
Apuntes de Lenguajes de Marcas Documentation

Publicación 1.3

Oscar Gomez

oct. 05, 2016

1. Introducción	1
1.1. Historia	1
1.2. Servidores web	1
1.3. Los nombres de dominio	2
1.4. Los sistemas de gestión de información	2
1.5. Tipos de lenguajes	2
1.6. Organismos de regulación	3
1.7. Gramáticas y DTD's	3
1.8. XML Schemas	4
1.9. Definiciones	4
2. HTML5	5
2.1. Introducción	5
2.2. Etiquetas estructurales	5
2.3. Etiquetas de formato	6
2.4. Gestión de espacios	6
2.5. Entidades	6
2.6. Texto preformateado	7
2.7. Listas	7
2.8. Tablas	8
2.9. Formularios	13
2.10. Ejercicios tipo examen	15
3. CSS	21
3.1. Introducción	21
3.2. Posicionamiento	21
3.3. Gestión de espacios	28
3.4. Colores	28
3.5. Tipografías	28
3.6. Alineación del texto	29
3.7. Decoración del texto	29
3.8. Medidas	29
3.9. Selectores	29
3.10. Bootstrap	33
4. Javascript	37
4.1. Introducción	37
4.2. Tipos de datos	37

4.3.	Incrustando Javascript	37
4.4.	Decisiones	38
4.5.	Vectores o Arrays	38
4.6.	Bucles	38
4.7.	Ejercicio: media aritmética	39
4.8.	Ejercicio: desviación media	40
4.9.	Ejercicio: la mediana	40
4.10.	Funciones	40
4.11.	Programación OO	42
4.12.	Programación con JQuery	44
4.13.	Procesado de atributos	48
4.14.	Configurador de coches	53
4.15.	Dinamismo con Google Maps	54
5.	XML	61
5.1.	Introducción	61
5.2.	Un ejemplo sencillo	61
5.3.	Construcción de XML	61
5.4.	Validez	62
5.5.	Gramáticas	63
5.6.	Ejercicio I (DTD)	66
5.7.	Ejercicio II (DTD)	67
5.8.	Ejercicio (con atributos)	68
5.9.	Ejercicio	69
5.10.	Otras características de XML	72
5.11.	Ejercicio	74
5.12.	Ejercicio: mayorista de libros	75
5.13.	Ejercicio: fabricante de tractores	76
5.14.	Ejercicio: repeticiones de opciones	77
5.15.	Ejercicio	78
5.16.	Ejercicio	79
5.17.	Examen	80
6.	Recuperación de información	81
6.1.	Introducción	81
6.2.	Fundamentos de DOM con Java	81
6.3.	Ejemplo de base	81
6.4.	La clase Document	82
6.5.	Ejercicios	83
6.6.	Anexo	96
6.7.	Ejercicios para preparar examen	105
7.	Sindicación y transformación de contenidos	107
7.1.	Introducción	107
7.2.	RSS	107
7.3.	Adaptación y transformación de XML	110
8.	Sistemas de gestión de información	125
8.1.	Introducción	125
8.2.	Niveles en la empresa	125
8.3.	Almacenamiento en los SGI	126
8.4.	Puntos de vista en los SGI	127

Introducción

1.1 Historia

Los lenguajes de marcas son bastante antiguos aunque solo se han popularizado con la llegada de Internet.

Los comienzos de los lenguajes de marcas se pueden situar en el lenguaje SGML (Standard Generalized Markup Language). En realidad, lo más usado hoy día es HTML (HyperText Markup Language)

El fundamento de Internet y todas las tecnologías asociadas se basa en estándares abiertos e independientes de la tecnología. El organismo que regula estos estándares sin ninguna contrapartida a cambio es el [World Wide Web Consortium \(W3C\)](#).

El futuro de todas estas tecnologías sigue estando en sistemas abiertos (Firefox OS). Ahora mismo las tres grandes plataformas por orden de supremacía en el mercado son:

1. Android.
2. iPhone
3. Windows Phone.

1.2 Servidores web

Es un programa que “entiende” los protocolos HTTP y HTTPS y que atiende peticiones de navegadores sirviendo páginas a medida que se van solicitando. Tener un servidor no implica obligatoriamente un nombre de dominio.

Cuando un servidor recibe una petición examina sus directorios para ver si tiene ese archivo en el directorio que le han dicho, si es así, el fichero se transmite al que hizo la petición. Si no existe, el navegador devuelve un código 404 (recurso no encontrado). Un servidor web puede ser privado o alquilado, existiendo grandes diferencias entre la forma de gestionar ambos.

- Uno privado, tiene la ventaja de ofrecer un control absoluto. No siempre es fácil mantener un ordenador doméstico conectado 24x7. En especial, en España, las conexiones ADSL no suelen ofrecer las capacidades necesarias para un sitio web de tamaño mediano.
- Los alquilados suelen conllevar un mayor precio cuando se necesita mas espacio para alojar nuestros ficheros. Ofrecen muchas garantías como por ejemplo anchos de banda muy aceptables a precios bastante competitivos y sobre todo que permiten al diseñador web liberarse de la gestión de los recursos de red

1.2.1 Como configurar un servidor en casa

1. Se necesita un servidor web instalado en un equipo (Apache del paquete XAMPP)
2. Apuntar la IP del equipo en el que se instala Apache.
3. Entrar en el router y **abrir el puerto 80** indicando que se debe reenviar el tráfico a la IP que se apuntó.
4. Se debe averiguar la IP pública del router (por ejemplo podemos visitar [WhatsMyIp](#) ,
5. La IP que aparece es la que se puede dar a clientes o amigos para que naveguen por nuestro sitio web.
6. (Optativo) se puede alquilar un nombre de dominio y solicitar que ese nombre (loquesea.com) sea redirigido a nuestra IP pública.

1.3 Los nombres de dominio

El servicio de nombres de dominio es un sistema que convierte de direcciones tipo [www.loquesea.com](#) a direcciones IP. Esto permite que sea más fácil recordar direcciones de páginas. Sin embargo, DNS es un sistema muy complejo que funciona de forma distribuida entre distintos países.

Los nombres de dominio se resuelven de final a principio. La última parte se llama TLD o Top Level Domain o dominio de primer nivel.. Estos dominios son administrados por países ocupándose cada uno de ellos de los nombres o marcas que hay dentro de dichos países.

1.4 Los sistemas de gestión de información

Definamos primero algunos términos:

- Sistema: conjunto de elementos interrelacionados que colaboran en la consecución de un objetivo.
- Gestión: conjunto de operaciones que resultan de relevancia para una persona o empresa.
- Información: conjunto de datos que resultan de utilidad a las funciones de la empresa.

Un SGI no tiene por qué estar informatizado.

Son programas que se pueden adaptar las necesidades de la empresa y que a veces necesitan importar (o exportar) datos e información.

El uso de un formato unificado (normalmente basado en marcas) facilita enormemente los trasvases de datos.

1.5 Tipos de lenguajes

Aparte de los lenguajes de marcas típicos, existen otros de uso bastante común. Siendo el caso más conocido LaTeX.

Programas como LaTeX tienen la desventaja de obligar a aprender “marcas”. Sin embargo, los algoritmos de colocación del texto suelen ser más sofisticados que otros programas. Los programas de redacción de documentos más utilizados suelen ser los del tipo MS-Word o LibreOffice que son del tipo WYSIWYG (What You See Is What You Get).

Dentro de los tipos de formatos, RTF es un mecanismo público que permite describir el aspecto de documentos. Al ser público, muchos procesadores pueden implementarlo si necesidad de pagar “royalties”.

Adobe lidera la especificación de documentos PDF (Portable Document Format). Antes de PDF, existía un lenguaje abierto denominado PostScript. Por otro lado RTF o LaTeX, son lenguajes descriptivos, mientras que PostScript es un lenguaje completo

1.6 Organismos de regulación

1. La International Standards Organization emite estándares para la documentación.
2. El W3C (o WorldWideWeb Consortium) emite “technical recommendations” (o TR’s) que los interesados en la web pueden seguir para garantizar la interoperabilidad. Su web es muy útil

1.7 Gramáticas y DTD’s

Las gramáticas HTML indican el conjunto de reglas para determinar lo que se acepta o no se acepta. Si se elige una gramática (como por ejemplo la de HTML5) es muy recomendable respetar las reglas de esa gramática. A modo de ejemplo, esto es una página válida

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    Mi primera página
  </body>
</html>
```

y esto no lo es.

```
<!DOCTYPE html>
  <title>Esto es el título de la página
<body>
```

A lo largo del tiempo ha habido diversas versiones de HTML (con sus correspondientes gramáticas) y tales documentos deben llevar en la cabecera algo que diga a qué estándar se ciñen.

Las tres últimas familias de estándares han sido

- HTML4: muy permisivo, lo que dificulta a los navegadores el procesar el HTML dando lugar a que fuera bastante difícil para ellos el mostrar correctamente y de igual forma todos los HTML
- XHTML: es HTML con las estrictas reglas que impuso XML. Esto simplificó el desarrollo de navegadores y se avanzó en facilidad para mostrar páginas en distintos navegadores.
- HTML5: es una nueva revisión de XHTML en el que se han incluido nuevas posibilidades como etiquetas <audio> y <video> así como posibilidad de hacer muchas cosas desde JavaScript.

Un ejemplo de DTD, sería esto:

```
<!ELEMENT lista_de_personas (persona*)>
<!ELEMENT persona (nombre, fechanacimiento?, sexo?, numeroseguridadsocial?)>
<!ELEMENT nombre (#PCDATA) >
<!ELEMENT fechanacimiento (#PCDATA) >
<!ELEMENT sexo (#PCDATA) >
<!ELEMENT numeroseguridadsocial (#PCDATA)>
```

Y un ejemplo de archivo aceptado por esa DTD sería este

```
<lista_de_personas>
  <persona>
    <nombre> Pepe Pérez </nombre>
```

```
<sexo> Varón </sexo>
<numeroseguridadsocial>555</numeroseguridadsocial>
</persona>
<persona>
  <nombre> Angela Lopez </nombre>
  <fechanacimiento>13-2-1995</fechanacimiento>
  <sexo> Mujer </sexo>
  <numeroseguridadsocial>355</numeroseguridadsocial>
</persona>
</lista_de_personas>
```

1.8 XML Schemas

Los XML Schemas surgen para mejorar las faltas de precisión que tenían las DTD. Sin embargo, la mejora en la precisión de la definición ha implicado que escribir XML Schemas sea mucho más complicado.

Un ejemplo de XML Schema (tomado de Wikipedia):

```
<?xml version="1.0" encoding="UTF-8"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="Libro">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="Título"
            type="xsd:string"/>
          <xsd:element name="Autores"
            type="xsd:string"
            maxOccurs="10"/>
          <xsd:element name="Editorial"
            type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="precio"
          type="xsd:double"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
```

1.9 Definiciones

Etiqueta: Es una secuencia de texto encerrada entre < y >

Elemento: Es todo lo que va entre una cierta etiqueta de apertura y cierre. En el ejemplo siguiente si nos hablan de la etiqueta libro se refieren simplemente a la etiqueta entre los paréntesis angulares. Si hay que procesar el elemento libro esto significa procesar los sub-elementos o “elementos hijo”.

Atributo: Es un texto junto a la etiqueta que amplía información sobre la misma. En el ejemplo anterior podemos ver un atributo precio en la etiqueta titulo

Árbol del documento: Todo documento XML y HTML5 puede representarse como un árbol que se puede recorrer desde distintos lenguajes. Este árbol a veces se llama el árbol DOM o simplemente el DOM (Document Object Model).

Relaciones de parentesco: En un árbol DOM, los distintos elementos (o nodos). Se dice que un nodo es hijo de otro si aparece más abajo en el árbol DOM. Se dice que dos nodos son hermanos si están en el mismo nivel del árbol DOM. Se dice que un nodo es padre de otro si está en un nivel más arriba en el árbol DOM.

2.1 Introducción

Es la última revisión del estándar HTML. Se incluyen algunas etiquetas nuevas cuyo significado se comentará a continuación. Aún está completándose.

2.2 Etiquetas estructurales

Crean la estructura para el resto de la página:

- doctype: identifica el estándar.
- todo documento debe ir entre las marcas `<html>` y `</html>`.
- Todo html tiene dos partes: head y body. El primero incluye otros elementos estructurales como `<title>` que indica el título de dicha página. Dentro del body se incluye el contenido real de la página.
- Existe una etiqueta vital para el correcto visionado de los símbolos de nuestra página. Esta etiqueta se denomina `<meta>` y lleva el atributo `charset=" "`
- Dentro del body pueden incluirse otras etiquetas que estructuran el contenido de la página:
 - ** `<section>` permite marcar contenido de una página relacionado con un tema concreto. **
 - `<article>` es una unidad de contenido sobre un tema específico que irá dentro de una sección. **
 - `<header>` se utiliza para indicar cual es la cabecera de un artículo o texto. ** `<hgroup>` permite agrupar un conjunto de encabezados y marcarlos como pertenecientes al mismo contenido.
- `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>` y `<h6>` establecen encabezados: trozos de texto que identifican la importancia del siguiente trozo de texto.
- Cualquier etiqueta puede ir comentada. Los comentarios no se muestran, son solo de interés para el programador en un futuro. Un comentario se abre con `<!--` y se cierra con `-->`
- La etiqueta `<nav>` se utilizará para crear barras de navegación.
- La etiqueta `<footer>` se reserva para los pies de página. Son de utilidad para que los buscadores obtengan información como fechas y nombres de los autores.
- La etiqueta `<aside>` se utiliza para indicar información relacionada con el artículo o texto pero que no tiene porque ser parte del mismo. El ejemplo más común es utilizarlo para publicidad relacionada o texto del tipo “artículos relacionados con este”.

2.3 Etiquetas de formato

Para el formateo elemental de textos se utilizan varias etiquetas:

- `` Formatea el texto en “negrita”.
- `<i>` Lo pone en “itálica” (cursiva).
- `<u>` Subraya el texto.
- Las diversas etiquetas se pueden meter unas dentro de otras para obtener efectos como “cursiva, y negrita” o “subrayado y cursiva”, sin embargo las etiquetas deben cerrarse en el orden inverso al que se abrieron.
- `<sup>` y `<sub>` fabrican respectivamente superíndices y subíndices.
- `` se utiliza para enfatizar un texto.
- `<p>` Se utiliza para marcar el comienzo y el fin de un párrafo.
- La etiqueta `br` se utiliza para hacer una ruptura en el flujo del texto. Se escribe en forma abreviada `
`

2.4 Gestión de espacios

Los navegadores web manejan el espacio de una forma un poco especial:

- Si se pone uno o varios espacios en blanco o si se pulsa la tecla ENTER muchas veces el navegador mostrará *un solo espacio en blanco*
- Para poner un espacio en blanco horizontal se puede usar la entidad ` `.
- Para hacer un salto de línea se puede usar la etiqueta `
` (esta etiqueta no lleva asociada una etiqueta de cierre, es *autocerrada*)
- Se puede indicar el comienzo y el final de un párrafo con `<p>` y `</p>`.

Una pregunta habitual es “¿Cuándo se debe usar `<p>` y cuándo `
`?”. La respuesta es “depende”. Una posible respuesta es que si se escriben varios párrafos relacionados es bastante habitual separarlos con `
` mientras que si se ponen varios párrafos que hablan de distintas cosas es habitual usar `<p>` con cada uno de ellos, sin embargo no hay una respuesta universal.,

2.5 Entidades

Las entidades HTML permiten escribir determinados símbolos especiales que podrían confundir al navegador, así como otros símbolos que no aparecen directamente en los teclados:

- `<` y `>` representan los símbolos `<` y `>`.
- `©`
- `™`
- `®`
- `€` y `¥`
- `&`

2.6 Texto preformateado

Algunas marcas, como `<pre>` permiten obligar al navegador a que respete los espacios en blanco tal y como aparecen en la página original.

Si se desea indicar que algo debe ser teclado por el usuario se usa la marca `<kbd>`

Si se desea indicar que algo es una variable se puede usar la marca `<var>`.

La etiqueta `<code>` permite indicar que un determinado es código en un lenguaje de programación.

2.7 Listas

Es una secuencia de elementos relacionados en torno a un mismo concepto Para abrir una lista de elementos se utilizan dos posibles marcas:

- `` Para crear una lista ordenada (numerada)
- `` Para crear una lista desordenada (no numerada)

Una vez creadas hay que etiquetar cada elemento de la lista con la etiqueta ``

2.7.1 Ejercicio

Comprueba que el siguiente código HTML crea unas listas dentro de otras. Prueba a crear listas desordenadas dentro de listas desordenadas.

```
<body>
Antes de programar
<ol>
  <li>
    Instalar JDK
    <ol>
      <li>Ir a oracle.com</li>
      <li>Buscar JDK</li>
      <li>Aceptar licencia</li>
      <li>Descargar</li>
      <li>
        Ejecutar setup.exe
        <ol>
          <li>
            Ejecutar como
            admin
          </li>
          <li>Comprobarli>
        </ol>
      </li>
    </ol>
  </li>
  <li>Modificar variables de entorno</li>
  <li>Asignar más memoria</li>
  <li>Reiniciar</li>
</ol>
Prerrequisitos
<ul>
  <li>Comprobar RAM</li>
```

```
<li>Comprobar disco</li>
<li>Comprobar arranque</li>
</ul>
</body>
```

2.8 Tablas

Una tabla muestra un conjunto de elementos relacionados en forma de matriz. No deberían usarse para maquetar la posición de los elementos. Todo el contenido de la tabla debe ir entre las etiquetas `<table>` y `</table>`. Las tablas se construyen de izquierda a derecha (por columnas) y de arriba a abajo (filas).

Una tabla puede tener una cabecera, un cuerpo y un pie, especificados por `<thead>`, `<tbody>` y `<tfoot>`. La primera etiqueta dentro de `<tbody>`, solo puede ser `<tr>`. **Cuidado al crear tablas, todo dato, o subtablas debe ir dentro de `<td>`, es absolutamente obligatorio**

Para ser exactos una tabla puede llevar estas tres etiquetas:

- `thead`: dentro de ella a su vez pondremos una fila (`<tr>`) con celdas en las que la etiqueta es `<th>`
- `tbody`: utiliza las filas y columnas normales.
- `tfooter`: también usa `<tr>` y `<td>` de la forma habitual, sin embargo permite describir mejor el contenido de la tabla. Se utiliza para celdas con los valores acumulados o similares.

2.8.1 Un ejemplo de tabla

```
<table>
  <thead>
    <tr>
      <th>País</th>
      <th>Oro</th>
      <th>Plata</th>
      <th>Bronce</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>USA</td>
      <td>110</td>
      <td>115</td>
      <td>99</td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <td>Total</td>
      <td>219</td>
      <td>247</td>
      <td>206</td>
    </tr>
  </tfoot>
</table>
```

2.8.2 Ejercicio sobre tablas

Crea una tabla con la estructura siguiente:

A	B
C	D1
	D2
	D3
	D4

2.8.3 Solución

Un posible HTML que resuelve esto sería:

2.8.4 Ejercicio sobre tablas (II)

Crea una tabla con la estructura siguiente:

A	
B1	C1 C2
B2	
B3	

2.8.5 Solución

Un posible HTML que resuelve esto sería:

2.8.6 Ejercicio sobre tablas (III)

Crea una tabla con la estructura siguiente:

A1	B1	B1-1
A2		B1-2
C1	C1-1	D1
	C1-2	D2

2.8.7 Solución

Un posible HTML que resuelve esto sería:

2.8.8 Ejercicio sobre tablas (IV)

Crea una tabla con la estructura siguiente:

A1	A2	A3	
B11 B12 B13 B14		B21 B22 B23	

2.8.9 Solución

Un posible HTML que resuelve esto sería:

2.8.10 Ejercicio sobre tablas (V)

Crea una tabla con la estructura siguiente

2.8.11 Solución

```
<table border="1">
  <thead>
    <tr>
      <th>País</th>
      <th>Datos econ.</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>España</td>
      <!--Atención
      ;Tabla dentro de celda-->
      <td>
        <table border="1">
          <tr>
            <td>
              PIB:-0,1%
            </td>
          </tr>
        </table>
      </td>
    </tr>
  </tbody>
</table>
```

País	Datos econ.
España	PIB:-0,1%
	Déficit:5%
	Paro 25,4%
USA	PIB:0,4%
	Déficit:3%
	Paro:11%

```

        <td>
            Déficit:5%
        </td>
    </tr>
    <tr>
        <td>
            Paro 25,4%
        </td>
    </tr>
</table>
</td>
</tr>
<tr>
    <td>USA</td>
    <!--Otra tabla dentro de celda-->
    <td>
        <table border="1">
            <tr>
                <td>
                    PIB:0,4%
                </td>
            </tr>
            <tr>
                <td>
                    Déficit:3%
                </td>
            </tr>
        </table>
    </td>

```

```

        <td>
            Paro:11%
        </td>
    </tr>
</table>
</td>
</tr>
</tbody>
</table>

```

2.8.12 Ejercicio sobre tablas (VI)

Crea una tabla con la estructura siguiente

España	I+D:7%	IRS:13%	
ALCA	I+D:3%	Felic:38	Resto:42%
UE	España	I+D:8%	SS:25%
	Resto UE	I+D:13%	Otros:28%

2.8.13 Solución

```

<table border="1">
  <tbody>
    <tr>
      <td>España</td>
      <td>
        <!--Subtabla-->
        <table border="1">
          <tr>
            <td>
              I+D:7%
            </td>
            <td>
              IRS:13%
            </td>
          </tr>
        </table>
      </td>
    </tr>
    <tr>
      <td>ALCA</td>
      <td>I+D:3%</td>
      <td>Felic:38</td>
      <td>Resto:42%</td>
    </tr>
    <tr>
      <td data-kind="parent" data-rs="2">UE</td>
      <td>España</td>
      <td>I+D:8%</td>
      <td>SS:25%</td>
    </tr>
    <tr>
      <td data-kind="ghost"></td>
      <td>Resto UE</td>
      <td>I+D:13%</td>
      <td>Otros:28%</td>
    </tr>
  </tbody>
</table>

```



```

</tr>
<tr>
  <td>ALCA</td>
  <td>
    <!--Subtabla-->
    <table border="1">
      <tr>
        <td>
          I+D:3%
        </td>
        <td>
          Felic:38
        </td>
        <td>
          Resto:42%
        </td>
      </tr>
    </table>
  </td>
</tr>
<tr>
  <td>UE</td>
  <td>
    <!--Subtabla-->
    <table border="1">
      <tr>
        <td>
          España
        </td>
        <td>
          I+D:8%
        </td>
        <td>
          SS:25%
        </td>
      </tr>
      <tr>
        <td>
          Resto UE
        </td>
        <td>
          I+D:13%
        </td>
        <td>
          Otros:28%
        </td>
      </tr>
    </table>
  </td>
</tr>
</tbody>
</table>

```

2.9 Formularios

Un formulario permite que el usuario interactúe con la página por medio de una serie de controles

2.9.1 Campo de texto

Permite crear una zona donde el usuario puede escribir y se muestra un ejemplo a continuación. Tiene algunos atributos que se usan muy a menudo:

- `type` indica el tipo de control
- `name` será el nombre de la variable en JS (no lo usaremos por ahora)
- `id` permitirá procesar el control de JS (no se usará por ahora)
- `value` permite indicar un valor por defecto
- `size` indica la anchura por defecto

```
<input type="text" name="nombre_usuario"
      id="id_nombre" value="Escriba su nombre aqui">
```

Un campo de texto puede llevar asociada una etiqueta `label` que indique al navegador que texto va con ese campo. Esto es de utilidad para programas lectores de páginas y en general para gente con discapacidad.

```
<label for="d_nombre">Nombre de usuario</label>
<input type="text" name="nombre_usuario"
      id="id_nombre" value="Escriba su nombre aqui">
```

Si el `type` de este elemento se sustituye por `password` se obtiene un control igual, pero que reemplaza el texto por símbolos que ocultan el texto.

2.9.2 Selector único (radio-button)

Permite elegir una sola opción de entre muchas, se necesita usar un `input` de tipo `radio`.

```
<br/>
<input type="radio" name="sexo">Masculino
<br/>
<input type="radio" name="sexo">Femenino
```

2.9.3 Selector múltiple (checkbox)

Permite elegir múltiples combinaciones de opciones. El nombre del control utilizará los corchetes para crear un vector que se procesará desde Javascript.

```
<br/>
<input type="checkbox" name="medios[]">
Coche
<br/>
<input type="checkbox" name="medios[]">
Moto
<br/>
<input type="checkbox" name="medios[]">
Bici
```

2.9.4 Lista desplegable

Permite elegir valores de una lista.

```
<select name="provincia">
  <option value="AB">Albacete</option>
  <option value="CR">Ciudad R.</option>
  <option value="CU">Cuenca</option>
</select>
```

Se debe recordar que el texto que ven los usuarios es lo que va entre las etiquetas option. El valor que comprobarán los programadores es lo que va en value. En una lista desplegable se pueden elegir muchos valores usando el atributo multiple.

2.9.5 Textareas

Permiten introducir textos muy largos:

```
<textarea rows="10" cols="15">
  Valor por defecto
</textarea>
```

2.9.6 Ejercicio

Crear un formulario que pregunte al usuario por su nombre, apellidos, fecha de nacimiento y país. ¿Qué controles habría que usar para cada tarea?

Es importante recalcar que cada vez más, muchas personas visitan páginas desde dispositivos móviles, por eso deberíamos intentar reducir la cantidad de texto que deben teclear. Así, por ejemplo, en lugar de pedir que se teclee la fecha de nacimiento, tal vez puedan seleccionarse opciones de un menú. Para generar la lista de años o de países, probablemente podamos crear un programa que nos genere los option automáticamente.

2.10 Ejercicios tipo examen

2.10.1 Enunciado

Crea una tabla con la estructura siguiente

Celda 1	Celda 2
	Celda 4a
Celda 3a Celda 3b Celda 3c	Celda 4b
	Celda 4c

2.10.2 Solución

El siguiente HTML produce algo muy parecido:

```
<table border="1">
  <tr>
    <td>Celda 1</td>
    <td>Celda 2</td>
  </tr>
```

```

<tr>
  <td>
    <table border="1">
      <tr>
        <td>Celda 3a</td>
        <td>Celda 3b</td>
        <td>Celda 3c</td>
      </tr>
    </table>
  </td>
  <td>
    <table border="1">
      <tr>
        <td>Celda 4a</td>
      </tr>
      <tr>
        <td>Celda 4b</td>
      </tr>
      <tr>
        <td>Celda 4c</td>
      </tr>
    </table>
  </td>
</tr>
</table>

```

2.10.3 Enunciado

Crea una página HTML que produzca este resultado

1a 1b	2a 2b 2c 2c	
3a	4a1 4a2 4a3	
3b1 3b2 3b3 3b4	4b	

Figura 2.1: Una tabla compleja

2.10.4 Solución

El HTML siguiente produce el resultado pedido:

```

<table border="1">
  <tr><!--Primera fila-->
    <td>
      <table border="1">
        <tr>

```

```

                                <td>1a</td>
                                <td>1b</td>
                            </tr>
                        </table>
                    </td>
                    <td>
                        <table border="1">
                            <tr>
                                <td>2a</td>
                                <td>2b</td>
                            </tr>
                            <tr>
                                <td>2c</td>
                                <td>2c</td>
                            </tr>
                        </table>
                    </td>
                </tr>
                <tr><!--Segunda fila-->
                <td>
                    3a
                </td>
                <td>
                    <table border="1">
                        <tr>
                            <td>4a1</td>
                            <td>4a2</td>
                            <td>4a3</td>
                        </tr>
                    </table>
                </td>
            </tr>
            <tr> <!--Tercera fila-->
            <td>
                <table border="1">
                    <tr>
                        <td>3b1</td>
                        <td>3b2</td>
                    </tr>
                    <tr>
                        <td>3b3</td>
                        <td>3b4</td>
                    </tr>
                </table>
            </td>
            <td>
                4b
            </td>
        </tr>
    </table>

```

2.10.5 Enunciado

Crea una página HTML que produzca este resultado

El proceso de **maquetado** de una página WebTM consta de:

1. Reunión con el cliente
2. Boceto HTML
3. Posicionamiento de elementos
4. Alojamiento y pago

El pseudocódigo podría ser

```
while not fin:
    revisar
    comprobar con el cliente
end while
```

2.10.6 Solución

2.10.7 Enunciado

Crea un formulario como este donde haya 3 opciones en la lista desplegable: “Más de 400”, “Menos de 400”, “Variables”

Datos fiscales

☐ En paro
☐ Autónomo
☐ Por c. ajena

Datos personales

Nombre:
Apellidos:
Sueldo:
☐ Con enfermedad profesional
☐ Con padres a cargo
☐ Con hijos a cargo

2.10.8 Solución

El HTML siguiente produce el resultado pedido

2.10.9 Enunciado

Crea un formulario como este

Datos fiscales

Automocion
Metal
Informatica
Finanzas

☐ Autonomo ☐ Cuenta ajena ☐ No sabe

Describe su función:

Escriba aqui

2.10.10 Enunciado

Crea un formulario como este

Laboral

☐ Por cuenta ajena ☐ Autónomo

¿Alguna vez en el extranjero?

☐ Sí Dentro de la UE

☐ No

Personal

Apellidos y nombre:

Conocimientos sobre:

Informática
Conducción
Finanzas
Leyes

2.10.11 Solución

```
<form>
  <fieldset>
    <legend>
      Laboral
    </legend>
    <input type="checkbox"
      name="contratos[]"
      id="ajena">
    Por cuenta ajena
    <input type="checkbox"
      name="contratos[]"
      id="autonomo">
    Autónomo
    <br/>
    ¿Alguna vez en el extranjero?
    <br/>
    <input type="radio"
      name="en_extranjero"
      id="si_en_extranjero">
    Sí
    <select name="lugar">
```

```
        <option id="en_ue">
            Dentro de la UE
        </option>
        <option id="en_asia">
            En Asia
        </option>
        <option id="en_hispanoamerica">
            En Hispanoamérica
        </option>
        <option id="en_eeuu">
            En EE.UU
        </option>
        <option id="en_otro">
            En otro
        </option>
    </select>
    <br/>
    <input type="radio"
        name="en_extranjero"
        id="no_en_extranjero">
        No
</fieldset>
<fieldset>
    <legend>
        Personal
    </legend>
    Apellidos y nombre:
    <input type="text"
        id="ap_nombre">
    <br/>
    Conocimientos sobre:<br/>
    <select name="cono" multiple>
        <option id="informatica">
            Informática
        </option>
        <option id="conduccion">
            Conducción
        </option>
        <option id="finanzas">
            Finanzas
        </option>
        <option id="leyes">
            Leyes
        </option>
    </select>
</fieldset>
</form>
```


3.1 Introducción

El lenguaje CSS permite cambiar el aspecto de páginas web utilizando enlaces a archivos de hojas de estilo. Si todos los HTML de un portal web cargan el mismo archivo CSS se puede cambiar todo un conjunto de HTML's modificando un solo CSS.

3.2 Posicionamiento

Para posicionar los elementos se suelen utilizar dos etiquetas que no hacen nada especial, salvo actuar de contenedores. Las etiquetas `` y `<div>`.

- `` se usa para no romper el flujo, es decir en principio todo va en la misma línea
- `<div>` sí rompe el flujo, por lo que va a una línea distinta

En cualquier etiqueta puede ocurrir que deseemos que el estilo no se aplique a todos los elementos o que queramos que se aplique a unos cuantos (pero no a todos). En ese caso, se deben utilizar los atributos `class` e `id`

- El `class` es un atributo que puede llevar el mismo valor en muchos elementos HTML y que nos permitirá después seleccionarlos a todos.
- El `id` es un atributo que debe tener distinto valor en todos los casos, no se puede repetir.

Para posicionar correctamente un `span` o un `div`, se deben tener en cuenta varias cosas:

- Todos deberían llevar un `id` o un `class` (o las dos cosas)
- **El posicionamiento tiene varias posibilidades:**
 - `fixed`: la caja va en cierta posición y no se mueve de allí
 - `absolute`: la caja va en cierta posición inicial y puede desaparecer al hacer scroll.
 - `relative`: podemos indicar una posición para indicar el desplazamiento relativo con respecto a la posición que le correspondería según el navegador
 - `static`: dar permiso al navegador para que coloque la caja donde corresponda
 - `float`: mover la caja a cierta posición permitiendo que otras cajas floten a su alrededor

3.2.1 Ejercicio de maquetación

Crear una página con la siguiente estructura:

- En el margen izquierdo debe aparecer una barra de enlaces que ocupe el 20 o 25 % de la anchura de la página y no debe desaparecer aunque el usuario se mueva.
- En el margen derecho debe aparecer una caja con el texto y resto de información de interés que debe ocupar el 80 o 75 % de la página y el texto se mueve cuando el usuario se mueve.
- Aplicar bordes y efectos visuales a ambas cajas para intentar que el efecto final sea estéticamente aceptable.

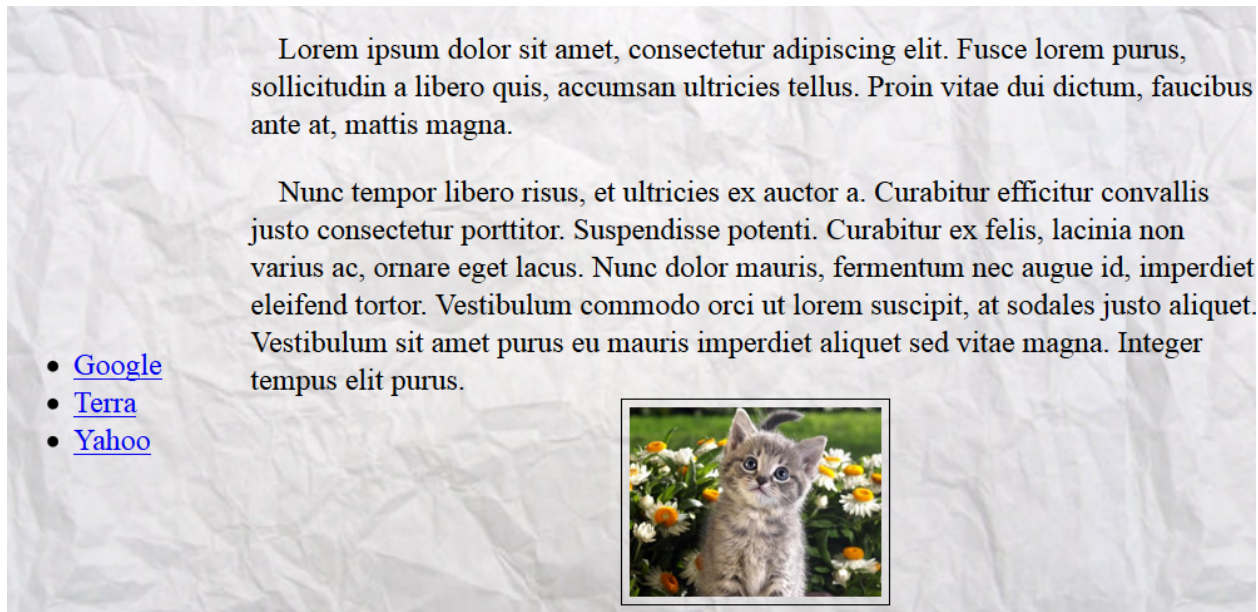


Figura 3.1: Resultado final

HTML

```
<div id="enlaces">
  <ul>
    <li>
      <a href="http://www.google.es">
        Google
      </a>
    </li>
    <li>
      <a href="http://www.terra.es">
        Terra
      </a>
    </li>
    <li>
      <a href="http://www.yahoo.es">
        Yahoo
      </a>
    </li>
  </ul>
</div>
```

```

<div id="contenido">

<p>
    Nunc tempor libero risus, et ultricies ex auctor a. Curabitur efficitur,
    ↪convallis justo consectetur porttitor. Suspendisse potenti. Curabitur ex felis,
    ↪lacinia non varius ac, ornare eget lacus. Nunc dolor mauris, fermentum nec augue,
    ↪id, imperdiet eleifend tortor. Vestibulum commodo orci ut lorem suscipit, at,
    ↪sodales justo aliquet. Vestibulum sit amet purus eu mauris imperdiet aliquet sed,
    ↪vitae magna. Integer tempus elit purus ...

    
</p>
</div>

```

CSS

```

body{
    background-image:
        url("textura.jpg");
    background-attachment: fixed;
}

div#enlaces{
    position: fixed;
    top:40%;
    left:0px;
    width:17%;
}

div#contenido{
    width: 80%;
    position: absolute;
    top:0px;
    right: 0px;
}

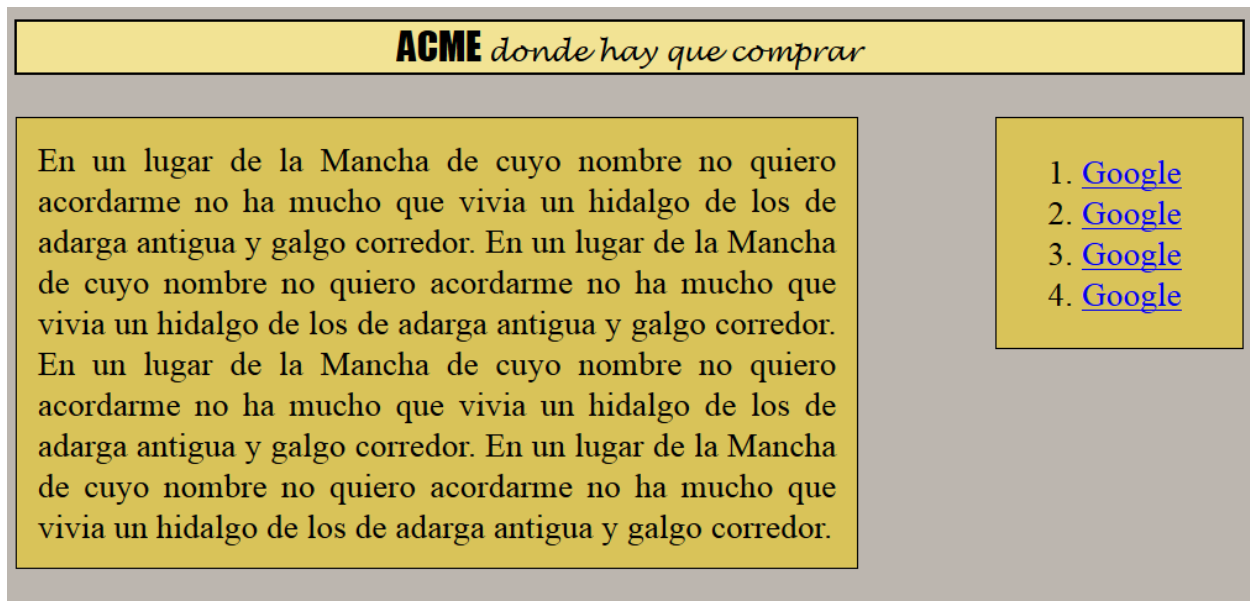
/* Todos los párrafos llevan
 * un pequeño sangrado extra
 * de 15 px en la primera línea*/
p{
    text-indent: 15px;
}

img{
    width: 25%;
    border: solid black 1px;
    padding: 4px;
    display: block;
    margin-left: auto;
    margin-right: auto;
}

```

3.2.2 Ejercicio 2 de maquetación

Conseguir una página como esta



3.2.3 HTML

```
<div id="contenedorglobal">
  <div id="cabecera">
    <span id="marcacabecera">
      ACME
    </span>
    <span id="lemacabecera">
      donde hay que comprar
    </span>
  </div> <!--Fin de la cabecera-->
  <div id="cuerpo">
    En un lugar de la Mancha ...
  </div>
  <div id="enlaces">
    <ol>
      <li>
        <a href="google.es">
          Google
        </a>
      </li>
      <li>
        <a href="google.es">
          Google
        </a>
      </li>
      <li>
        <a href="google.es">
          Google
        </a>
      </li>
      <li>
        <a href="google.es">
          Google
        </a>
      </li>
    </ol>
  </div>
</div>
```

```

        </ol>
    </div>
</div>

```

3.2.4 CSS

```

#cabecera{
    text-align: center;
    background-color:
        rgb(242,227,148)
}
#marcacabecera{
    font-size: larger;
    font-family: "Impact";
}
#lemacabecera{
    font-style: italic;
    font-size: smaller;
    font-family: "Lucida Handwriting";
}

div#cuerpo, div#enlaces{
    background-color:
        rgb(217,195,89);
}

div#cuerpo{
    width:65%;
    float:left;
    margin-top: 20px;
    padding:10px;
    text-align: justify;
}
div#enlaces{
    width: 20%;
    float:right;
    margin-top:20px;
}

div{
    border-width: 1px;
    border-style: solid;
    border-color: black;
    background-color:
        rgb(230,230, 230);
}

div#contenedorglobal{
    background-color:
        rgb(188,182,175);
}

```

3.2.5 Ejercicio: barra de herramientas

Crear una página con dos cajas diferenciadas. Una de ellas, que ocupará el 30 % de la página contendrá enlaces a diferentes sitios web. La caja no se moverá aunque el usuario haga scroll. Por otro lado, la otra caja ocupará el 70 % de la página y habrá que llenarla de texto para poder desplazarse por él y comprobar que la caja de enlaces no se mueve.

El HTML sería algo así:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="solucion1.css" type="text/css"/>
  <title>Ejercicio 1</title>
</head>
<body>
<div id="enlaces">
  <ul>
    <li>
      <a href="http://cocacola.com">CocaCola</a>
    </li>
    <li>
      <a href="http://google.com">Google</a>
    </li>
    <li>
      <a href="http://terra.es">Terra</a>
    </li>
  </ul>
</div>
<div id="contenido">
  En un lugar de la Mancha..
  En ...
</div>
</body>
</html>
```

Y el CSS que resuelve el problema sería:

3.2.6 Ejercicio: ampliación

Ampliar el ejemplo anterior para hacer que el contenido solo ocupe el 50 % y añadir una barra de publicidad fija en el centro vertical que ocupe el 20 %.

El HTML sería

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="solucion2.css" type="text/css"/>
  <title>Ejercicio 1</title>
</head>
<body>
<div id="enlaces">
  <ul>
    <li>
      <a href="http://cocacola.com">CocaCola</a>
    </li>
    <li>

```

```

        <a href="http://google.com">Google</a>
      </li>
      <li>
        <a href="http://terra.es">Terra</a>
      </li>
    </ul>
  </div>
  <div id="publi">
    <ul>
      <li>
        <a href="http://iesmaestredecatalrava.es">IES</a>
      </li>
    </ul>
  </div>
  <div id="contenido">
    En un lugar de la Mancha.. (repetido)
  </div>

</body>
</html>

```

Y el CSS que resuelve el problema sería:

3.2.7 Ejercicio

Crear una estructura de página con una cabecera que ocupe el 100 % de la página, con texto centrado y algunos enlaces. A continuación un bloque de contenido que ocupe el 70 % y a su izquierda un bloque de publicidad que ocupe el 30 %. Debe haber un pie de página con el copyright que ocupe el 100 % de la página, con el texto centrado y que no se mueva cuando el usuario desplace el texto.

El HTML sería

```

<!DOCTYPE html>

<html>
<head>
  <link href="solucion4.css" rel="stylesheet" type="text/css">
  <title>Ejercicio</title>
</head>

<body>

  <header id="cabecera">
    <a href="http://google.es">Google</a>
    <a href="http://terra.es">Terra</a>
  </header>
  <section id="contenido">
    Texto texto texto ...
  </section>
  <aside id="publi">
    <a href="http://cocacola.es">Beba Coca-Cola</a>
  </aside>
  <footer>
    &copy; Pepe Perez, IES Maestre 2013-2014
  </footer>

```

```
</footer>
</body>
</html>
```

Y el CSS que resuelve el problema sería:

3.2.8 Posicionamiento float

En el posicionamiento `float` solo indicaremos la anchura de una caja. El resto de los elementos se encajará automáticamente en el espacio restante dejado por dicha caja.

3.2.9 Ejercicio

Crear una página con una cabecera que ocupe el 100 %, que tenga el texto centrado y una zona debajo que tenga 3 partes: contenido (60 %), enlaces_relacionados (20 %) y publicidad(20 % restante). Crear un pie de página con una anchura del 100 %.

3.3 Gestión de espacios

En CSS se puede controlar el espacio interno y externo por medio de las propiedades `padding-` y `margin-` pudiendo usar `margin-top` o `padding-left`. Las cuatro posiciones son `top`, `bottom`, `left` y `right`

3.4 Colores

Los colores en CSS se pueden especificar de varias maneras:

- Por nombre: `red`, `yellow`, `green`
- Mediante `rgb(rojo, verde, azul)`, donde entre comas se pone la cantidad de cada color de 0 a 255. Así, `rgb(0, 0, 0)` es negro y `rgb(255, 255, 255)` es blanco.
- Se puede usar directamente la nomenclatura hexadecimal `#ffffff`. Donde cada dos letras se indica un número hexadecimal de 00 a ff, que indica respectivamente la cantidad de color rojo, verde o azul.

Se pueden encontrar en Internet listas de colores denominados “seguros” (buscando por “web safe colors”) que indican nombres de color que se ven igual en los distintos navegadores.

3.5 Tipografías

En tipografía se habla de dos términos distintos: el “typeface” y la “font”.

- Hay tipos “Serif”, que llevan “rabito”.
- Hay tipos “Sans-serif” que no lo llevan
- Hay tipos monoespaciados

Lo más relevante, es que cuando usamos `font-family: "Arial";`, el navegador puede decidir poner otro tipo de letra de la misma familia.

Se pueden indicar varios tipos de letra por orden de preferencia.

Google Fonts permite el “embebido” de fuentes de manera muy segura.

3.6 Alineación del texto

Se puede usar la propiedad `text-align: left` para modificar la alineación del texto, usando `left`, `center`, `right` o `justify`

3.7 Decoración del texto

Se pueden usar otras propiedades para cambiar el aspecto del texto como estas:

- `text-decoration: underline`
- `text-decoration: overline`
- `text-decoration: line-through`

3.8 Medidas

Normalmente, lo más seguro es usar medidas en forma de porcentajes, pero hay otras

- `margin: 1cm`
- `margin: 1in`: esta y la anterior son más útiles cuando creamos hojas de estilo enfocadas a que la página quede bien cuando se imprima.
- `margin: 1px`: muy dependiente de la resolución
- `margin: 1%`: es la más apropiada al modificar elementos `div` en pantalla.
- `margin: 1em`: equivale aproximadamente a la anchura de una letra “m”.

3.9 Selectores

<http://10.9.0.252:8000>

Explica qué hacen los siguientes selectores y crea un ejemplo HTML donde se pueda ver que realmente funcionan como esperas

- `p#destacado`
- `p.destacado`
- `p.destacado, span#id1`
- `p.destacado > li.elemento_enumeracion`
- `p.destacado > .elemento_numeracion`
- `.destacado > #id1`

Supongamos que tenemos un archivo HTML como este:

```
<p>
    Párrafo sin class ni ide
</p>
<p class="cita" id="destacado">
    Párrafo con el class 'cita'
    y el id 'p_destacado'
</p>
<p class="p_destacado">Párrafo con el class
    destacado que no
    contiene nada
</p>
<p class="p_destacado">
    Párrafo con el class destacado.
    <span id="id1">
        Este texto va dentro de
        un span con el
        id id1
    </span>
</p>
<p class="destacado">
    <ol>
        <li class="elemento_numeracion">
            Esto es un li
        </li>
        <li class="elemento_numeracion">
            Esto es otro li
        </li>
    </ol>
</p>
<p class="destacado">
    Este párrafo tiene el class
    destacado y en él enumeramos
    cosas como
    <span class="elemento_numeracion">
        A
    </span>,
    <span class="elemento_numeracion">
        B
    </span> o también
    <span class="elemento_numeracion">
        C
    </span>
</p>
<div class="destacado">
    Aquí hay un
    <span id="id1">
        span con el id id1
    </span>
</div>
```

3.9.1 Solución p#destacado

Si tenemos un estilo como este:

```
p#destacado{
    border:solid black 1px;
}
```

Lo que ocurrirá es que se pondrá un borde solo al párrafo cuyo `id` sea `destacado`

Párrafo sin `class` ni `id`

Párrafo con el `class` 'cita' y el `id` 'p_destacado'

Párrafo con el `class` `destacado` que no contiene nada

Párrafo con el `class` `destacado`. Este texto va dentro de un `span` con el `id` `id1`

Aquí hay una numeración

1. Esto es un `li`
2. Esto es otro `li`

Este párrafo tiene el `class` `destacado` y en él enumeramos cosas como A , B o también C ,

Aquí hay un `span` con el `id` `id1`

Figura 3.2: Resultado

3.9.2 Solución `p.destacado`

Los cambios se aplican a todos los párrafos con el `class` `destacado`

3.9.3 Solución `p.destacado, span#id1`

Los cambios se aplican a todos los párrafos con el `class` `destacado` y también al `span` cuyo `id` sea `id1`

Advertencia: Obsérvese que en el HTML hay **dos elementos con el mismo ID**. No se debe hacer esto, ya que corremos el riesgo de que todo se vea mal.

3.9.4 Solución `p.destacado > li.elemento_enumeracion`

Los cambios solo se aplican a los `li` cuyo `class` sea `elemento_enumeracion` y que además sean hijos de un `p` cuyo `class` sea `destacado`

¿Por qué los cambios no afectan a ninguno?

Párrafo sin class ni ide

Párrafo con el class 'cita' y el id 'p_destacado'

Párrafo con el class destacado que no contiene nada

Párrafo con el class destacado. Este texto va dentro de un span con el id id1

Aquí hay una numeración

1. Esto es un li
2. Esto es otro li

Este párrafo tiene el class destacado y en él enumeramos cosas como A , B o también C ,

Aquí hay un span con el id id1

Figura 3.3: Resultado

Párrafo sin class ni ide

Párrafo con el class 'cita' y el id 'p_destacado'

Párrafo con el class destacado que no contiene nada

Párrafo con el class destacado. Este texto va dentro de un span con el id id1

Aquí hay una numeración

1. Esto es un li
2. Esto es otro li

Este párrafo tiene el class destacado y en él enumeramos cosas como A , B o también C ,

Aquí hay un span con el id id1

Figura 3.4: Resultado

3.9.5 Solución `p.destacado > .elemento_numeracion`

Ocurre lo mismo de antes, ¿por qué?

3.9.6 Solución `.destacado > #id1`

Ahora el resultado es este

Párrafo sin class ni ide

Párrafo con el class 'cita' y el id 'p_destacado'

Párrafo con el class destacado que no contiene nada

Párrafo con el class destacado. Este texto va dentro de un span con el id id1

Aquí hay una numeración

1. Esto es un li
2. Esto es otro li

Este párrafo tiene el class destacado y en él enumeramos cosas como A , B o también C ,

Aquí hay un span con el id id1

Figura 3.5: Resultado

¿Por qué ahora sí funciona?

3.10 Bootstrap

Bootstrap define una estructura básica de clases CSS para facilitar el desarrollo web. En concreto CSS consigue que crear páginas que se vean igual en dispositivos muy distintos sea algo relativamente sencillo.

3.10.1 Estructura básica

El siguiente HTML define lo mínimo que se necesita para crear una página con Bootstrap. Dentro de `<body>` podremos poner lo que necesitemos y el *framework* colocará todo automáticamente y le aplicará cierto estilismo.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Plantilla bootstrap</title>
```

```

<link href="css/bootstrap.min.css" rel="stylesheet">

<!--[if lt IE 9]>
  <script src="js/html5shiv.min.js"></script>
  <script src="js/respond.min.js"></script>
<![endif]-->
</head>
<body>
  <h1>Página con Bootstrap</h1>
  <div class="container">
    </div>
  <script src="js/jquery.min.js"></script>

  <script src="js/bootstrap.min.js"></script>
</body>
</html>

```

Lo único que se debe asumir es que debe existir un `<div>` cuyo `class` sea `container`

3.10.2 Rejilla o *grid*

Bootstrap asume que cualquier pantalla tiene una anchura básica de 12 columnas. Podremos crear una fila de elementos y hacer que cada una de ellas ocupe cierta proporción de esas columnas.

Por ejemplo, si deseamos que una fila de contenidos tenga una columna que ocupe la mitad de esas 12 columnas (6) y dos columnas que ocupen la mitad restante, podremos hacer lo siguiente.

```

<div class="container">
  <div class="row">
    <div class="col-md-6">
      Mitad del contenedor
    </div>
    <div class="col-md-3">
      Esto ocupa un cuarto
    </div>
    <div class="col-md-3">
      Esto ocupa otro cuarto
    </div>
  </div>
</div>

```

Página con Bootstrap

Mitad del contenedor

Esto ocupa un cuarto

Esto ocupa otro cuarto

Figura 3.6: Ejemplo del grid

En realidad Bootstrap define muchos tipos de columna dependiendo del tipo de dispositivo al que nos hayamos enfocado más:

- `col-xs-3`: ocupa 3 de las doce columnas de un dispositivo que se ha dividido en 12 pero tiene una anchura “muy pequeña/extrasmall” (menos de 768)

- col-sm-3: ocupa 3 de las doce columnas de un dispositivo que se ha dividido en 12 pero tiene una anchura “pequeña/*sm*all” (más de 768 y menos de 992)
- col-md-6: ocupa 6 de las doce columnas de un dispositivo que se ha dividido en 12 pero tiene una anchura “media” (unos 992 px)
- col-lg-9: ocupa 9 de las doce columnas de un dispositivo que se ha dividido en 12 pero tiene una anchura “grande/large” (unos 992 px)

3.10.3 Tipografía

Bootstrap modifica la tipografía por defecto e incluso permite destacar algunos elementos. Por ejemplo un párrafo con la clase `lead` destacará:

```
<p class="lead">
    Este párrafo es muy importante
</p>
<p>Este párrafo es normal</p>
```

Página con Bootstrap

Este párrafo es muy importante

Este párrafo es normal

Figura 3.7: Párrafo destacado

Se puede destacar texto usando lo siguiente:

```
<mark>Texto subrayado en amarillo</mark>
```


4.1 Introducción

Surgió como una iniciativa de Netscape. **No hay ninguna relación entre Java y Javascript.** Algunas características de Javascript son:

- No se convierte en bytecodes, lo interpreta el navegador
- Está estandarizado aunque no todos los navegadores se ciñen al 100 % al estándar. El nombre del estándar es “ECMAScript”
- No está orientado a objetos, sino que está basado en objetos.
- Es de tipado débil: esto significa que podemos cambiar una variable de tipo sin problemas, el intérprete intentará hacer las conversiones correctas.

4.2 Tipos de datos

Javascript acepta los siguientes tipos de datos:

- Números.
- Cadenas
- Booleanos (lógicos)
- undefined: se utiliza cuando intentamos acceder a una variable que no contiene nada porque no se ha creado.
- null: se utiliza habitualmente para indicar algo vacío.

4.3 Incrustando Javascript

Javascript se inserta en HTML con la etiqueta `<script>`. Esta etiqueta puede ir en cualquier sitio del HTML, dentro de `<head>` o dentro de `<body>`.

Un programa muy simple sería este:

```
var una_variable
una_variable=42
document.write(una_variable)
```

4.4 Decisiones

Las decisiones se toman con la sentencia `if` que funciona exactamente igual que en Java. Se pueden utilizar los mismos operadores `&&` y `||` al igual que en Java.

```
if (una_variable > otra_variable) {
    document.write ("La primera es mayor que la segunda")
} else {
    document.write ("La segunda es mayor que la primera")
}
```

4.5 Vectores o Arrays

En Javascript los arrays pueden almacenar elementos de distinto tipo. Al crearlos podemos indicar el tamaño o no, pero no habrá problemas si queremos almacenar más elementos de los previstos.

```
/* Una forma de crear un array*/
vector_nombres=new Array()
vector_nombres[0]="Juan Perez"
vector_nombres[1]="Pedro Diaz"
document.write ("El primer nombre es:"+vector_nombres[0])

/* Otra forma de crearlos*/
vector_numeros=new Array(2)
vector_numeros[0]=23
vector_numeros[1]=-45.23
vector_numeros[2]=45e2
```

4.6 Bucles

4.6.1 Bucles for

Bucles for estilo clásico

En estos bucles hay que poner la inicialización, la condición de final y la actualización:

```
for (var i=0; i<vector_numeros.length; i++){
    document.write("<br/>")
document.write ("En la posición "+i)
document.write (" está el número " + vector_numeros[i])
}
```

Obsérvese que hemos introducido el atributo `length` de la clase `Array` que nos indica la longitud del vector.

Ejercicio

Crear un vector de 6 posiciones y rellenarlo con estos números: 9.98, 7.86, 4.53, 8.91, 5.76, 2.31.

Ordenar el vector y mostrar el contenido del vector ordenado por pantalla.

```

var v=new Array()
v=[9.98, 7.86, 4.53,
   8.91, 5.76, 2.31]

/* Vamos cogiendo cada elemento...*/
for (var i=0; i<v.length; i++){
  /* Y se compara con
   * todos los demas*/
  for (var j=0; j<v.length; j++) {
    if (v[j]>v[i]) {
      aux=v[i]
      v[i]=v[j]
      v[j]=aux
    } /* Fin del if*/
  } /* Fin del for interno*/
} /* fin del for externo*/

/* Se imprime el contenido*/
for (var i=0; i<v.length; i++){
  alert ("Pos "+i+ ":"+v[i])
}

```

Bucles foreach

Funciona igual que el anterior pero es mucho más corto.

```

for (var posicion in vector_numeros) {
  document.write("<br/>")
  document.write ("En la posición "+posicion)
  document.write (" está el número " + vector_numeros[posicion])
}

```

4.6.2 Bucles while

Los bucles while funcionan igual que en Java

```

var posicion=0
while (posicion<vector_numeros.length){
  document.write("<br/>")
  document.write ("En la posición "+posicion)
  document.write (" está el número " + vector_numeros[posicion])
  posicion++
}

```

4.7 Ejercicio: media aritmética

Crear un programa que calcule la media aritmética del vector de números.

```

var suma=0
for (var pos in vector_numeros){
  suma=suma+vector_numeros[pos]
}

```

```
}  
var media=suma / vector_numeros.length  
document.write("<br/>La media es:" + media)
```

4.8 Ejercicio: desviación media

Crear un programa que calcule la desviación media del vector de números.

```
/* Para calcular la desviación media*/  
suma=0  
for (var pos in vector_numeros) {  
    var desviacion= Math.abs ( vector_numeros[pos] - media )  
    suma = suma + desviacion  
}  
/* En este punto la variable suma contiene la suma de las desviaciones*/  
var desv_media = suma / vector_numeros.length  
document.write("<br/>La desv media es:"+desv_media)
```

4.9 Ejercicio: la mediana

Calcular la mediana del vector

```
if (v.length%2==0) {  
    var pos1=v.length/2  
    var pos2=pos1-1  
    var elem1=v[pos1]  
    var elem2=v[pos2]  
    var mediana=(elem1+elem2)/2  
} else {  
    var pos_central=(v.length-1)/2  
    var mediana=v[pos_central]  
}  
doc      ument.write("La mediana es:"+mediana)
```

4.10 Funciones

Para crear una función usaremos la palabra `function`, pondremos el nombre, luego los parámetros, dentro irá el código de la función, y si queremos devolver algo usaremos `return`.

```
/* Función a la que le pasamos un vector de números y que  
 * nos devuelve la media de sus valores*/  
  
function calcularMedia(vector_valores){  
    var suma=0  
    for (var pos in vector_valores){  
        suma = suma + vector_valores[pos]  
    }  
    return suma / vector_valores.length  
}
```

```

var vector=new Array(4)
vector[0]=5
vector[1]=2
vector[2]=7
vector[3]=8

var media=calcularMedia(vector)
document.write("<br/>La media es:"+media)

```

Una cuestión importante es que las funciones son valores asignables. Cuando queramos asignar una función a una variable **no pondremos paréntesis**. Cuando sí queramos ejecutar una función (ya sea con su nombre original o con el de la variable, sí pondremos los paréntesis con los parámetros que queramos pasar**.

```

function saludar(nombre) {
    document.write("Hola "
        +nombre+"<br/>")
}
function despedir(nombre){
    document.write("Adios "
        +nombre+"<br/>")
}
saludar("Antonio")
despedir("Antonio")
/* Las funciones son valores
 * asignables*/
var f=despedir
f("Tomas")

```

4.10.1 Ejercicio

Crear un programa que tenga una función que calcule la desviación media de valores de un vector.

```

/* Función que calcula la desviacion media de
 * un vector de valores numericos*/
function calcularDesviacionMedia(vector_valores){
    var media=calcularMedia(vector_valores)
    var suma=0
    for (var pos in vector_valores){
        suma= suma + Math.abs ( vector_valores[pos] - media )
    }
    return suma / vector_valores.length
}

```

4.10.2 Ejercicio

Crear un programa que tenga una función que calcule la moda.

```

/* Este vector nos dice cuantas veces aparece un número
 * en un vector*/
function calcularFrecuencia(numero, vector){
    var num_veces=0
    for (var pos in vector) {
        if (vector[pos]==numero) {
            num_veces++
        }
    }
    return num_veces
}

```

```
    }  
  }  
  return num_veces  
}  
  
/* Dado un vector de números se nos devuelve la posición  
 * del número mayor*/  
function obtenerPosMayor(vector_valores){  
  var posMayor=0  
  var numMayor=vector_valores[0]  
  for (var pos in vector_valores){  
    if (vector_valores[pos]>numMayor) {  
      numMayor=vector_valores[pos]  
      posMayor=pos  
    }  
  }  
  return posMayor  
}  
  
/* Función que devuelve el número "moda" de un vector*/  
function obtenerModa(vector_valores){  
  var frecuencias=new Array(vector_valores.length)  
  for (var pos in vector_valores){  
    var numero=vector_valores[pos]  
    frecuencias[pos]=calcularFrecuencia(numero, vector_valores)  
  }  
  var posModa=obtenerPosMayor(frecuencias)  
  return vector_valores[posModa]  
}  
  
var vector=new Array(4)  
vector[0]=7  
vector[1]=7  
vector[2]=7  
vector[3]=5  
var moda=obtenerModa(vector)  
document.write("<br/>La moda es:"+moda)
```

4.11 Programación OO

Se ha dicho anteriormente que Javascript es “basado en objetos” y no “orientado a objetos”, es decir la POO es optativa. No por ello es menos potente.

En primer lugar, es posible crear objetos sin crear clases.

```
var empleado={  
  nombre:"Pepe Perez",  
  edad:27,  
  fiijo:true,  
  estaJubilado:function () {  
    if (this.edad>65) {  
      return true  
    } else {  
      return false  
    }  
  }  
}
```

```

}
document.write("<br/>El nombre es:"+empleado.nombre)
document.write("<br/>¿Jubilado?" + empleado.estaJubilado() )

```

4.11.1 Ejercicio

Añadir un método llamado `nivelExperiencia` que nos diga una de estas cosas:

- Nos debe devolver “junior” si la edad está entre 18 y 25
- Nos debe devolver “asociado” si la edad está entre 26 y 45
- Nos debe devolver “senior” si la edad está entre 46 y 60
- Nos debe devolver “experto” si la edad está entre 61 y 65
- Nos debe devolver “no aplicable” si la edad es mayor de 65

```

var empleado={
  nombre:"Pepe Perez",
  edad:27,
  fijo:true,
  estaJubilado:function () {
    if (this.edad>65) {
      return true
    } else {
      return false
    }
  },
  nivelExperiencia:function() {
    if ( (this.edad>18)  && (this.edad<=25) ) {
      return "junior"
    }
    if ( (this.edad>=26)  && (this.edad<=45) ) {
      return "asociado"
    }
  }
}

```

4.11.2 Ejercicio

Crear una clase `GestorVectores` que tenga los principales métodos estadísticos vistos hasta ahora: media, desviación media, mediana y moda.

```

gestor_vectores={
  vector_numeros:new Array(),
  setDatos:function(vector) {
    this.vector_numeros=vector
  }
  , //Importante: separar métodos y atributos con ,
  getMedia:function() {
    var suma=0
    var media=0
    for (pos in this.vector_numeros) {
      suma=suma + this.vector_numeros[pos]
    }
  }
}

```

```
        }
        media=suma / this.vector_numeros.length
        return media
    }
    ,
    getModa:function() {
    }
    ,
    getMediana:function() {
        this.vector_numeros.sort()
    }
}

var vector_prueba=new Array(3)
vector_prueba[0]=5
vector_prueba[1]=10
vector_prueba[2]=8
gestor_vectores.setDatos ( vector_prueba )
var media=gestor_vectores.getMedia()
document.write ("La media es:"+media)
```

4.12 Programación con JQuery

Existen muchos navegadores que a veces muestran pequeñas diferencias entre ellos. Para evitar problemas los programadores tenían que incluir muchos código para comprobar qué navegador ejecutaba su JS y en función de eso actuar. Para resolver estas diferencias John Resig creó JQuery.

4.12.1 Inicio

A partir de ahora todos nuestros archivos HTML tendrán que cargar al comienzo la biblioteca JQuery con una etiqueta como esta:

```
<script src="jquery.js" language="Javascript">
</script>
<script src="nuestroprograma.js" language="Javascript">
</script>
```

El orden es importante

4.12.2 La función \$

La función \$ selecciona elementos de la página para que podamos hacer cosas con ellos. Es la función más utilizada de JQuery y veremos que podemos pedir que nos seleccione grupos de cosas de forma muy sencilla.

La función \$ devuelve siempre objetos. Los atributos y métodos de esos objetos los iremos aprendiendo poco a poco.

En general, antes de poder procesar un elemento, deberemos seleccionarlo utilizando los mismos selectores que en CSS.

4.12.3 Gestión de eventos

Utilizando `click` podemos indicar a la biblioteca que queremos que cuando alguien haga click en un elemento se ejecute una cierta función. El siguiente código HTML y JS ilustra una posibilidad

```
<!DOCTYPE html>
<html>
<head>
  <script src="jquery.js"></script>
  <script src="ejemplo.js"></script>
  <title>Ejemplos</title>
  <style>
    div#texto{
      background-color:yellow;
    }
  </style>
</head>

<body>
<form>
  <input type="button" value="fadeOut" id="botonizq">
  <input type="button" value="fadeIn" id="botonder">
</form>

<div id="texto">
  Texto texto texto
</div>

</body>
</html>
```

El código Javascript asociado al HTML anterior es este.

```
/* Esperaremos hasta que el documento esté cargado y listo
 * para ser procesado por nuestro programa*/

var obj_documento = $(document)

/* Cuando esté cargado ejecutaremos la función cuyo nombre aparezca aquí*/
obj_documento.ready(inicio)

/* Error gravísimo*/
//obj_documento.ready( inicio() )

function inicio(){
  var obj_izq=$("#botonizq")
  obj_izq.click ( fn_click_izq )
  var obj_der=$("#botonder")
  obj_der.click ( fn_click_der )
}

function fn_click_izq(){
  var obj_div=$("#texto")
  obj_div.fadeOut()
}

function fn_click_der(){
  var obj_div=$("#texto")
```

```
obj_div.fadeIn()  
}
```

Solución HTML (párrafos)

```
<div data-role="content">  
  <div class="ui-grid-c">  
    <div class="ui-block-a">  
      <input type="submit"  
        id="mostrar_pares"  
        value="Mostrar pares">  
    </div>  
    <div class="ui-block-b">  
      <input type="submit"  
        id="ocultar_pares"  
        value="Ocultar pares">  
    </div>  
    <div class="ui-block-c">  
      <input type="submit"  
        id="mostrar_impares"  
        value="Mostrar impares">  
    </div>  
    <div class="ui-block-d">  
      <input type="submit"  
        id="ocultar_impares"  
        value="Ocultar impares">  
    </div>  
  </div>  
  <p class="p_impar">  
    Soy un párrafo impar  
  </p>  
  <p class="p_par">  
    Soy un párrafo par  
  </p>  
  <p class="p_impar">  
    Soy un párrafo impar  
  </p>  
  <p class="p_par">  
    Soy un párrafo par  
  </p>  
  <p class="p_impar">  
    Soy un párrafo impar  
  </p>  
  <p class="p_par">  
    Soy un párrafo par  
  </p>  
</div>
```

Solución Javascript (párrafos)

```
$(document).ready(main)  
  
function mostrar_pares() {  
  var objetos=$(".p_par")
```

```
        objetos.slideDown()
    }
    function mostrar_impares() {
        var objetos=$( ".p_impar" )
        objetos.slideDown()
    }
    function ocultar_pares() {
        var objetos=$( ".p_par" )
        objetos.slideUp()
    }
    function ocultar_impares() {
        var objetos=$( ".p_impar" )
        objetos.slideUp()
    }

    function main() {

        $( "#mostrar_pares" ).click( mostrar_pares )
        $( "#mostrar_impares" ).click( mostrar_impares )

        $( "#ocultar_pares" ).click( ocultar_pares )
        $( "#ocultar_impares" ).click( ocultar_impares )

    }
}
```

Existen diversos eventos aunque los más utilizados son:

- click
- dblclick
- mouseover

4.12.4 Ejercicio

Crear un programa que tenga varios párrafos con 4 botones que permitan que cuando se haga click en ellos ocurran distintas cosas

- Habrá un botón con el texto “Ocultar pares”. Cuando se hace click en él se ocultan los párrafos pares.
- Habrá un botón con el texto “Ocultar impares”. Cuando se hace click en él se ocultan los párrafos impares.
- Habrá un botón con el texto “Mostrar pares”. Cuando se hace click en él se muestran los párrafos pares (que tal vez estaban ocultos).
- Habrá un botón con el texto “Mostrar impares”. Cuando se hace click en él se muestran los párrafos impares (que tal vez estaban ocultos).

4.12.5 Ejercicio

Crear una página en la que hay un div con texto y al pasar el ratón por encima de ella, la caja cambia de color.

Antes de poder resolver este ejercicio, hay que echar un vistazo a varias posibilidades de JQuery.

4.13 Procesado de atributos

En JQuery sabemos que podemos procesar elementos utilizando su `id` con cosas como esta:

```
var objeto=$("#identificador1")
objeto.metodo( ... )
```

Una de las cosas que se puede hacer es leer y escribir diversos atributos de los objetos. Además, se pueden leer propiedades especiales como comprobar si un radio o un checkbox están en el estado `checked`.

Supongamos este formulario:

```
<form>
<input type="radio" name="sexo" value="h" id="opc_h">Hombre
<br/>
<input type="radio" name="sexo" value="m" id="opc_m">Mujer
<br/>
<input type="text" id="informe">
<br/>
<input type="checkbox" name="medios[]" id="bus">Autobús
<br/>
<input type="checkbox" name="medios[]" id="coche">Coche
<br/>
<input type="checkbox" name="medios[]" id="moto">Moto
<br/>
<input type="checkbox" name="medios[]" id="bici">Bici
<br/>
<input type="checkbox" name="medios[]" id="tren">Tren
<br/>
</form>
```

Podemos usar el método `val` para cambiar el valor de un objeto cualquiera:

```
function inicio(){
    var opc_h=$("#opc_h")
    opc_h.click ( click_hombre )

    var opc_m=$("#opc_m")
    opc_m.click ( click_mujer )
}

function click_hombre() {
    var cuadro_texto=$("#informe")
    cuadro_texto.val("Bienvenido Sr.")
}

function click_mujer(){
    var cuadro_texto=$("#informe")
    cuadro_texto.val("Bienvenido Sra/Srta.")
}
```

Por ejemplo, en los checkboxes y en los radios, podemos comprobar si uno de ellos está marcado comprobando la propiedad `checked` con el método `prop`.

Supongamos que deseamos saber cuantos checkboxes se marcan. Si se marcan cero, una o dos, mostraremos el texto “poca variedad”, si se marcan tres mostraremos “cierta variedad” y si se marcan cuatro o cinco, mostraremos “mucha variedad”.

Aquí hay dos posibles soluciones, siendo una de ellas más corta y flexible que la otra.

La primera:

```
var opc_coche=$("#coche")
opc_coche.click ( cuantas_pulsadas )

var opc_moto=$("#moto")
opc_moto.click ( cuantas_pulsadas )

var opc_bici=$("#bici")
opc_bici.click ( cuantas_pulsadas )

var opc_bus=$("#bus")
opc_bus.click ( cuantas_pulsadas )

var opc_tren=$("#tren")
opc_tren.click ( cuantas_pulsadas )

function cuantas_pulsadas() {
    //Aquí contaríamos cuantas tienen la propiedad checked
}
```

El segundo implica que todos los controles tengan el mismo atributo class. Ahora la solución tendría un HTML como este:

```
<input type="checkbox" name="medios[]" id="bus" class="medio">Autobús
<br/>
<input type="checkbox" name="medios[]" id="coche" class="medio">Coche
<br/>
<input type="checkbox" name="medios[]" id="moto" class="medio">Moto
<br/>
<input type="checkbox" name="medios[]" id="bici" class="medio">Bici
<br/>
<input type="checkbox" name="medios[]" id="tren" class="medio">Tren
<br/>
```

Y el JS sería así:

```
var medios_de_locomocion$(".medio")
medios_de_locomocion.click ( cuantas_pulsadas )
function cuantas_pulsadas() {
    var cuantas_marcadas=0
    var vector_ids=["#bus", "#coche", "#moto", "#bici", "#tren"]

    for (pos in vector_ids) {
        var objeto = $( vector_ids[pos] )
        if (objeto.prop("checked")) {
            cuantas_marcadas=cuantas_marcadas+1
        }
    }

    if ((cuantas_marcadas>=0 ) && (cuantas_marcadas<=2)){
        alert ("Poca variedad")
    }
    if (cuantas_marcadas==3) {
        alert ("Variedad media")
    }
    if (cuantas_marcadas>=4) {
```

```

        alert ("Mucha variedad")
    }
}

```

4.13.1 Ejercicio: recuento de medios de locomoción

Crear un programa que permita al usuario indicar cinco posibles medios de locomoción (usar checkboxes), a saber: coche, moto, bus, tren y avión. El programa debe contabilizar cuantos se usan en informar del número de medios usados en un textbox.

Solución: recuento de medios (HTML)

```

<div class="ui-grid-d">
<div class="ui-block-a">
  <input type="checkbox"
    name="medio"
    id="coche">
  <label for="coche">Coche</label>
</div>
<div class="ui-block-b">
  <input type="checkbox"
    name="medio"
    id="moto">
  <label for="moto">Moto</label>
</div>
<div class="ui-block-c">
  <input type="checkbox"
    name="medio"
    id="bus">
  <label for="bus">Bus</label>
</div>
<div class="ui-block-d">
  <input type="checkbox"
    name="medio"
    id="tren">
  <label for="tren">Tren</label>
</div>
<div class="ui-block-e">
  <input type="checkbox"
    name="medio"
    id="avion">
  <label for="avion">Avión</label>
</div>
</div>
<input type="text" id="informe">

```

Solución: recuento de medios (JS)

Variante 1: Sin vectores, implica usar muchos `if`. Aunque funcione supone cortar y pegar, que aunque en este caso no sea un trabajo muy grande nos obliga a adoptar malos hábitos

```
$(document).ready(main)
```

```

function contar() {
    var contador=0

    if ($("#coche").prop("checked"))
    {
        contador=contador+1
    }
    if ($("#moto").prop("checked"))
    {
        contador=contador+1
    }
    var mensaje="Medios:"+contador
    $("#informe").val(mensaje)
}

function main() {
    $("#coche").click(contar)
    $("#moto").click(contar)
    $("#bus").click(contar)
    $("#tren").click(contar)
    $("#avion").click(contar)
}

```

Variante 2: Con vectores

```

$(document).ready(main)
function contar() {
    var contador=0
    var ids=new Array()
    var ids=["#coche", "#moto",
            "#bus", "#tren",
            "#avion"]

    for (pos in ids) {
        var medio=$("#"+ids[pos])
        if (medio.prop("checked"))
        {
            contador=contador+1
        }
    }
    var mensaje="Medios:"+contador
    $("#informe").val(mensaje)
}

function main() {
    $("#coche").click(contar)
    $("#moto").click(contar)
    $("#bus").click(contar)
    $("#tren").click(contar)
    $("#avion").click(contar)
}

```

4.13.2 Ejercicio configurador

Se desea tener una aplicación que permita configurar un equipo al gusto del usuario:

- Se debe elegir entre un procesador Intel o AMD. El primero cuesta 250 euros y el segundo 230.
- Se debe elegir entre 2, 4 y 8 GB de memoria. El coste es respectivamente 90, 145, 210
- Hay extras que se pueden elegir o no, ya que son completamente optativos (es decir, usar checkboxes). En concreto se puede tener un grabador de Blu-ray (190 euros), tarjeta gráfica aceleradora (430 euros) y un monitor LED (185 euros).

4.13.3 Solución HTML configurador

```
<label for="intel">Intel i5</label>
<input type="radio"
  name="procesador" id="intel">
<label for="amd">AMD</label>
<input type="radio"
  name="procesador" id="amd">
<label for="2gb">2GB</label>
<input type="radio"
  name="memoria" id="2gb">
<label for="4gb">4 GB</label>
<input type="radio"
  name="memoria" id="4gb">
<label for="8gb">8 GB</label>
<input type="radio"
  name="memoria" id="8gb">
<label for="bluray">Blu-ray</label>
<input type="checkbox" name="extra[]"
  id="bluray">
<label for="aceleradora">Aceleradora</label>
<input type="checkbox" name="extra[]"
  id="aceleradora">
<label for="monitor">Monitor 25</label>
<input type="checkbox" name="extra[]"
  id="monitor">
<input type="text" id="total">
```

4.13.4 Solución JS configurador

```
$(document).ready(main)

function calcular_precio() {
  var precio=0
  if ($("#intel").prop("checked")) {
    precio=precio+250
  }
  if ($("#amd").prop("checked")) {
    precio=precio+210
  }
  if ($("#2gb").prop("checked")) {
    precio=precio+90
  }
  if ($("#4gb").prop("checked")) {
```



```

        precio=precio+140
    }
    if ($("#8gb").prop("checked")) {
        precio=precio+210
    }
    if ($("#bluray").prop("checked")) {
        precio=precio+190
    }
    if ($("#aceleradora").prop("checked")) {
        precio=precio+430
    }
    if ($("#monitor").prop("checked")) {
        precio=precio+185
    }

    $("#total").val(precio)
}
function main(){
    $("#intel").click (calcular_precio)
    $("#amd").click (calcular_precio)

    $("#2gb").click (calcular_precio)
    $("#4gb").click (calcular_precio)
    $("#8gb").click (calcular_precio)

    $("#bluray").click (calcular_precio)
    $("#monitor").click (calcular_precio)
    $("#aceleradora").click (calcular_precio)
}

```

4.13.5 Ejercicio configurador de PCs ampliado

En el ejercicio anterior ocurre que por un problema hardware no es posible tener procesadores AMD con aceleradora, por lo que cuando se marque un AMD se debe desactivar el checkbox de la aceleradora y si hubiera una marca, también se debe desactivar y por supuesto recalcular el precio.

4.14 Configurador de coches

Un fabricante de automóviles desea ofrecer a sus clientes una aplicación que les permita configurar sus vehículos según sus preferencias y ver el precio final del coche. Los precios y las restricciones son los siguientes:

- Se pueden tener dos tipos de motor: gasolina (precio base 7000 euros) y diésel (precio base 8200).
- Se pueden tener 3 potencias: 1100, 1800 y 2300 centímetros cúbicos. Los precios de cada uno son 800, 1900 y 2500. Sin embargo **no es posible fabricar motores diésel de 2300**.
- Hay dos tipos de pintura: normal y metalizada. Los precios respectivos son 750 y 1580 euros.
- Hay seis colores: negro, blanco, rojo, azul polar, verde turquesa y gris marengo. **No se pueden fabricar colores de pintura normal de ninguno de los tres últimos colores.**
- Se dispone de diversos extras: alerón deportivo (190 euros ******pero solo se puede elegir si se elige pintura metalizada), radio-CD con MP3 (230 euros más), altavoces traseros (320 euros más, ******pero solo si se elige antes el Radio-CD), y GPS incorporado (520 euros más).

Crear la aplicación que respete las restricciones exigidas por el cliente.

4.15 Dinamismo con Google Maps

Google Maps ofrece un servicio de mapas con una limitación de 25.000 peticiones diarias. El código básico sería así:

4.15.1 HTML de GMaps

```
<!DOCTYPE html>

<html>
<head>
  <!--En móviles poner la escala inicial a 1-->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!--Cargamos los estilos y los efectos de Bootstrap-->
  <link
    rel="stylesheet"
    type="text/css" href="bootstrap/css/bootstrap.css"/>
  <script src="bootstrap/js/bootstrap.js"></script>
  <script src="js/jquery.js"></script>
  <script
    src="http://maps.googleapis.com/maps/api/js?key=AIzaSyDpv9zCj9szIIu--
    ↪LuNmDsry2fZCRrOqfY&sensor=false">

    </script>
  <style>
    #mapa{
      width: 500px;
      height: 500px;
      float: right;
      background-color: rgb(180,190,240);
    }
    #controles{
      float: left;
    }
  </style>
  <script src="js/programa.js"></script>
  <title>Plantilla JQuery</title>
</head>

<body>
<div class="container">
  <h1>Javascript con mapas</h1>
  <div id="controles">
    Introduzca latitud:<input type="text" id="latitud">
    <br/>
    Introduzca longitud:<input type="text" id="longitud">
    <br/>
    <input type="submit" id="mover" value="¡Viajar!">
    <select id="ciudades">
      <option value="CR">Ir a Ciudad Real</option>
      <option value="BA">Ir a Barcelona</option>
      <option value="PO">Ir a Pontevedra</option>
    </select>
    <br/>
    Calculador de distancias desde CR a otras ciudades
    <select id="ciudades">
      <option value="CR">Ir a Ciudad Real</option>
```

```

        <option value="BA">Ir a Barcelona</option>
        <option value="PO">Ir a Pontevedra</option>
    </select>

</div>
<div id="mapa">

</div>
</div>
</body>
</html>

```

4.15.2 Javascript de GMaps

```

var latitud=38.59
var longitud=-3.55
var mi_nivel_de_zoom=8
var obj_mapa
function inicio() {
    var div_mapa=document.getElementById("mapa")
    var obj_coordenadas=new google.maps.LatLng(latitud,longitud)
    var obj_opciones={
        center:obj_coordenadas,
        zoom:mi_nivel_de_zoom
    }
    obj_mapa=new google.maps.Map(div_mapa, obj_opciones)

    $("#mover").click (mover_el_mapa)
}

function mover_el_mapa() {
    var obj_latitud=$("#latitud")
    var valor_latitud=obj_latitud.val()

    var valor_longitud=$("#longitud").val()
    var nuevas_coordenadas=new google.maps.LatLng(
        valor_latitud, valor_longitud)
    obj_mapa.panTo(nuevas_coordenadas)
}

```

4.15.3 Ejercicio

Ampliar el programa con una lista de ciudades del mundo. Cuando el usuario elija una de ellas, nuestro programa nos dirá la distancia desde Ciudad Real a dichas ciudades. Considerar las siguientes coordenadas en formato (latitud, longitud):

- Ciudad Real: (38.59, -3.55)
- Nueva York: (40.73, -73.87)
- Sidney: (-33.90, 151.13)
- Berlin: (52.31, 13.39)
- París: (48.85, 2.35)

Para poder conseguir esto, hay que modificar la URL de carga de GoogleMaps para solicitar que se cargue una biblioteca que nos ayudará a resolver este punto. En concreto, ahora pasaremos un parámetro `libraries` con el valor `geometry` que nos permitirá utilizar la biblioteca en concreto. Ahora el HTML es así:

```
<script src="http://maps.googleapis.com/maps/api/js?key=AIzaSyDpv9zCj9szIIu--
↳LuNmDsry2fZCRrOqfY&sensor=false&libraries=geometry">

</script>
```

Ahora una función que nos calcula la distancia sería algo como esto:

```
/* Nos da la distancia en metros entre CR
 * y el punto (latitud_destino, longitud_destino)
 * (abreviados lat_dest y lng_dest) */
function distancia(lat_dest, lng_dest)
{
    var latitud_cr=38.59
    var longitud_cr=-3.55
    var coords_origen=new google.maps.LatLng(
        latitud_cr, longitud_cr)
    var coords_destino=new google.maps.LatLng(
        lat_dest, lng_dest)
    var distancia=google.maps.geometry.spherical.computeDistanceBetween(
        coords_origen, coords_destino
    )
    return distancia
}
```

4.15.4 Ejercicio

La empresa Automobile Creation for Millenium Enterprise (ACME) planea lanzar una página web en la que se permita al usuario configurar los coches a su medida, ofreciendo las distintas opciones en pantalla para que el usuario las elija. Sin embargo no todas las combinaciones se permiten en fábrica por lo que deberán tenerse en cuentas las siguientes

Especificaciones

- Hay dos motores: gasolina (5000) y diésel (6800)
- Hay dos carrocerías: monovolumen (4500) y berlina (3700)
- Hay tres accesorios: radio-cd con MP3 (180), alerones deportivos (220) y llantas de aleación (200)

Por diversos problemas, no es posible combinar las siguientes opciones:

- No se pueden tener berlinas de gasolina.
- No se puede integrar el alerón en los monovolúmenes.
- No se puede poner el radio-cd a los monovolúmenes.

Cuando se marque cualquiera de estas opciones, hay que limpiar todo el configurados y avisar de que no se puede hacer eso. No se pueden usar alerts

```
var vector_ids=["#gasolina", "#diesel", "#monovolumen",
               "#berlina", "#radiocd", "#alerones", "#llantas"]
var precios=[5000, 6800, 4500,
             3700, 180, 220, 200]
```

```

function inicio(){

    for (var pos in vector_ids){
        var el_id=vector_ids[pos]
        $(el_id).click ( calcularPrecio )
    }
}

function cocheEsFabricable() {
    //Caso 1: nada de berlinas de gasolina
    var marcada_la_berlina=$( "#berlina" ).prop("checked")
    var marcada_la_gasolina=$( "#gasolina" ).prop("checked")
    if (marcada_la_berlina && marcada_la_gasolina) {
        alert ("No se pueden fabricar berlinas de gasolina")
        return false
    }

    var marcado_aleron=$( "#aleron" ).prop("checked")
    var marcado_monovolumen=$( "#monovolumen" ).prop("checked")
    if (marcado_aleron && marcado_monovolumen ) {
        alert ("No podemos integrar los alerones en monovolúmenes")
        return false
    }

    var marcado_radiocd=$( "#radiocd" ).prop("checked")
    if (marcado_radiocd && marcado_monovolumen) {
        alert ("No podemos fabricar un monovol. con radio-cd")
        return false
    }
    return true
}

/* Calcula el precio del coche en función de lo que esté marcado
 * y lo que no.*/
function calcularPrecio() {
    var todo_bien=cocheEsFabricable()
    if (todo_bien!=true) {
        return
    }
    var precioCoche=0
    for (var pos in vector_ids){
        var el_id=vector_ids[pos]
        if ($(el_id).prop("checked")) {
            var precio_accesorio=precios[pos]
            precioCoche=precioCoche+precio_accesorio
        }
    }
    alert ("El precio es:"+precioCoche)
}

```

4.15.5 Ampliación

Se desea que el usuario puede elegir entre los siguientes colores con los siguientes precios:

- Blanco: 700 euros
- Rojo, Verde y Azul básicos: 950
- Gris, Negro y Naranja: 1400 euros por ser colores metalizados

Además, se desea ver una muestra de color en algún punto de la página. Para lograrlo se necesitará utilizar un método que proporciona JQuery y que se llama `addClass`

Solución HTML

```
<head>
  <style>
    .muestrarojo{
      background-color:red;
    }
    .muestraverde{
      background-color: green;
    }
    .muestraazul{
      background-color: blue;
    }
    .muestragris{
      background-color: grey;
    }
    .muestrablanco{
      background-color: white;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>Motores</h1>
    <input type="radio" id="gasolina" name="motor">Motor Gasolina
    <br/>
    <input type="radio" id="diesel" name="motor">Motor Diésel
    <br/>
    <h1>Carrocerías</h1>
    <input type="radio" id="monovolumen"
    name="carroceria">Monovolumen
    <br/>
    <input type="radio" id="berlina" name="carroceria">Berlina
    <br/>
    <h1>Accesorios</h1>
    <input type="checkbox" name="accesorios[]"
    id="radiocd">Radio-CD
    <br/>
    <input type="checkbox" name="accesorios[]"
    id="alerones">Alerones
    <br/>
    <input type="checkbox" name="accesorios[]"
    id="llantas">Llantas
    <br/>
    <h1>Colores</h1>
    <!--Los colores irán
    en una columna y la muestra en otra-->
    <div class="row">
      <div class="col-md-3">
        <input type="radio"
        name="colores"
        id="blanco">Blanco
        <br/>
        <input type="radio"
```

```

        name="colores"
        id="rojo">Rojo
        <br/>
        <input type="radio"
        name="colores" id="gris">Gris
    </div>
    <div class="col-md-9 center-block">
        <h2>Muestra de color
        <small>Observe y compare</small></h2>
    </div>
    </div><!--Fin de la fila-->
</div>
</body>

```

Solución JS

Añadiremos este código a nuestro programa anterior.

```

var obj_documento = $(document)
obj_documento.ready(inicio)

var vector_ids=["#gasolina", "#diesel", "#monovolumen",
               "#berlina", "#radiocd", "#alerones", "#llantas"]
var precios=[5000, 6800, 4500,
             3700, 180, 220, 200]

function inicio(){

    for (var pos in vector_ids){
        var el_id=vector_ids[pos]
        $(el_id).click ( calcularPrecio )
    }
    $("#blanco").click ( ponerColorBlanco )
    $("#rojo").click ( ponerColorRojo )
    $("#gris").click ( ponerGris )
}

function limpiarColores(){
    var clases=["muestrarojo", "muestrablanca",
               "muestragris"]
    for (var pos in clases) {
        $("#muestra").removeClass( clases[pos] )
    }
}

function ponerGris() {
    limpiarColores()
    $("#muestra").addClass("muestragris")
}

function ponerColorRojo(){
    limpiarColores()
    $("#muestra").addClass ("muestrarojo")
}

function ponerColorBlanco(){
    limpiarColores()
    $("#muestra").addClass ("muestrablanca")
}

```


5.1 Introducción

Los lenguajes de marcas como HTML tienen una orientación muy clara: describir páginas web.

En un contexto distinto, muy a menudo ocurre que es muy difícil intercambiar datos entre programas.

XML es un conjunto de tecnologías orientadas a crear nuestros propios lenguajes de marcas. A estos lenguajes de marcas “propios” se les denomina “vocabularios”.

5.2 Un ejemplo sencillo

```
<clientes>
  <cliente>
    <nombre>AcerSA</nombre>
    <cif>5664332</cif>
  </cliente>
  <cliente>
    <nombre>Mer SL</nombre>
    <cif>5111444</cif>
  </cliente>
</clientes>
```

Lo fundamental es que podemos crear nuestros propios “vocabularios” XML.

5.3 Construcción de XML

Para crear XML es importante recordar una serie de reglas:

- XML es “case-sensitive”, es decir que no es lo mismo mayúsculas que minúsculas y que por tanto no es lo mismo `<cliente>`, que `<Cliente>` que `<CLIENTE>`.
- Obligatorio: solo un elemento raíz.
- En general, la costumbre es poner todo en minúsculas.
- Solo se puede poner una etiqueta que empiece por letra o `_`. Es decir, esta etiqueta no funcionará en los programas `<12Cliente>`.
- Aparte de eso, una etiqueta sí puede contener números, por lo que esta etiqueta sí es válida `<Cliente12>`.

5.4 Validez

Un documento XML puede “estar bien formado” o “ser válido”. Se dice que un documento “está bien formado” cuando respeta las reglas XML básicas. Si alguien ha definido las reglas XML para un vocabulario, podremos además decir si el documento es válido o no, lo cual es mejor que simplemente estar bien formado.

Por ejemplo, los siguientes archivos ni siquiera están bien formados.

```
<clientes>
  <cliente>
    <nombre>AcerSA
    <CIF>5666333</CIF>
  </cliente>
</clientes>
```

En este caso la etiqueta <nombre> no está cerrada.

```
<clientes>
  <cliente>
    <nombre>AcerSA</nombre>
    <cif>5666333</CIF>
  </cliente>
</clientes>
```

En este caso, se ha puesto <cif> cerrado con </CIF> (mayúsculas).

```
<clientes>
  <cliente>
    <nombre!>AcerSA</nombre!>
    <CIF>5666333</CIF>
  </cliente>
</clientes>
```

Se ha utilizado la admiración, que no es válida.

Atención a este ejemplo:

```
<cliente>
  <nombre>AcerSA</nombre>
  <CIF>5666333</CIF>
</cliente>
<cliente>
  <nombre>ACME</nombre>
  <CIF>455321</CIF>
</cliente>
```

En este caso, el problema es que hay más de un elemento raíz.

En general, podemos asumir que un documento puede estar en uno de estos estados que de peor a mejor podríamos indicar así:

1. Mal formado (lo peor)
2. Bien formado.
3. Válido: está bien formado y además nos han dado las reglas para determinar si algo está bien o mal y el documento XML cumple dichas reglas. Este es el mejor caso.

Para determinar si un documento es válido o no, se puede usar el validador del W3C situado en <http://validator.w3c.org>

5.5 Gramáticas

Pensemos en el siguiente problema, un programador crea aplicaciones con documentos que se almacenan así:

```
<clientes>
  <cliente>
    <nombre>AcerSA</nombre>
    <cif>455321</cif>
  </cliente>
  <cliente>
    <nombre>ACME</nombre>
    <cif>455321</cif>
  </cliente>
</clientes>
```

Sin embargo, otro programador de la misma empresa lo hace así:

```
<clientes>
  <cliente>
    <cif>455321</cif>
    <nombre>AcerSA</nombre>
  </cliente>
  <cliente>
    <cif>455321</cif>
    <nombre>ACME</nombre>
  </cliente>
</clientes>
```

Está claro, que ninguno de los dos puede leer los archivos del otro, sería crítico ponerse de acuerdo en lo que se puede hacer, lo que puede aparecer y en qué orden debe hacerlo. Esto se hará mediante las DTD.

DTD significa Declaración de Tipo de Documento, y es un mecanismo para expresar las reglas sobre lo que se va a permitir y lo que no en archivos XML.

Por ejemplo, supongamos el mismo ejemplo anterior en el que queremos formalizar lo que puede aparecer en un fichero de clientes. Se debe tener en cuenta que en un DTD se pueden indicar reglas para lo siguiente:

- Se puede indicar si un elemento aparece o no de forma opcional
- Se puede indicar si un elemento debe aparecer de forma obligatoria.
- Se puede indicar si algo aparece una o muchas veces.
- Se puede indicar si algo aparece cero o muchas veces.

Supongamos que en nuestros ficheros deseamos indicar que el elemento raíz es `<listaclientes>`. Dentro de `<listaclientes>` deseamos permitir uno o más elementos `<cliente>`. Dentro de `<cliente>` todos deberán tener `<cif>` y `<nombre>` y en ese orden. Dentro de `<cliente>` puede aparecer o no un elemento `<diasentrega>` para indicar que ese cliente exige un máximo de plazos. Como no todo el mundo usa plazos el `<diasentrega>` es optativo.

Por ejemplo, este XML sí es válido:

```
<listaclientes>
  <cliente>
    <cif>5676443</cif>
    <nombre>Mercasa</nombre>
  </cliente>
</listaclientes>
```

Este también lo es:

```
<listaclientes>
  <cliente>
    <cif>5676443</cif>
    <nombre>Mercasa</nombre>
    <diasentrega>30</diasentrega>
  </cliente>
</listaclientes>
```

Este también:

```
<listaclientes>
  <cliente>
    <cif>5676443</cif>
    <nombre>Mercasa</nombre>
    <diasentrega>30</diasentrega>
  </cliente>
  <cliente>
    <cif>5121554</cif>
    <nombre>Acer SL</nombre>
  </cliente>
</listaclientes>
```

Sin embargo, estos no lo son:

```
<listaclientes>
</listaclientes>
```

Este archivo no tenía clientes (y era obligatorio al menos uno)

```
<listaclientes>
  <cliente>
    <cif>5676443</cif>
    <diasentrega>30</diasentrega>
  </cliente>
</listaclientes>
```

Este archivo no tiene nombre de cliente.

```
<listaclientes>
  <cliente>
    <nombre>Mercasa</nombre>
    <cif>5676443</cif>
  </cliente>
  <cliente>
    <cif>5121554</cif>
    <nombre>Acer SL</nombre>
  </cliente>
</listaclientes>
```

En este archivo no se respeta el orden cif, nombre.

5.5.1 Sintaxis DTD

Una DTD es como un CSS, puede ir en el mismo archivo XML o puede ir en uno separado. Para poder subirlos al validador, meteremos la DTD junto con el XML.

La primera línea de todo XML debe ser esta:

```
<?xml version="1.0"?>
```

Al final del XML pondremos los datos propiamente dichos

```
<listaclientes>
  <cliente>
    <nombre>Mercasa</nombre>
    <cif>5676443</cif>
  </cliente>
  <cliente>
    <cif>5121554</cif>
    <nombre>Acer SL</nombre>
  </cliente>
</listaclientes>
```

La DTD tiene esta estructura

```
<!DOCTYPE listaclientes [
    <!ELEMENT listaclientes (cliente+)>
    <!ELEMENT cliente (nombre, cif, diasentrega?)>
    <!ELEMENT nombre (#PCDATA)>
    <!ELEMENT cif (#PCDATA)>
    <!ELEMENT diasentrega (#PCDATA)>
]
>
```

Esto significa lo siguiente:

- Se establece el tipo de documento `listaclientes` que consta de una serie de elementos (dentro del corchete)
- Un elemento `listaclientes` ``consta de uno o más clientes. El signo ``+ significa “uno o más”.
- Un cliente tiene un nombre y un cif. También puede tener un elemento `diasentrega` que puede o no aparecer (el signo ? significa “0 o 1 veces”).
- Un nombre no tiene más elementos dentro, solo caracteres (#PCDATA)
- Un CIF solo consta de caracteres.
- Un elemento `diasentrega` consta solo de caracteres.

La solución completa sería así:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE listaclientes [
    <!ELEMENT listaclientes (cliente+)>
    <!ELEMENT cliente (nombre, cif, diasentrega?)>
    <!ELEMENT nombre (#PCDATA)>
    <!ELEMENT cif (#PCDATA)>
    <!ELEMENT diasentrega (#PCDATA)>
]
>
<listaclientes>
  <cliente>
    <nombre>Mercasa</nombre>
    <cif>5676443</cif>
  </cliente>
  <cliente>
    <nombre>Acer SL</nombre>
```

```
        <cif>5121554</cif>
    </cliente>
</listaclientes>
```

5.6 Ejercicio I (DTD)

Unos programadores necesitan un formato de fichero para que sus distintos programas intercambien información sobre ventas. El acuerdo al que han llegado es que su XML debería tener esta estructura:

- El elemento raíz será <listaventas>
- Toda <listaventas> tiene una o más ventas.
- Toda <venta> tiene los siguientes datos:
 - Importe.
 - Comprador.
 - Vendedor.
 - Fecha (optativa).
 - Un código de factura.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE listaventas[
  <!ELEMENT listaventas (venta+)>
  <!ELEMENT venta (importe, comprador, vendedor, fecha?, codigofactura)>
  <!ELEMENT importe (#PCDATA)>
  <!ELEMENT comprador (#PCDATA)>
  <!ELEMENT vendedor (#PCDATA)>
  <!ELEMENT fecha (#PCDATA)>
  <!ELEMENT codigofactura (#PCDATA)>
]>

<listaventas>
  <venta>
    <importe>1500</importe>
    <comprador>Wile E.Coyote</comprador>
    <vendedor>ACME</vendedor>
    <codigofactura>E17</codigofactura>
  </venta>
  <venta>
    <importe>750</importe>
    <comprador>Elmer Fudd</comprador>
    <vendedor>ACME</vendedor>
    <fecha>27-2-2015</fecha>
    <codigofactura>E18</codigofactura>
  </venta>
</listaventas>
```

5.7 Ejercicio II (DTD)

Crear un XML de ejemplo y la DTD asociada para unos programadores que programan una aplicación de pedidos donde hay una lista de pedidos con 0 o más pedidos. Cada pedido tiene un número de serie, una cantidad y un peso que puede ser opcional.

5.7.1 Solución

Este ejemplo es un documento XML válido.

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

<!DOCTYPE listapedidos [
    <!ELEMENT listapedidos (pedido*)>
    <!ELEMENT pedido (numeroserie, cantidad, peso?)>
    <!ELEMENT numeroserie (#PCDATA)>
    <!ELEMENT cantidad (#PCDATA)>
    <!ELEMENT peso (#PCDATA)>
]>

<listapedidos>
</listapedidos>
```

Este documento **no es válido**

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

<!DOCTYPE listapedidos [
    <!ELEMENT listapedidos (pedido*)>
    <!ELEMENT pedido (numeroserie, cantidad, peso?)>
    <!ELEMENT numeroserie (#PCDATA)>
    <!ELEMENT cantidad (#PCDATA)>
    <!ELEMENT peso (#PCDATA)>
]>

<listapedidos>
    <pedido>
        <numeroserie>23332244</numeroserie>
    </pedido>
</listapedidos>
```

Este documento **sí es válido**. Las DTD solo se ocupan de determinar qué elementos hay y en qué orden, pero no se ocupan de lo que hay dentro de los elementos.

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

<!DOCTYPE listapedidos [
    <!ELEMENT listapedidos (pedido*)>
    <!ELEMENT pedido (numeroserie, cantidad, peso?)>
    <!ELEMENT numeroserie (#PCDATA)>
    <!ELEMENT cantidad (#PCDATA)>
    <!ELEMENT peso (#PCDATA)>
]>

<listapedidos>
    <pedido>
```

```

                <numeroserie>23332244</numeroserie>
                <cantidad>ññlñ</cantidad>
            </pedido>
</listapedidos>

```

5.8 Ejercicio (con atributos)

Unos programadores necesitan estructurar la información que intercambiarán los ficheros de sus aplicaciones para lo cual han determinado los requisitos siguientes:

- Los ficheros deben tener un elemento `<listafacturas>`
- Dentro de la lista debe haber una o más facturas.
- Las facturas tienen un atributo `fecha` que es optativo.
- Toda factura tiene un `emisor`, que es un elemento obligatorio y que debe tener un atributo `cif` que es obligatorio. Dentro de `emisor` debe haber un elemento `nombre`, que es obligatorio y puede o no haber un elemento `volumenventas`.
- Toda factura debe tener un elemento `pagador`, el cual tiene exactamente la misma estructura que `emisor`.
- Toda factura tiene un elemento `importe`.

5.8.1 Solución ejercicio con atributos

La siguiente DTD refleja los requisitos indicados en el enunciado.

```

<!ELEMENT listafacturas (factura+)>
<!ELEMENT factura (emisor, pagador, importe)>
<!ATTLIST factura fecha CDATA #IMPLIED>
<!ELEMENT emisor (nombre, volumenventas?)>
<!ELEMENT nombre (#PCDATA)>
<!ATTLIST emisor cif CDATA #REQUIRED>
<!ELEMENT volumenventas (#PCDATA)>
<!ELEMENT pagador (nombre, volumenventas?)>
<!ATTLIST pagador cif CDATA #REQUIRED>
<!ELEMENT importe (#PCDATA)>

```

Y el XML siguiente refleja un posible documento. Puede comprobarse que es válido con respecto a la DTD.

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE listafacturas SYSTEM "ListaFacturas.dtd">
<listafacturas>
  <factura fecha="11-2-2015">
    <emisor cif="123">
      <nombre>ACME</nombre>
    </emisor>
    <pagador cif="234">
      <nombre>ACME Inc</nombre>
      <volumenventas>2000</volumenventas>
    </pagador>
    <importe>2500</importe>
  </factura>
</listafacturas>

```


5.9 Ejercicio

Un instituto necesita registrar los cursos y alumnos que estudian en él y necesita una DTD para comprobar los documentos XML de los programas que utiliza:

- Tiene que haber un elemento raíz `listacursos`. Tiene que haber uno o más cursos.
- Un curso tiene uno o más alumnos
- Todo alumno tiene un DNI, un nombre y un apellido, puede que tenga segundo apellido o no.
- Un alumno escoge una lista de asignaturas donde habrá una o más asignaturas. Toda asignatura tiene un nombre, un atributo código y un profesor.
- Un profesor tiene un NRP (Número de Registro Personal), un nombre y un apellido (también puede tener o no un segundo apellido).

5.9.1 Solución punto 1

La raíz es `listacursos` y tiene uno o más cursos

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

<!DOCTYPE listacursos [
    <!ELEMENT listacursos (curso+)>
]>
```

5.9.2 Solución punto 2

Un curso tiene uno o más elementos alumno.

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

<!DOCTYPE listacursos [
    <!ELEMENT listacursos (curso+)>
    <!ELEMENT curso (alumno+)>
]>
```

5.9.3 Solución punto 3

Todo alumno tiene DNI, nombre y apellido 1, pero puede que tenga el segundo o no

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

<!DOCTYPE listacursos [
    <!ELEMENT listacursos (curso+)>
    <!ELEMENT curso (alumno+)>
    <!ELEMENT alumno (dni, nombre, ap1, ap2?)>
]>
```

5.9.4 Solución al punto 4

Todo alumno tiene una lista de asignaturas que consta de una o más asignaturas. Ampliamos el elemento alumno:

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

<!DOCTYPE listacursos [
    <!ELEMENT listacursos (curso+)>
    <!ELEMENT curso (alumno+)>
    <!ELEMENT alumno (dni, nombre, ap1, ap2?, listaasignaturas)>
    <!ELEMENT listaasignaturas (asignatura+)
]>
```

5.9.5 Solución punto 5

Una posible solución implicaría usar dos elementos nombre. *Está permitido* pero solo si ambos elementos se usan de la misma forma (por ejemplo que usen (#PCDATA)). Para evitar problemas cambiaremos algunos nombres de elementos.

Este punto pedía contemplar que una asignatura tiene un nombre, un código y un profesor.

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

<!DOCTYPE listacursos [
    <!ELEMENT listacursos (curso+)>
    <!ELEMENT curso (alumno+)>
    <!ELEMENT alumno (dni, nombre_alumno, ap1, ap2?, listaasignaturas)>
    <!ELEMENT listaasignaturas (asignatura+)>
    <!ELEMENT asignatura (nombre_asig, profesor)>
    <!--ATTLIST asignatura codigo CDATA #REQUIRED-->
    <!ELEMENT nombre_asig (#PCDATA)>
]>
```

5.9.6 Solución punto 6

Se indica que un profesor tiene una serie de elementos dentro. Aquí hay un claro ejemplo en el que repetir el elemento ap1 o el ap2 hubiera sido apropiado, ya que los apellidos de alumnos o profesores en realidad no se distinguen en nada.

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

<!DOCTYPE listacursos [
    <!ELEMENT listacursos (curso+)>
    <!ELEMENT curso (alumno+)>
    <!ELEMENT alumno (dni, nombre_alumno, ap1, ap2?, listaasignaturas)>
    <!ELEMENT listaasignaturas (asignatura+)>
    <!ELEMENT asignatura (nombre_asig, profesor)>
    <!--ATTLIST asignatura codigo CDATA #REQUIRED-->
    <!ELEMENT nombre_asig (#PCDATA)>
    <!ELEMENT profesor (nro, nombre_prof, ap_prof1, ap_prof2?)>
]>
```

5.9.7 Solución completa

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

```

<!DOCTYPE listacursos [
  <!ELEMENT listacursos (curso+)>
  <!ELEMENT curso (alumno+)>
  <!ELEMENT alumno (dni, nombre_alumno, ap1, ap2?, listaasignaturas)>
  <!ELEMENT listaasignaturas (asignatura+)>
  <!ELEMENT asignatura (nombre_asig, profesor)>
  <!ATTLIST asignatura codigo CDATA #REQUIRED>
  <!ELEMENT nombre_asig (#PCDATA)>
  <!ELEMENT profesor (nrp, nombre_prof, ap_prof1, ap_prof2?)>

  <!ELEMENT dni (#PCDATA)>
  <!ELEMENT nombre_alumno (#PCDATA)>
  <!ELEMENT ap1 (#PCDATA)>
  <!ELEMENT ap2 (#PCDATA)>
  <!ELEMENT nrp (#PCDATA)>
  <!ELEMENT nombre_prof (#PCDATA)>
  <!ELEMENT ap_prof1 (#PCDATA)>
  <!ELEMENT ap_prof2 (#PCDATA)>
]>

```

```

<listacursos>
  <curso>
    <alumno>
      <dni>1234567</dni>
      <nombre_alumno>Juan</nombre_alumno>
      <ap1>Sanchez</ap1>
      <listaasignaturas>
        <asignatura>
          <nombre_asig>
            Lenguajes de marcas
          </nombre_asig>
          <codigo>
            XML-DAM1
          </codigo>
          <profesor>
            <nrp>
              03409435898W0303
            </nrp>
            <nombre_prof>
              Andres
            </nombre_prof>
            <ap_prof1>
              Ruiz
            </ap_prof1>
          </profesor>
        </asignatura>
      </listaasignaturas>
    </alumno>
  </curso>
</listacursos>

```

5.10 Otras características de XML

5.10.1 Atributos

Un atributo XML funciona exactamente igual que un atributo HTML, en concreto un atributo es un trozo de información que acompaña a la etiqueta, en lugar de ir dentro del elemento.

```
<pedido codigo="20C">
  <contenido>
    ...
</pedido>
```

En este caso, la etiqueta `pedido` tiene un atributo `codigo`.

¿Cuándo debemos usar atributos y cuándo debemos usar elementos? Resulta que el ejemplo anterior también se podría haber permitido hacerlo así:

```
<pedido>
  <codigo>20C</codigo>
  <contenido>
    ...
</pedido>
```

Hay muchas discusiones sobre qué meter dentro de elemento o atributo. Sin embargo, los expertos coinciden en señalar que en caso de duda es mejor el segundo.

La definición de atributos se hace por medio de una directiva llamada `ATTLIST`. En concreto si quisieramos permitir un atributo `código` en el elemento `pedido` se haría algo así.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE pedido[
  <!ELEMENT pedido (contenido)>
  <!ELEMENT contenido (#PCDATA)>
  <!ATTLIST pedido codigo CDATA #REQUIRED>
]>

<pedido codigo="20C">
  <contenido>Pedido de cosas</contenido>
</pedido>
```

En concreto este código pone que el elemento `pedido` tiene un atributo `código` con datos carácter dentro y que es obligatorio que esté presente (un atributo optativo en vez de `#REQUIRED` usará `#IMPLIED`)

Si probamos esto, también validará porque el atributo es *optativo*

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE pedido[
  <!ELEMENT pedido (contenido)>
  <!ELEMENT contenido (#PCDATA)>
  <!ATTLIST pedido codigo CDATA #IMPLIED>
]>

<pedido>
  <contenido>Pedido de cosas</contenido>
</pedido>
```

5.10.2 Elementos vacíos

En ocasiones, un elemento en especial puede interesarnos que vaya vacío porque simplemente no contiene mucha información de relevancia. Por ejemplo en HTML podemos encontrarnos esto:

```
<b>Texto texto...</b>
<br/>
```

Los elementos vacíos suelen utilizar para indicar pequeñas informaciones que no deseamos meter en atributos y que de todas formas tampoco son de demasiada relevancia.

Un elemento vacío se indica poniendo EMPTY en lugar de #PCDATA

Por supuesto, estas dos formas de usar un atributo son válidas:

```
<pedido>
  <pagado></pagado>
  <contenido>...</contenido>
</pedido>
```

```
<pedido>
  <pagado/>
  <contenido>...</contenido>
</pedido>
```

La definición completa sería así:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE pedido[
  <!ELEMENT pedido (pagado?, contenido)>
  <!ELEMENT pagado EMPTY>
  <!ELEMENT contenido (#PCDATA)>
  <!ATTLIST pedido codigo CDATA #IMPLIED>
]>

<pedido>
  <pagado/>
  <contenido>Pedido de cosas</contenido>
</pedido>
```

5.10.3 Alternativas

Hasta ahora hemos indicado elementos donde un elemento puede aparecer o puede no aparecer, pero ¿qué ocurre si deseamos obligar a que aparezca una posibilidad entre varias?

Por ejemplo, supongamos que en un nuestro ejemplo de pedidos deseamos indicar si el pedido se entregó en almacén o a domicilio. A la fuerza todo pedido se entrega de alguna manera, sin embargo queremos exigir que en los XML aparezca una de esas dos alternativas. Los elementos alternativos se indican con la barra vertical `almacen|domicilio`

Una tentación sería hacer esto (que está **mal**):

```
<!DOCTYPE pedido[
  <!ELEMENT pedido (pagado?, contenido, almacen?, domicilio?)>
  <!ELEMENT pagado EMPTY>
  <!ELEMENT contenido (#PCDATA)>
  <!ELEMENT almacen (#PCDATA)>
```

```
<!ELEMENT domicilio (#PCDATA)>
]>
```

Está mal porque se permite esto:

```
<pedido>
  <pagado/>
  <contenido>Ordenadores</contenido>
  <almacen>Entregado el 20-2-2011</almacen>
  <domicilio>Entregado el 20-2011</domicilio>
</pedido>
```

La forma **correcta** es esta:

```
<!DOCTYPE pedido[
  <!ELEMENT pedido (pagado?, contenido, (almacen|domicilio)?>
  <!ELEMENT pagado EMPTY>
  <!ELEMENT contenido (#PCDATA)>
  <!ELEMENT almacen (#PCDATA)>
  <!ELEMENT domicilio (#PCDATA)>
]>
<pedido>
  <contenido>Ordenadores</contenido>
</pedido>
```

5.11 Ejercicio

Un mayorista informático necesita especificar las reglas de los elementos permitidos en las aplicaciones que utiliza en sus empresas, para ello ha indicado los siguientes requisitos:

- Una entrega consta de uno o más lotes.
- Un lote tiene uno o más palés
- Todo palé tiene una serie de elementos: número de cajas, contenido y peso y forma de manipulación.
- El contenido consta de una serie de elementos: nombre del componente, procedencia (puede aparecer 0, 1 o más países), número de serie del componente, peso del componente individual y unidad de peso que puede aparecer o no.

5.11.1 Solución

Observa como en la siguiente DTD se pone `procedencia?` y dentro de ella `pais+`. Esto nos permite que si aparece la procedencia se debe especificar uno o más países. Sin embargo si no queremos que aparezca ningún país, el XML **no necesita contener un elemento vacío**.

```
<!ELEMENT entrega (lote+)>
<!ELEMENT lote (pale+)>
<!ELEMENT pale (numcajas, contenido, peso, formamanipulacion?)>
<!ELEMENT numcajas (#PCDATA)>
<!ELEMENT peso (#PCDATA)>
<!ELEMENT formamanipulacion (#PCDATA)>
<!ELEMENT contenido (nombrecomponente, procedencia?,
                     numserie, peso, unidades)>
<!ELEMENT nombrecomponente (#PCDATA)>
```

```

<!ELEMENT procedencia (pais+)>
<!ELEMENT pais (#PCDATA)>
<!ELEMENT numserie (#PCDATA)>
<!ELEMENT unidades (#PCDATA)>

```

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE entrega SYSTEM "mayorista.dtd">
<entrega>
  <lote>
    <pale>
      <numcajas>3</numcajas>
      <contenido>
        <nombrecomponente>Fuentes</nombrecomponente>
        <numserie>3A</numserie>
        <peso>2kg</peso>
        <unidades>50</unidades>
      </contenido>
      <peso>100kg</peso>
      <formamanipulacion>Manual</formamanipulacion>
    </pale>
  </lote>
  <lote>
    <pale>
      <numcajas>2</numcajas>
      <contenido>
        <nombrecomponente>CPUs</nombrecomponente>
        <procedencia>
          <pais>China</pais>
          <pais>Corea del Sur</pais>
        </procedencia>
        <numserie>5B</numserie>
        <peso>100g</peso>
        <unidades>1000</unidades>
      </contenido>
      <peso>100kg</peso>
      <formamanipulacion>Manual</formamanipulacion>
    </pale>
  </lote>
</entrega>

```

5.12 Ejercicio: mayorista de libros

Se desea crear un formato de intercambio de datos para una empresa mayorista de libros con el fin de que sus distintos programas puedan manejar la información interna. El formato de archivo debe tener la siguiente estructura:

- Un archivo tiene una serie de operaciones dentro.
- Las operaciones pueden ser “venta”, “compra”, o cualquier combinación y secuencia de ellas, pero debe haber al menos una.
- Una venta tiene:
 - Uno o más títulos vendidos.
 - La cantidad total de libros vendidos.
 - Puede haber un elemento “entregado” que indique si la entrega se ha realizado.

- Debe haber un elemento importe con un atributo obligatorio llamado “moneda”.
- Una compra tiene:
 - Uno o más títulos comprados.
 - Nombre de proveedor.
 - Una fecha de compra, que debe desglosarse en elementos día, mes y año

5.12.1 Solución al mayorista de libros

5.13 Ejercicio: fabricante de tractores

Un fabricante de tractores desea unificar el formato XML de sus proveedores y para ello ha indicado que necesita que los archivos XML cumplan las siguientes restricciones:

- Un pedido consta de uno o más tractores.
- Un tractor consta de uno o más componentes.
- Un componente tiene los siguientes elementos: nombre del fabricante (atributo obligatorio), fecha de entrega (si es posible, aunque puede que no aparezca, si aparece el día es optativo, pero el mes y el año son obligatorios). También se necesita saber del componente si es frágil o no. También debe aparecer un elemento peso del componente y dicho elemento peso tiene un atributo unidad del peso (kilos o gramos), un elemento número de serie y puede que aparezca o no un elemento km_maximos indicando que el componente debe sustituirse tras un cierto número de kilómetros.

Una posible solución sería esta (aunque se puede mejorar en algunos aspectos):

```
<!ELEMENT pedido (tractor+)>
<!ELEMENT tractor (componente+)>
<!ELEMENT componente (fecha?, peso, fragil?, km_maximos?)>
<!ELEMENT fragil EMPTY>
<!ELEMENT peso (#PCDATA)>
<!ELEMENT km_maximos (#PCDATA)>
<!ATTLIST componente nombre_fabricante CDATA #REQUIRED>
<!ATTLIST peso unidades CDATA #IMPLIED>
<!ELEMENT fecha (dia?, mes, anio)>
<!ELEMENT dia (#PCDATA)>
<!ELEMENT mes (#PCDATA)>
<!ELEMENT anio (#PCDATA)>
```

Un ejemplo de XML sería este:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pedido SYSTEM "tractores.dtd">
<pedido>
  <tractor>
    <componente nombre_fabricante="John Deere">
      <peso unidades="kilos">2.5</peso>
      <fragil/>
    </componente>
    <componente nombre_fabricante="Agrotrans">
      <peso unidades="gramos">50</peso>
    </componente>
  </tractor>
</pedido>
```



```

    <componente nombre_fabricante="John Deere">
      <fecha>
        <mes>Enero</mes>
        <anio>2015</anio>
      </fecha>
      <peso>150</peso>
      <fragil/>
    </componente>
    <componente nombre_fabricante="Agrotrans">
      <peso unidades="gramos">50</peso>
    </componente>
  </tractor>
</pedido>

```

5.14 Ejercicio: repeticiones de opciones

Se necesita un formato de archivo para intercambiar productos entre almacenes de productos de librería y se desea una DTD que incluya estas restricciones:

- Debe haber un elemento raíz pedido que puede constar de libros, cuadernos y/o lápices. Los tres elementos pueden aparecer repetidos y en cualquier orden. También pueden aparecer por ejemplo 4 libros, 2 lápices y luego 4 lápices de nuevo.
- Todo libro tiene un atributo obligatorio título.
- Los elementos cuaderno tienen un atributo optativo num_hojas.
- Todo elemento lápiz debe tener dentro un elemento obligatorio número.

La solución a la DTD:

```

<!ELEMENT pedido (libro|cuaderno|lapis)+>
<!ELEMENT libro (#PCDATA)>
<!ATTLIST libro titulo CDATA #REQUIRED>
<!ELEMENT cuaderno (#PCDATA)>
<!ATTLIST cuaderno num_hojas CDATA #IMPLIED>
<!ELEMENT lapis (numero)>
<!ELEMENT numero (#PCDATA)>

```

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pedido SYSTEM "libreria.dtd">
<pedido>
  <libro titulo="Java 8"></libro>
  <cuaderno></cuaderno>
  <libro titulo="HTML y CSS"/>
  <libro titulo="SQL para Dummies"/>
  <cuaderno num_hojas="150"/>
  <lapis>
    <numero>2H</numero>
  </lapis>
  <cuaderno num_hojas="250"/>
  <cuaderno num_hojas="100"/>
  <lapis>
    <numero>2B</numero>
  </lapis>
  <lapis>
    <numero>1HB</numero>
  </lapis>
</pedido>

```

```
</lapiz>
</pedido>
```

5.15 Ejercicio

Una multinacional que opera en bolsa necesita un formato de intercambio de datos para que sus programas intercambien información sobre los mercados de acciones.

En general todo archivo constará de un listado de cosas como se detalla a continuación

- En el listado aparecen siempre uno o varios futuros, después una o varias divisas, después uno o varios bonos y una o varias letras.
- Todos ellos tienen un atributo precio que es **obligatorio**
- Todos ellos tienen un elemento vacío que indica de donde es el producto anterior: “Madrid”, “Nueva York”, “Frankfurt” o “Tokio”.
- Las divisas y los bonos tienen un atributo optativo que se usa para indicar si el producto ha sido estable en el pasado o no.
- Un futuro es un valor esperado que tendrá un cierto producto en el futuro. Se debe incluir este producto en forma de elemento. También puede aparecer un elemento mercado que indique el país de procedencia del producto.
- Todo bono tiene un elemento país_de_procedencia para saber a qué estado pertenece. Debe tener tres elementos extra llamados “valor_deseado”, “valor_mínimo” y “valor_máximo” para saber los posibles precios.
- Las divisas tienen siempre un nombre pueden incluir uno o más tipos de cambio para otras monedas.
- Las letras tienen siempre un tipo de interés pagadero por un país emisor. El país emisor también debe existir y debe ser siempre de uno de los países cuyas capitales aparecen arriba (es decir “España”, “EEUU”, “Alemania” y “Japón”)

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE listado [
  <!ELEMENT listado (futuro+, divisa+, bono+, letra+)>
  <!ATTLIST futuro precio CDATA #REQUIRED>
  <!ATTLIST divisa precio CDATA #REQUIRED>
  <!ATTLIST bono precio CDATA #REQUIRED>
  <!ATTLIST letra precio CDATA #REQUIRED>
  <!ELEMENT ciudad_procedencia (madrid|nyork|frankfurt|tokio)>
  <!ELEMENT madrid EMPTY>
  <!ELEMENT nyork EMPTY>
  <!ELEMENT frankfurt EMPTY>
  <!ELEMENT tokió EMPTY>
  <!ATTLIST divisa estable CDATA #IMPLIED>
  <!ATTLIST bono estable CDATA #IMPLIED>
  <!ELEMENT futuro (producto, mercado?, ciudad_procedencia)>
  <!ELEMENT producto (#PCDATA)>
  <!ELEMENT mercado (#PCDATA)>
  <!ELEMENT bono (pais_de_procedencia, valor_deseado,
                 valor_minimo, valor_maximo, ciudad_procedencia)>
  <!ELEMENT valor_deseado (#PCDATA)>
  <!ELEMENT valor_minimo (#PCDATA)>
  <!ELEMENT valor_maximo (#PCDATA)>
  <!ELEMENT pais_de_procedencia (#PCDATA)>
  <!ELEMENT divisa (nombre_divisa,
                   tipo_de_cambio+, ciudad_procedencia)>
```

```

<!ELEMENT nombre_divisa (#PCDATA)>
<!ELEMENT tipo_de_cambio (#PCDATA)>
<!ELEMENT letra (tipo_de_interes, pais_emisor, ciudad_procedencia)>
<!ELEMENT tipo_de_interes (#PCDATA)>
<!ELEMENT pais_emisor (espania|eeuu|alemania|japon)>
<!ELEMENT espania EMPTY>
<!ELEMENT eeuu EMPTY>
<!ELEMENT alemania EMPTY>
<!ELEMENT japon EMPTY>
]>

<listado>
  <futuro precio="11.28">
    <producto>Cafe</producto>
    <mercado>América Latina</mercado>
    <ciudad_procedencia>
      <frankfurt/>
    </ciudad_procedencia>
  </futuro>
  <divisa precio="183">
    <nombre_divisa>Libra esterlina</nombre_divisa>
    <tipo_de_cambio>2.7:1 euros</tipo_de_cambio>
    <tipo_de_cambio>1:0.87 dólares</tipo_de_cambio>
    <ciudad_procedencia>
      <madrid/>
    </ciudad_procedencia>
  </divisa>
  <bono precio="10000" estable="si">
    <pais_de_procedencia>
      Islandia
    </pais_de_procedencia>
    <valor_deseado>9980</valor_deseado>
    <valor_minimo>9950</valor_minimo>
    <valor_maximo>10020</valor_maximo>
    <ciudad_procedencia>
      <tokio/>
    </ciudad_procedencia>
  </bono>
  <letra precio="45020">
    <tipo_de_interes>4.54%</tipo_de_interes>
    <pais_emisor>
      <espania/>
    </pais_emisor>
    <ciudad_procedencia>
      <madrid/>
    </ciudad_procedencia>
  </letra>
</listado>

```

5.16 Ejercicio

La Seguridad Social necesita un formato de intercambio unificado para distribuir la información personal de los afiliados.

- Todo archivo XML contiene un listado de uno o mas afiliados

- Todo afiliado tiene los siguientes elementos:
 - DNI o NIE
 - Nombre
 - Apellidos
 - Situación laboral: que tiene que ser una y solo una de entre estas posibilidades: “en_paro”, “en_activo”, “jubilado”, “edad_no_laboral”
 - Fecha de nacimiento: que se desglosa en los elementos obligatorios día, mes y año.
 - Listado de bajas: que indica las situaciones de baja laboral del empleado. Dicho listado consta de una repetición de 0 o más bajas:
 - Una baja consta de tres elementos: causa (obligatoria), fecha de inicio (obligatorio) y fecha de final (optativa),
 - Listado de prestaciones cobradas: consta de 0 o más elementos prestación, donde se indicará la cantidad percibida (obligatorio), la fecha de inicio (obligatorio) y la fecha de final (obligatorio)

5.17 Examen

El examen de este tema tendrá lugar el viernes 7 de marzo de 2014.

Recuperación de información

6.1 Introducción

En líneas generales hay dos grandes formas de usar un lenguaje de programación para leer o escribir archivos XML

- DOM: significa Document Object Model (o Modelo del objeto documento). DOM en general almacena los archivos en memoria lo que es mucho más rápido y eficiente.
- SAX: en algunos casos, los archivos muy grandes, pueden no caber en memoria. SAX proporciona otras clases y métodos distintos para ir procesando un archivo por partes. SAX significa Simple Access for XML, pero en general es un poco más complicado. En este módulo, no lo veremos.

DOM es un estándar y sus clases y métodos existen en muchos otros lenguajes.

6.2 Fundamentos de DOM con Java

En primer lugar va a ser necesario importar las clases correctas para poder usar DOM. La línea correcta es

```
import javax.xml.parsers.*;
import org.w3c.dom.*;
```

Un parser es un programa que analiza la sintaxis de un fichero, en nuestro caso un fichero XML. En castellano se debería decir analizador o analizador gramatical.

6.3 Ejemplo de base

```
package com.ies;
import javax.xml.parsers.*;
import java.io.File;
import org.w3c.dom.*;

public class ProcesadorXML {
    public void procesarArchivo(String nombreArchivo) {
        DocumentBuilderFactory fabrica;
        DocumentBuilder constructor;
        Document documentoXML;
        File fichero=new File(nombreArchivo);
        fabrica=
            DocumentBuilderFactory.newInstance();
```

```
        System.out.println("Procesando "+nombreArchivo);
        try {
            constructor=
                fabrica.newDocumentBuilder();
            documentoXML=constructor.parse(fichero);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

    public static void main (String[] argumentos){
        System.out.println("Probando...");
        ProcesadorXML proc=new ProcesadorXML();
        proc.procesarArchivo("bolsas.xml");
    }
}
```

6.4 La clase Document

La clase Document es una representación Java que almacena en memoria un archivo XML. Mediante esta clase y otras clases compañeras podremos recorrer cualquier punto del archivo XML.

Este recorrido se basa siempre en la visita de nodos hijo o nodos hermano. No todos los nodos son iguales y se debe tener presente que en un nodo podríamos encontrar que los saltos de línea pueden ser un problema a la hora de recorrer el árbol DOM.

Por ejemplo, dado un documento, se debe empezar obteniendo la raíz. Este elemento se llama también el “elemento documento” y podemos obtenerlo así:

```
documento=constructor.parse(archivoXML);
Element raiz=documento.getDocumentElement();
System.out.println(raiz.getNodeName());
```

La clase principal que nos interesa es la clase Node, siendo Document su clase Hija. Algunos métodos de interés son estos:

- `getDocumentElement()` obtiene el elemento raíz, a partir del cual podremos empezar a “navegar” a través de los elementos.
- `getFirstChild()` obtiene el primer elemento hijo del nodo que estemos visitando.
- `getParentNode()` nos permite obtener el nodo padre de un cierto nodo.
- `getChildNodes()` obtiene todos los nodos hijo.
- `getNextSibling()` obtiene el siguiente nodo hermano.
- `getChildNodes()` devuelve un `NodeList` con todos los hijos de un elemento. Esta `NodeList` se puede recorrer con un `for`, obteniendo el tamaño de la lista con `getLength()` y extrayendo los elementos con el método `item(posicion)`
- `getNodeType()` es un método que nos indica el tipo de nodo (devuelve un `short`). Podemos compararlo con `Node.ELEMENT_NODE` para ver si el nodo es realmente un elemento.

- Otro método de utilidad es `getElementsByTagName` que extrae todos los subelementos que tengan un cierto nombre de etiqueta.

Consejo: En general, hay muchas clases que proporcionan más métodos de utilidad, como por ejemplo la clase `Element`. En muchas ocasiones, podremos hacer un `cast` y aprovecharnos de ellos.

Cuando se procesan archivos, se debe tener especial importancia a los espacios en blanco que pueda haber. Estos dos archivos no son iguales:

```
El primer hijo de listado es <futuro>
<listado><futuro>...</futuro></listado>
```

```
Aquí el primer hijo de listado es \n
<listado>
    <futuro>...</futuro>
</listado>
```

6.5 Ejercicios

6.5.1 Ejercicio

Dado el siguiente archivo XML crear un programa que muestre todos los nombres:

```
<listaempleados>
  <empleado edad="27">
    <nombre>Pepe Perez</nombre>
    <categoria>Empleado</categoria>
  </empleado>
  <empleado edad="34">
    <nombre>Juan Sanchez</nombre>
    <categoria>Gerente</categoria>
  </empleado>
</listaempleados>
```

La solución podría ser algo así:

```
public class ProcesadorXML {
    String ruta;
    public ProcesadorXML(String ruta){
        this.ruta=ruta;
    }
    public Element getRaiz()
        throws ParserConfigurationException, SAXException, IOException
    {
        DocumentBuilderFactory fabrica;
        fabrica=
        DocumentBuilderFactory.newInstance();
        /* A partir de un fichero XML
         * crea el objeto documento en memoria*/
        DocumentBuilder creadorObjDocumento;
        creadorObjDocumento=
            fabrica.newDocumentBuilder();
        FileInputStream fich;
```

```

        fich=new FileInputStream(this.ruta);

        /* Analiza el XML y
         * lo carga en memoria */
        Document documento;
        documento=
            creadorObjDocumento.parse(fich);
        Element raiz;
        raiz=documento.getDocumentElement();
        return raiz;
    }
    /* Este método imprime todos los nombres*/
    public void todosNombres()
        throws ParserConfigurationException,
            SAXException, IOException
    {
        Element raiz=getRaiz();
        Node hijo=raiz.getFirstChild();
        while (hijo!=null){
            String nombreElemento;
            nombreElemento=hijo.getNodeName();
            if (nombreElemento.equals("empleado")){
                Node hijoFinLinea=hijo.getFirstChild();
                Element hijoNombre=(Element) hijoFinLinea.
↳getNextSibling();

                String contenido=hijoNombre.getTextContent();
                System.out.println("Empleado "+contenido);
            }
            hijo=hijo.getNextSibling();
        }
    }
    public static void main(String[] args) throws ParserConfigurationException,
↳SAXException, IOException {
        ProcesadorXML procesador;
        procesador=new ProcesadorXML(
            "D:/oscar/empleados.xml");
        procesador.todosNombres();
    }
}

```

Ampliaciones:

- Añadir uno que devuelva todas las edades.
- Añadir uno que devuelva los nombres de los empleados mayores de 30 (mostrando los nombres pero no las edades).
- Añadir un método que diga cuantos empleados hay. El método debe ser capaz de tolerar que haya muchas líneas en blanco seguidas.

El siguiente código resuelve el problema de mostrar los mayores de cierta edad:

```

public void mostrarMayoresDe(int edadMinima)
    throws ParserConfigurationException,
        SAXException, IOException
{
    Element raiz=getRaiz();
    Node finLinea=raiz.getFirstChild();
    Element empleado=(Element) finLinea.getNextSibling();
}

```



```

while (empleado!=null){
    String edad=empleado.getAttribute("edad");
    int iEdad=Integer.parseInt(edad);
    if (iEdad>edadMinima){
        finLinea=empleado.getFirstChild();
        Element elemNombre=(Element)
            finLinea.getNextSibling();
        String nombreEmpleado=
            elemNombre.getTextContent();
        System.out.println(nombreEmpleado +
            " es mayor de "+iEdad);
    } //Fin del if
    finLinea=empleado.getNextSibling();
    empleado=(Element) finLinea.getNextSibling();
} //Fin del while
} //Fin del método

```

El método `getElementsByTagName` puede facilitar mucho el resolver ciertas tareas. Por ejemplo, supongamos que queremos resolver el problema de contar cuantos empleados hay:

```

public int contarEmpleados()
    throws ParserConfigurationException, SAXException, IOException{
    int numEmpleados=0;
    Element raiz=getRaiz();
    NodeList lista=raiz.getElementsByTagName("empleado");
    numEmpleados=lista.getLength();
    return numEmpleados;
}

```

6.5.2 Ejercicio

Extraer la raíz de un archivo XML

```

public Node extraerRaiz(String nombreArchivo){
    DocumentBuilderFactory fabrica;
    DocumentBuilder constructor;
    Document documentoXML=null;
    File fichero=new File(nombreArchivo);
    fabrica=
        DocumentBuilderFactory.newInstance();
    System.out.println("Procesando "+nombreArchivo);
    try {
        constructor=
            fabrica.newDocumentBuilder();
        documentoXML=constructor.parse(fichero);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return documentoXML.getDocumentElement();
}

```

6.5.3 Ejercicio

Imprimir todos los elementos hijo del archivo XML.

Una posibilidad es la siguiente:

- Usar `getNextSibling` para ir recorriendo hermano a hermano hasta que se encuentre un `null`
- Para evitar los nodos texto, solo imprimiremos cosas cuando el `getNodeTipo()` nos devuelva un tipo `Node.ELEMENT_NODE`

```
public void imprimirHijos(Node nodoRaiz){
    if (nodoRaiz==null){
        System.out.println("Imposible procesar raiz null");
        return ;
    }
    Node nodo=nodoRaiz.getFirstChild();
    while (nodo!=null){
        short tipo=nodo.getNodeTipo();
        if (tipo==nodo.ELEMENT_NODE){
            System.out.println("Nodo hijo:"+nodo.getNodeName());
        }
        nodo=nodo.getNextSibling();
    }
}
```

Otra posibilidad sería usar el `getChildNodes` que nos devuelve un vector con todos los hijos. Sin embargo ocurrirá lo mismo, deberemos evitar el visitar nodos texto, solo nos interesan los nodos Elemento.

```
public void imprimirHijos2(Node nodoRaiz){
    if (nodoRaiz==null){
        System.out.println("Imposible procesar raíz nula");
        return;
    } //Fin del if null
    NodeList lista=nodoRaiz.getChildNodes();
    for (int i=0; i<lista.getLength();i++){
        Node nodo=lista.item(i);
        short tipo=nodo.getNodeTipo();
        if (tipo==Node.ELEMENT_NODE){
            System.out.println("Hijo:"+nodo.getNodeName());
        } //Fin del if
    } //Fin del for
} //Fin del método
```

6.5.4 Ejercicio

Ampliar el programa para que nos diga cuantos elementos “divisa” hay en el archivo.

Para practicar esto y de paso practicar programación genérica, fabricaremos un método al que le pasaremos el nombre del elemento a buscar y el método nos dirá cuantos elementos con ese nombre hay.

```
public int contadorElementos(Node raiz,String nombreElemento){
    int contador=0;
    NodeList nodosHijo=raiz.getChildNodes();
    for (int i=0; i<nodosHijo.getLength();i++){
        Node nodo=nodosHijo.item(i);
        short tipo=nodo.getNodeTipo();
        if (tipo==Node.ELEMENT_NODE){
```

```

        String nombre=nodo.getNodeName();
        if (nombre==nombreElemento){
            contador++;
        } //Fin del if interno
    } //Fin del if externo
} //Fin del for
return contador;
} //Fin del método

```

6.5.5 Ejercicio

Nos interesa conocer el precio de todos los bonos. Crear un programa que ejecute esta tarea.

```

private void comprobarSiEsBono(Node n){
    String nombre=n.getNodeName();
    if (nombre=="bono"){
        System.out.println("Encontrado un bono");
    }
}
public void imprimirPrecioBonos(Node raiz){
    if (raiz==null){
        System.out.println("Imposible procesar null");
        return;
    }
    NodeList nodos=raiz.getChildNodes();
    for (int i=0; i<nodos.getLength(); i++){
        Node nodo=nodos.item(i);
        short tipo=nodo.getNodeType();
        if (tipo==Node.ELEMENT_NODE){
            this.comprobarSiEsBono(nodo);
        }
    }
}

```

6.5.6 Ejercicio

Crear un programa que nos diga cuantos productos financieros del listado no son estables. Es decir, que tengan el atributo estable y lo tengan a false.

En su momento, en la DTD se permitió que el atributo estable fuera #IMPLIED, es decir **optativo**. Al ser la DTD como un contrato, esto nos obliga a preparar nuestro código para manejar la posibilidad de que el atributo no esté presente.

```

public int cuantosInestables (Node raiz){
    int cuantos=0;
    NodeList lista=raiz.getChildNodes();
    for (int i=0; i<lista.getLength(); i++){
        Node n=lista.item(i);
        if (n.getNodeType()!=Node.ELEMENT_NODE) continue;
        Element e=(Element) lista.item(i);
        if (e.getNodeName()=="divisa" ||
            e.getNodeName()=="bono"){
            String atEstable=e.getAttribute("estable");
            if (atEstable!=null){
                System.out.println("Atributo:"+atEstable);
            }
        }
    }
    return cuantos;
}

```

```
        if (atEstable.equals("no")) {
            cuantos+=1;
        } //Fin del if interno
    } //Fin del if atEstable
} //Fin de if nodo es divisa o bono
} //Fin del for que recorre los nodos
return cuantos;
} //Fin del método cuantosInestables
```

6.5.7 Ejercicio

Sumar los precios de todos los productos financieros.

```
public float sumarAtributosPrecio(Node raiz){
    float precioTotal=0;
    NodeList hijos=raiz.getChildNodes();
    for (int i=0; i<hijos.getLength(); i++){
        Node hijo=hijos.item(i);
        if (hijo.getNodeType() !=Node.ELEMENT_NODE) continue;
        Element e=(Element) hijo;
        String precio=e.getAttribute("precio");
        Float f=Float.parseFloat(precio);
        precioTotal+=f;
    }
    return precioTotal;
} //Fin del método sumarAtributosPrecio
```

6.5.8 Ejercicio

Contar cuantos productos financieros tienen algo que ver con el país “Islandia”

Se deben tener presentes varias cosas:

- Si no se tiene claro lo que nos piden, preguntar.
- En cualquier caso, si se tiene DTD, hay una buena pista.
 - Aparece un elemento llamado <pais_de_procedencia>, que puede contener cualquier cosa (incluido Islandia)
 - La ciudad de procedencia no incluye la capital o ninguna ciudad de dicho país, así que podemos ignorar eso.
 - También aparece un elemento llamado <pais_emisor>, pero tampoco incluye Islandia, en principio también podemos saltarlo.

Análisis del problema

Después de haber examinado la DTD se llega a la conclusión de que el único elemento que puede transportar alguna clase de información relacionada con “Islandia” es el nodo `pais_de_procedencia`, que es un elemento hijo del elemento `bono`.

Solución

- La clase `Element` tiene un método llamado `getElementsByTagName` que nos permite recuperar de una sola vez todos los elementos con el nombre `bono`.
- Se debe tener en cuenta que para llegar al elemento que nos interesa podemos seguir usando los métodos `getFirstChild` o `getNextSibling` para ir al primer hijo o para ir al siguiente hermano.
- El contenido textual de un nodo se puede extraer con `getTextContent`
- Al procesar un contenido textual, podríamos encontrar muchos espacios en blanco, tabuladores u otros elementos que alteren las comparaciones entre cadenas, por lo que deberemos usar métodos como `trim()` que limpian los espacios en blanco.

```
public int algoQueVerCon(Node raiz, String nombrePais){
    int cuantos=0;
    Element elementoRaiz=(Element) raiz;
    NodeList lista=elementoRaiz.getElementsByTagName("bono");
    for (int i=0; i<lista.getLength(); i++){
        Node nodoBono=lista.item(i);
        Node primerHijoTexto=nodoBono.getFirstChild();
        Node segHijoPais=primerHijoTexto.getNextSibling();
        String paisExtraido=segHijoPais.getTextContent();
        //Limpiamos espacios
        paisExtraido=paisExtraido.trim();
        System.out.println("Pais extraido:"+paisExtraido);
        if (paisExtraido.equals(nombrePais)){
            cuantos++;
        }
    }
    return cuantos;
}
```

6.5.9 Ejercicio

Se desea crear un método que indique cuantos elementos tienen relación de alguna forma con “España”.

Análisis

- Se dispone del método anterior `algoQueVerCon` que nos permite contabilizar cuantos bonos tienen el país “España”.
- Al analizar la DTD, se ha encontrado que la ciudad de procedencia de un elemento `futuro` puede ser Madrid.
- Al analizar la DTD también se ha encontrado que elemento `pais_emisor` de un elemento `letra` puede ser `espania`
- Al analizar las divisas se debe comprobar si el elemento `ciudad_procedencia` es el elemento `madrid`

Diseño

Crearemos dos métodos extra, uno para calcular la solución para el segundo punto (ver cuantos elementos `futuro` tienen como `ciudad_procedencia` el valor Madrid y otro método para el tercer punto.

```

public int cuantosFuturosTienenCiudadProcedencia(
    Node raiz, String ciudad)
{
    int cuantos=0;
    Element nodoRaiz=(Element) raiz;
    NodeList lista=nodoRaiz.getElementsByTagName("futuro");
    for (int i=0; i<lista.getLength(); i++){
        Element e=(Element)lista.item(i);
        NodeList listaHijos=e.getChildNodes();
        //El elemento ciudad procedencia es el quinto hijo
        Node nodoCiudad=listaHijos.item(5);
        NodeList hijosCiudad=nodoCiudad.getChildNodes();
        Node nodoElemCiudad=hijosCiudad.item(1);
        String nombreCiudad=nodoElemCiudad.getNodeName();
        if (nombreCiudad.equals(ciudad)){
            cuantos++;
        } //Fin del if
    } //Fin del for
    return cuantos;
}

```

Para resolver el último punto nos bastaría un método como este:

```

public int letrasConPaisEmisor(Node raiz, String nombrePais){
    int cuantos=0;
    Element eRaiz=(Element) raiz;
    NodeList listaLetras=eRaiz.getElementsByTagName("letra");
    for (int i=0; i<listaLetras.getLength(); i++){
        Node nodo=listaLetras.item(i);
        Element eNodo=(Element) nodo; //Devuelve elemento letra
        NodeList hijosLetra=eNodo.getChildNodes();
        Node nodoPaisEmisor=hijosLetra.item(3);
        NodeList hijosPais=nodoPaisEmisor.getChildNodes();
        Node nodoPais=hijosPais.item(1);
        String nombreNodoPais=nodoPais.getNodeName();
        if (nombreNodoPais.equals(nombrePais)){
            cuantos++;
        }
    } //Fin del for
    return cuantos;
}

```

Ahora el método que resuelve este ejercicio es tan simple como esto:

```

public int algoQueVerConEspania(Node raiz){
    int cuantasLetras=this.letrasConPaisEmisor(raiz, "espania");
    int cuantosFuturos=this.cuantosFuturosTienenCiudadProcedencia(raiz,
        "madrid");
    int cuantosBonos=this.algoQueVerCon(raiz, "España");
    return cuantasLetras+cuantosFuturos+cuantosBonos;
}

```

6.5.10 Ejercicio

Crear un programa que indique el país de procedencia de todos aquellos bonos en los que el precio deseado tenga un valor comprendido entre el precio mínimo y el máximo.

Una posible solución sería esta:

```
public void imprimirBonos(Node raiz){

    Element eRaiz=(Element) raiz;
    NodeList listaBonos=eRaiz.getElementsByTagName("bono");
    for (int i=0; i<listaBonos.getLength(); i++){
        Node bono=listaBonos.item(i);
        Element eBono=(Element) bono;
        // Element eBono=(Element) listaBonos.item(i);
        NodeList listaParaValorDeseado=
            eBono.getElementsByTagName("valor_deseado");
        NodeList listaParaValorMinimo=
            eBono.getElementsByTagName("valor_minimo");
        NodeList listaParaValorMaximo=
            eBono.getElementsByTagName("valor_maximo");
        Node nodoValorDeseado=listaParaValorDeseado.item(0);
        Node nodoValorMinimo=listaParaValorMinimo.item(0);
        Node nodoValorMaximo=listaParaValorMaximo.item(0);

        Element eValorDeseado=(Element) nodoValorDeseado;
        Element eValorMinimo=(Element) nodoValorMinimo;
        Element eValorMaximo=(Element) nodoValorMaximo;

        String cadValorDeseado=eValorDeseado.getTextContent();
        String cadValorMinimo=eValorMinimo.getTextContent();
        String cadValorMaximo=eValorMaximo.getTextContent();

        int valorDeseado=Integer.parseInt(cadValorDeseado);
        int valorMinimo=Integer.parseInt(cadValorMinimo);
        int valorMaximo=Integer.parseInt(cadValorMaximo);

        if ((valorDeseado>valorMinimo) &&
            (valorDeseado<valorMaximo) ){
            System.out.println("Encontrado un bono!");
        }
        //Element eValorDeseado=(Element)
        //        listaParaValorDeseado.item(0);

    }

}
```

Una solución mejor sería esta:

```
public int extraerHijoNumero (Element padre,
    String nombreHijo){
    int valor=0;
    //Esta lista tiene solo un elemento
    NodeList listaHijos=
        padre.getElementsByTagName(nombreHijo);
    Element hijoNumerico=(Element) listaHijos.item(0);
    String contenidoTextual=hijoNumerico.getTextContent();
    valor=Integer.parseInt(contenidoTextual);
    return valor;
}

public void imprimirBonos(Node raiz){

    Element eRaiz=(Element) raiz;
```

```

NodeList listaBonos=eRaiz.getElementsByTagName("bono");
for (int i=0; i<listaBonos.getLength(); i++){
    Node bono=listaBonos.item(i);
    Element eBono=(Element) listaBonos.item(i);

    int valorDeseado=this.extraerHijoNumero(
        eBono, "valor_deseado");
    int valorMinimo=this.extraerHijoNumero(
        eBono, "valor_minimo");
    int valorMaximo=this.extraerHijoNumero(
        eBono, "valor_maximo");

    if ((valorDeseado>valorMinimo) &&
        (valorDeseado<valorMaximo) ){
        System.out.println("Encontrado un bono!");
    }
    //Element eValorDeseado=(Element)
    //    listaParaValorDeseado.item(0);
}
}

```

6.5.11 Ejercicio

Imprimir, los productos financieros con la misma ciudad de procedencia.

Análisis

En general, todos los problemas donde nos piden algo como *comprobar todos los elementos que tengan las mismas características* implican hacer una comprobación de *todos con todos*.

Diseño

Todos los elementos tienen un elemento `ciudad_de_procedencia`, por lo cual, probablemente sea útil crear algún pequeño método de utilidad que dado un elemento nos devuelva un string con la ciudad de procedencia.

Por otro lado, comparar *todos con todos* suele implicar un doble bucle, donde el primer irá extrayendo elementos y el otro irá extrayendo todos los demás.

Implementación

```

public String getCiudadProcedencia(Element e){
    NodeList listaHijos=
        e.getElementsByTagName("ciudad_procedencia");
    Element eCiudad=(Element) listaHijos.item(0);
    NodeList listaHijosCiudad=eCiudad.getChildNodes();
    Element eCiudadConcreto=
        (Element) listaHijosCiudad.item(1);
    String nombre=eCiudadConcreto.getNodeName();
    return nombre;
}

```



```

public void imprimirMismaCiudad(Node raiz){
    NodeList hijos=raiz.getChildNodes();
    for (int i=0; i<hijos.getLength(); i++){
        Node hijo=hijos.item(i);
        if (hijo.getNodeType() !=Node.ELEMENT_NODE){
            continue;
        }
        for (int j=0; j<hijos.getLength(); j++){
            Node otroHijo=hijos.item(j);
            if (otroHijo.getNodeType() !=Node.ELEMENT_NODE){
                continue;
            }
            String ciudadHijo=
                this.getCiudadProcedencia((Element)hijo);
            String ciudadOtro=
                this.getCiudadProcedencia((Element)otroHijo);
            if (ciudadHijo.equals(ciudadOtro)){
                System.out.println(
                    "Encontré dos elementos con la ciudad
↪"+ciudadHijo);

                } //Fin del if ciudadHijo
            } //Fin del for interno
        } //Fin del for externo
    } //Fin del método

public void imprimirMismaCiudad(Node raiz){
    NodeList hijos=raiz.getChildNodes();
    for (int i=0; i<hijos.getLength(); i++){
        Node hijo=hijos.item(i);
        if (hijo.getNodeType() !=Node.ELEMENT_NODE){
            continue;
        }
        String ciudadHijo=
            this.getCiudadProcedencia((Element)hijo);
        for (int j=i+1; j<hijos.getLength(); j++){
            Node otroHijo=hijos.item(j);
            if (otroHijo.getNodeType() !=Node.ELEMENT_NODE){
                continue;
            }

            String ciudadOtro=
                this.getCiudadProcedencia((Element)otroHijo);
            if (ciudadHijo.equals(ciudadOtro)){
                System.out.println(
                    "Encontré dos elementos con la ciudad
↪"+ciudadHijo);

                } //Fin del if ciudadHijo
            } //Fin del for interno
        } //Fin del for externo
    } //Fin del método

```

6.5.12 Ejercicio

Contar cuantos productos que no sean estables tienen como ciudad de procedencia Tokio. Si hay más de 2, devolver los precios en un vector y si no devolver un vector vacío.

Análisis

El atributo `estable`, solo lo tienen los productos `bono`. Ese atributo es optativo, puede que esté o puede que no. Si existe, debemos comprobar si tiene un “no”.

Por otro lado, no sabemos a priori si habrá más de 2 o no.

- Podríamos hacer la cuenta, y si da más de 2 repetir operaciones y meter los bonos correctos en un vector.
- Podríamos ir haciendo las operaciones y a la vez las inserciones en un vector y ahorrarnos operaciones.

Optaremos por la segunda.

Diseño

- Aprovecharemos los métodos que nos permiten extraer el elemento raíz de un fichero.
- Necesitaremos crear vectores que tengan un cierto tamaño. Crearemos vectores muy grandes y los dejaremos con la inicialización que hace Java por defecto llenándolo con valores `null`.
- Podemos aprovechar métodos ofrecidos por Java como `getElementsByTagName`.
- Después recorreremos los elementos, comprobaremos si sus atributos cumplen las condiciones y si las cumplen almacenaremos en el vector a devolver la ciudad de ese bono.
- Nuestro método devolverá siempre algo como `String[]`, ese vector puede que vaya lleno o no.

Solución 1

```
public class ProcesadorXML {
    public Element getRaiz(String nombreFichero)
        throws ParserConfigurationException, SAXException, IOException
    {
        DocumentBuilderFactory
            fabrica = DocumentBuilderFactory.newInstance();
        DocumentBuilder constructor=
            fabrica.newDocumentBuilder();
        FileInputStream fichero=
            new FileInputStream(nombreFichero);
        Document documento=
            constructor.parse(fichero);
        Element raiz=documento.getDocumentElement();
        return raiz;
    }
    public int[] getPreciosInestables()
        throws ParserConfigurationException, SAXException, IOException
    {
        int[] vPrecios=null;
        int contador=0;
        Element raiz=
            getRaiz("d:/oscar/productos.xml");
        Element hijo=(Element)
```

```

        raiz.getFirstChild();
        /* Mientras le queden hijos a la raíz...*/
        while (hijo!=null){
            String atrEstable=
                hijo.getAttribute("estable");
            if (atrEstable.equals("no")){
                NodeList vector=
                    hijo.getElementsByTagName(
                        "tokio");
                if (vector.getLength()>0){
                    //El producto sí es de Tokio
                    String precio=
                        hijo.getAttribute(
                            "precio");
                    System.out.println("Precio:"+precio);
                }
            }
            Node finLinea=hijo.getNextSibling();
            hijo=(Element)
                finLinea.getNextSibling();
        }
        return vPrecios;
    }

    public static void main(String[] args)
        throws ParserConfigurationException, SAXException,
↳IOException {
        ProcesadorXML procesador=
            new ProcesadorXML();
        int[] precios=
            procesador.getPreciosInestables();
    }
}

```

Solución 2

```

public String[] obtenerListaBonos(Node raiz){
    int tamañoMaximo=1000;
    String[] ciudades=new String[tamañoMaximo];
    String[] aux=new String[tamañoMaximo];
    int posPrecio=0;
    Element eRaiz=(Element) raiz;
    NodeList listaBonos=eRaiz.getElementsByTagName("bono");
    for (int i=0; i<listaBonos.getLength(); i++){
        Element bono=(Element) listaBonos.item(i);
        String atEstable=bono.getAttribute("estable");
        if (atEstable!=null){
            if (atEstable.equals("no")){
                //Examinamos la ciudad aprovechando
                //un método ya construido.
                String ciudad=this.getCiudadProcedencia(bono);
                if (ciudad.equals("tokio")){
                    //Si es de tokió, copiamos el precio
                    String precio=bono.getAttribute("precio");
                    aux[posPrecio]=precio;
                    posPrecio++;
                }
            }
        }
    }
    return aux;
}

```

```
        } //Fin del if para tokiio
    } //Fin del if para el "no"
    } //Fin del if para atEstable
} //Fin del for que recorre los bonos
if (posPrecio>2){
    return aux;
}
return ciudades;
}
```

6.6 Anexo

6.6.1 Código Java

A continuación se muestra el código Java completo:

```
package com.ies;
import javax.xml.parsers.*;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import java.io.File;

import org.w3c.dom.*;

public class ProcesadorXML {
    public Node extraerRaiz(String nombreArchivo){
        DocumentBuilderFactory fabrica;
        DocumentBuilder constructor;
        Document documentoXML=null;
        File fichero=new File(nombreArchivo);
        fabrica=
            DocumentBuilderFactory.newInstance();
        System.out.println("Procesando "+nombreArchivo);
        try {
            constructor=
                fabrica.newDocumentBuilder();
            documentoXML=constructor.parse(fichero);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return documentoXML.getDocumentElement();
    }
    public void imprimirNombreDeLaRaiz(Node nodo){
        if (nodo!=null){
            String nombre=nodo.getNodeName();
            System.out.println("La raíz se llama:"+nombre);
            Node primerHijo=nodo.getFirstChild();
            String nombreHijo=primerHijo.getNodeName();
        }
    }
}
```

```

        System.out.println("El primer hijo se llama <"+nombreHijo+">
↪");
    } else {
        System.out.println("No se pudo leer la raíz por ser nula");
    }
}
public void imprimirHijos(Node nodoRaiz){
    if (nodoRaiz==null){
        System.out.println("Imposible procesar raíz null");
        return ;
    }
    Node nodo=nodoRaiz.getFirstChild();
    while (nodo!=null){
        short tipo=nodo.getNodeType();
        if (tipo==nodo.ELEMENT_NODE){
            System.out.println("Nodo hijo:"+nodo.getNodeName());
        }
        nodo=nodo.getNextSibling();
    }
}

public void imprimirHijos2(Node nodoRaiz){
    if (nodoRaiz==null){
        System.out.println("Imposible procesar raíz nula");
        return;
    } //Fin del if null
    NodeList lista=nodoRaiz.getChildNodes();
    for (int i=0; i<lista.getLength();i++){
        Node nodo=lista.item(i);
        short tipo=nodo.getNodeType();
        if (tipo==Node.ELEMENT_NODE){
            System.out.println("Hijo:"+nodo.getNodeName());
        } //Fin del if
    } //Fin del for
} //Fin del método

public int contadorElementos(Node raiz,String nombreElemento){
    int contador=0;
    NodeList nodosHijo=raiz.getChildNodes();
    for (int i=0; i<nodosHijo.getLength();i++){
        Node nodo=nodosHijo.item(i);
        short tipo=nodo.getNodeType();
        if (tipo==Node.ELEMENT_NODE){
            String nombre=nodo.getNodeName();
            if (nombre==nombreElemento){
                contador++;
            } //Fin del if interno
        } //Fin del if externo
    } //Fin del for
    return contador;
} //Fin del método

private void comprobarSiEsBono(Node n){
    String nombre=n.getNodeName();
    if (nombre=="bono"){
        Element e=(Element) n;
        String precio=e.getAttribute("precio");
        System.out.println("Precio:"+precio);
    }
}

```

```

    }

    public int cuantosInestables (Node raiz){
        int cuantos=0;
        NodeList lista=raiz.getChildNodes();
        for (int i=0; i<lista.getLength(); i++){
            Node n=lista.item(i);
            if (n.getNodeType() != Node.ELEMENT_NODE) continue;
            Element e=(Element) lista.item(i);
            if (e.getNodeName()=="divisa" ||
                e.getNodeName()=="bono"){
                String atEstable=e.getAttribute("estable");
                if (atEstable!=null){
                    System.out.println("Atributo:"+atEstable);
                    if (atEstable.equals("no")){
                        cuantos+=1;
                    } //Fin del if interno
                } //Fin del if atEstable
            } //Fin de if nodo es divisa o bono
        } //Fin del for que recorre los nodos
        return cuantos;
    } //Fin del método cuantosInestables

    public float sumarAtributosPrecio(Node raiz){
        float precioTotal=0;
        NodeList hijos=raiz.getChildNodes();
        for (int i=0; i<hijos.getLength(); i++){
            Node hijo=hijos.item(i);
            if (hijo.getNodeType() != Node.ELEMENT_NODE) continue;
            Element e=(Element) hijo;
            String precio=e.getAttribute("precio");
            Float f=Float.parseFloat(precio);
            precioTotal+=f;
        }
        return precioTotal;
    } //Fin del método sumarAtributosPrecio
    public void imprimirPrecioBonos(Node raiz){
        if (raiz==null){
            System.out.println("Imposible procesar null");
            return;
        }
        NodeList nodos=raiz.getChildNodes();
        for (int i=0; i<nodos.getLength(); i++){
            Node nodo=nodos.item(i);
            short tipo=nodo.getNodeType();
            if (tipo==Node.ELEMENT_NODE){
                this.comprobarSiEsBono(nodo);
            }
        }
    }

    /**
     *
     * @param raiz
     * @param nombrePais
     * @return Numero de elementos dentro del nodo en
     * los cuales aparece de alguna forma el país
     */
    public int algoQueVerCon(Node raiz, String nombrePais){

```

```

        int cuantos=0;
        Element elementoRaiz=(Element) raiz;
        NodeList lista=elementoRaiz.getElementsByTagName("bono");
        for (int i=0; i<lista.getLength(); i++){
            Node nodoBono=lista.item(i);
            Node primerHijoTexto=nodoBono.getFirstChild();
            Node segHijoPais=primerHijoTexto.getNextSibling();
            String paisExtraido=segHijoPais.getTextContent();
            //Limpiamos espacios
            paisExtraido=paisExtraido.trim();
            System.out.println("Pais extraido:"+paisExtraido);
            if (paisExtraido.equals(nombrePais)){
                cuantos++;
            }
        }
        return cuantos;
    }

    /**
     * Este método averigua cuantos elementos futuro
     * tienen una cierta ciudad procedencia
     * @param argumentos
     */
    public int cuantosFuturosTienenCiudadProcedencia(
        Node raiz, String ciudad
    )
    {
        int cuantos=0;
        Element nodoRaiz=(Element) raiz;
        NodeList lista=nodoRaiz.getElementsByTagName("futuro");
        for (int i=0; i<lista.getLength(); i++){
            Element e=(Element) lista.item(i);
            NodeList listaHijos=e.getChildNodes();
            //El elemento ciudad procedencia es el quinto hijo
            Node nodoCiudad=listaHijos.item(5);
            NodeList hijosCiudad=nodoCiudad.getChildNodes();
            Node nodoElemCiudad=hijosCiudad.item(1);
            String nombreCiudad=nodoElemCiudad.getNodeName();
            if (nombreCiudad.equals(ciudad)){
                cuantos++;
            } //Fin del if
        } //Fin del for
        return cuantos;
    }

    /**
     *
     * @param raiz Raíz del documento
     * @param nombrePais (Debe ser "espania" para España)
     * @return
     */
    public int letrasConPaisEmisor(Node raiz, String nombrePais){
        int cuantos=0;
        Element eRaiz=(Element) raiz;
        NodeList listaLetras=eRaiz.getElementsByTagName("letra");
        for (int i=0; i<listaLetras.getLength(); i++){
            Node nodo=listaLetras.item(i);
            Element eNodo=(Element) nodo; //Devuelve elemento letra
            NodeList hijosLetra=eNodo.getChildNodes();

```

```

        Node nodoPaisEmisor=hijosLetra.item(3);
        NodeList hijosPais=nodoPaisEmisor.getChildNodes();
        Node nodoPais=hijosPais.item(1);
        String nombreNodoPais=nodoPais.getNodeName();
        if (nombreNodoPais.equals(nombrePais)){
            cuantos++;
        }
    } //Fin del for
    return cuantos;
}

public int algoQueVerConEspania(Node raiz){
    int cuantasLetras=this.letrasConPaisEmisor(raiz, "espania");
    int cuantosFuturos=this.cuantosFuturosTienenCiudadProcedencia(raiz,
        "madrid");
    int cuantosBonos=this.algoQueVerCon(raiz, "España");
    return cuantasLetras+cuantosFuturos+cuantosBonos;
}

public int extraerHijoNumero (Element padre,
    String nombreHijo){
    int valor=0;
    //Esta lista tiene solo un elemento
    NodeList listaHijos=
        padre.getElementsByTagName(nombreHijo);
    Element hijoNumerico=(Element) listaHijos.item(0);
    String contenidoTextual=hijoNumerico.getTextContent();
    valor=Integer.parseInt(contenidoTextual);
    return valor;
}

public void imprimirBonos(Node raiz){
    Element eRaiz=(Element) raiz;
    NodeList listaBonos=eRaiz.getElementsByTagName("bono");
    for (int i=0; i<listaBonos.getLength(); i++){
        Node bono=listaBonos.item(i);
        Element eBono=(Element) listaBonos.item(i);

        int valorDeseado=this.extraerHijoNumero(
            eBono, "valor_deseado");
        int valorMinimo=this.extraerHijoNumero(
            eBono, "valor_minimo");
        int valorMaximo=this.extraerHijoNumero(
            eBono, "valor_maximo");

        if ((valorDeseado>valorMinimo) &&
            (valorDeseado<valorMaximo) ){
            System.out.println("Encontrado un bono!");
            String ciudad=this.getCiudadProcedencia(eBono);
            System.out.println("Su ciudad era:"+ciudad);
        }
        //Element eValorDeseado=(Element)
        // listaParaValorDeseado.item(0);
    }
}

public String getCiudadProcedencia(Element e){
    NodeList listaHijos=

```



```

        e.getElementsByTagName("ciudad_procedencia");
        Element eCiudad=(Element) listaHijos.item(0);
        NodeList listaHijosCiudad=eCiudad.getChildNodes();
        Element eCiudadConcreto=
            (Element) listaHijosCiudad.item(1);
        String nombre=eCiudadConcreto.getNodeName();
        return nombre;
    }

    public void imprimirMismaCiudad(Node raiz){
        NodeList hijos=raiz.getChildNodes();
        for (int i=0; i<hijos.getLength(); i++){
            Node hijo=hijos.item(i);
            if (hijo.getNodeType() !=Node.ELEMENT_NODE){
                continue;
            }
            String ciudadHijo=
                this.getCiudadProcedencia((Element)hijo);
            for (int j=i+1; j<hijos.getLength(); j++){
                Node otroHijo=hijos.item(j);
                if (otroHijo.getNodeType() !=Node.ELEMENT_NODE){
                    continue;
                }

                String ciudadOtro=
                    this.
↪getCiudadProcedencia((Element)otroHijo);
                if (ciudadHijo.equals(ciudadOtro)){
                    System.out.println(
                        "Encontré dos elementos con_
↪la ciudad "+ciudadHijo);

                } //Fin del if ciudadHijo
            } //Fin del for interno
        } //Fin del for externo
    } //Fin del método

    public String[] obtenerListaBonos(Node raiz){
        int tamañoMaximo=1000;
        String[] ciudades=new String[tamañoMaximo];
        String[] aux=new String[tamañoMaximo];
        int posPrecio=0;
        Element eRaiz=(Element) raiz;
        NodeList listaBonos=eRaiz.getElementsByTagName("bono");
        for (int i=0; i<listaBonos.getLength(); i++){
            Element bono=(Element) listaBonos.item(i);
            String atEstable=bono.getAttribute("estable");
            if (atEstable!=null){
                if (atEstable.equals("no")){
                    //Examinamos la ciudad aprovechando
                    //un método ya construido.
                    String ciudad=this.getCiudadProcedencia(bono);
                    if (ciudad.equals("tokio")){
                        //Si es de tokió, copiamos el precio
                        String precio=bono.getAttribute(
↪"precio");

                        aux[posPrecio]=precio;
                        posPrecio++;
                    }
                }
            }
        }
    }

```

```

        } //Fin del if para tokio
    } //Fin del if para el "no"
    } //Fin del if para atEstable
} //Fin del for que recorre los bonos
if (posPrecio>2){
    return aux;
}
return ciudades;
}

public void crearElemento(Document d,
    String nombre,String contenido){
    Element e=d.createElement(nombre);
    e.setTextContent(contenido);
}

public void crearRSS(){
    DocumentBuilderFactory fabrica;
    DocumentBuilder constructor;
    Document documentoXML;
    try{
        fabrica=
            DocumentBuilderFactory.newInstance();
        constructor=fabrica.newDocumentBuilder();
        documentoXML=constructor.newDocument();
        TransformerFactory fabricaConv =
            TransformerFactory.newInstance();
        Transformer transformador =
            fabricaConv.newTransformer();
        DOMSource origenDOM =
            new DOMSource(documentoXML);
        Element e=documentoXML.createElement("rss");
        documentoXML.appendChild(e);
        StreamResult resultado=
            new StreamResult(
                new File("D:\\oscar\\archivo.
↪rss"));

        transformador.transform(origenDOM, resultado);
    }
    catch (Exception e){
        System.out.print("No se han podido crear los");
        System.out.println(" objetos necesarios.");
        e.printStackTrace();
        return ;
    }
}

public static void main (String[] argumentos){
    System.out.println("Probando...");
    ProcesadorXML proc=new ProcesadorXML();
    Node nodoRaiz=proc.extraerRaiz("bolsas.xml");
    proc.imprimirNombreDeLaRaiz(nodoRaiz);
    proc.imprimirHijos(nodoRaiz);
    proc.imprimirHijos2(nodoRaiz);
    int cuantosFuturos=proc.contadorElementos(nodoRaiz, "futuro");
    System.out.println("Hay "+cuantosFuturos);
    proc.imprimirPrecioBonos(nodoRaiz);
    int inestables=proc.cuantosInestables(nodoRaiz);
}

```

```
//      System.out.println("Inestables hay:"+inestables);
//      float preciosTotales=proc.sumarAtributosPrecio(nodoRaiz);
//      System.out.println("La suma total es:"+preciosTotales);
//      int cuantos=proc.algoQueVerCon(nodoRaiz, "Islandia");
//      System.out.println("Num paises Islandia:"+cuantos);
//      cuantos=proc.cuantosFuturosTienenCiudadProcedencia(
//          nodoRaiz, "madrid");
//      System.out.println("Futuros de Madrid:"+cuantos);
//      cuantos=proc.letrasConPaisEmisor(nodoRaiz, "espania");
//      System.out.println("Letras con espania:"+cuantos);
//      cuantos=proc.algoQueVerConEspania(nodoRaiz);
//      System.out.println("Productos rel. con España:"+cuantos);
//      proc.imprimirBonos(nodoRaiz);
//      proc.imprimirMismaCiudad(nodoRaiz);
//      String[] resultados=proc.obtenerListaBonos(nodoRaiz);
//      System.out.println("La ciudad 0 es:"+resultados[0]);
//      proc.crearRSS();
//  }
}
```

6.6.2 Archivo XML

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE listado [
    <!--ELEMENT listado (futuro+, divisa+, bono+, letra+)-->
    <!--ATTLIST futuro precio CDATA #REQUIRED-->
    <!--ATTLIST divisa precio CDATA #REQUIRED-->
    <!--ATTLIST bono precio CDATA #REQUIRED-->
    <!--ATTLIST letra precio CDATA #REQUIRED-->
    <!--ELEMENT ciudad_procedencia (madrid|nyork|frankfurt|tokio)-->
    <!--ELEMENT madrid EMPTY-->
    <!--ELEMENT nyork EMPTY-->
    <!--ELEMENT frankfurt EMPTY-->
    <!--ELEMENT tokio EMPTY-->
    <!--ATTLIST divisa estable CDATA #IMPLIED-->
    <!--ATTLIST bono estable CDATA #IMPLIED-->
    <!--ELEMENT futuro (producto, mercado?, ciudad_procedencia)-->
    <!--ELEMENT producto (#PCDATA)-->
    <!--ELEMENT mercado (#PCDATA)-->
    <!--ELEMENT bono (pais_de_procedencia,valor_deseado,
        valor_minimo, valor_maximo, ciudad_procedencia)-->
    <!--ELEMENT valor_deseado (#PCDATA)-->
    <!--ELEMENT valor_minimo (#PCDATA)-->
    <!--ELEMENT valor_maximo (#PCDATA)-->
    <!--ELEMENT pais_de_procedencia (#PCDATA)-->
    <!--ELEMENT divisa (nombre_divisa,
        tipo_de_cambio+, ciudad_procedencia)-->
    <!--ELEMENT nombre_divisa (#PCDATA)-->
    <!--ELEMENT tipo_de_cambio (#PCDATA)-->
    <!--ELEMENT letra (tipo_de_interes, pais_emisor,ciudad_procedencia)-->
    <!--ELEMENT tipo_de_interes (#PCDATA)-->
    <!--ELEMENT pais_emisor (espania|eeuu|alemania|japon)-->
    <!--ELEMENT espania EMPTY-->
    <!--ELEMENT eeuu EMPTY-->
    <!--ELEMENT alemania EMPTY-->
    <!--ELEMENT japon EMPTY-->
]
```

```
]>
```

```
<listado><futuro precio="11.28">
  <producto>Cafe</producto>
  <mercado>América Latina</mercado>
  <ciudad_procedencia>
    <madrid/>
  </ciudad_procedencia>
</futuro>
<divisa precio="183">
  <nombre_divisa>Libra esterlina</nombre_divisa>
  <tipo_de_cambio>2.7:1 euros</tipo_de_cambio>
  <tipo_de_cambio>1:0.87 dólares</tipo_de_cambio>
  <ciudad_procedencia>
    <madrid/>
  </ciudad_procedencia>
</divisa>
<bono precio="100" estable="si">
  <pais_de_procedencia>
    España
  </pais_de_procedencia>
  <valor_deseado>9980</valor_deseado>
  <valor_minimo>9950</valor_minimo>
  <valor_maximo>10020</valor_maximo>
  <ciudad_procedencia>
    <tokio/>
  </ciudad_procedencia>
</bono>
<bono precio="10000" estable="si">
  <pais_de_procedencia>
    España
  </pais_de_procedencia>
  <valor_deseado>9980</valor_deseado>
  <valor_minimo>9950</valor_minimo>
  <valor_maximo>10020</valor_maximo>
  <ciudad_procedencia>
    <tokio/>
  </ciudad_procedencia>
</bono>
<bono precio="10000" estable="no">
  <pais_de_procedencia>
    España
  </pais_de_procedencia>
  <valor_deseado>9980</valor_deseado>
  <valor_minimo>9950</valor_minimo>
  <valor_maximo>10020</valor_maximo>
  <ciudad_procedencia>
    <tokio/>
  </ciudad_procedencia>
</bono>
<bono precio="10000" estable="no">
  <pais_de_procedencia>
    España
  </pais_de_procedencia>
  <valor_deseado>9980</valor_deseado>
  <valor_minimo>9950</valor_minimo>
  <valor_maximo>10020</valor_maximo>
```

```
        <ciudad_procedencia>
            <tokio/>
        </ciudad_procedencia>
    </bono>
    <letra precio="45020">
        <tipo_de_interes>4.54%</tipo_de_interes>
        <pais_emisor>
            <espania/>
        </pais_emisor>
        <ciudad_procedencia>
            <madrid/>
        </ciudad_procedencia>
    </letra>
</listado>
```

6.7 Ejercicios para preparar examen

1. Indicar cuantas letras tienen un tipo de interés inferior al 3 % e indicar para cada una de ellas el país emisor.
2. Comprobar si hay alguna divisa con nombre “Euro” cuyo precio sea mayor de 195.
3. Indicar todos los productos que tengan la misma ciudad de procedencia.

Sindicación y transformación de contenidos

7.1 Introducción

Muchas páginas web disponen de “feeds”. Un “feed” es un mecanismo de suscripción que facilita la recepción de información.

En general, los feed utilizan un vocabulario XML llamado RSS o uno llamado Atom.

7.2 RSS

RSS es un estándar para la “sindicación” o “agregación” de recursos (recursos web normalmente).

Su objetivo principal era permitir a un sitio web publicar las novedades con facilidad y que el usuario puede ir directamente al lugar que le interese.

7.2.1 Formato de archivo RSS

Todo archivo RSS es, por supuesto, un XML

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
  <channel>
    <title>
      Canal de noticias de SS00 de DAM
    </title>
    <link>
      http://ssoo.iesmaestredecatalrava.es
    </link>
    <description>
      En este canal...
    </description>
    <item>
      <title>Nueva versión de Ubuntu</title>
      <link>http://ubuntu.org</link>
      <description>
        Nueva versión...
      </description>
    </item>
  </channel>
</channel>
```

```
<title>
    Canal de Lenguajes de marcas
</title>
<link>
    http://xml.iesmaestredecalatrava.es
</link>
<description>
    En este canal...
</description>
<item>
    <title>Publicado nuevo validador del W3C</title>
    <link>http://validator.w3c.org</link>
    <description>
        Hay nuevo validador...
    </description>
</item>
</channel>
</rss>
```

Las reglas serían las siguientes:

- Todo archivo de descripción de recursos en RSS utiliza el preámbulo típico de los documentos xml

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

- Todo RSS tiene un solo elemento raíz en el cual se puede indicar la versión RSS a la que nos ceñimos

```
<rss version="2.0">
```

- Un RSS tiene uno o más canales

```
<channel>
</channel>
```

- Todo canal debe tener, al menos un título, un enlace base (la dirección del propio sitio web) y una descripción:

```
<channel>
    <title>
    </title>
    <link>
    </link>
    <description>
    </description>
    <item>
        ...
    </item>
    <item>
        ...
    </item>
</channel>
```

Resumiendo los puntos más importantes:

1. Usar como elemento raíz `rss`.
2. Todo RSS tiene uno o más `channel`
3. Todo `channel` tiene al menos `title`, `link` y `description`

4. Después de estos elementos, un `channel` puede tener uno o más elementos `item` (que son los que contienen las noticias)
5. Todo `item` también debe tener un `title`, un `link` y `description`

7.2.2 Ejercicio

Crear un fichero Java que construya el siguiente fichero XML:

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>Prueba</title>
    <link>http://www.google.es</link>
    <description>Prueba de descripcion</description>
  </channel>
</rss>
```

Una posible solución es esta:

```
public class CreadorRSS {
    public byte[] getEtiquetas(
        String titulo,
        String enlace,
        String descripcion)
    {
        String resultado="";
        resultado+="<title>";
        resultado+=titulo;
        resultado+="</title>\n";
        resultado+="<link>";
        resultado+=enlace;
        resultado+="</link>\n";
        resultado+="<description>";
        resultado+=descripcion;
        resultado+="</description>";
        return resultado.getBytes();
    }

    public void crearArchivo(String nombre)
        throws IOException{
        FileOutputStream fos=
            new FileOutputStream(nombre);
        String cabecera="<?xml version='1.0'?>\n";
        fos.write(cabecera.getBytes());
        String rss="<rss version='1.0'>\n";
        fos.write(rss.getBytes());
        byte[] etiquetas=this.getEtiquetas(
            "Titulo de la noticia",
            "http://www.algo.com",
            "Noticia muy importante");
        fos.write(etiquetas);
        String rssCierre="</rss>";
        fos.write(rssCierre.getBytes());
        fos.close();
    }

    public static void main(String[] args)
        throws IOException {
```

```

        CreadorRSS creador=new CreadorRSS();
        creador.crearArchivo("D:/oscar/archivo.rss");
    }
}

```

El siguiente código Java ilustra otra forma de hacerlo:

```

public void crearRSS(){
    DocumentBuilderFactory fabrica;
    DocumentBuilder constructor;
    Document documentoXML;
    try{
        fabrica=
            DocumentBuilderFactory.newInstance();
        constructor=fabrica.newDocumentBuilder();
        documentoXML=constructor.newDocument();
        TransformerFactory fabricaConv =
            TransformerFactory.newInstance();
        Transformer transformador =
            fabricaConv.newTransformer();
        DOMSource origenDOM =
            new DOMSource(documentoXML);
        Element e=documentoXML.createElement("rss");
        documentoXML.appendChild(e);
        StreamResult resultado=
            new StreamResult(
                new File("D:\\resul\\archivo.rss"));
        transformador.transform(origenDOM, resultado);
    }
    catch (Exception e){
        System.out.print("No se han podido crear los");
        System.out.println(" objetos necesarios.");
        e.printStackTrace();
        return ;
    }
}

```

7.3 Adaptación y transformación de XML

Muy a menudo va a ocurrir que un cierto formato XML va a ampliarse o a modificarse o simplemente se necesita convertir un documento XML en otro con un formato distinto.

Supongamos una estructura como la siguiente:

```

<?xml version="1.0" encoding="UTF-8"?>
<catalogo>
    <libro>
        <title>Don Quijote</title>
        <autor>Cervantes</autor>
    </libro>
    <libro>
        <title>
            Poeta en Nueva York
        </title>
        <autor>Lorca</autor>
    </libro>
</catalogo>

```

```

    </libro>
</catalogo>

```

Supongamos que un cierto sitio se necesita almacenar la información de esta forma:

```

<?xml version="1.0" encoding="UTF-8"?>
<listadolibros>
  <libro>
    <titulo autor="Cervantes">Don Quijote</title>
  </libro>
  <libro>
    <titulo autor="Lorca">
      Poeta en Nueva York
    </title>
  </libro>
</listadolibros>

```

En general, para poder modificar o presentar los XML se puede hacer varias cosas:

- En primer lugar, se puede usar CSS para poder cargar los documentos XML en un navegador y mostrarlos de forma aceptable.
- Se pueden utilizar otras tecnologías para transformar por completo la estructura del XML.
 - Se puede usar un lenguaje llamado XSLT (Xml Stylesheet Language Transformation) para convertir el XML en otro distinto.
 - Se puede utilizar XSL:FO (Xml Stylesheet Language: Formatting Objects) cuando se desee convertir el documento en algo que se desee imprimir (normalmente un PDF)

7.3.1 CSS con XML

Supongamos de nuevo el archivo anterior, el cual ahora queremos mostrar en un navegador:

```

<?xml version="1.0" encoding="UTF-8"?>
<catalogo>
  <libro>
    <title>Don Quijote</title>
    <autor>Cervantes</autor>
  </libro>
  <libro>
    <title>
      Poeta en Nueva York
    </title>
    <autor>Lorca</autor>
  </libro>
</catalogo>

```

Si usamos el archivo `estilo.css` de esta forma:

```

catalogo{
  background-color:rgb(220, 230, 220);
  display:block;
}

libro{
  display:block;
  border: solid black 1px;
}

```

```

        margin-bottom:20px;
    }
    title{
        margin: 10px;
        display:block;
    }
    autor{
        display:block;
        font-face:Arial;
        text-decoration:underline;
    }
}

```

Veremos algo como esto:

Don Quijote	Cervantes
Poeta en Nueva York	Lorca

Ejercicio

Crear una hoja de estilo asociada al catálogo anterior, que muestre la información de cada libro de forma parecida a una tabla, en la que el `title` utilice un color de fondo distinto del `autor`.

Don Quijote	Cervantes
Poeta en Nueva York	Lorca

```

catalogo{
    background-color:rgb(220, 230, 220);
    display:block;
}

libro{
    display:block;
    width:100%;
    margin-bottom:40px;
    clear:both;
}

title{
    float:left;
    width:45%;
    border:solid black 1px;
    padding:5px;
    text-align:center;
    background-color:rgb(180,180,240);
}

autor{
    float:left;
    text-align:center;
    width:45%;
    border:solid black 1px;
}

```

```
padding:5px;
background-color:rgb(340,180,240);
}
```

7.3.2 Transformación de XML

Si deseamos *transformar un XML en otro XML* necesitaremos usar XSLT. Un archivo XSLT tiene la extensión XSL e indica las reglas para convertir entre formatos XML.

El documento XSL básico sería así (los navegadores la darán por mala, ya que no hace absolutamente nada):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet>
</xsl:stylesheet>
```

En este caso xsl es el espacio de nombres. Un espacio de nombres es un contenedor que permite evitar que haya confusiones entre unas etiquetas y otras que se llamen igual. En este caso, queremos usar la etiqueta <stylesheet> definida por el W3C. Una hoja básica sería esta

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>
          Resultado
        </title>
      </head>
      <body>
        Documento resultado
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Algunos navegadores no ejecutan XSL por seguridad. Los detalles de como “abrir” la seguridad de cada uno de estos navegadores deben investigarse en el manual de cada uno de ellos.

Cabe destacar que esta hoja simplemente genera HTML básico pero no recoge ningún dato del XML original.

7.3.3 Ejercicio

Hacer que el archivo XML de libros cargue esta hoja de estilos.

Solución: consiste en añadir una línea al archivo que referencie el archivo de transformación y el tipo de lenguaje usado para transformar.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="hoja1.xsl" type="text/xsl"?>

<catalogo>
  ... (El resto es igual)
</catalogo>
```

7.3.4 Ejercicio

Hacer que el XSL genere un HTML con información del archivo XML de libro.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.1" xmlns:xsl=
  "http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>Ejemplo de transformación</title>
      </head>
      <body>
        <h1>Resultado</h1>
        <xsl:for-each select="catalogo/libro">
          <p>
            <xsl:value-of select="title"/>
          </p>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

7.3.5 Ejercicio

Extraer los títulos de los libros pero consiguiendo encerrarlos en una lista ordenada HTML para que aparezcan numerados.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.1" xmlns:xsl=
  "http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>Ejemplo de transformación</title>
      </head>
      <body>
        <h1>Resultado</h1>
        <ol>
          <xsl:for-each select="catalogo/libro">
            <li>
              <xsl:value-of select="title"/>
            </li>
          </xsl:for-each>
        </ol>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

7.3.6 Ejercicio

Supongamos que ahora un libro tiene varios autores y el XML es algo así:

```

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="hojal.xml" type="text/xsl"?>

<catalogo>
    <libro>
        <title>Don Quijote</title>
        <autores>
            <autor>Cervantes</autor>
        </autores>
    </libro>
    <libro>
        <title>Patrones de diseño en programación</title>
        <autores>
            <autor>Erich Gamma</autor>
            <autor>John Vlissides</autor>
            <autor>Ralph Johnson</autor>
        </autores>
    </libro>
</catalogo>

```

¿Como mostrar en HTML el título y todos los autores de cada libro?

```

<?xml version="1.0"?>
<xsl:stylesheet version="1.1" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <head>
                <title>Ejemplo de transformación</title>
            </head>
            <body>
                <h1>Resultado</h1>
                <ol>
                    <xsl:for-each select="catalogo/libro">
                        <li>
                            <xsl:value-of select="title"/>
                            <ol>
                                <xsl:for-each select="autores/
↪ autor">
                                    <li>
                                        <xsl:value-of
↪ select="."/>
                                    </li>
                                </xsl:for-each> <!--Fin del
↪ bucle autores-->
                            </ol>
                        </li>
                    </xsl:for-each> <!--Fin del recorrido de libro-->
                </ol>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>

```

7.3.7 Ejercicio

Se desea hacer lo mismo que en el ejercicio anterior pero haciendo que los autores aparezcan de forma ordenada.

La solución está fundamentada en el uso de la etiqueta siguiente:

```
<xsl:for-each ...>
  <xsl:sort select="..." ordering="...">
    ..cosas del bucle...
  </xsl>
</xsl>
```

La solución completa sería así:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.1" xmlns:xsl=
  "http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>Ejemplo de transformación</title>
      </head>
      <body>
        <h1>Resultado</h1>
        <ol>
          <xsl:for-each select="catalogo/libro">
            <li>
              <xsl:value-of select="title"/>
              <ol>
                <xsl:for-each select="autores/"
                  <xsl:sort order="descending"/>
                <li>
                  <xsl:value-of
                    <select="."/>
                </li>
              </ol>
            </li>
          </xsl:for-each> <!--Fin del recorrido de libro-->
        </ol>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

7.3.8 Ejercicio

Suponiendo que además todos los libros tienen además un elemento <fechaedicion> mostrar los libros editados después del 2000.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="hoja1.xsl" type="text/xsl"?>

<catalogo>
  <libro>
    <title>Don Quijote</title>
```



```

        <autores>
            <autor>Cervantes</autor>
        </autores>
        <fechaedicion>1984</fechaedicion>
    </libro>
</libro>
    <title>Patrones de diseño en programación</title>
    <autores>
        <autor>Ralph Johnson</autor>
        <autor>Erich Gamma</autor>
        <autor>John Vlissides</autor>
    </autores>
    <fechaedicion>2007</fechaedicion>
</libro>
</catalogo>

```

```

<?xml version="1.0"?>
<xsl:stylesheet version="1.1" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <head>
                <title>Ejemplo de transformación</title>
            </head>
            <body>
                <h1>Resultado</h1>
                <ol>
                    <xsl:for-each select="catalogo/libro">
                        <xsl:if test="fechaedicion > 2000">
                            <li>
                                <xsl:value-of select="title"/>
                                <ol>
                                    <xsl:for-each select="autores/
↪ autor">
                                        <xsl:sort order="descending"/>
                                        <li>
                                            <xsl:value-of ↪
↪ select="."/>
                                        </li>
                                    </xsl:for-each>
                                </ol>
                            </li>
                        </xsl:if>
                    </xsl:for-each> <!--Fin del recorrido de libro-->
                </ol>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>

```

En general, las condiciones se escriben así:

- > o mayor que o > ;
- < o menor que o < ;
- >= o mayor o igual o >= ;

- `<=` o menor o igual o `≤`;
- `=` o igual o `&eq;`;
- `<>` o distinto o `&neq;`;

7.3.9 Ejercicio XSL, paso a paso

Dado el siguiente XML crear un programa con XSLT que muestre los títulos y los autores de los libros cuya fecha de edición sea posterior al 2000.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="ejercicio1.xsl"?>
  <catalogo>
    <libro fechaedicion="1999">
      <titulo>Don Quijote</titulo>
      <autores>
        <autor>Cervantes</autor>
      </autores>
    </libro>
    <libro fechaedicion="2005">
      <titulo>
        La sociedad civil moderna
      </titulo>
      <autores>
        <autor>Luis Diaz</autor>
        <autor>Pedro Campos</autor>
      </autores>
    </libro>
  </catalogo>
```

Hagámoslo paso a paso. En primer lugar tendremos que crear el fichero `ejercicio1.xsl` y crear la estructura básica:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.1">
<xsl:template match="/">

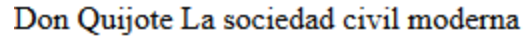
</xsl:template>
</xsl:stylesheet>
```

Ahora recorramos los libros que hay en el catalogo (recordemos que la estructura es `catalogo/libro`. Simplemente por ver si funciona, de momento el navegador solo muestra los títulos y en una sola línea.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.1">
<xsl:template match="/">
  <xsl:for-each select="catalogo/libro">
    <xsl:value-of select="titulo"/>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Avancemos un poco más y creemos una estructura HTML válida

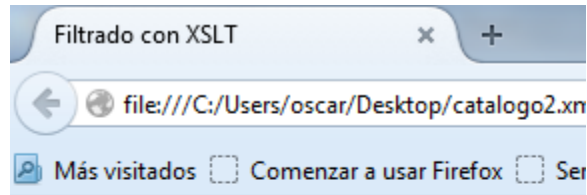
```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.1">
<xsl:template match="/">
```



Don Quijote La sociedad civil moderna

Figura 7.1: Paso inicial del XSL

```
<html>
  <head>
    <title>Filtrado con XSLT</title>
  </head>
  <body>
    <h1>Filtrado con XSLT</h1>
    <ol>
      <xsl:for-each select="catalogo/libro">
        <li>
          <xsl:value-of select="titulo"/>
        </li>
      </xsl:for-each>
    </ol>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```



Filtrado con XSLT

1. Don Quijote
2. La sociedad civil moderna

Figura 7.2: Extrayendo los titulos con XSL

Ahora vamos a procesar solo los libros cuya fechaedicion sea posterior al 2000. Añadamos un if

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.1">
<xsl:template match="/">
<html>
  <head>
    <title>Filtrado con XSLT</title>
  </head>
  <body>
    <h1>Filtrado con XSLT</h1>
```

```

<ol>
  <xsl:for-each select="catalogo/libro">

    <xsl:if test="@fechaedicion > 2000">

      <li>
        <xsl:value-of select="titulo"/>
      </li>

    </xsl:if>
  </xsl:for-each>
</ol>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```



Filtrado con XSLT

1. La sociedad civil moderna

Figura 7.3: Procesando los que son > 2000

Ahora para cada libro queremos también mostrar los elementos autor con su propia lista

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.1">
<xsl:template match="/">
<html>
  <head>
    <title>Filtrado con XSLT</title>
  </head>
  <body>
    <h1>Filtrado con XSLT</h1>
    <ol>
      <xsl:for-each select="catalogo/libro">

        <xsl:if test="@fechaedicion > 2000">

          <li>
            <xsl:value-of select="titulo"/>
          </li>

          <ol>
            <xsl:for-each select="autores/autor">
              <li>

```

```

                                <!--El elemento actual es .-->
                                <xsl:value-of select="."/>
                            </li>
                        </xsl:for-each>
                    </ol>
                </xsl:if>
            </xsl:for-each>
        </ol>
    </body>
</html>
</xsl:template>
</xsl:stylesheet>

```

Y el navegador muestra lo siguiente



Filtrado con XSLT

1. La sociedad civil moderna

1. Luis Díaz
2. Pedro Campos

Figura 7.4: Mostrando también los autores

7.3.10 Transformación de pedidos

Dado el siguiente archivo XML:

```

<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet href="estilo1.xsl" type="text/xsl"?>
<pedido>
    <portatiles>
        <portatil>
            <peso>1430</peso>
            <ram unidad="GB">4</ram>
            <disco tipo="ssd">500</disco>
            <precio>499</precio>
        </portatil>
        <portatil>
            <peso>1830</peso>
            <ram unidad="GB">6</ram>
            <disco tipo="ssd">1000</disco>
            <precio>1199</precio>
        </portatil>
    </portatiles>
</pedido>

```

```

        </portatil>
        <portatil>
            <peso>1250</peso>
            <ram unidad="GB">2</ram>
            <disco tipo="ssd">750</disco>
            <precio>699</precio>
        </portatil>
    </portatiles>
    <tablets>
        <tablet>
            <plataforma>Android</plataforma>
            <caracteristicas>
                <memoria medida="GB">2</memoria>
                <tamano medida="pulgadas">6</tamano>
                <bateria>LiPo</bateria>
            </caracteristicas>
        </tablet>
        <tablet>
            <plataforma>iOS</plataforma>
            <caracteristicas>
                <memoria medida="GB">4</memoria>
                <tamano medida="pulgadas">9</tamano>
                <bateria>LiIon</bateria>
            </caracteristicas>
        </tablet>
    </tablets>
</pedido>

```

Crear un fichero de estilos que permita mostrar la información de los portátiles en forma de tabla.

Resultado

Peso	RAM	Disco	Precio
1430	4	500	499
1830	6	1000	1199
1250	2	750	699

Figura 7.5: Transformacion XSL

Una posible solución sería esta:

```

<?xml version="1.0"?>
<xsl:stylesheet version="1.1"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
        <head>
            <title>Ejercicio 1</title>
        </head>
    </template>

```

```

<body>
<h1>Resultado</h1>
<table border="1">
<tr>
<td>Peso</td>
<td>RAM</td>
<td>Disco</td>
<td>Precio</td>

</tr>
<xsl:for-each select=
"pedido/portatiles/portatil">
<tr>
<td>
<xsl:value-of select="peso"/>
</td>
<td>
<xsl:value-of select="ram"/>
</td>
<td>
<xsl:value-of select="disco"/>
</td>
<td>
<xsl:value-of select="precio">
</td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

7.3.11 Transformación de pedidos (II)

Con el mismo fichero de pedidos crear una sola tabla que tenga 3 columnas y aglutine información tanto de portátiles como de tablets:

- Cuando procesemos portátiles, las columnas serán respectivamente “precio”, “ram” y “disco”. Solo se procesan portátiles con más de 2GB de RAM.
- Cuando procesemos tablets, las columnas serán “plataforma”, “ram” y “batería”. Solo se procesan los tablets con más de 2GB de RAM y que además tengan un tamaño superior a 7 pulgadas.

El fichero siguiente ilustra una posible forma de hacerlo:

```

<?xml version="1.0"?>
<xsl:stylesheet version="1.1"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>Ejercicio 1</title>
      </head>
      <body>
        <h1>Resultado</h1>
        <table border="1">
          <xsl:for-each select=

```

```

"pedido/portatiles/portatil">
  <xsl:if test="ram > 2">
    <tr>
      <td>
        Precio:<xsl:value-of select="precio"/>
      </td>
      <td>
        Memoria:<xsl:value-of select="ram"/>
      </td>
      <td>
        Disco duro:<xsl:value-of select="disco"/>
      </td>
    </tr>
  </xsl:if>
</xsl:for-each>
<xsl:for-each select="pedido/tablets/tablet">
  <xsl:if test="caracteristicas/memoria > 2">
    <xsl:if test="caracteristicas/tamano > 7">
      <tr>
        <td>
          <xsl:value-of select="plataforma"/>
        </td>
        <td>
          <xsl:value-of select="caracteristicas/memoria
        </td>
        <td>
          <xsl:value-of select="caracteristicas/bateria
        </td>
      </tr>
    </xsl:if>
  </xsl:if>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

7.3.12 Ejercicio (no se da la solución)

Poner en una lista ordenada (elemento `ol`) todas las capacidades RAM que se encuentren en el fichero XML.

Sistemas de gestión de información

8.1 Introducción

Primero definiremos los términos:

- Sistema: “conjunto de elementos interrelacionados que colaboran para la consecución de un objetivo”.
- Gestión de información: movimientos de información que facilitan la consecución de los objetivos de la empresa.

Entre los objetivos más habituales están:

- Objetivos económicos.
- Objetivos temporales (supervivencia).
- Objetivos legales, entre los que destacan los fiscales.

Procesar la información hoy en día es muy difícil, por lo que la informática es de gran ayuda en dicho procesamiento: (INFORmación+autoMÁTICA).

¿Todos los sistemas de una empresa son informáticos? NO. Un ejemplo es el sistema postal que no requiere (y no siempre ha requerido) parte informática.

8.2 Niveles en la empresa

Toda empresa se puede ver desde tres puntos de vista temporales distintos

1. Nivel estratégico: en este nivel están las personas o elementos que tienen un punto de vista más global y más largo plazo (normalmente plazos de varios años).
2. Nivel táctico: en este nivel están los mandos intermedios de la empresa con plazos medios (máximo un año)
3. Nivel operativo: está el personal “de a pie”, que recoge la información a diario y la procesa a diario.

Ejemplo: Mercadona.

8.2.1 Tipos de SGI informáticos

- DSS o Decision Support System: ayudan a tomar decisiones en función de la información disponible.
- CRM o Customer Relationship Managements o Gestores de la relación con clientes. Son sistemas que gestionan información muy personalizada a los clientes, tales como sistemas de recomendación, sistemas de fidelización. Los CRM manejan la comunicación entre información interna y externa de la empresa.

- ERP o Enterprise Resource Planning o sistemas de planificación de recursos corporativos gestionan toda la información interna de la empresa.

Los ERP han ganado mucho auge con el tiempo, ya que pueden hacer operaciones muy sofisticadas, entre ellas:

1. Optimización de procesos: implica resolver complejos problemas matemáticos que con una herramienta informática se resuelven al instante.
2. Logística: implica resolver problemas de transporte de productos en base a diversas restricciones (no todas matemáticas). Un problema muy común es la optimización de rutas que implica resolver el “problema del viajante”.
3. Fiscalidad: permite aprovechar las diferentes ventajas que en materia de fiscalidad se ofrecen en distintos casos, en distintos territorios...
4. Contabilidad: la participación en diversas sociedades empresariales puede complicar las operaciones de contabilidad, operaciones que un ERP puede registrar.
5. Optimización de recursos humanos: optimizar el volumen de contrataciones y despidos, nóminas, etc...
6. Operaciones de informes y estadística: para conocer el estado real de la empresa en sus distintos departamentos y períodos.
7. Trazabilidad: permite conocer el punto exacto de ubicación de un producto a lo largo de toda la cadena de abastecimiento (supply chain).

8.3 Almacenamiento en los SGI

La mayor parte de SGI se apoyan en tecnologías conocidas como SGBD con SQL e incluso XML.

La mayor parte de SGBD permite procesar y extraer información en forma de SQL.

En general, cualquier consulta sobre cualquier tabla puede convertirse en XML. Cuando por ejemplo hacemos

```
select * from marcas;
```

obtenemos un resultado como este:

```
+----+-----+-----+
| id | nombre | ap      |
+----+-----+-----+
| 1  | Juan   | Lopez Lopez |
| 2  | Andres | Ruiz Gomez  |
| 3  | Tomas  | Perez Diaz  |
+----+-----+-----+

3 rows in set (0.00 sec)
```

Sin embargo, podemos ejecutar el siguiente comando que se conecta a MySQL ejecuta la consulta dada y devuelve un fichero XML:

```
mysql -uroot -Ddatos -e"select * from usuarios" --xml
```

El resultado será algo como esto:

```
<?xml version="1.0"?>

<resultset statement="select * from usuarios
" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

<row>
    <field name="id">1</field>
    <field name="nombre">Juan</field>
    <field name="ap">Lopez Lopez</field>
</row>

<row>
    <field name="id">2</field>
    <field name="nombre">Andres</field>
    <field name="ap">Ruiz Gomez</field>
</row>

<row>
    <field name="id">3</field>
    <field name="nombre">Tomas</field>
    <field name="ap">Perez Diaz</field>
</row>
</resultset>

```

8.4 Puntos de vista en los SGI

8.4.1 Desarrollador

Para un desarrollador hay ciertos temas fundamentales a conocer del SGI con el que trabaje:

- Programable ¿qué lenguaje de programación usa? ¿usa OOP? ¿es un tipo de lenguaje distinto?.
- ¿Se puede conectar desde algún lenguaje general?
- ¿Se pueden usar otras bibliotecas de uso general? (Hibernate)

8.4.2 Administrador

- ¿Se pueden controlar los accesos?
- ¿Como cumplir la LOPD y sus 3 niveles?

8.4.3 Usuario

- ¿Como se usa el SGI? En unos casos hay muchos requisitos pero cada vez más abundan los SGI con interfaz Web.
- La adaptación específica que ofrece el sistema a la empresa. El PGC en España es un elemento al cual un programa puede estar adaptado o no. Otra posibilidad es que el programa permita INCOTERMS.

8.4.4 Ejercicio

Modelar el diagrama de estados para el cajero en el caso “Sacar dinero”