12. Aplicaciones web en Java (JSP)

Hasta el momento, los programas realizados funcionan en una austera ventana de terminal. No se trata de algo casual, hemos elegido esta manera de trabajar de forma deliberada, para que el estudiante se centre en resolver el problema y no se distraiga dándole sombra a un botón o escogiendo la mejor imagen de fondo para un listado.

Llegamos, no obstante, a un punto en que algunos programas piden a gritos una apariencia más vistosa.

En este capítulo vamos a aprender cómo utilizar páginas web como interfaz para programas en Java. Usaremos JSP (JavaServer Pages) que nos permitirá mezclar código en Java con código HTML. El código en Java que utilizaremos será muy parecido al que hemos venido utilizando hasta ahora. Cambiará únicamente lo relativo a mostrar información por pantalla (ahora se volcará todo a HTML) y la manera en que se introducen los datos, que se realizará mediante formularios.

El lector deberá tener unos conocimientos básicos de HTML para entender bien y sacar provecho de este capítulo. Una web excelente para aprender HTML es W3Schools[1]

El estándar de programación de Google para el código fuente escrito en Java[2] no especifica ninguna regla en cuanto a la manera de nombrar los ficheros correspondientes a las aplicaciones JSP. Por tanto vamos a adoptar las convenciones de Oracle[3] en esta materia. Según estas convenciones, los nombres de los ficheros JSP deben estar escritos en *lowerCamelCase*, es decir, la primera letra siempre es minúscula y, en caso de utilizar varias palabras dentro del nombre, éstas van seguidas una de otra empezando cada una por mayúscula. Finalmente se añade la extensión `.jsp`. Por ejemplo, `listadoSocios.jsp` es un nombre correcto, mientras que `ListadoSocios.jsp`, `listado_socios.jsp` o `Listadosocios.jsp` son incorrectos.

12.1 Hola Mundo en JSP

Para desarrollar aplicaciones en JSP utilizaremos el entorno de desarrollo **Netbeans**. Este IDE se puede descargar de forma gratuita en https://netbeans.org/downloads/ [4]

Sigue estos pasos para crear una aplicación en JSP con **Netbeans**:

[1] http://w3schools.com

[2] http://google-styleguide.googlecode.com/svn/trunk/javaguide.html

[3] http://www.oracle.com/technetwork/articles/javase/code-convention-138726.html

[4] Para más detalles sobre la instalación y configuración de **Netbeans**, ver el Apéndice B.

- En el menú principal, selecciona **Archivo** y a continuación **Nuevo Proyecto**. Se puede hacer más rápido con la combinación **Control + Mayúscula + N**.
- Aparecerá una ventana emergente para elegir el tipo de proyecto. Selecciona **Java Web** en el apartado "Categorías" y **Aplicación Web** en el apartado "Proyectos". Haz clic en **Siguiente**.
- Ahora hay que darle el nombre al proyecto, lo puedes llamar por ejemplo **Saludo1**. Recuerda no utilizar caracteres especiales ni signos de puntuación. Se creará una carpeta con el mismo nombre que el proyecto que contendrá todos los ficheros necesarios. Haz clic en **Siguiente**.
- En la siguiente ventana tendrás que elegir el servidor, la versión de Java y la ruta. Elige **Glass Fish Server**. En caso de no haber ningún servidor disponible, **Netbeans** permite añadir uno sobre la marcha (se lo descarga y lo instala). Haz clic en **Finalizar**.

A la izquierda, en la ventana de proyectos, debe aparecer el que acabas de crear: **Saludo1**. Navegando por la estructura del proyecto, verás que hay una carpeta con nombre **Web Pages**; en esta carpeta se deberán guardar los archivos correspondientes a la aplicación: archivos jsp, html, css, imágenes, archivos de audio, etc. Para que los archivos estén ordenados, se pueden organizar, a su vez, en subcarpetas dentro de **Web Pages**.

Por defecto, cuando se crea un proyecto JSP, se crea el archivo `index.html`. Elimina este archivo de tu proyecto y luego crea `index.jsp` haciendo clic con el botón derecho en **Web Pages → Nuevo → JSP**. Al indicar el nombre no hay que poner la extensión.

Ya tienes un "Hola Mundo" en inglés. Si lo quieres en español solo tienes que cambiar "Hello World" por "Hola Mundo", con lo que la aplicación quedaría como sigue:

```
<%-- saludo1.jsp --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <h1>¡Hola Mundo!</h1>
    En esta página se usa únicamente HTML.
  </body>
</html>
```

Para ejecutar la aplicación hay que hacer clic en el botón verde **Ejecutar Proyecto** o pulsar **F6**. En ese momento, **Netbeans** buscará un fichero con el nombre `index.jsp` y lo lanzará en el navegador.

Observa que el programa del ejemplo se llama `saludo1.jsp` y no `index.jsp`. En este caso, para ejecutar la aplicación hay que hacer clic con el botón derecho encima de `saludo1.jsp` y seleccionar `Ejecutar`.

De momento, solo hemos visto código HTML ¿dónde está Java? Bien, vamos a verlo, pero antes hay entender bien cómo funciona JSP.

Una aplicación JSP consistirá en una o varias páginas web que normalmente contendrá código HTML y que llevarán "insertado" código en Java. Este código en Java puede colocarse en cualquier parte del archivo y se delimita mediante las etiquetas `<%` y `%>`. Cuando se pulsa **F6** para ejecutar la aplicación sucede lo siguiente:

1. Todo el código (tanto HTML como JAVA) se envía al servidor (Glass Fish o Apache Tomcat por ejemplo).
2. El servidor deja intacto el código HTML y compila y ejecuta el código en Java. Generalmente el código Java genera código HTML mediante las instrucciones `out.print` y `out.println`.
3. Por último, todo el código (ya solo queda HTML) se envía al navegador que es el que muestra la página. Recuerda que el navegador no entiende Java (entiende Javascript que es otro lenguaje diferente) y es imprescindible que todo el código Java haya sido traducido previamente.

12.2 Mezclando Java con HTML

A continuación se muestra un ejemplo que sí tiene Java. Como ves, el código Java está delimitado por los caracteres `<%` y `%>`.

```
<%-- saludo2.jsp --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <% out.println("<h1>¡Hola Mundo!</h1>"); %>
  </body>
</html>
```

Mediante Java, se puede generar cualquier contenido HTML, por ejemplo para modificar estilos, crear tablas, mostrar imágenes, etc. Trabajando en consola, lo que se incluye en un `print` es exactamente lo que se muestra por pantalla. Ahora, con JSP, lo que se incluye dentro del `print` es "renderizado" a posteriori por el navegador. En el siguiente ejemplo se puede ver claramente este comportamiento.

```
<%-- saludo3.jsp --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <h1>¡Hola Caracola!</h1>
    <% out.print("<b><i>"); %>
    Esta línea se ha puesto en negrita y cursiva mediante Java.
    <% out.print("</i></b>"); %>
  </body>
</html>
```

Es muy útil ver el código fuente que se ha generado una vez que se puede visualizar la aplicación en el navegador. Para ver el código fuente tanto en **Firefox** como en **Chrome**, hay que hacer clic con el botón derecho en la página y seleccionar la opción "Ver código fuente de la página".

Cuando el contenido que se quiere volcar en HTML es el resultado de una expresión, se pueden utilizar de forma opcional las etiquetas `<%=` y `%>`.

```
<%-- saludo4.jsp --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <h1>¡Hola Caracola!</h1>
    <%= "<b><i>" %>
    Esta línea se ha puesto en negrita y cursiva mediante Java.
    <%= "</i></b>" %>
  </body>
</html>
```

En el código Java que se inserta en las aplicaciones JSP se pueden utilizar todos los elemento del lenguaje vistos anteriormente: bucles, sentencias de control, arrays, diccionarios, etc.

En el siguiente ejemplo se utiliza un bucle `for` para mostrar texto en diferentes tamaños, utilizando las etiquetas desde la `<h6>` (cabecera pequeña) hasta la `<h1>` (cabecera más grande).

```
<%-- pruebaVariable1 --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <%
      for (int i = 1; i < 7; i++)
        out.println("<h" + (7-i) + ">"+ i + "</h" + (7-i) + ">");
    %>
  </body>
</html>
```

El código en Java se puede insertar en cualquier parte dentro del código HTML, e incluso, se pueden insertar varios trozos como se puede ver en el siguiente ejemplo. Aunque una variable se haya definido en una determinada zona, su ámbito de actuación no se ve reducido a ese fragmento de código sino que se puede utilizar posteriormente en otro lugar de la aplicación.

Observa que en el siguiente ejemplo, la definición e inicialización de la variable `x` se realiza en un bloque de código en Java y que luego esta variable se utiliza en otros dos bloques diferentes.

```
<%-- pruebaVariable2 --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <% int x = 3;  %>
    <h<% out.print(x); %>>hola</h<% out.print(x); %>>
  </body>
</html>
```

En ocasiones puede ser útil recabar información del sistema: versión de Java en uso, sistema operativo, nombre de usuario, etc. Estos datos se pueden obtener mediante `System.getProperty(propiedad)`.

```
<%-- muestraInfo.jsp --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <h1>Información del entorno de trabajo</h1>
    <%
      out.print("Fabricante de Java: " + System.getProperty("java.vendor"));
      out.print("<br>Url del fabricante: " + System.getProperty("java.vendor.url"));
      out.print("<br>Versión: " + System.getProperty("java.version"));
      out.print("<br>Sistema operativo: " + System.getProperty("os.name"));
      out.print("<br>Usuario: " + System.getProperty("user.name"));
    %>
  </body>
</html>
```

Para mostrar información en una página de forma ordenada es muy frecuente utilizar tablas. Las tablas son especialmente útiles para mostrar listados obtenidos de una base de datos como veremos en el siguiente capítulo.

En el siguiente ejemplo se muestra una tabla con dos columnas; en la primera columna se muestran los números del 0 al 9 y en la segunda, sus correspondientes cuadrados.

```
<%-- tabla.jsp --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <body>
    <h1>Ejemplo de tabla</h1>
    <table border="2">
      <tr>
        <td>Número</td><td>Cuadrado</td>
      </tr>
      <%
        for(int i = 0; i < 10; i++) {
          out.println("<tr>");
          out.println("<td>" + i + "</td>");
          out.println("<td>");
          out.println(i * i);
          out.println("</td></tr>");
        }
```

```
      %>
    </table>
  </body>
</html>
```

12.3 Recogida de datos en JSP

En cualquier aplicación web, la introducción de datos se realiza mediante formularios. Aunque se puede hacer todo en la misma página, de momento vamos a tener dos. La primera página contendrá un formulario y será la encargada de recoger la información y enviarla a una segunda página. Esta última página recibirá los datos, realizará una serie de cálculos u operaciones si procede y, por último, mostrará un resultado.

En el primer ejemplo - un proyecto que llamaremos `PasoDeCadena` - tenemos una página con nombre `index.jsp` que contiene un formulario HTML. Este formulario contiene una entrada de texto donde el usuario introducirá una cadena de caracteres. Es muy importante darle un valor a la etiqueta `name`. En el caso que nos ocupa tenemos `name="cadenaIntro"`, por tanto el dato que recoge el formulario se llama de esa manera, sería el equivalente al nombre de la variable en Java. No menos importante es indicar la página que recibirá los datos que recoge el formulario. Esta información se indica con la etiqueta `action`, en nuestro ejemplo `action="frase.jsp"` indica que será el fichero `frase.jsp` el que recibirá `cadenaIntro`.

El fichero `index.jsp` únicamente contiene código HTML, por tanto se podría llamar `index.html` y la aplicación funcionaría exactamente igual.

```
<%-- index.jsp (proyecto PasoDeCadena) --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Paso de cadena</title>
  </head>
  <body>
    <h1>Pasando una cadena de caracteres</h1>
    <form method="post" action="frase.jsp">
      Introduzca el nombre de una fruta:
      <input type="text" name="cadenaIntro">
      <input type="submit" value="OK">
    </form>
  </body>
</html>
```

Para recoger los datos enviados por un formulario se utiliza `request.getParameter("nombreDeVariable")` donde `nombreDeVariable` es el dato que se envía desde el formulario. En caso de que el dato enviado sea un texto que pueda contener tildes o eñes, hay que especificar la codificación mediante `request.setCharacterEncoding("UTF-8")` antes de recoger el dato.

```
<%-- frase.jsp (proyecto PasoDeCadena) --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Paso de cadena</title>
  </head>
  <body>
    <% request.setCharacterEncoding("UTF-8"); %>
    Me gusta mucho comer
    <% out.print(request.getParameter("cadenaIntro"));%>
  </body>
</html>
```

En el siguiente ejemplo, llamado `Incrementa5`, se muestra cómo manipular en JSP un dato numérico. La aplicación recoge un número y luego muestra ese número incrementado en 5 unidades.

```
<%-- index.jsp (proyecto Incrementa5) --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <form method="get" action="incrementa5.jsp">
      Introduzca un número (puede tener decimales):
      <input type="text" name="numeroIntro">
      <input type="submit" value="OK">
    </form>
  </body>
</html>
```

Observa en el fichero `incrementa5.jsp` cómo se transforma la cadena de caracteres que se recibe en `numeroIntro` en un dato numérico con el método `Double.parseDouble`; exactamente igual que si estuviéramos leyendo un número desde teclado en la ventana de terminal.

```
<%-- incrementa5.jsp (proyecto Incrementa5) --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    El número introducido más cinco es
    <%
      double resultado;
      resultado = Double.parseDouble(request.getParameter("numeroIntro")) + 5;
      out.print(resultado);
    %>
  </body>
</html>
```

El siguiente ejemplo ilustra la recogida y envío de dos variables, `x` e `y`. Observa que ahora el formulario contiene dos entradas de texto, una para cada variable.

```
<%-- index.jsp (proyecto Suma) --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Suma</title>
  </head>
  <body>
    <h1>Supercalculadora</h1>
    <form method="get" action="resultado.jsp">
      x <input type="text" name="x"/></br>
      y <input type="text" name="y"/></br>
      <input type="submit">
    </form>
  </body>
</html>
```

En el fichero `resultado.jsp` se reciben las variables recogidas en `index.jsp` y se suman. Recuerda que por defecto la información enviada por un formulario es una cadena de caracteres. Si queremos sumar los valores introducidos, debemos transformar las cadenas de caracteres a números enteros con el método `Integer.valueOf()`.

```
<%-- resultado.jsp (proyecto Suma) --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Suma</title>
  </head>
  <body>
    La suma es
    <%
      int primerNumero = Integer.valueOf(request.getParameter("x"));
      int segundoNumero = Integer.valueOf(request.getParameter("y"));
      out.println(primerNumero + segundoNumero);
    %>
  </body>
</html>
```

El siguiente proyecto de ejemplo es `Animales`. En la página principal (`index.jsp`) se puede seleccionar un animal - gato o caracol - y un número. Una vez introducidos los datos, éstos se envían a `animales.jsp`.

```
<%-- index.jsp (proyecto Animales) --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Animales</title>
  </head>
  <body>
    <form method="post" action="animales.jsp">
      Seleccione animal a visualizar
      <select name="animal">
        <option>Gato</option>
        <option>Caracol</option>
      </select>
      </br>
      Número de animales <input type="text" name="numero" size="3">
      </br>
      <input type="submit">
    </form>
  </body>
</html>
```

La página `animales.jsp` nos mostrará una imagen del animal elegido repetida el número

de veces que hemos indicado.

```
<%-- animales.jsp (proyecto Animales) --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Animales</title>
  </head>
  <body>
    <%
      String nombreAnimal = request.getParameter("animal");
      String nombreImagen;
      if (nombreAnimal.equals("Gato")) {
        nombreImagen = "gato.jpg";
      } else {
        nombreImagen = "caracol.jpg";
      }

      int veces = Integer.parseInt(request.getParameter("numero"));

      for (int i = 0; i < veces; i++) {
        out.print("<img src=\"" + nombreImagen +"\" width=\"20%\">");
      }
     %>
  </body>
</html>
```

12.4 POO en JSP

En las aplicaciones realizadas en JSP se pueden incluir clases definidas por el usuario para posteriormente crear objetos de esas clases. Lo habitual es que el fichero que contiene la definición de la clase se encuentre separado del programa principal.

En un nuevo proyecto con nombre **GatosConClase** vamos a crear la clase `Gato`; para ello haz clic derecho sobre el icono del proyecto, selecciona **Nuevo** y luego selecciona **Clase de Java**. El nombre de la clase será `Gato` y el nombre del paquete será por ejemplo `daw1`.

El fichero `Gato.java` quedaría como el que se muestra a continuación.

```
/* Gato.java (proyecto GatosConClase) */

package daw1;

public class Gato {
  private String nombre;
  private String imagen;

  public Gato(String nombre, String imagen) {
    this.nombre = nombre;
    this.imagen = imagen;
  }

  public String getNombre() {
    return nombre;
  }

  public String getImagen() {
    return imagen;
  }

  @Override
  public String toString() {
    return "<img src='" + imagen + "' width='80'>Hola, soy " + nombre + "<br>";
  }

  public String maulla() {
    return "<img src='" + imagen + "' width='80'>Miauuuuuuuu<br>";
  }

  public String come(String comida) {
    return "<img src='" + imagen + "' width='80'>Estoy comiendo " + comida + "<br>";
  }
}
```

Hemos creado una clase `Gato` muy sencilla que únicamente contiene dos atributos: el nombre del gato y el nombre del fichero de su foto. Como métodos, se han implementado `toString()`, `maulla()` y `come()`.

```
<%-- index.jsp (proyecto GatosConClase) --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="daw1.Gato"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Gatos con clase</title>
  </head>
  <body>
    <h1>Gatos con clase</h1>
    <hr>
    <%
      Gato g1 = new Gato("Pepe", "gato.jpg");
      Gato g2 = new Gato("Garfield", "garfield.jpg");
      Gato g3 = new Gato("Tom", "tom.png");

      out.println(g1);
      out.println(g2);
      out.println(g3);
      out.println(g1.maulla());
      out.println(g2.come("sardinas"));
    %>
  </body>
</html>
```

El código Java del programa principal es casi idéntico al que tendríamos en un programa para consola.

El siguiente ejemplo - GatosConClaseYBocadillos - es una mejora del anterior. Se añaden estilos[5] para mostrar lo que dicen los gatos, igual que en un cómic.

```
/* Gato.java (proyecto GatosConClaseYBocadillos) */

package daw1;

public class Gato {
  private String nombre;
  private String imagen;

  public Gato(String nombre, String imagen) {
    this.nombre = nombre;
    this.imagen = imagen;
  }
```

[5] https://github.com/LuisJoseSanchez/aprende-java-con-ejercicios/blob/master/ejemplos/12_JSP/GatosConClaseYBocadillos/estilos.css

```
  public String getNombre() {
    return nombre;
  }

  public String getImagen() {
    return imagen;
  }

  @Override
  public String toString() {
    return "<div class=\"acciongato\"><img src=\"" + imagen + "\" width=\"80\"><div class=\"ar\
row_box\"> Hola, soy " + nombre + " </div></div>";
  }

  public String maulla() {
    return "<div class=\"acciongato\"><img src=\"" + imagen + "\" width=\"80\"><div class=\"ar\
row_box\"> Miauuuuuuuu </div></div>";
  }

  public String come(String comida) {
    return "<div class=\"acciongato\"><img src=\"" + imagen + "\" width=\"80\"><div class=\"ar\
row_box\"> Estoy comiendo " + comida + " </div></div>";
  }
}
```

El único añadido al programa principal es una línea situada en la cabecera de la página que carga la hoja de estilos `estilos.css`.

```
<%-- index.jsp (proyecto GatosConClaseYBocadillos) --%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="daw1.Gato"%>
<!DOCTYPE html>
<html>
    <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Gatos con clase y bocadillos</title>
                  <link rel="stylesheet" type="text/css" href="estilos.css" />
    </head>
    <body>
      <h1>Gatos con clase</h1>
      <hr>
      <%
        Gato g1 = new Gato("Pepe", "gato.jpg");
        Gato g2 = new Gato("Garfield", "garfield.jpg");
        Gato g3 = new Gato("Tom", "tom.png");
```

```
        out.println(g1);
        out.println(g2);
        out.println(g3);
        out.println(g1.maulla());
        out.println(g2.come("sardinas"));
      %>
    </body>
</html>
```

12.5 Ejercicios

Ejercicio 1

Crea una aplicación web en Java que muestre tu nombre y tus datos personales por pantalla. La página principal debe ser un archivo con la extensión `jsp`. Comprueba que se lanzan bien el servidor y el navegador web. Observa los mensajes que da el servidor. Fíjate en la dirección que aparece en la barra de direcciones del navegador.

Ejercicio 2

Mejora el programa anterior de tal forma que la apariencia de la página web que muestra el navegador luzca más bonita mediante la utilización de CSS (utiliza siempre ficheros independientes para CSS para no mezclarlo con HTML).

Ejercicio 3

Escribe una aplicación que pida tu nombre. A continuación mostrará “Hola” seguido del nombre introducido. El nombre se deberá recoger mediante un formulario.

Ejercicio 4

Realiza una aplicación que calcule la media de tres notas.

Ejercicio 5

Realiza un conversor de euros a pesetas.

Ejercicio 6

Realiza un conversor de pesetas a euros.

Ejercicio 7

Combina las dos aplicaciones anteriores en una sola de tal forma que en la página principal se pueda elegir pasar de euros a pesetas o de pesetas a euros. Adorna la página con alguna foto o dibujo.

Ejercicio 8

Realiza una aplicación que pida un número y que luego muestre la tabla de multiplicar de ese número. El resultado se debe mostrar en una tabla (`<table>` en HTML).

Ejercicio 9

Realiza una aplicación que pinte una pirámide. La altura se pedirá en un formulario. La pirámide estará hecha de bolitas, ladrillos o cualquier otra imagen.

Ejercicio 10

Realiza un cuestionario con 10 preguntas tipo test sobre las asignaturas que se imparten en el curso. Cada pregunta acertada sumará un punto. El programa mostrará al final la calificación obtenida. Pásale el cuestionario a tus compañeros y pídeles que lo hagan para ver qué tal andan de conocimientos en las diferentes asignaturas del curso. Utiliza *radio buttons* en las preguntas del cuestionario.

Ejercicio 11

Escribe una aplicación que genere el calendario de un mes. Se pedirá el nombre del mes, el año, el texto que queremos que aparezca sobre el calendario, el día de la semana en que cae el día 1 y el número de días que tiene el mes.

Ejercicio 12

Mejora la aplicación anterior de tal forma que no haga falta introducir el día de la semana en que cae el día 1 y el número de días que tiene el mes. El programa debe deducir estos datos del mes y el año. Pista: puedes usar la clase `Calendar` (`java.util.Calendar`).

Ejercicio 13

Transforma el test de infidelidad realizado anteriormente para consola en una aplicación web.

Ejercicio 14

Escribe un programa que muestre los n primeros términos de la serie de Fibonacci. El primer término de la serie de Fibonacci es 0, el segundo es 1 y el resto se calcula sumando los dos anteriores, por lo que tendríamos que los términos son 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144… El número n se debe introducir por teclado.

Ejercicio 15

Realiza una aplicación que genere 100 números aleatorios del 1 al 200. Los primos deberán aparecer en un color diferente al resto.

Ejercicio 16

Realiza una aplicación que muestre la tirada aleatoria de tres dados de póker. Utiliza imágenes de dados reales.

Ejercicio 17

Realiza un configurador del interior de un vehículo. El usuario puede elegir, mediante un formulario, el color de la tapicería – blanco, negro o berenjena - y el material de las molduras – madera o carbono. Se debe mostrar el interior del coche tal y como lo quiere el usuario.

Ejercicio 18

Crea la aplicación “El Trile”. Deben aparecer tres cubiletes por pantalla y el usuario deberá seleccionar uno de ellos. Para dicha selección se puede usar un formulario con radio-button, una lista desplegable, hacer clic en el cubilete o lo que resulte más fácil. Se levantarán los tres cubiletes y se verá si estaba o no la bolita dentro del que el usuario indicó, así mismo, se mostrará un mensaje diciendo “Lo siento, no has acertado” o “¡Enhorabuena!, has encontrado la bolita”. La probabilidad de encontrar la bolita es de 1/3.

Ejercicio 19

Crea el juego “Apuesta y gana”. El usuario debe introducir inicialmente una cantidad de dinero. A continuación aparecerá por pantalla una imagen de forma aleatoria. Si sale una calavera, el usuario pierde todo su dinero y termina el juego. Si sale medio limón, el usuario pierde la mitad del dinero y puede seguir jugando con esa cantidad o puede dejar de jugar. Si sale el gato chino de la suerte, el usuario multiplica por dos su dinero y puede seguir jugando con esa cantidad o puede dejar de jugar.

Ejercicio 20

Crea una aplicación que dibuje un tablero de ajedrez mediante una tabla HTML generada con bucles usando JSP y que sitúe dentro del tablero un alfil y un caballo en posiciones aleatorias. Las dos figuras no pueden estar colocadas en la misma casilla. Las filas y las columnas del tablero deben estar etiquetadas correctamente.

Ejercicio 21

Implementa una máquina de helados. El usuario indica los **porcentajes de helado de chocolate, de fresa y de vainilla**. Los porcentajes deben ser números comprendidos entre 0 y 100. Si los porcentajes suman más de 100, se debe dar un mensaje al usuario que diga **“La suma de los porcentajes debe ser menor o igual que 100. Por favor, introduzca de nuevo los porcentajes”**. En caso de que los porcentajes sean correctos, se mostrará la tarrina con los sabores adecuados y los tamaños bien dimensionados según los porcentajes, tal y como se muestra en los ejemplos. Hay que tener en cuenta que si los sabores no suman el 100% de la tarrina, se debe mostrar el hueco correspondiente. En caso de que no se incluya algún sabor (sabor al 0%), no debe aparecer ninguna referencia a dicho sabor en la tarrina.

Ejemplo 1:

Máquina de helados

Seleccione los porcentajes para preparar un helado a su gusto.


Chocolate 0 %


Fresa: 0 %


Vainilla: 0 %

Preparar helado

La suma de porcentajes no pueden ser mayor que el 100%
Por favor, introduzca de nuevo los porcentajes.

Ejemplo 2:

Máquina de helados

Seleccione los porcentajes para preparar un helado a su gusto.


Chocolate 40 %

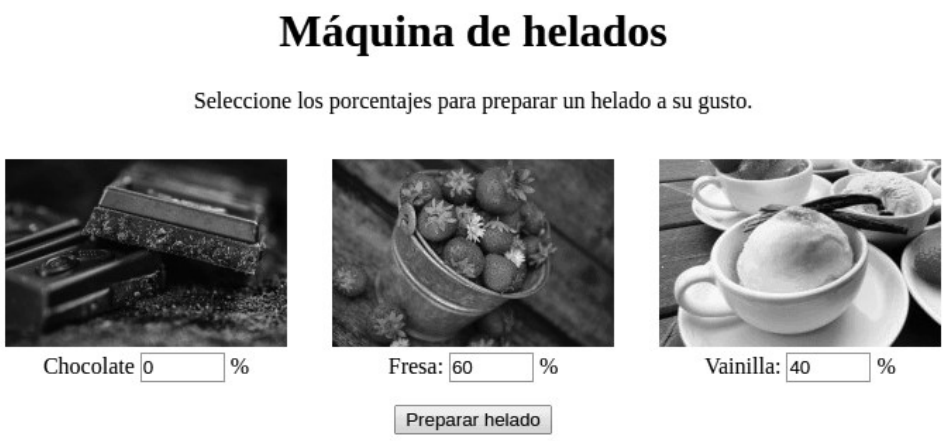

Fresa: 30 %


Vainilla: 15 %

Preparar helado

Aquí tiene su tarrina de helado

Ejemplo 3:



Ejercicio 22

Una empresa que sirve comida vegetariana a domicilio necesita una aplicación para que los clientes puedan hacer sus pedidos por internet. La primera versión incluirá cuatro comidas y tres bebidas. A continuación se muestra una tabla con las diferentes opciones y precios.

Quinoa con verdura	Pizza caprese	Pasta al pesto	Hamburguesa vegetariana	Agua	Cerveza	Refresco
6.95 €	5.50 €	4.90 €	6.20 €	1.00 €	1.50 €	1.40 €

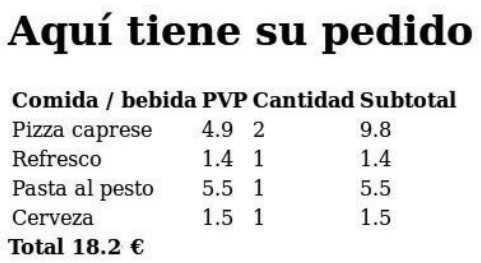
Un pedido puede contener varias comidas del mismo o de distinto tipo y también varias bebidas.
Ejemplos de pedidos:

- 1 quinoa con verdura, 2 hamburguesas vegetarianas, un botellín de agua, una cerveza y un refresco. En total sería 6.95 + 2 x 6.20 + 1 + 1.50 + 1.40 = 23.25 euros.
- 2 pizzas caprese sin bebida. En total sería 2 x 4.90 = 9.80 euros.

Ejemplo:

Formulario de pedido:

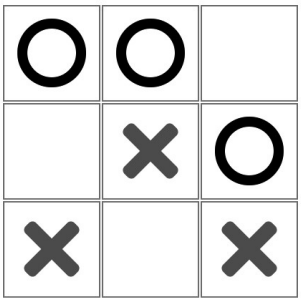
Pantalla con el resultado:



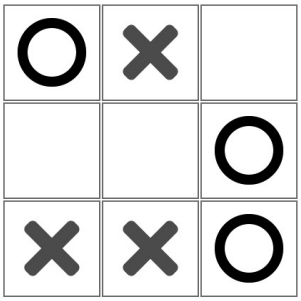
Ejercicio 23

Realiza una aplicación que genere de forma aleatoria una partida finalizada del juego del tres en raya teniendo en cuenta que el tablero tiene tres filas por tres columnas y hay tres círculos y tres cruces que no se pueden solapar. No hay que programar el juego, simplemente mostrar cómo quedaría una partida una vez que ha finalizado.

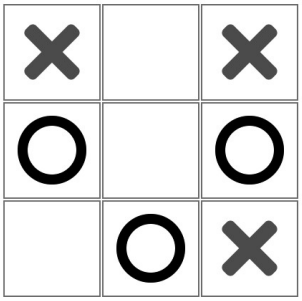
Ejemplo 1:



Ejemplo 2:



Ejemplo 3:



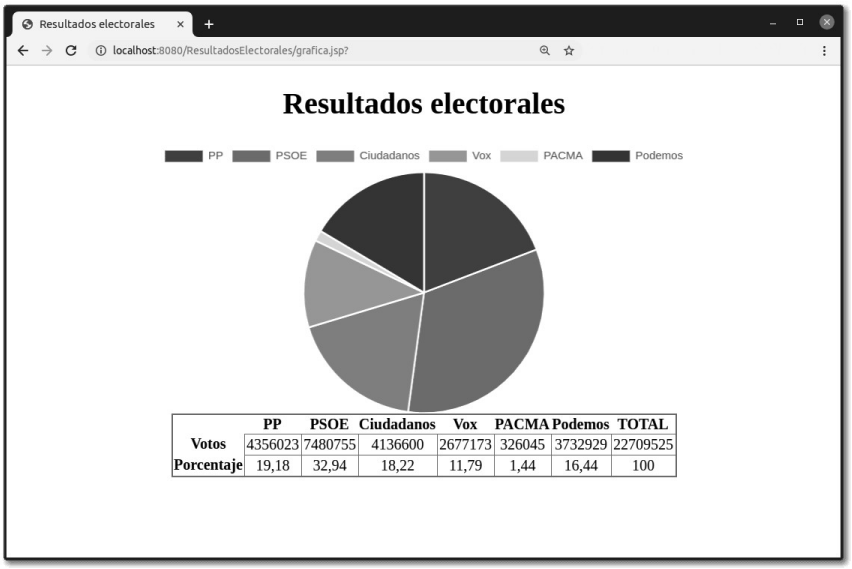
Ejercicio 24

Realiza una aplicación en JSP que recoja mediante un formulario los votos de los diferentes partidos políticos que concurren a las elecciones. A continuación, se debe mostrar una gráfica circular y una tabla con los votos y los porcentajes tal y como se muestra en el ejemplo. Se recomienda usar la librería ChartJs[6]

Ejemplo 1:

[6] https://www.chartjs.org/

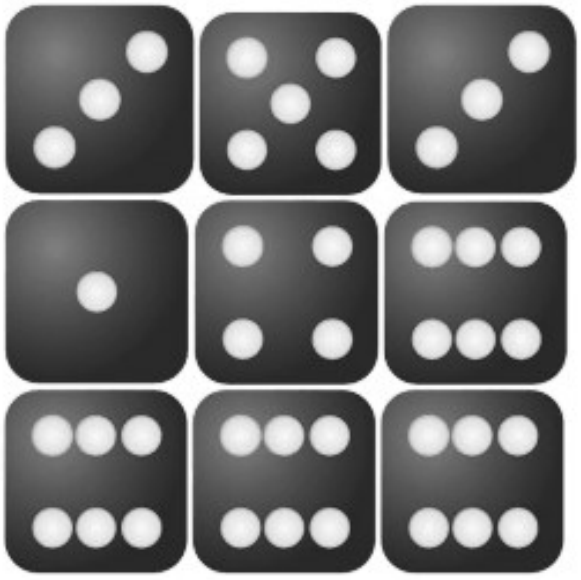
Ejemplo 2:



Ejercicio 25

Realiza un programa en JSP que haga tiradas consecutivas de tres dados (en la misma pantalla). El programa parará cuando, en la misma tirada, los tres dados tengan el mismo valor.

Ejemplo 1:



Ejemplo 2: