    }
                    listaIsbns.add(isbn);
                    session.setAttribute("tienda.carro", listaIsbns);
                } else {
                   PrintWriter out = response.getWriter();
                   out.print("" + isbn);
                   out.close();
                }
                String url = "/Tienda.jsp";
                ServletContext sc = getServletContext();
                RequestDispatcher rd = sc.getRequestDispatcher(url);
                rd.forward(request, response);
            } else if (accion.equals("comprar")) {
                String url = "/Compra.jsp";
                ServletContext sc = getServletContext();
                RequestDispatcher rd = sc.getRequestDispatcher(url);
                rd.forward(request, response);
            }
        }

    }
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    public String getServletInfo() {
        return "Short description";
    }
}
```

5. Libro.java

```
package libreria;

public class Libro {
    private String titulo = null;
    private String autor = null;
    private int cantidad = -1;
    private int isbn = -1;
    private double precio = Double.NaN;
    public Libro() {
        this.setAutor("");
        this.setTitulo("");
        this.setPrecio(0.);
        this.setCantidad(0);
        this.setIsbn(0);
    }
    public String getTitulo() {
        return titulo;
    }
    public boolean equals(Object olibro) {
        boolean eq = false;
        Libro libro = (Libro) olibro;
        if (this.getTitulo().equals(libro.getTitulo())
                && this.getAutor().equals(libro.getAutor())
                && this.getPrecio() == libro.getPrecio()
                && this.getIsbn() == libro.getIsbn()) {
            eq = true;
        }
        return eq;
    }
    public void setIsbn(int isbn) {
        this.isbn = isbn;
    }
    public int getIsbn() {
        return this.isbn;
    }
    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }
    public String getAutor() {
        return autor;
    }
    public void setAutor(String autor) {
        this.autor = autor;
    }
    public int getCantidad() {
        return cantidad;
    }
    public void incrementaCantidad() {
        this.cantidad++;
    }
    public void disminuyeCantidad() {
        this.cantidad--;
    }
    public void setCantidad(int cantidad) {
        this.cantidad = cantidad;
    }
    public double getPrecio() {
        return precio;
    }
    public void setPrecio(double precio) {
        this.precio = precio;
    }
}
```

6. Almacen.java

```
package libreria;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.ListIterator;
import java.util.Map;
import java.util.Set;

public class Almacen {
    public static List<Libro> consultaListaLibrosSolicitados(
            List<String> listaIsbns) throws Exception {
        Map<String, Libro> conjuntoLibros =
                new HashMap<String, Libro>(listaIsbns.size());
        ListIterator<String> iterator = listaIsbns.listIterator();
        while (iterator.hasNext()) {
            String isbn = iterator.next();
            Libro libro = null;
            if (!conjuntoLibros.containsKey(isbn)) {
                libro = Almacen.consultaLibro(Integer.parseInt(isbn));
            } else {
                libro = conjuntoLibros.get(isbn);
                conjuntoLibros.remove(isbn);
                libro.incrementaCantidad();
            }
            conjuntoLibros.put(isbn, libro);
        }
        //pasamos a lista
        Set<String> conjuntoIsbns = conjuntoLibros.keySet();
        Iterator<String> it = conjuntoIsbns.iterator();
        List<Libro> listaLibros = new ArrayList<Libro>(conjuntoIsbns.size());
        while (it.hasNext()) {
            listaLibros.add(conjuntoLibros.get(it.next()));
        }

        //pasamos a
        return listaLibros;
    }
    public static Libro consultaLibro(int isbn) throws Exception {
        if (!Almacen.consultaDisponibilidadLibro(isbn)) {
            throw new Exception("Libro con ISBN " + isbn + " no esta disponible.");
        }
        boolean disponible = false;
        List<Libro> lista = Almacen.consultaLibrosDisponibles();
        Iterator it = lista.iterator();
        Libro libro = null;
        while (it.hasNext() && !disponible) {
            libro = (Libro) it.next();
            if (libro.getIsbn() == isbn) {
                disponible = true;
                libro.setCantidad(1);
            }
        }
        return libro;
    }

    public static boolean consultaDisponibilidadLibro(int isbn) {
        boolean disponible = false;
        List<Libro> lista = Almacen.consultaLibrosDisponibles();
        Iterator it = lista.iterator();
        while (it.hasNext() && !disponible) {
            Libro libro = (Libro) it.next();
            if (libro.getIsbn() == isbn) {
                disponible = true;
            }
        }
        return disponible;
    }
```

```
public static List<Libro> consultaLibrosDisponibles() {
        List<Libro> lista = new ArrayList<Libro>(10);
        //libro 1
        Libro libro1 = new Libro();
        libro1.setAutor("Arturo Pérez Reverte");
        libro1.setTitulo("El Capitán Alatriste");
        libro1.setPrecio(10.00);
        libro1.setCantidad(0);
        libro1.setIsbn(101);
        lista.add(libro1);
        //libro 2
        Libro libro2 = new Libro();
        libro2.setAutor("Neil Gaiman");
        libro2.setTitulo("Humos y Espejos");
        libro2.setPrecio(12.00);
        libro2.setCantidad(0);
        libro2.setIsbn(102);
        lista.add(libro2);
        //libro 3
        Libro libro3 = new Libro();
        libro3.setAutor("Lewis Carroll");
        libro3.setTitulo("Alicia en el País de las Maravillas");
        libro3.setPrecio(15.00);
        libro3.setCantidad(0);
        libro3.setIsbn(103);
        lista.add(libro3);
        //libro 4
        Libro libro4 = new Libro();
        libro4.setAutor("Philip K. Dick");
        libro4.setTitulo("Cuentos Completos I");
        libro4.setPrecio(14.50);
        libro4.setCantidad(0);
        libro4.setIsbn(104);
        lista.add(libro4);
        //libro 5
        Libro libro5 = new Libro();
        libro5.setAutor("H. P. Lovecraft");
        libro5.setTitulo("La Llamada de Cthulhu");
        libro5.setPrecio(11.00);
        libro5.setCantidad(0);
        libro5.setIsbn(105);
        lista.add(libro5);
        //libro 6
        Libro libro6 = new Libro();
        libro6.setAutor("Arturo Pérez Reverte");
        libro6.setTitulo("La Piel del Tambor");
        libro6.setPrecio(18.00);
        libro6.setCantidad(0);
        libro6.setIsbn(106);
        lista.add(libro6);
        //libro 7
        Libro libro7 = new Libro();
        libro7.setAutor("Eduardo Punset");
        libro7.setTitulo("El Viaje a la Felicidad");
        libro7.setPrecio(13.50);
        libro7.setCantidad(0);
        libro7.setIsbn(107);
        lista.add(libro7);
        //libro 8
        Libro libro8 = new Libro();
        libro8.setAutor("Juan Pérez Mercader");
        libro8.setTitulo("¿Qué Sabemos del Universo?");
        libro8.setPrecio(11.50);
        libro8.setCantidad(0);
        libro8.setIsbn(108);
        lista.add(libro8);
        return lista;
    }
}
```

Tras analizar y probar el código, responda a las siguientes preguntas:

- Describa el cometido de cada uno de los componentes.
- ¿Es el *servlet* de la aplicación el que controla en todo el momento el flujo de interacción con la misma? ¿Cómo consigue tal cosa?
- ¿Respetan los componentes de la aplicación los roles asignados por el patrón arquitectónico MVC? ¿Dónde se produce divergencias? ¿Cómo solventar las mismas?

Bibliografía básica

1. Understanding JavaServer Pages Model 2 architecture. JavaWorld. http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html
2. Professional JSP. Karl Avedal et al. Wrox Press, 2000. Capítulo 12. http://www.javacoffeebreak.com/books/samples/professionaljsp/index.html

Ejercicios

EJ1 (50 mins) Implementar una aplicación basada en la arquitectura de dos capas para simular un sistema de gestión de automatizado de aparcamientos en zona azul. Para ello, deberá registrarse, por cada coche:

- Matrícula
- Modelo
- Hora de entrada
- Hora de salida
- Tiempo permitido

La única acción que se ha de implementar en la página principal será la de visualizar la lista de vehículos registrados por la aplicación. De esta forma, NO se pide que se implemente el alta de vehículos, SÓLO que se muestren los que haya registrados. Ejemplo:

Matrícula	Modelo	Hora de entrada	Hora de salida	Tiempo permitido
12784HIH	BMW	10:05	--	30
76234AAC	Toyota	10:07	10:35	30
32162BAQ	Audi	10:45	--	90
87823CDA	Mercedes	10:46	11:05	15

Para simular el acceso a la información de los vehículos, se pueden leer los datos de un archivo o generar dichos datos directamente en una clase específica para esto (de forma similar a como se ha hecho en el experimento).

Problemas

P1. (40 mins) Implementar el ejercicio 1 utilizando una arquitectura a 1 capa.

P2. (40 mins) Se desea mostrar agrupados en diferentes tablas los vehículos que han excedido el tiempo permitido en la zona azul y los que no. Para ello se va a añadir a lo realizado en el ejercicio 1 y el problema 1, una nueva vista que muestre dichas agrupaciones. Es requisito indispensable que sólo se muestre una tabla a la vez, pudiendo pasar de una tabla a otra usando unos botones en la página. (Pista: tanto la primera tabla como la segunda han de mostrarse en la misma vista)

P3. (40 mins) Vamos ahora a ofrecer la posibilidad de buscar vehículos a la aplicación. Se quieren realizar dos búsquedas distintas:

1. Localizar coches en concreto, búsqueda que se hará por matrícula introducida por teclado por el usuario. El criterio de búsqueda será que la matrícula empiece por lo que haya en dicha caja de texto.

2. Vehículos que todavía se encuentran en el aparcamiento.

Añada esta funcionalidad a las aplicaciones desarrolladas en el ejercicio 1 y en los problemas 1 y 2.

P4. (60 mins) Implementar el ejercicio 1 utilizando una arquitectura a 3 capas utilizando también las vistas implementadas en los problemas 2 y 3.

Ampliación de la bibliografía

1. JavaServer pages. Bergsten, Hans. O'Reilly, 2001.
2. JavaServer Pages: pocket reference. Bergsten, Hans. O'Reilly, 2001.
3. JavaServer Pages: Manual de usuario y tutorial. Froufe Quintas, Agustín. Ra-ma, 2002.
4. Beginning JavaServer Pages. Vivek Chopra y otros. , 2005.