Práctica 1: HTML básico

El objetivo de esta práctica es familiarizarse con el lenguaje HTML, por lo que se trabajará con un editor de ficheros ASCII como el **Notepad**, aunque existen otros métodos para conseguir el mismo resultado (editores de HTML o conversores desde otras aplicaciones).

Esta práctica consiste en un *website* que se irá completando progresivamente. Crea una carpeta en tu cuenta y descarga en ella el archivo *Practica01.zip* de la web de Informática III.

Ejercicio 1.1: Estructura del HTML; FRAMES.

Crea la página *index.html* con las siguientes características:

- Título: CULTURSITE.
- Dos *frames*: margen izquierdo (20% de la pantalla) y parte principal (80%).
- Contenido del margen izquierdo: página *menu.html*. Target de nombre: *navegador*.
- Contenido de la parte principal: página *principal.html*. Target de nombre: *main*.

Ejercicio 1.2: TAGs para estructurar y formatear texto; LISTAS.

Formatea el texto de *instituciones.html* según se muestra.

INSTITUCIONES VASCAS

- Administración de Justicia
- Araneko
- Asociación de Municipios Vascos-Eudel
- Ayuntamientos
- Consejo de Relaciones Laborales
- Consejo Economico y Social Vasco
- Consejo Superior de Cooperativas de Euskadi
- Consejo Vasco de la Abogacía
- Cuerpo consular de Bilbao
- Diputación Foral de Alava
- Diputación Foral de Bizkaia
- Diputacion Foral de Gipuzkoa
- Gobierno Vasco
- Juntas Generales de Alava
- Juntas Generales de Bizkaia
- Juntas Generales de Gipuzkoa
- Parlamento Vasco
- Tribunal Vasco de Cuentas Públicas

Consejo de Relaciones Laborales
Es una institución pública de la **Comunidad Autónoma del País Vasco** , integrada paritariamente por personas designadas por las confederaciones sindicales y empresariales. Sus fines básicos son dos:

1. Promover el encuentro y diálogo entre dichas confederaciones.
2. Actuar como órgano consultivo en materia sociolaboral respecto del Gobierno y del Parlamento Vasco.

Tiene naturaleza pública, personalidad jurídica propia y actúa con plena independencia en el desarrollo de sus funciones. Fue creado por la Ley 9/1981, de 30 de septiembre, si bien su regulación actual se encuentra en la Ley 11/1997, de 27 de junio (*Boletín Oficial del País Vasco nº 142/1997*, de 28 de julio).

Consejo Económico y Social Vasco

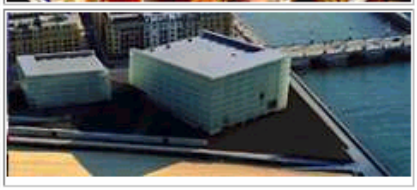
El Consejo Económico y Social Vasco es un ente consultivo del Gobierno y del Parlamento cuyo objetivo es hacer efectiva la participación de los distintos intereses económicos y sociales en la política económica del País Vasco. El Consejo goza de personalidad jurídica propia, distinta de la de la Administración de la Comunidad Autónoma, con plena capacidad e independencia para el ejercicio de sus funciones.

Formatos:

- **Color de fondo**: #8CACD4
- **Texto de título**: fuente arial, color blanco, tamaño 2 puntos mayor que el estándar.
- **Resto del documento**: listas numeradas y no numeradas; negrita; subrayado; itálica (cursiva); fuente arial / por defecto.

Ejercicio 1.3: IMÁGENES; TABLAS.

Añade la última columna de la tabla de *turismo.html* y modifica la tabla para que muestre este aspecto:

<table>
<tr><td rowspan="5">Gastronomía</td><td>Restaurantes</td><td rowspan="5">  </td></tr>
<tr><td>Sidrerías y asadores</td></tr>
<tr><td>Bodegas de vino y txakoli</td></tr>
<tr><td>Productos típicos</td></tr>
<tr><td>Escuelas de hostelería</td></tr>
<tr><td rowspan="4">Cultura</td><td>archivos y bibliotecas</td><td rowspan="4">  </td></tr>
<tr><td>museos</td></tr>
<tr><td>palacios de congresos</td></tr>
<tr><td>ferias de muestras</td></tr>
<tr><td rowspan="3">Ocio</td><td>Entretenimiento y diversión</td><td rowspan="3">  </td></tr>
<tr><td>Ocio cultural</td></tr>
<tr><td>Excursiones y deporte</td></tr>
</table>

Imágenes: *receta1.jpg*, *receta2.jpg*, *cultura.jpg*, *playa.jpg*.

Ejercicio 1.4: LINK en IMAGEN (mapa de bits); TARGETs especiales.

Añade links en la imagen de *principal.html*:

- Las dos imágenes de la izquierda: link a *turismo.html* en ventana nueva.
- Las dos imágenes de la derecha: link a *instituciones.html* en ventana actual completa.

Práctica 2: Formularios en HTML

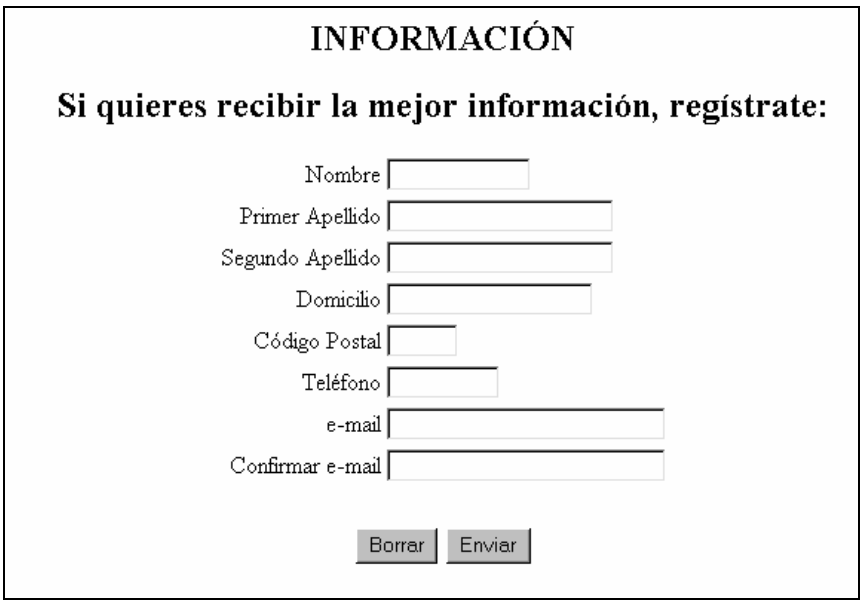
En esta práctica se añadirán dos nuevas páginas al *website* de la práctica anterior. Se trata de crear dos formularios para que el usuario pueda registrarse como tal y pedir información.

Descarga en la misma carpeta de la práctica 1 el contenido de *Practica02.zip*, que está en la zona de material de la web de la asignatura. Redirecciona los links de *menu.html* y *menu_tur.html* donde corresponda a *información.html* y *arte.html* (antes *enconstruccion.html*).

Los formularios deben estar dirigidos a: http://www.tecnun.es/cgi-bin/ii/CGI0.exe.

Ejercicio 2.1

Introduce el formulario y sus cajas de texto en *informacion.html* y ordénalos de esta forma:



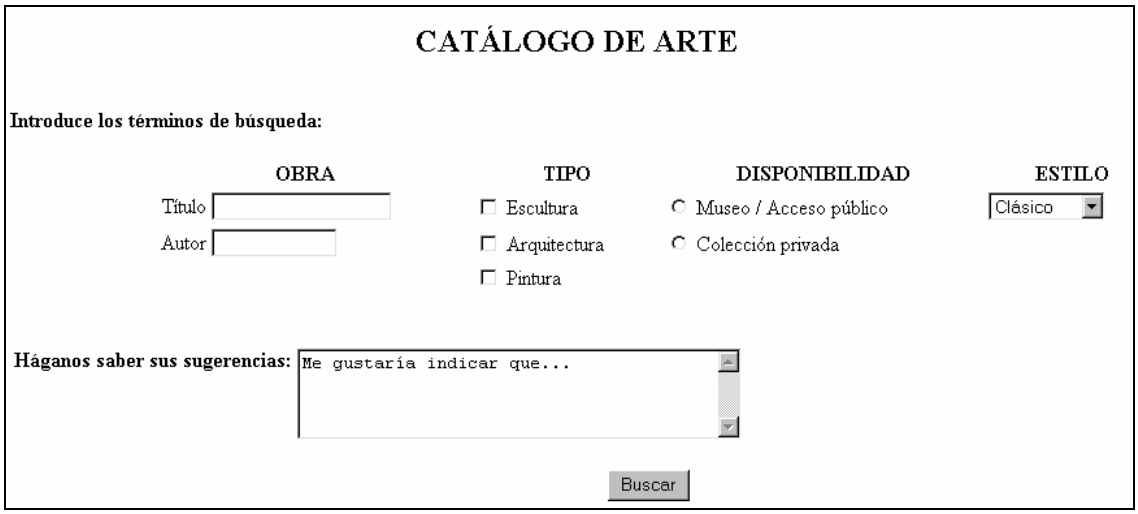
Características de los campos del formulario:

Campo	Nombre de la caja de texto	Tamaño
Nombre	nombre	12
Primer Apellido	apell1	(por defecto)
Segundo Apellido	apell2	(por defecto)
Domicilio	domicilio	18
Código Postal	CP	5
Teléfono	tel	9
e-mail	mail	25

Confirmar e-mail	c_mail	25

Ejercicio 2.2.

Completa el formulario de *arte.html* disponiéndolo de la siguiente manera:



Elementos del formulario:

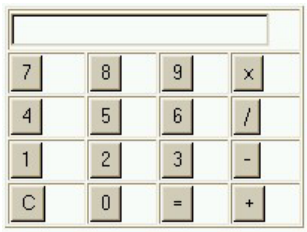
- **Tipo**: casillas *checkbox* de nombre ***tipo*** y valores ***esc***, ***arq*** y ***pint***.
- **Disponibilidad**: casillas *radio* de nombre ***disp*** y valores ***publ*** y ***priv***.
- **Estilo**: ventana de selección de nombre ***estilo*** y valores ***clas***, ***abs***, ***surr*** y ***otro***, representados por los textos ***Clásico***, ***Abstracto***, ***Surrealista*** y ***Otros***.
- **Área de texto:** 4 filas y 40 columnas y con un texto por defecto.

Práctica 3: Lenguaje JavaScript

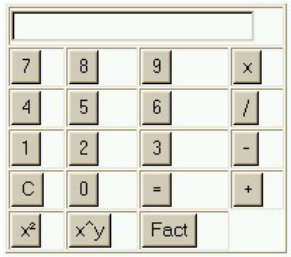
Esta práctica tiene dos partes. En la primera de ellas, correspondiente a los ejercicios 3.1, 3.2 y 3.3, se analizarán varias funcionalidades de JavaScript que pueden ayudarnos en la creación de nuestros propios formularios. En la segunda parte de esta práctica, correspondiente al ejercicio 3.4, se realizará la validación de un formulario de la práctica anterior utilizando recursos vistos en los ejercicios previos.

Ejercicio 3.1: Calculadora

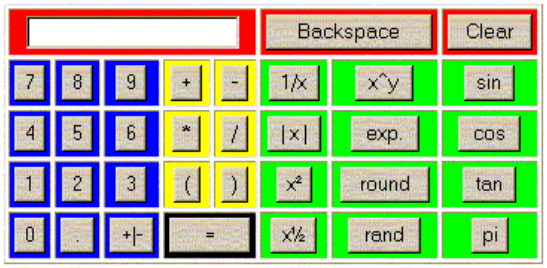
Se parte del ejemplo de una calculadora realizada con JavaScript: *Calculator.html*. Se pide analizar su funcionamiento. ¿Sería capaz de realizar las variaciones necesarias para lograr la calculadora presentada en *CalculatorNew.html*? ¿Lograría realizar la presentada en *CalculatorScientific.html*?



Calculator.html



CalculatorNew.html

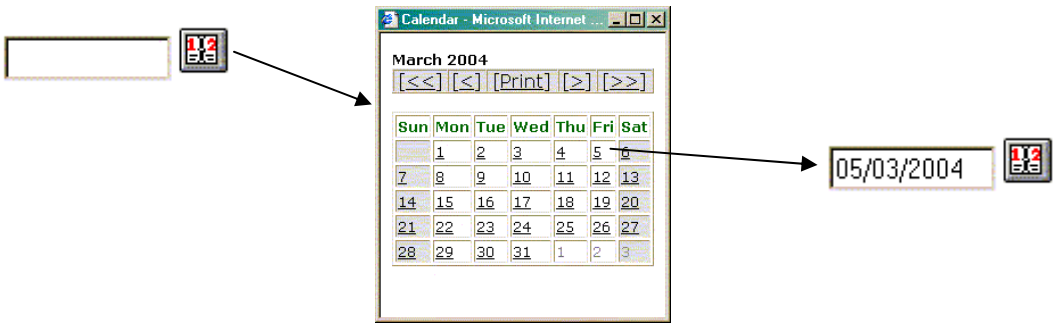


CalculatorScientific.html

Ejercicio 3.2: Selector de Fechas

El fichero *CalendarExample.html* muestra una aplicación muy útil de JavaScript. Se trata de un selector de fechas que acompaña a un campo de introducción de fechas y que facilita la introducción de fechas mostrando un calendario completo en una ventana a parte. Desde esta ventana, clicando en el día seleccionado, se rellena el campo en el formulario original. El código fuente JavaScript se encuentra en el fichero *date-picker.js* y la imagen que acompaña al campo y que al clicarla muestra el calendario se llama *show-calendar.gif*.

Se pide estudiar su funcionamiento y ser capaz de utilizar este recurso en formularios que creemos nosotros.



Ejercicio 3.3: Validación de Formularios 1

El formulario de la figura se encuentra en el fichero *Registro.html*. Desde este fichero se hace referencia al fichero con el código fuente JavaScript de nombre *Validacion.js*.

Se pide estudiar su funcionamiento de forma que el alumno aprenda cómo se valida el contenido de un formulario antes de proceder a su envío, evitando así que se produzcan errores innecesarios en el servidor por incongruencia de datos.

Ejercicio 3.4: Validación de Formularios 2

Se trata de hacer una función en JavaScript para que se valide el formulario del ejercicio 2.1 de la práctica anterior, de forma que el Código Postal y Teléfono sólo admitan números y los campos para la introducción del e-mail coincidan y contengan el carácter "@".

Práctica 4: Toma de contacto con Java

El objetivo de esta primera práctica es la toma de contacto con el ***lenguaje Java***, con el ***Java 2 Software Development Kit (J2SDK)*** y con la documentación del ***J2SDK*** donde podremos ver las definiciones de las clases que vayamos utilizando en los ejercicios. Escribiremos 5 programas, los compilaremos y ejecutaremos de forma que durante el proceso vayamos descubriendo la estructura de los programas en ***Java*** y cómo se trabaja con las clases de la ***API*** de ***Java***.

Ejercicio 4.1: Pasando argumentos al programa

El siguiente programa tiene por objetivo enseñarte la forma en que puedes pasar parámetros a un programa en Java. Los dos números que le pasarás al programa, desde la línea de comandos, se van a comparar y el programa te dirá cuál es el mayor.

Después de compilar el programa, ejecútalo de la siguiente manera:

```
java Ejer1 20 25
```

Observa cómo se le pasan los dos parámetros al programa (los dos números que va a comparar, en este caso el 20 y el 25).

```
/* Obtener el mayor de 2 números, pasados como argumentos */
// Fichero Ejer1.java

public class Ejer1{
   public static void main(String args[]){
      float x1=0,x2=0;

      if ( args.length<2 ) {
         System.out.println("Faltan los dos numeros");
      } else {
         x1 = Float.parseFloat(args[0]);
         x2 = Float.parseFloat(args[1]);
         if (x1>x2) System.out.println("Mayor: " + x1);
         else if (x1<x2) System.out.println("Mayor: " + x2);
         else if (x1==x2) System.out.println("Iguales");

         System.out.println("Otra Forma:");
         System.out.println("El mayor es: " + Math.max(x1,x2));
      }
   } // Fin de main()
} // Fin de clase Ejer1
```

Se han definido dos variables nuevas (**x1** y **x2**) de tipo ***float***. Observa la forma en que puedes inicializar una variable en el momento de declararla y cómo se utilizan los condicionales ***If-Else***.

Se utiliza la clase ***Float***, con uno de sus métodos, ***parseFloat***, para convertir el "string" o cadena de caracteres en un número flotante, para así tratarlo luego como números en la comparación. Conviene que leas un poco en el manual de Java sobre esta clase.

Ejercicio 4.2: Lectura de datos desde el Teclado

Este programa te enseñará la forma de introducir datos al programa desde el teclado. Se definen dos variables (**str1** y **str2**) del tipo ***String*** (realmente son objetos de la clase `String`) donde se almacenarán los caracteres dados por el teclado. Notarás al final del ejercicio que usamos el método o función ***max()*** de la clase ***Math*** para comparar el mayor de los números pasados como argumentos.

```
/* Lectura de datos desde el teclado */
// Fichero Ejer2.java

import java.io.*;    //no olvidar poner esta línea

public class Ejer2 {
   public static void main(String args[]) throws IOException {
      float x1=0,x2=0;
      String str1,str2;

      BufferedReader InBuf =
         new BufferedReader(new InputStreamReader(System.in));

      System.out.println("Primer Numero: "); //Leer Primer número
      str1=InBuf.readLine();
      x1 = Float.parseFloat(str1);

      System.out.println("Segundo Numero: "); //Leer Segundo número
      str2=InBuf.readLine();
      x2 = Float.parseFloat(str2);

      //Obtener el mayor
      System.out.println("El mayor es: " + Math.max(x1,x2));
   }
}
```

Las clases que conviene que aprendas a utilizar son: ***BufferedReader*** e ***InputStreamReader***, ya que las vas a utilizar muy a menudo. Son las que permiten acceder al teclado y leer lo que escribes. Consulta la ayuda sobre java para que te familiarices con estas clases.

Ejercicio 4.3: Bucle For - Factorial de un número

A continuación veremos cómo se hace un bucle con la sentencia ***For***.

Aquí usamos la clase `Integer` para convertir un string (con su método `parseInt`) en un número entero, al cual vamos a calcular su factorial. Conviene que te anotes esta clase para que la repases, junto con la `Float` del ejercicio 1.

```
/* Bucle For: Factorial de un número */
// Fichero Ejer3.java

import java.io.*;

public class Ejer3 {

   public static void main(String args[]) throws IOException {
      int x1;
      long Fact;
      String str1;

      BufferedReader InBuf =
            new BufferedReader(new InputStreamReader(System.in));
```

```
        System.out.println("Factorial de un Numero\r");
        System.out.println("Dar el Numero: ");
        str1=InBuf.readLine();
        x1 = Integer.parseInt(str1); //convertimos a un número entero

        Fact=x1;
        for (int i=x1-1; i>0; i--){
              Fact *= i;
        }
        System.out.println("\rFactorial de "+x1+" es: "+Fact);
    }
}
```

Prueba haciendo el bucle ***For*** con la variable ascendente, de la siguiente forma:

```
        for (int i=1; i<=x1; i++){
              ...
        }
```

Ejercicio 4.4: Bucle While - Generación aleatoria de números

En este ejercicio aprenderás a hacer un bucle con la sentencia While. Este programa generará una cantidad determinada (que el usuario dará por medio del teclado) de números reales de forma aleatoria, comprendidos entre dos límites.

```
/* Bucle While: Generación aleatoria de números */
// Fichero Ejer4.java

import java.io.*;

public class Ejer4 {

   public static void main(String args[]) throws IOException {
      int x1;
      char c;
      double val;
      String str1="";

      System.out.println("Generacion Aleatoria de Numeros\r");

      //Ahora no usaremos printl, sino print. Ya no saltará una línea
      System.out.print("Cuantos Numeros?: ");

      //Leeremos caracter por caracter del teclado, hasta presionar Enter
      //read() devuelve un byte y por eso hay que hacer un cast
      //read() detiene la ejecución del programa hasta que se pulsa Enter
      while ( (c=(char)System.in.read()) != '\r' ) {
         if ( c>='0' && c<='9') {
            str1 = str1 + c; //Sólo tomamos los dígitos
         }
      }
      x1 = Integer.parseInt(str1);
      while((x1--)>0) {
         val = Math.random();
         val *= 10.0;
         System.out.println("Numero: " + val);
      }
   }
}
```

Si ya has acabado el programa, prueba hacer una variación de éste: los límites de generación de números los des por teclado (Por ejemplo, generar números entre 10 y 40).

Ejercicio 4.5: Métodos (funciones) de clase y Variables de clase

Este ejercicio te enseñará cómo crear métodos (o funciones) de una clase. Así mismo verás cómo las variables pueden ser declaradas locales (dentro de un método) o "globales" (variables de la clase), y cómo éstas pueden ser accedidas.

```
/* Métodos de clase y Variables de clase */
// Observar también la visibilidad de las variables
// Fichero Ejer5.java

public class Ejer5 {
       double area; //Variable de la Clase

       public static void main(String args[]){
              double radio,area; //variables de la función main()

              Ejer5 ej = new Ejer5();

              if ( args.length < 1){
                 System.out.println("Pasar el radio. Ej. : >java Ejer5 12.5");
                 System.exit(0); //Terminar el programa
              }
              radio = Double.parseDouble(args[0]);
              area = ej.Area(radio);  //Acceder a función de Objeto creado
              System.out.println("Area del circulo r="+radio+" m. = " +area+" m2");

              area = ej.area;
              System.out.println("Area del circulo r="+radio+" m. = "+area+" Has.");
       }

       //Función o método Area de la clase Ejer5
       public double Area (double rd ){
              double area; //variable local de la funcion Area

              //usar la variable local
              area = Math.PI*Math.pow(rd,2.0);

              //acceder a la variable de la clase
              this.area = area/10000; //obtener el area en Has.

              return area; //devuelve el valor
       }
}
```

Observa que la variable ***area***, declarada como **double**, ha sido definida tanto en la clase como dentro de los métodos ***main()*** y ***Area()***. La variable ***area*** de la clase puede ser accedida por cualquier método (o función) de la clase, no así las variables declaradas dentro de cada método, ya que éstas son locales.

Si ya has acabado todos los ejercicios y ves que te sobra tiempo, sería bueno que fueses consultando la documentación de la ***API***, de tal forma que empieces a aprender la sintaxis y las clases de Java.

Práctica 5: AWT y applets de Java

Ejercicio 5.1: Usando Componentes Gráficos

En este primer ejercicio aprenderemos a introducir componentes elementales en una aplicación y a utilizar contenedores y diseños para que nuestra aplicación siempre tenga la misma organización independientemente del tamaño de la ventana donde esté desplegada.

En una primera versión de la aplicación para realizar pedidos en una empresa de comida rápida se introducen componentes elementales directamente sobre un `Container` de la clase `Frame` con un `LayoutManager` de la clase `FlowLayout`. Compila y ejecuta el fichero `Ejer1_01.java`.

El programa anterior despliega una ventana como la presentada en la parte izquierda de la figura 5.1. Ahora bien, cambiemos el tamaño de la ventana mientras esta está activa. Observamos que los componentes se mueven y se acomodan a las nuevas dimensiones, pero las nuevas posiciones de los elementos no cumplen el cometido que tenían al principio (parte derecha de la figura 5.1).

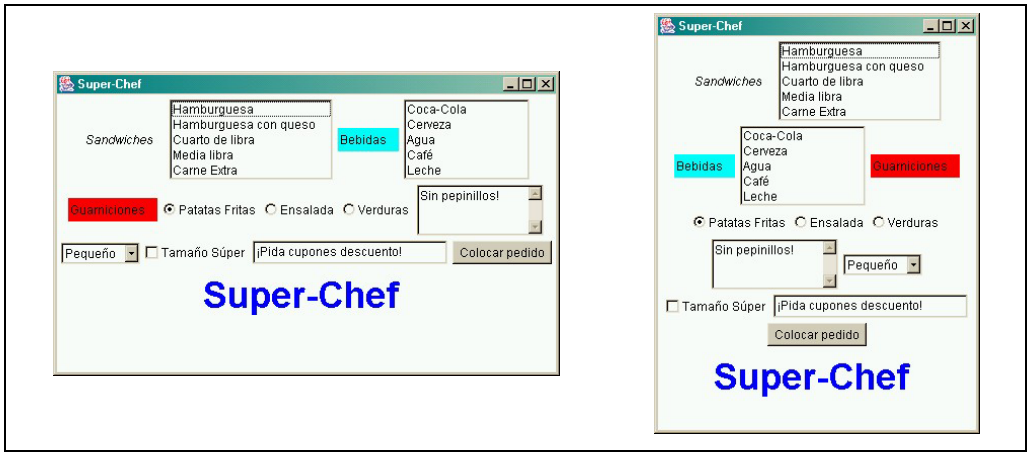


Figura 5.1: Inconveniente del Layout `FlowLayout`

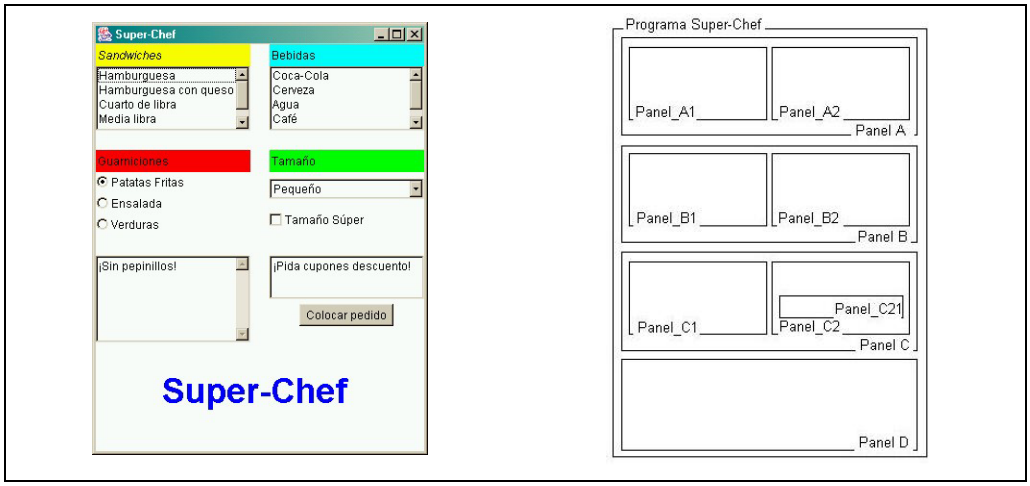


Figura 5.2. Diseño Visual de un programa y su jerarquía de contención

Para evitar que el aspecto de nuestra aplicación cambie de un sistema a otro, se utilizan jerarquías de contención de los elementos y otros diseños además del ya visto de `FlowLayout`. El programa `Ejer1_02.java` resuelve el problema anterior.

La figura 5.2 muestra el aspecto que tendrá la aplicación con los cambios introducidos. Pruébese en este caso a modificar el tamaño de la ventana y se comprobará que los elementos siguen manteniendo sus posiciones relativas, aunque no así sus tamaños que se van ajustando según las dimensiones de la ventana. También se puede observar en la figura 5.2 la estructura jerárquica de los diferentes elementos contenedores.

Ejercicio 5.2: Usando Eventos

En este segundo ejercicio vamos a hacer que la aplicación desarrollada en el ejercicio 1 sea sensible a ciertos eventos. En concreto vamos a introducir las siguientes características:

- Que la aplicación se cierre cuando se pulse en el botón de cerrar ventana de la esquina superior derecha.
- Que el `CheckBox` de "Tamaño Super" esté inactivo y que sólo se active cuando en el `Choice` se seleccione el tamaño "Grande".
- Que cuando se pulse el `Button` de "Colocar Pedido" se genere una página ***HTML*** "Pedido.html" que contenga toda la información del pedido.

Para conseguir todo lo anterior debemos introducir ciertos elementos en el código que enumeramos a continuación:

1. Se deben incluir dos nuevos paquetes correspondientes a los eventos y a la entrada/salida:

```
import java.awt.event.*;
import java.io.*;
```

2. La clase Ejer2 debe implementar tres interfaces de *escucha*:

```
public class Ejer2 extends Frame implements WindowListener, ItemListener,
ActionListener {
```

3. En el constructor de la clase se deben registrar los *escuchas* para los generadores de eventos que queremos controlar y hacer que el `CheckBox` esté inactivo al comienzo de la aplicación:

```
addWindowListener(this);
sizes.addItemListener(this);
order.addActionListener(this);

supersize.setEnabled(false);
```

4. Se deben instrumentar las interfaces correspondientes:

```
//Instrumentación de la Interfaz WindowListener
public void windowActivated(WindowEvent e){}
public void windowClosed(WindowEvent e){}
public void windowClosing(WindowEvent e){
   System.exit(0);
}
public void windowDeactivated(WindowEvent e){}
public void windowDeiconified(WindowEvent e){}
public void windowIconified(WindowEvent e){}
public void windowOpened(WindowEvent e){}

//Instrumentación de la Interfaz ItemListener
public void itemStateChanged(ItemEvent e){
   String label = "" + e.getItem();
   if(label.equals("Grande")){
       supersize.setEnabled(true);
   }else{
       supersize.setEnabled(false);
       supersize.setState(false);
```

```
   }
}

//Instrumentación de la interfaz ActionListener
public void actionPerformed(ActionEvent e){
   try{
       BufferedWriter salida = new BufferedWriter(new FileWriter("Pedido.html"));
       String title = "Su Pedido";
       String Bebidas[], Comidas[];
       int i;

       Bebidas = drinks.getSelectedItems();
       Comidas = sandwiches.getSelectedItems();
       salida.write("<HTML><HEAD><TITLE>");
       salida.newLine();
       salida.write(title);
       salida.write("</TITLE></HEAD><BODY>");
       salida.newLine();
       salida.write("<H1>" + title + "</H1>");
       salida.newLine();
       salida.write("<P><STRONG>Comidas:</STRONG> ");
       for(i=0;i<Comidas.length-1;i++){
          salida.write(Comidas[i] + "," );
       }
       salida.write(Comidas[i]);
       salida.newLine();
       salida.write("<BR><STRONG>Bebidas:</STRONG> " );
       for(i=0;i<Bebidas.length-1;i++){
          salida.write(Bebidas[i] + ",");
       }
       salida.write(Bebidas[i]);
       salida.newLine();
       salida.write("<BR><STRONG>Guarnición:</STRONG> " +
                    sides.getSelectedCheckbox().getLabel());
       salida.newLine();
       salida.write("<BR><STRONG>Tamaño:</STRONG> " + sizes.getSelectedItem());
       if(supersize.isEnabled() == true){
          salida.write(" SUPER");
       }
       salida.newLine();
       salida.write("<BR><STRONG>Ordenes:</STRONG> " + comments.getText());
       salida.newLine();
       salida.write("<BR><STRONG>Recuerde:</STRONG> " + reminder.getText());
       salida.newLine();
       salida.write("</BODY></HTML>");
       salida.close();
   }catch(IOException ex){}
}
```

Una vez realizados los cambios anteriores, compilar y ejecutar la aplicación comprobando que responde a los eventos introducidos. Cuando se pulse el botón de "Colocar Pedido" se generará una página "Pedidos.html" en el mismo directorio donde esté la aplicación. Abrámosla y veamos cómo queda.

Ejercicio 5.3: Applet para pedir datos de usuario.

```
/* Applet para pedir datos de usuario */
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Ejer3 extends Applet implements ActionListener{

    Label LblExplicacion =
             new Label("Rellena los siguientes campos y pulsa ACEPTAR");
    Label Etiqueta1 = new Label("Nombre Usuario:", Label.RIGHT);
    Label Etiqueta2 = new Label("Contraseña:", Label.RIGHT);
    Label Etiqueta3 = new Label("", Label.CENTER);
    TextField Campo1 = new TextField(15);
    TextField Campo2 = new TextField(15);
    Button BtnAceptar = new Button("ACEPTAR");
    String Nombre = "informatica3";
    String Password = "java";

    public void init(){
```

```
        setLayout(new BorderLayout());

        //Fuente usada en el rótulo y caracter de eco de la contraseña
        LblExplicacion.setFont(new Font("SansSerif", Font.BOLD, 16));
        Campo2.setEchoChar('*');
        Etiqueta3.setForeground(Color.red);

        //Añadir el rótulo de explicación
        add(BorderLayout.NORTH, LblExplicacion);

        //Construir el panel central y añadirlo
        Panel pc = new Panel();
        pc.setLayout(new FlowLayout());
        Panel pc1 = new Panel();
        pc1.setLayout(new GridLayout(2, 2, 8, 2));
        pc1.add(Etiqueta1);
        pc1.add(Campo1);
        pc1.add(Etiqueta2);
        pc1.add(Campo2);
        pc.add(pc1);
        //Poner el botón ACEPTAR en un panel y añadirlo también
        Panel pc2 = new Panel();
        pc2.add(BtnAceptar);
        pc.add(pc2);
        add(BorderLayout.CENTER,pc);
        add(BorderLayout.SOUTH, Etiqueta3);

        //Registramos el Botón como escucha
        BtnAceptar.addActionListener(this);
    }

    //Instrumentamos la Interfaz ActionListener
    public void actionPerformed(ActionEvent e){
        if (Nombre.equals(Campo1.getText()) == false){
                Etiqueta3.setText("Nombre de Usuario erróneo");
        }else if(Password.equals(Campo2.getText()) == false){
                Etiqueta3.setText("Contraseña errónea");
        }else{
                Etiqueta3.setText("Acceso Permitido");
        }
    }
}
```

Para desplegar el applet o subprograma anterior en un navegador o en el appletviewer, se necesita un archivo html como el siguiente:

```
<HTML>
    <APPLET code = "Ejer3.class" width = 400 height = 100>
    </APPLET>
</HTML>
```

En la figura 5.3 se muestra el despliegue del applet en un navegador. Si introducimos un nombre de usuario o una contraseña incorrecta, aparecerá un aviso en rojo. Si los datos son correctos el aviso nos dirá: "Acceso Permitido".

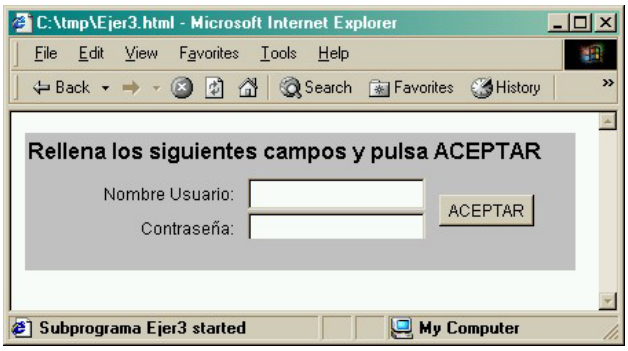


Figura 5.3. Despliegue del Applet en un navegador

Práctica 6: JDBC: Acceso a base de datos desde Java

Ejercicio 6.1: Toma de contacto con JDBC

En este primer ejercicio se pide seguir los pasos expuestos en el apartado 8.3 del libro de la asignatura. En dicho apartado aparecen dos ejemplos sencillos de utilización de la JDBC 3.0 API. Para poder utilizar el puente JDBC-OBDC habrá que realizar unos pasos previos que vienen detallados en dicho apartado. El objetivo de este ejercicio es familiarizarse con:

- Bases de Datos en Access
- ODBC y creación de DSN (Data Source Name)
- JDBC y su utilización

Una vez creado el DSN y compilado el código fuente Java, se trata de ir ejecutando las sentencias que aparecen en el libro e ir comprobando que los resultados son los mismos. Además, el alumno debe entender completamente el código fuente Java y saber qué hace cada una de las sentencias que en él aparecen.

Ejercicio 6.2: Realizando consultas de actualización

Los ejemplos que se han estudiado en el ejercicio anterior realizaban **consultas de Selección de Datos**, es decir, hacían peticiones de registros que cumpliesen ciertos requisitos a la Base de Datos. Corresponden a sentencias SQL tipo SELECT.

Esto se hacía mediante el método `executeQuery(String sql)` de la clase `Statement`. Este método devuelve un objeto de tipo `ResultSet` con los registros que cumplan las condiciones especificadas en la sentencia SQL. Este objeto `ResultSet` puede ser recorrido registro a registro y obtener los valores de los campos deseados.

En este segundo ejercicio se van a realizar **consultas de Actualización de Datos**, es decir, ahora se trata de insertar, modificar o eliminar un registro de una tabla de la Base de Datos. Corresponden a sentencias SQL tipo INSERT INTO, UPDATE o DELETE.

En este caso, se realizará mediante el método `executeUpdate(String sql)` de la clase `Statement`. Este método no devuelve ningún objeto de la clase `ResultSet`, sino que devuelve un entero correspondiente al número de registros modificados mediante la sentencia SQL ejecutada o cero si la sentencia SQL no devuelve nada.

A continuación se muestra el código fuente en Java que vamos a utilizar. Como puede observarse, es similar en gran parte al utilizado en el programa `firstJDBC.java`. De hecho sólo cambia la sentencia que contiene el método `executeUpdate(String sql)`, que no devuelve un `ResultSet` sino un entero.

Se pide compilar dicho código fuente y ejecutar la secuencia de sentencias que vienen a continuación. Después de cada una de las sentencias, se deben comprobar los cambios producidos en la Base de Datos.

```
import java.sql.*;

class secondJDBC {

  public static void main(String args[]) throws ClassNotFoundException, SQLException {

    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    String url="jdbc:odbc:" + args[0];

    Connection connection = DriverManager.getConnection(url);

    Statement statement = connection.createStatement();

    String sql = args[1];
    int nrows = statement.executeUpdate(sql);

    System.out.println ("Numero de registros modificados: " + nrows);

    connection.close();
  }
}
```

Consulta de Inserción de Registros

La siguiente sentencia introduce un registro en la tabla `Datos` de la Base de Datos `Libros.mdb`, señalada por el DSN `pruebaODBC`:

```
java secondJDBC pruebaODBC "INSERT INTO Datos (Codigo,Titulo,Autor) VALUES ('N006','El jinete polaco','A. Muñoz Molina')"
```

La salida de la ejecución de la sentencia anterior es:

```
Numero de registros modificados: 1
```

Consulta de Actualización de Registro

La siguiente sentencia modifica el campo `Autor` del registro con `Codigo='N006'` de la tabla `Datos` de la Base de Datos `Libros.mdb`, señalada por el DSN `pruebaODBC`:

```
java secondJDBC pruebaODBC "UPDATE Datos SET Autor='Antonio Muñoz Molina' WHERE Codigo='N006'"

Numero de registros modificados: 1
```

Si ninguno de los registros cumple las condiciones no se produce ninguna modificación, como por ejemplo la siguiente consulta:

```
java secondJDBC pruebaODBC "UPDATE Datos SET Autor='Antonio Muñoz Molina' WHERE Codigo='N999'"

Numero de registros modificados: 0
```

Consulta de Eliminación de Registros

La siguiente sentencia elimina el registro con `Codigo='N006'` de la tabla `Datos` de la Base de Datos `Libros.mdb`, señalada por el DSN `pruebaODBC`:

```
java secondJDBC pruebaODBC "DELETE FROM Datos WHERE Codigo='N006'"

Numero de registros modificados: 1
```

Práctica 7: Servlets

En esta práctica se instalará un servlet y se comprobará su funcionamiento, para posteriormente realizar las modificaciones que se solicitan. El servlet con el que se trabajará será el ejemplo **`SurveyServlet.java`** que introduce los datos de una encuesta en un fichero.

Primeramente se debe realizar la instalación y configuración del servlet y del servidor de servlets. Para ello se dispone de los ficheros **`JdcSurvey.html`**, **`SurveyServlet.java`** y **`servlet.properties`** comprimidos en el fichero ***Practica07.zip***, que se encuentra en la zona de material de la página web de la asignatura. Descarga el fichero y descomprímelo en un directorio propio.

Modificar **`JdcSurvey.html`** para que el formulario envíe los datos al servlet **`SurveyServlet.java`** del propio directorio y comprobar el funcionamiento viendo que inserta los datos correctamente en el fichero especificado. A continuación realizar las modificaciones oportunas en el servlet **`SurveyServlet.java`** para que este presente los datos que se han introducido en la respuesta al cliente de la forma mostrada en la Figura 7.1.

Figura 7.1. Salida del servlet `SurveyServlet.java`

El servlet **`SurveyServlet.java`** sólo imprime una de las opciones elegidas de IDE. En el caso de seleccionar varias, para imprimir todas se debe sustituir el código que se muestra en la siguiente tabla (se recomienda crear un nuevo servlet: **`SurveyServletNew.java`**):

`SurveyServlet.java`	**`SurveyServletNew.java`**

`while(values.hasMoreElements()) {`
`  String name = (String)values.nextElement();`
`  String value =`
`  req.getParameterValues(name)[0];`
`  if(name.compareTo("submit") != 0) {`
`  toFile.println(name + ": " + value);`
`  }`
`}`
	`while(values.hasMoreElements()) {`
	`  String name = (String)values.nextElement();`
	`  String[] valueArray =`
	`  req.getParameterValues(name);`
	`  for (int i=0;i<valueArray.length;i++) {`
	`  String value = valueArray[i];`
	`  if(name.compareTo("submit") != 0) {`
	`  toFile.println(name + ": " + value);`
	`  }`
	`  }`
	`}`

Pasos a realizar para la puesta en funcionamiento del servlet:

1. Crea un nuevo directorio de trabajo y descomprime allí el fichero ***Practica07.zip***.

2. Modifica el texto "Thank you" por "Gracias" las dos veces que aparece en el fichero **`SurveyServlet.java`**. De esta forma sabrás si el servlet al que se está llamando es el propio.

3. Modifica el título de la página HTML a "Mi página JdcSurvey" en el fichero **`JdcSurvey.html`**.

4. En el fichero de proceso por lotes (*.BAT) en el que tengas establecidas las variables de entorno PATH, JAVAPATH y CLASSPATH, añade las siguientes líneas:

   ```
   SET PATH=%PATH%;q:\jsdk2.0\bin
   SET CLASSPATH=%CLASSPATH%;q:\jsdk2.0\lib\jsdk.jar;.
   ```

   De esta forma se tendrá acceso a la aplicación ***servletrunner.exe*** y a las clases de la ***Servlet API 2.0***

5. Abre una consola de MS-DOS y ejecuta dicho fichero de proceso por lotes (*.BAT).

6. Compila el código fuente java con el comando:

   ```
   javac SurveyServlet.java
   ```

7. En el fichero **`servlet.properties`** cambia la última línea para indicar que el directorio en el que se escribirá es el `c:/temp` en lugar de `/tmp`.

8. Arranca el servidor de servlets con el comando:

   ```
   servletrunner -d "path completo del directorio de trabajo"
   ```

9. Arranca el navegador y solicita la página **`JdcSurvey.html`**.

10. Selecciona una opción referente al tamaño de la compañía, introduce un comentario e indica un IDE. Pulsa el botón "Submit Query". Comprueba que el resultado que se obtiene tiene el título "Gracias" y el mensaje "Gracias por su participación" (es decir, se está usando el servlet que hemos modificado). En el servidor de servlets se debe haber impreso la línea:

    ```
    SurveyServlet: INIT
    ```

11. Comprueba que se ha escrito el siguiente fichero: `c:\temp\Survey01Results.txt`

Puede ocurrir que no se reconozca la máquina que hace de servidor de servlets, en este caso el ordenador local, cuando el formulario envía la información:

```
<FORM action="http://localhost:8080/servlet/survey" method="POST">
```

Entonces se debe modificar el nombre que aparece actualmente "**`localhost`**" por:

- el número de IP de la máquina en la que se está trabajando, por ejemplo: **`193.145.251.42:8080`**
- el nombre de la máquina en la que se está trabajando, por ejemplo: **`A01.tecnun.es:8080`**.

Lo habitual es colocar el número propio de IP que se puede obtener con el comando "`ipconfig`". El nombre de la máquina en la que se está trabajando o "*host*" se obtiene con el comando "`ipconfig/all`".

Práctica 8: Servlets con acceso a DB

En esta práctica se modificará el servlet de la práctica anterior, **`SurveyServlet.java`**, para que introduzca los datos que le llegan en una Base de Datos en lugar de escribirlos en un fichero.

Pasos para insertar los datos en una tabla de una base de datos:

Los dos primeros pasos del proceso corresponden a la creación de la Base de Datos que almacenará los datos de las encuestas y la definición del *Data Source Name* (DSN) que nos permitirá acceder a ella mediante el puente JDBC-ODBC:

1. Crea una Base de Datos y en ella la tabla **"SurveyData"** con los campos:

Nombre del Campo	Tipo	Comentarios
Tamano	Text	Almacena el nº de empleados de la empresa
Comentarios	Memo	Almacena los comentarios
UtilizaJavaWorkShop	Boolean	Los campos almacenarán `True` o `False` dependiendo de si se seleccionó cada una de las opciones mediante su *checkbox* correspondiente en el formulario.
UtilizaJpp	Boolean	
UtilizaCafe	Boolean	

2. Crea un *Data Source Name* (DSN) para la Base de Datos creada en el paso anterior mediante el administrador de ODBC. Pon como nombre del DNS: "**surveyODBC**".

Los siguientes pasos corresponden a las modificaciones que se deben realizar en el fichero **`SurveyServlet.java`** para que este lleve a cabo la inserción del nuevo registro en la Base de Datos que se ha creado a tal efecto:

3. Inserta la siguiente sentencia para importar el paquete `java.sql`:

```
import java.sql.*;
```

4. Inserta un objeto `Connection` como miembro de la clase `SurveyServlet`:

```
Connection conn;
```

5. Inserta en el método `init()` la conexión con la base de datos ODBC, que sustituye al cogido que antes obtenía el directorio donde guardar el fichero con la salida:

```
try {
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  String url = "jdbc:odbc:surveyODBC";
  conn = DriverManager.getConnection(url);
} catch(Exception e) {
  System.out.println("Error al crear la conexión");
  e.printStackTrace();
```

```
}
```

6. Inserta en el método `doPost()` el código que ejecuta las inserciones en la base de datos, que sustituirá a aquel que realizaba la escritura en el fichero:

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
                                throws ServletException, IOException {

  res.setContentType("text/html");

  PrintWriter toClient = res.getWriter();

  String sql;
  String sql1 = "INSERT INTO SurveyData (Tamano,Comentarios";
  String sql2 = ") VALUES ('" + req.getParameter("employee") + "','"
              + req.getParameter("comment") + "'";
  String[] valueArray = req.getParameterValues("ide");
  for (int i=0;i<valueArray.length;i++) {
    String value = valueArray[i];
    if(value.equals("JavaWorkShop") == true){
      sql1 += ",UtilizaJavaWorkShop";
      sql2 += ",True";
    } else if(value.equals("J++") == true){
      sql1 += ",UtilizaJpp";
      sql2 += ",True";
    } else if(value.equals("Cafe'") == true){
      sql1 += ",UtilizaCafe";
      sql2 += ",True";
    }
  }
  sql = sql1 + sql2 + ")";
  System.out.println(sql);

  try {
    Statement stmt = conn.createStatement();
    stmt.executeUpdate(sql);
  } catch(SQLException e) {
    e.printStackTrace();
  }

  toClient.println("<HTML>");
  toClient.println("<TITLE>Thank you!</TITLE>");
  toClient.println("Thank you for participating");
  toClient.println("</HTML>");

  toClient.close();
}
```

Nótese que el servlet, además de insertar el nuevo registro en la Base de Datos, imprime en la consola la sentencia SQL que realiza la inserción.

Práctica 9: Sesión en Servlets

Ejercicio 9.1: Escritura de valores en una sesión

En **`SessionServlet.java`** añadir a la salida de la página HTML un formulario que recoja el valor de dos variables, una conteniendo el nombre de un atributo de una sesión y la otra el valor de dicho atributo. Estas variables se deben enviar al mismo servlet. Para crear el formulario añadir después de:

```
out.println("</table></center>");
```

las líneas:

```
out.println("<H2>Envío de atributo al servlet</H2>");
out.println("<FORM action=\"SessionServlet\" method=\"GET\">");
out.println("<INPUT type=\"text\" size=\"20\" name=\"dataname\">");
out.println("<BR>");
out.println("<INPUT type=\"text\" size=\"20\" name=\"datavalue\">");
out.println("<BR>");
out.println("<INPUT type=\"submit\">");
out.println("</FORM>");
```

En el mismo servlet, una vez obtenida la sesión, se deben leer las dos variables anteriores y si las dos no son nulas añadir el nuevo atributo con su valor a la sesión. Para leer los parámetros y crear el atributo, añadir después de:

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

las líneas:

```
String dataName = request.getParameter("dataname");
String dataValue = request.getParameter("datavalue");
if (dataName != null && dataValue != null) {
   session.putValue(dataName, dataValue);
}
```

Llamar al servlet desde el navegador:

```
http://localhost:8080/servlet/SessionServlet
```

Comprobar que los nuevos atributos que se van añadiendo se muestran en la lista de atributos, así como que se puede cambiar el valor de un atributo existente.

Ejercicio 9.2: Guardar distintas solicitudes de un mismo usuario

Se trata de simular el funcionamiento de un carro de la compra de un comercio electrónico. Se utilizará para ello la base de datos de **`biblio.mdb`** que contiene una lista de libros. El objetivo es mostrar una relación de libros e ir añadiéndolos a una lista de reservas del usuario.

9.2.1. Servlet que muestra la relación de libros

Primeramente es necesario crear el DSN "***libro***", con la base de datos **`biblio.mdb`**. Se dispone de un servlet, `Libros.java`, que muestra los libros que cumplan la condición de contener la palabra que se le da como parámetro. Se puede llamar al servlet sin ningún parametro, con lo que

se muestran todos los libros de la tabla, o con el parámetro "`filtro`", en cuyo caso muestra solo los libros que contienen la palabra o parte de la palabra especificada en el parámetro:

```
http://localhost:8080/servlet/Libros
http://localhost:8080/servlet/Libros?filtro=handbook
```

9.2.2. Servlet que muestra la relación de libros con posibilidad de selección de libros

A partir del servlet **`Libros.java`** anterior, se ha realizado el servlet **`LibrosCheck.java`** con las siguientes modificaciones:

- La tabla debe estar incluida en un formulario que llame al servlet: **`RealizarReserva.java`**
- Cada línea debe tener un checkbox que en el caso de marcarse envía el ISBN del libro

Se pide probar este servlet. La salida originada por el servlet **`LibrosCheck.java`** es la que se muestra en la Figura 9.1.

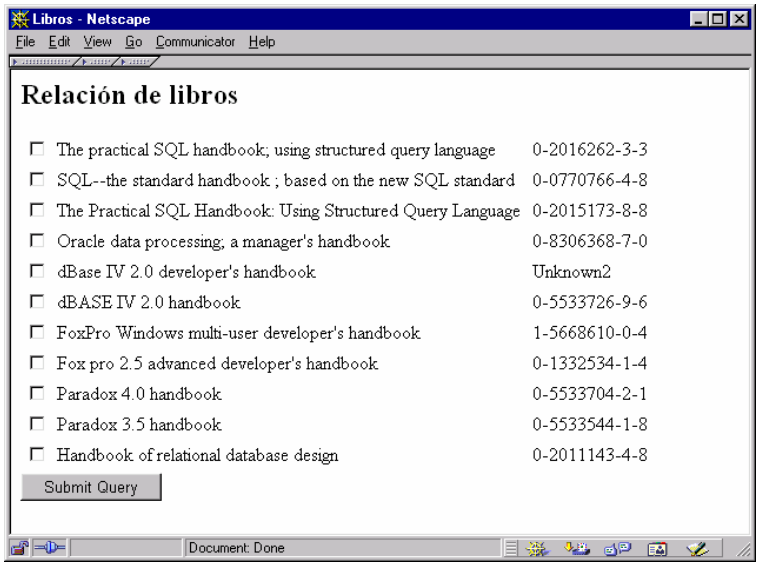


Figura 9.1. Relación de libros

9.2.3. Servlet que realiza la reserva de un libro seleccionado

Por último se dispone del servlet **`RealizarReserva.java`** que graba en una tabla el ISBN del libro solicitado y el ID de la sesión que realiza la reserva. Este servlet muestra a continuación todas las reservas de la sesión correspondiente.

Este servlet se llama cuando se pulsa "Submit Query" en el formulario que muestra el servlet **`LibrosCheck.java`** (ver Figura 9.1).

A continuación se resumen los servlets que aparecen en la práctica con las características generales de cada uno de ellos:

Servlet	JDBC	HttpSession
SessionServlet	No	Sí
Libros	Sí	No
LibrosCheck	Sí	No
RealizarReserva	Sí	Sí

Práctica 10: Puesta en marcha de una aplicación de Comercio Electrónico

El objetivo de esta práctica es estudiar el funcionamiento de una aplicación completa que incluye todos los elementos y tecnologías que se desarrollan a lo largo de la asignatura.

Se trata de una tienda de discos virtual que permite consultar el catálogo de discos disponibles a cualquier usuario, registrarse como usuario y añadir discos al carrito de la compra. A su vez, existe un administrador que puede añadir nuevos discos al catálogo y ver los datos de todos los usuarios registrados.

La práctica consta de dos partes principales: puesta en marcha y análisis de la aplicación. Una tercera parte opcional será el planteamiento de mejoras sobre la aplicación actual.

Ejercicio 10.1: Puesta en marcha de la aplicación

Descarga el fichero ***Practica10.zip*** de la zona de Material de la web de la asignatura y descomprímelo en un directorio propio, por ejemplo: `G:\Infor3\Practica10\`

La aplicación consta de los siguientes archivos:

Tipo de Archivo	Nombre	
Base de Datos	`Tienda.mdb`	
Html	`index.html`	
	`login.html`	
	`menu.html`	
	`principal.html`	
	`registro.html`	
Servlet	`Actualizar.java`	`Inicio_Catalogo.java`
	`Actualizar_Datos.java`	`Login.java`
	`Add_Carrito.java`	`Logout.java`
	`Add_Disco.java`	`Registro.java`
	`Baja_Disco.java`	`Registro_Disco.java`
	`Datos_Cliente.java`	`Ver_Carrito.java`
	`Datos_Disco.java`	`Ver_Catalogo.java`
	`Ejecutar_Actualizacion.java`	`Ver_Clientes.java`
	`Eliminar.java`	

En primer lugar hay que crear un DSN para la Base de Datos **`Tienda.mdb`** con nombre **`tienda`**.

Los servlets de la aplicación se encontrarán en un subdirectorio llamado `\Servlets` del directorio donde se haya descomprimido el fichero ***Practica10.zip***. Se deben compilar todos los servlets. Esto se puede realizar mediante la instrucción:

```
javac *.java
```

Después de esto, arrancar el servletrunner. Si el directorio en el que se ha guardado la práctica fuese el mismo que el ejemplo (`G:\Infor3\Practica10\`), se arrancaría con la instrucción:

```
servletrunner -p 8081 -d G:\Infor3\Practica10\Servlets
```

La página HTML de inicio de la aplicación es **`index.html`**.

Ejercicio 10.2: Análisis del funcionamiento de la aplicación

Una vez que hemos sido capaces de poner en marcha la aplicación de Tienda de Discos, el objetivo de esta segunda parte de la práctica es aprender cómo funciona y analizar su estructura interna.

El alumno debe usar la aplicación desde los dos roles predefinidos, el de **usuario normal** y el de **administrador** y probar todas las funcionalidades que ésta pone a su disposición para cada uno de los roles.

Una vez que el alumno se ha familiarizado con las funcionalidades de la aplicación, debe centrar su atención en descubrir y analizar cómo están programadas.

La aplicación contiene servlets que realizan operaciones muy variadas contra la Base de Datos y que pueden servir al alumno como plantilla para sus propios desarrollos. Entre otros, existen servlets que realizan las siguientes acciones contra la Base de Datos:

- Consultas de selección simples a una sola tabla
- Consultas de selección complejas que incluyen varias tablas relacionadas
- Consultas de inserción de nuevos registros en una tabla
- Consultas de actualización de registros en una tabla
- Consultas de eliminación de registros de una tabla

Por último, una cuestión para pensar sobre ella... ¿de dónde saca la aplicación las imágenes de las carátulas de los discos y los iconos? ¿a qué se debe que las extraiga de allí?

Ejercicio 10.3: Planteamiento de mejoras

En este ejercicio el alumno se debe plantear posibles mejoras a la aplicación de la Tienda de Discos. Las mejoras pueden estar orientadas a:

- ***Interfaces de usuario***: Forma de mostrar la información, accesibilidad a las funcionalidades, economía de pasos para realizar operaciones por parte del usuario, etc.
- ***Funcionalidades***: Otro tipo de funcionalidades que puedan ser de utilidad para un usuario o para un administrador.
- ***Operativa***: Obligación de autenticarse antes de comenzar la compra, antes de entrar en la propia tienda, sólo cuando se va a cursar un pedido, etc.

Para obtener ideas, se anima al alumno a visitar *Web Sites* de comercio electrónico de reconocido prestigio como por ejemplo **Amazon.com**:

`http://www.amazon.com`

`http://www.amazon.co.uk`

Práctica 11: SAX & DOM Java XML APIs

El objetivo de esta práctica es familiarizarse con los ficheros XML y comenzar a utilizar las APIs de Java que permiten analizar (*parse*) dichos ficheros. Se ha diseñado una aplicación bajo entorno Web que permite guardar en un fichero XML los datos extraídos de una Base de Datos y también leer dichos ficheros XML, analizarlos y mostrar los datos contenidos en ellos en formato HTML. Para ello la aplicación dispone de dos servlets para esta última tarea, uno que utiliza la API SAX de análisis secuencial y otro que usa la API DOM de análisis mediante estructura de árbol.

Puesta en marcha

Descarga el fichero ***Practica11.zip*** de la zona de Material de la web de la asignatura y descomprímelo en un directorio propio, por ejemplo: `G:\Infor3\Practica11\`

La aplicación consta de los siguientes archivos:

Tipo de Archivo	Nombre
HTML	`index.html`
	`menu.html`
	`Leer_Factura_SAX.html`
	`Leer_Factura_DOM.html`
Servlets	`Facturacion.java`
	`Emitir_Factura_xml.java`
	`Leer_Factura_xml_SAX.java`
	`Leer_Factura_xml_DOM.java`
Propiedades	`servlet.properties`

La aplicación tomará los datos para emitir las facturas de la Base de Datos **`Tienda.mdb`** de la aplicación "Tienda de Discos" de la práctica 10, a través del DSN con nombre **`tienda`**, el cual ya debe estar presente en el sistema.

Los servlets de la aplicación se encontrarán en un subdirectorio llamado `\Servlets` del directorio donde se haya descomprimido el fichero ***Practica11.zip***. Se deben compilar todos los servlets como en prácticas anteriores.

Ahora bien, para que el compilador pueda entender las clases de las SAX y DOM APIs para XML, debemos ponerle a disposición dichas clases y modificar las VARIABLES DE ENTORNO de MS-DOS para que sepa dónde encontrarlas.

Las clases se encuentran en el fichero **`xerces.jar`** que puedes descargar zona de recursos, sección de XML, de la página Web de la asignatura. Una vez que guardes este fichero en un directorio propio, por ejemplo **`G:\Infor3\`**, en el fichero de proceso por lotes (*.BAT) en el que tengas establecidas las variables de entorno PATH, JAVAPATH y CLASSPATH, añade la siguiente línea:

```
SET CLASSPATH=%CLASSPATH%;G:\Infor3\xerces.jar;.
```

Se debe modificar el fichero **`servlet.properties`** para que los servlets tengan acceso a la propiedad **`facturasPath`** que les indicará el directorio donde deben guardar y de donde deben leer los ficheros XML. Por ejemplo:

```
# Emitir_Factura_xml servlet
servlet.Emitir_Factura_xml.code=Emitir_Factura_xml
servlet.Emitir_Factura_xml.initArgs=\
   facturasPath=G:/Infor3/Practica11
```

Después de esto, arrancar el servletrunner. Si el directorio en el que se ha guardado la práctica fuese el mismo que el ejemplo (`G:\Infor3\Practica11\`), se arrancaría con la instrucción:

```
servletrunner -p 8081 -d G:\Infor3\Practica11\Servlets
```

La página HTML de inicio de la aplicación es **`index.html`**, que muestra una pantalla con dos frames (figura 11.1), uno con un menú y el otro con el contenido resultante de las llamadas a las distintas opciones. Por defecto aparece la primera opción: *Emitir Factura*.

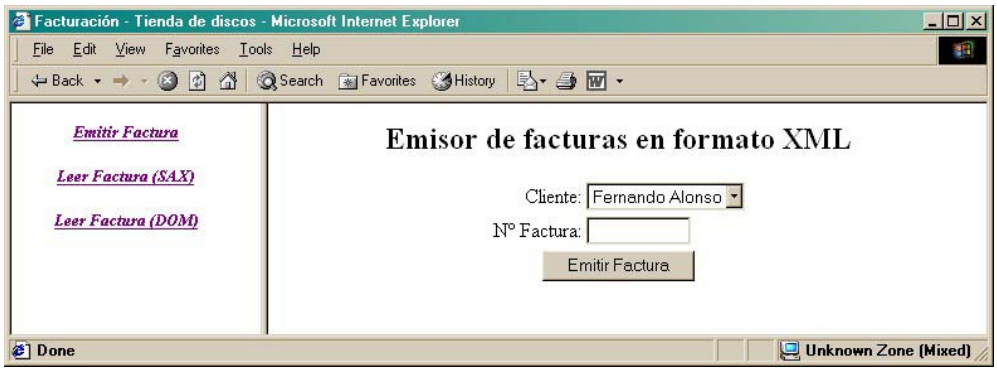


Figura 11.1. Aspecto de la aplicación de Facturación

Escribiendo un fichero en formato XML

La escritura del fichero en formato XML con los datos de una factura la lleva a cabo el servlet `Emitir_Factura_xml.java`, que sigue un esquema como el de la figura 11.2.

El servlet recibe a través de HTTP el identificador del cliente del cual se quiere la factura y el número de factura (que utilizará como nombre del fichero XML). Accede a la Base de Datos y obtiene los datos del carrito de la compra de ese cliente y escribe un fichero XML con el nombre indicado y en el directorio señalado por la propiedad **`facturasPath`**.

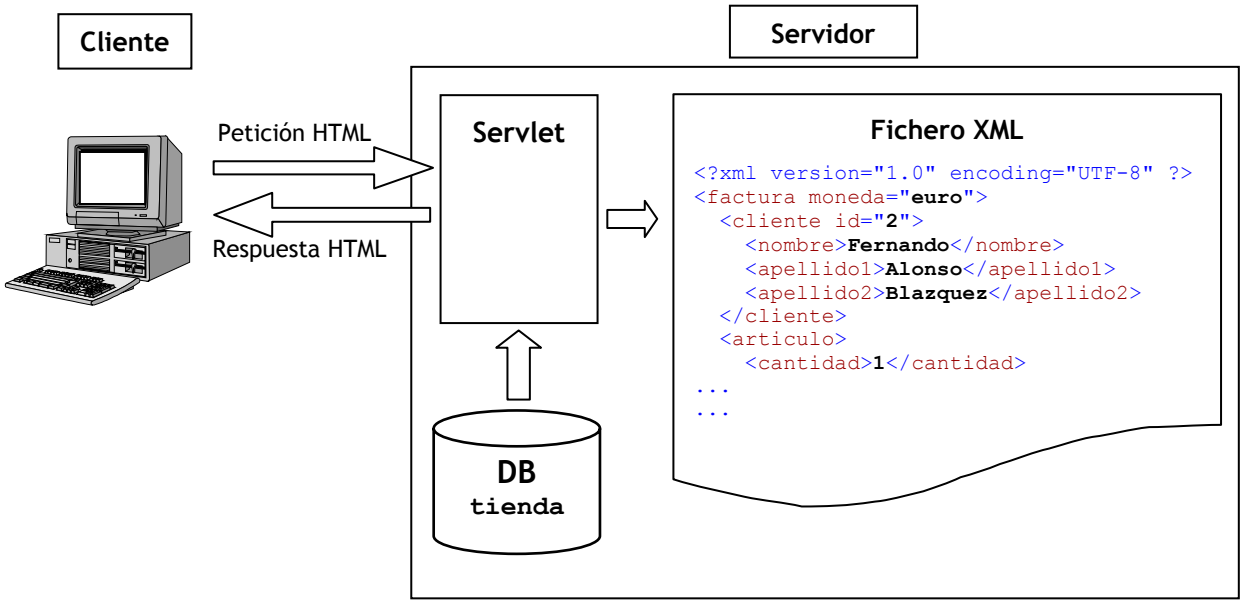


Figura 11.2. Operativa del servlet `Emitir_Factura_xml.java`

Analizando un documento XML mediante las APIs de Java para XML

El análisis de un fichero en formato XML con los datos de una factura lo llevan a cabo los servlet `Leer_Factura_xml_SAX.java` y `Leer_Factura_xml_DOM.java`, que siguen un esquema como el de la figura 11.3.

Ambos servlets reciben a través de HTTP el número de una factura previamente grabada en formato XML (que utilizará como nombre del fichero XML a leer). Lee el fichero XML con el nombre indicado del directorio señalado por la propiedad **`facturasPath`** y lo analiza mediante un analizador sintáctico (*parser*). Mediante este análisis sintáctico genera una factura en formato HTML que devuelve a través de HTTP.

El servlet `Leer_Factura_xml_SAX.java` usa el modelo SAX (*Simple API for XML*) de *parser*, que se trata de un analizador sintáctico secuencial, es decir, recorre el fichero XML de principio a fin de forma secuencial y lanza "eventos" a medida que encuentra etiquetas o TAGs.

El servlet `Leer_Factura_xml_DOM.java` usa el modelo DOM (*Document Object Model*) de parser, que se trata de un analizador sintáctico que genera un modelo del documento XML en forma de árbol (forma natural de los ficheros XML) que más tarde se puede recorrer para dar formato.

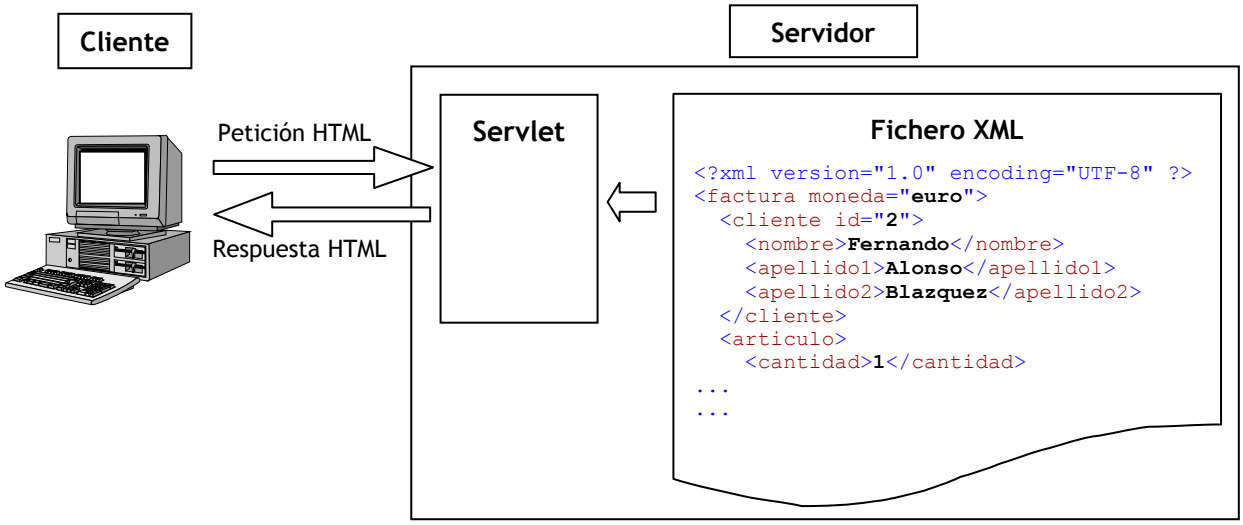


Figura 11.3. Operativa de los servlets `Leer_Factura_xml_SAX.java` y `Leer_Factura_xml_DOM.java`

Práctica 12: Petición en varias capas de servidores mediante XML

En esta práctica se trata de utilizar el formato XML para la transmisión de datos desde un servidor a otro como se muestra en la figura 12.1. Según la figura, el cliente hace una petición HTML de una factura al servidor 1 pero la información se encuentra en el servidor 2. El servidor 1 hace la petición al servidor 2, el cual accede a la Base de Datos, obtiene la información, la pone en formato XML y la devuelve al servidor 1. Este lee y analiza la respuesta XML y le da formato HTML para devolvérselo al cliente.

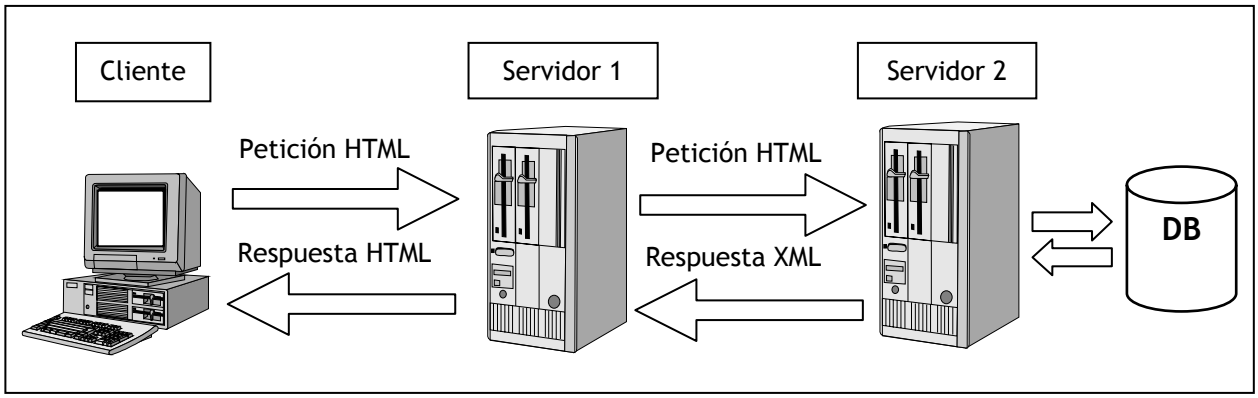


Figura 12.1. Petición en varias capas de servidores mediante XML

Puesta en marcha

Descarga el fichero ***Practica12.zip*** de la zona de Material de la web de la asignatura y descomprímelo en un directorio propio, por ejemplo: `G:\Infor3\Practica12\`

La aplicación consta de los siguientes archivos:

Tipo de Archivo	Nombre
Servlets	`Facturacion.java`
	`Servidor1.java`
	`Servidor2.java`

La aplicación tomará los datos para emitir las facturas de la Base de Datos **`Tienda.mdb`** de la aplicación "Tienda de Discos" de la práctica 10, a través del DSN con nombre **`tienda`**, el cual ya debe estar presente en el sistema.

Se deben compilar todos los servlets como en prácticas anteriores. Ahora bien, para que el compilador pueda entender las clases de la DOM API para XML. Si se ha realizado la práctica 11 esto ya debe estar resuelto y no debe suponer un problema.

Para simular la operativa de dos servidores nos vamos a servir de dos **`servletrunner`** ejecutándose en dos consolas de MS-DOS diferentes y escuchando a dos puertos diferentes.

Si el directorio en el que se ha guardado la práctica fuese el mismo que el ejemplo (`G:\Infor3\Practica12\`), los servidores se arrancarían con las instrucciones:

- Consola 1 de MS-DOS (simula Servidor 1):

```
servletrunner -p 8081 -d G:\Infor3\Practica12\
```

- Consola 2 de MS-DOS (simula Servidor 2):

```
servletrunner -p 8082 -d G:\Infor3\Practica12\
```

Para iniciar la aplicación se debe invocar el servlet `Facturacion.java` desde la barra del navegador mediante la instrucción:

```
http://localhost:8081/servlet/Facturacion
```

que muestra una pantalla como la de la figura 12.2.

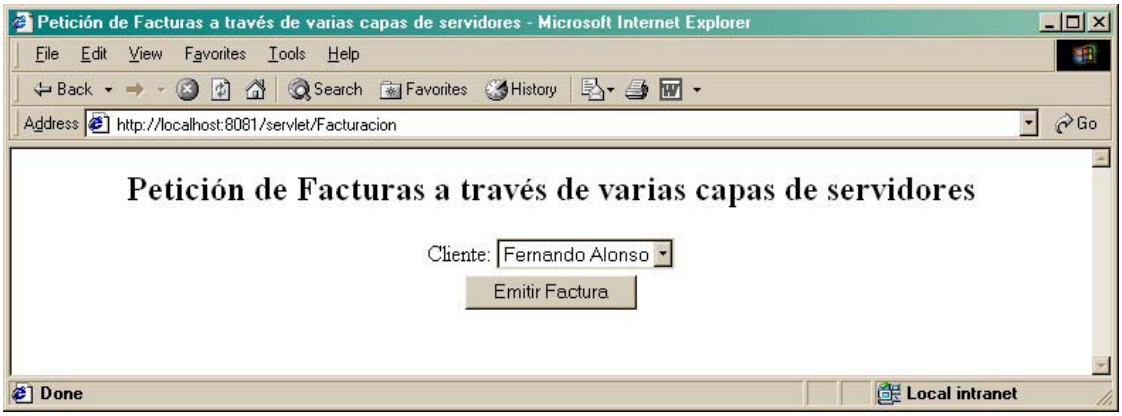


Figura 12.2. Pantalla de inicio, resultado del servlet `Facturacion.java`

Análisis de los servlets

El servlet `Servidor1.java` que se encuentra en el primer servidor (comprobar la consola 1 de MS-DOS para ver que dicho servlet ha sido inicializado) recibe a través de HTTP el identificador del cliente del cual se quiere la factura e invoca a su vez al servlet `Servidor2.java` que se encuentra en el segundo servidor (comprobar la consola 2 de MS-DOS para ver que dicho servlet ha sido inicializado), pasándole como parámetro dicho identificador de cliente.

El servlet `Servidor2.java` accede a la Base de Datos, obtiene los datos del carrito de la compra de ese cliente, transforma la información de dicho carrito a XML y se la devuelve al servlet `Servidor1.java` en este mismo formato.

Es entonces cuando el servlet `Servidor1.java` analiza el documento XML recibido mediante un *parser* que usa el modelo DOM y pone la información de la factura en formato HTML, que devuelve posteriormente al cliente a través de HTTP.