
Apuntes de Lenguajes de Marcas Documentation

Versión 1.3

Oscar Gomez

06 de marzo de 2018

1. Introducción	1
1.1. Historia	1
1.1.1. El pasado	1
1.1.2. El presente	1
1.1.3. El futuro	1
1.2. Servidores web	1
1.2.1. Como configurar un servidor en casa	2
1.3. Los nombres de dominio	2
1.4. Los sistemas de gestión de información	3
1.5. Tipos de lenguajes	3
1.6. Organismos de regulación	3
1.7. Gramáticas y DTD's	3
1.8. XML Schemas	5
1.9. Definiciones	5
2. HTML5	7
2.1. Introducción	7
2.2. Etiquetas estructurales	7
2.3. Etiquetas de formato	8
2.4. Gestión de espacios	8
2.5. Entidades	9
2.6. Texto preformateado	9
2.7. Listas	9
2.7.1. Ejercicio	10
2.8. Tablas	10
2.8.1. Un ejemplo de tabla	11
2.8.2. Ejercicio sobre tablas	11
2.8.3. Solución	12
2.8.4. Ejercicio sobre tablas (II)	12
2.8.5. Solución	12
2.8.6. Ejercicio sobre tablas (III)	13
2.8.7. Solución	14
2.8.8. Ejercicio sobre tablas (V)	15
2.8.9. Solución	15
2.8.10. Ejercicio sobre tablas (VI)	17
2.8.11. Solución	17

2.8.12.	Ejercicio sobre tablas (VII)	19
2.8.13.	Solución	19
2.8.14.	Ejercicio sobre tablas (VIII)	20
2.8.15.	Solución	20
2.9.	Formularios	21
2.9.1.	Campo de texto	21
2.9.2.	Selector único (radio-button)	22
2.9.3.	Selector múltiple (checkbox)	22
2.9.4.	Lista desplegable	22
2.9.5.	Textareas	22
2.10.	Ejercicios tipo examen	23
2.10.1.	Enunciado	23
2.10.2.	Solución	23
2.10.3.	Enunciado	23
2.10.4.	Solución	24
2.10.5.	Enunciado	25
2.10.6.	Solución	25
2.10.7.	Enunciado	25
2.10.8.	Solución	26
2.10.9.	Enunciado	27
2.10.10.	Enunciado	28
2.10.11.	Solución	28
2.11.	Examen	30
3.	CSS	31
3.1.	Introducción	31
3.2.	Sintaxis	31
3.3.	Los atributos <code>class</code> e <code>id</code>	31
3.4.	Posicionamiento	33
3.4.1.	Ejercicio propuesto	33
3.4.2.	Ejercicio propuesto (II)	35
3.4.3.	Ejercicio de maquetación	36
3.4.4.	Ejercicio 2 de maquetación	38
3.4.5.	HTML	38
3.4.6.	CSS	39
3.4.7.	Ejercicio: barra de herramientas	40
3.4.8.	Ejercicio: ampliación	41
3.4.9.	Ejercicio	41
3.5.	Posicionamiento float	42
3.5.1.	Ejercicio	43
3.6.	Maquetación avanzada con <code>grid-layouts</code>	44
3.7.	Media queries	47
3.8.	Gestión de espacios	50
3.9.	Colores	50
3.10.	Tipografías	50
3.11.	Alineación del texto	50
3.12.	Decoración del texto	50
3.13.	Medidas	51
3.14.	Selectores	51
3.14.1.	Solución <code>p#destacado</code>	52
3.14.2.	Solución <code>p.destacado</code>	52
3.14.3.	Solución <code>p.destacado, span#id1</code>	52
3.14.4.	Solución <code>p.destacado > li.elemento_enumeracion</code>	54
3.14.5.	Solución <code>p.destacado > .elemento_numeracion</code>	54

3.14.6. Solución .destacado > #id1	54
3.15. Bootstrap	54
3.15.1. Estructura básica	54
3.15.2. Rejilla o <i>grid</i>	56
3.15.3. Tipografía	56
3.16. Ejercicio responsive I	57
3.17. Ejercicio responsive II	59
4. Javascript	63
4.1. Introducción	63
4.2. Tipos de datos	63
4.3. Incrustando Javascript	64
4.4. Decisiones	64
4.5. Vectores o Arrays	64
4.6. Bucles	64
4.6.1. Bucles for	64
4.6.2. Bucles while	65
4.7. Ejercicio: media aritmética	66
4.8. Ejercicio: desviación media	66
4.9. Ejercicio: la mediana	66
4.10. Funciones	67
4.10.1. Ejercicio	67
4.10.2. Ejercicio	68
4.11. Programación OO	69
4.11.1. Ejercicio	69
4.11.2. Ejercicio	70
4.12. Programación con JQuery	70
4.12.1. Inicio	70
4.12.2. La función \$	71
4.12.3. Gestión de eventos	71
4.12.4. Ejercicio	73
4.12.5. Ejercicio	74
4.13. Procesado de atributos	74
4.13.1. Ejercicio: recuento de medios de locomoción	76
4.13.2. Ejercicio configurador	78
4.13.3. Solución HTML configurador	78
4.13.4. Solución JS configurador	79
4.13.5. Ejercicio configurador de PCs ampliado	79
4.14. Configurador de coches	80
4.14.1. HTML del configurador	80
4.14.2. JS del configurador (con JQuery (DAM))	80
4.15. Dinamismo con Google Maps	80
4.15.1. HTML de GMaps	80
4.15.2. Javascript de GMaps	81
4.15.3. Ejercicio	82
4.15.4. Ejercicio	83
4.15.5. Ampliación	84
4.16. Otro configurador de coches	86
4.16.1. Configurador en HTML	86
4.16.2. Configurador en JS	87
4.17. Calculo de impuestos	88
4.17.1. HTML calculador	88
4.17.2. JS Calculador (con JQuery (DAM))	89
4.17.3. JS Calculador (sin JQuery (ASIR))	90

4.18.	Comparador de telefonía	91
4.18.1.	HTML del comparador	91
4.18.2.	JS del comparador	92
5.	XML	95
5.1.	Introducción	95
5.2.	Un ejemplo sencillo	95
5.3.	Construcción de XML	95
5.4.	Validez	96
5.5.	Gramáticas	97
5.5.1.	Sintaxis DTD	99
5.5.2.	Ejemplo de DTD (productos)	100
5.6.	Ejercicio I (DTD)	101
5.7.	Ejercicio II (DTD)	102
5.7.1.	Solución	102
5.8.	Ejercicio (con atributos)	103
5.8.1.	Solución ejercicio con atributos	103
5.9.	Ejercicio	104
5.9.1.	Solución completa	104
5.10.	Otras características de XML	105
5.10.1.	Atributos	105
5.10.2.	Elementos vacíos	106
5.10.3.	Alternativas	106
5.11.	Ejercicio	107
5.11.1.	Solución	107
5.12.	Ejercicio: mayorista de libros	108
5.12.1.	Solución al mayorista de libros	110
5.13.	Ejercicio: fabricante de tractores	111
5.13.1.	Solución: DTD fabricante tractores	111
5.14.	Ejercicio: repeticiones de opciones	112
5.15.	Ejercicio: multinacional	113
5.16.	Ejercicio	114
5.17.	Esquemas XML	115
5.17.1.	Un ejemplo	115
5.17.2.	Tipos de datos básicos	116
5.17.3.	Derivaciones	116
5.17.4.	Tipos simples y complejos	118
5.17.5.	Ejercicio: edad de los trabajadores	118
5.17.6.	Ejercicio: peso de productos	119
5.17.7.	Ejercicio: pagos validados	119
5.17.8.	Ejercicio: validación de DNIs	119
5.17.9.	Uniendo la herencia y el sistema de tipos	120
5.17.10.	Restricciones	120
5.17.11.	Atributos	121
5.18.	Ejercicios de XML Schemas	121
5.18.1.	Cantidades limitadas	121
5.18.2.	Solución a las cantidades limitadas	121
5.18.3.	Cantidades limitadas con atributo divisa	122
5.18.4.	Solución a las cantidades limitadas con atributo divisa	122
5.18.5.	Cantidades limitadas con atributo divisa con solo ciertos valores	122
5.18.6.	Solución al atributo con solo ciertos valores	123
5.18.7.	Ejercicio: codigos y sedes	124
5.18.8.	Solución a los códigos y sedes	124
5.18.9.	Ejercicio: productos con atributos	125

5.18.10.	Ejercicio: clientes con información adicional	125
5.18.11.	Lista de clientes como XML Schemas	125
5.18.12.	Ampliación del esquema para clientes	127
5.18.13.	Ejercicio: lista de códigos	128
5.18.14.	Ejercicio: otra lista de clientes	129
5.18.15.	Ejercicio: lista de alumnos	130
5.18.16.	Ejercicio: lista de artículos (con atributos optativos)	131
5.18.17.	Ejercicio: lista de componentes (un enfoque distinto)	132
5.18.18.	Ejercicio: listas con choice	134
5.18.19.	Ejercicio: listas de productos	135
5.18.20.	Ejercicio: validación de componentes	136
5.18.21.	Ejercicio: inventario	138
5.19.	Ejercicio tipo examen	142
5.20.	Ejercicio tipo examen (II)	145
5.21.	Examen	145
6.	Recuperación de información	147
6.1.	Introducción	147
6.2.	Fundamentos de DOM con Java	147
6.3.	Ejemplo de base	147
6.4.	La clase Document	148
6.5.	Ejercicios	149
6.5.1.	Ejercicio	149
6.5.2.	Ejercicio	151
6.5.3.	Ejercicio	152
6.5.4.	Ejercicio: extracción de información financiera	152
6.5.5.	Ejercicio	155
6.5.6.	Ejercicio	155
6.5.7.	Ejercicio	156
6.5.8.	Ejercicio	156
6.5.9.	Ejercicio	157
6.5.10.	Ejercicio	158
6.5.11.	Ejercicio	159
6.5.12.	Ejercicio	161
6.5.13.	Ejercicio	162
6.6.	Anexo	164
6.6.1.	Código Java	164
6.6.2.	Archivo XML	171
6.7.	Procesamiento con SAX	173
6.7.1.	Ejercicio: encontrar producto	174
6.7.2.	Ejercicio: precios	175
6.8.	Ejercicios para preparar examen	175
7.	Sindicación y transformación de contenidos	177
7.1.	Introducción	177
7.2.	RSS	177
7.2.1.	Formato de archivo RSS	177
7.2.2.	Ejercicio	179
7.3.	XPath	180
7.3.1.	Acceso a elementos	181
7.4.	Adaptación y transformación de XML	183
7.4.1.	CSS con XML	184
7.4.2.	Transformación de XML	185
7.4.3.	Ejercicio (carga de estilos)	186

7.4.4.	Ejercicio (conversion entre XMLs)	186
7.4.5.	Ejercicio (generación de atributos)	187
7.4.6.	Ejercicio (tabla HTML)	187
7.4.7.	Ejercicio (generacion)	188
7.4.8.	Ejercicio	188
7.4.9.	Ejercicio	189
7.4.10.	Ejercicio	190
7.4.11.	Ejercicio	191
7.4.12.	Ejercicio XSL, paso a paso	192
7.4.13.	Ejercicio: condiciones complejas	195
7.4.14.	Transformación en tabla	197
7.4.15.	Transformacion de pedidos	197
7.4.16.	Transformación de pedidos (II)	199
7.4.17.	Ejercicio (no se da la solución)	200
8.	Sistemas de gestión de información	201
8.1.	Introducción	201
8.2.	Niveles en la empresa	201
8.2.1.	Tipos de SGI informáticos	202
8.3.	Almacenamiento en los SGI	202
8.4.	Puntos de vista en los SGI	203
8.4.1.	Desarrollador	203
8.4.2.	Administrador	203
8.4.3.	Usuario	203
8.4.4.	Ejercicio	204
9.	Anexo: ejercicios sobre tablas	205
9.1.	Tabla 1	205
9.2.	Tabla 2	207
9.3.	Tabla 3	208
9.4.	Tabla 4	209
9.5.	Tabla 5	212
9.6.	Tabla 6	214
9.7.	Tabla 7	215
9.8.	Tabla 8	216
9.9.	Tabla 9	218
9.10.	Tabla 10	219
9.11.	Tabla 11	221
9.12.	Tabla 12	223
9.13.	Tabla 13	224
9.14.	Tabla 14	225
9.15.	Tabla 15	227
9.16.	Tabla 16	227
9.17.	Tabla 17	229
9.18.	Tabla 18	230
9.19.	Tabla 19	231
10.	Anexo: ejercicios sobre formularios	233
10.1.	Formulario 1	233
10.2.	Formulario 2	234
10.3.	Formulario 3	235
10.4.	Formulario 4	236
10.5.	Formulario 5	237
10.6.	Formulario 6	239

10.7. Formulario 7	241
10.8. Formulario 8	243
10.9. Formulario 9	245
10.10. Formulario 10	246
10.11. Formulario 11	248
10.12. Formulario 12	250
10.13. Formulario 13	252
10.14. Formulario 14	254
10.15. Formulario 15	256
10.16. Formulario 16	258
10.17. Formulario 17	259
10.18. Formulario 18	260
10.19. Formulario 19	262
10.20. Formulario 20	264
11. Anexo: ejercicios sobre XSLT	267
11.1. Fichero origen	267
11.2. Generación de lista con puntos	267
11.3. Recuperación de elementos pesados	268
11.4. Productos del edificio B	269
11.5. Tabla de localizaciones	269
11.6. Tablas con edificios separados	270
11.7. Productos del aula 6	272
11.8. Productos del edificio B	272
11.9. Condiciones múltiples: peso ligero y edificio A	272
11.10. Ejercicio de examen XSLT	273
11.11. Transformación de un XML bancario	274
11.11.1. Análisis del problema	275
11.11.2. Solución paso a paso	276

1.1 Historia

1.1.1 El pasado

Los lenguajes de marcas son bastante antiguos aunque solo se han popularizado con la llegada de Internet. De hecho, los comienzos de los lenguajes de marcas se pueden situar en el lenguaje SGML (Standard Generalized Markup Language). Dicho lenguaje en realidad nunca llegó al gran público, aunque sigue existiendo y usándose en entornos muy específicos. Lo más usado y conocido hoy día es HTML (HyperText Markup Language).

1.1.2 El presente

El lenguaje HTML ha ido cambiando mucho a lo largo del tiempo. La versión más actual es la conocida como HTML5 y es la que se contará en estos apuntes. 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 \(W3C\)](#). Todos sus estándares (que ellos llaman «Technical Reports») pueden descargarse gratuitamente en su web.

1.1.3 El futuro

El futuro de todas estas tecnologías sigue estando en sistemas abiertos, que facilitan la competencia y por tanto benefician al usuario. La web ha tenido un crecimiento exponencial debido a la existencias de las plataformas móviles, que han multiplicado los accesos a las páginas web (y que también han dificultado el diseño web, hablaremos más de ello en el tema sobre CSS).

1.2 Servidores web

Este apartado intenta ilustrar muy por encima como funciona un servidor web. Un servidor 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. Es decir cuando en el navegador escribimos «<http://www.google.es>» contactamos con un programa que se está ejecutando en un ordenador al que se le ha dado el nombre «www.google.es» y le pedimos establecer una conexión para intercambiar información usando el protocolo HTTP. Así, 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) y nos mostrará la página web asociada a ese código 404. Si todo va bien nuestro navegador recibe un código 200



Figura 1.1: Intercambio de mensajes entre cliente web (o navegador) y servidor web.

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 Internation Standards Organization emite estándares para la documentación.
2. El W3C (o World Wide Web 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.

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:
 - La etiqueta `<section>` permite marcar contenido de una página relacionado con un tema concreto.
 - `<article>` es una unidad de contenido sobre un tema específico el cual puede ser independiente de otros «artículos».
 - `<header>` se utiliza para indicar cual es la cabecera de un artículo o sección.
 - `<footer>` permite definir un «pie de página», normalmente con indicación de derechos de autor, fecha o datos similares.
 - `<address>` se usa para marcar información de contacto.
 - `<aside>` se usa para definir contenido con «una relación vaga con el resto de la página» (definición tomada del estándar).

- `<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 `<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».
- La etiqueta `<base>` define la URL raíz de toda la página. Permite cambiar fácilmente las URL de los enlaces de una página.
- Las etiquetas `<script>` y `<noscript>` se utilizan para marcar pequeños programas o la ausencia de ellos.
- El elemento `<main>` indica **el contenido principal de una página**

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 ``.

En un plano distinto se pueden encontrar las **listas de definiciones**. Con estos elementos se puede especificar una secuencia de términos para los cuales proporcionamos una definición. Su estructura es la siguiente:

- `<dl>` y `</dl>` marcan el inicio y el final de la lista de definiciones. Dentro de estas etiquetas pondremos los dos siguientes.
- `<dt>` y `</dt>` especifican el *término* que vamos a definir.
- `<dd>` y `</dd>` indican la definición asociada al término anterior.

Ejemplo:

```
<dl>
  <dt>Etiqueta</dt>
  <dd>Todo lo contenido...</dd>
  <dt>Elemento</dt>
  <dd>
    Se define así a todo el árbol
    de nodos comprendido
    entre dos etiquetas
```

```
de apertura y cierre.  
</dd>  
</dl>
```

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>Comprobar</li>  
          </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

Se desea crear una tabla que represente los datos del medallero de unas olimpiadas y que se muestre de forma parecida a lo que muestra la figura:

País	Oro	Plata	Bronce
USA	110	115	99
Total	219	247	206

```
<table border="1">
  <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:

```
<table border="1">
  <tbody>
    <tr>
      <td>A</td><td>B</td>
    </tr>
    <tr>
      <td>C</td>
      <td>
        <table>
          <tbody>
            <tr><td>D1</td></tr>
            <tr><td>D2</td></tr>
            <tr><td>D3</td></tr>
            <tr><td>D4</td></tr>
          </tbody>
        </table>
      </td>
    </tr>
  </tbody>
</table>
```

2.8.4 Ejercicio sobre tablas (II)

Crea una tabla con la estructura siguiente:

A						
<table><tr><td>B1</td><td rowspan="3"><table><tr><td>C1</td><td>C2</td></tr></table></td></tr><tr><td>B2</td></tr><tr><td>B3</td></tr></table>	B1	<table><tr><td>C1</td><td>C2</td></tr></table>	C1	C2	B2	B3
B1	<table><tr><td>C1</td><td>C2</td></tr></table>		C1	C2		
C1			C2			
B2						
B3						

2.8.5 Solución

Un posible HTML que resuelve esto sería:

```
<table border="1">
  <tbody>
```



2.8.6 Ejercicio sobre tablas (III)

Crea una tabla con la estructura siguiente:

<table><tr><td>A1</td></tr><tr><td>A2</td></tr></table>	A1	A2	<table><tr><td>B1</td><td><table><tr><td>B1-1</td></tr><tr><td>B1-2</td></tr></table></td></tr></table>	B1	<table><tr><td>B1-1</td></tr><tr><td>B1-2</td></tr></table>	B1-1	B1-2		
A1									
A2									
B1	<table><tr><td>B1-1</td></tr><tr><td>B1-2</td></tr></table>	B1-1	B1-2						
B1-1									
B1-2									
<table><tr><td>C1</td><td><table><tr><td>C1-1</td></tr><tr><td>C1-2</td></tr></table></td></tr></table>	C1	<table><tr><td>C1-1</td></tr><tr><td>C1-2</td></tr></table>	C1-1	C1-2	<table><tr><td>D1</td><td></td></tr><tr><td>D2</td><td></td></tr></table>	D1		D2	
C1	<table><tr><td>C1-1</td></tr><tr><td>C1-2</td></tr></table>	C1-1	C1-2						
C1-1									
C1-2									
D1									
D2									

2.8.7 Solución

Un posible HTML que resuelve esto sería:

```
<!DOCTYPE html>

<html>
<head>
  <title>Solución al ejercicio propuesto el viernes</title>
  <meta charset="utf-8">
</head>

<body>
<table border="1">
  <tbody>
    <tr>
      <td>
        <table border="1">
          <tbody>
            <tr><td>A1</td></tr>
            <tr><td>A2</td></tr>
          </tbody>
        </table>
      </td>
      <td>
        <table border="1">
          <tbody>
            <tr>
              <td>B1</td>
              <td>
                <table border="1">
                  <tr><td>B1-1</td></tr>
                  <tr><td>B1-2</td></tr>
                </table>
              </td>
            </tr>
          </tbody>
        </table>
      </td>
    </tr>
    <tr>
      <td>
        <table>
          <tbody>
            <tr>
              <td>
                <table>
                  <tbody>
                    <tr><td>C1-1</td></tr>
                    <tr><td>C1-2</td></tr>
                  </tbody>
                </table>
              </td>
              <td>
                <table>
                  <tbody>
                    <tr><td>D1</td></tr>
                    <tr><td>D2</td></tr>
                  </tbody>
                </table>
              </td>
            </tr>
          </tbody>
        </table>
      </td>
    </tr>
  </tbody>
</table>
```



```

        <tbody>
        <tr>
            <td>C1</td>
            <td>
                <table border="1">
                    <tbody>
                        <tr><td>C1-1</td></tr>
                        <tr><td>C1-2</td></tr>
                    </tbody>
                </table>
            </td>
        </tr>
        </tbody>
    </table>
</td>
<td>
    <table border="1">
        <tbody>
            <tr><td>D1</td></tr>
            <tr><td>D2</td></tr>
        </tbody>
    </table>
</td>
</tr>
</tbody>
</table>

</body>
</html>

```

2.8.8 Ejercicio sobre tablas (V)

Crea una tabla con la estructura siguiente

2.8.9 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>
    </tr>
    <tr>
        <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>
    </tr>
    <td>
      Paro:11%
    </td>
  </tr>
</table>
</td>
</tr>
</tbody>
</table>

```

2.8.10 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%

2.8.11 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 rowspan="2">UE</td>
      <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>
  </tbody>
</table>

```

```

        </tr>
      </table>
    </td>
  </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>

```

2.8.12 Ejercicio sobre tablas (VII)

Crea una tabla con la estructura siguiente

A						
<table><tr><td>B1</td></tr><tr><td>B2</td></tr><tr><td>B3</td></tr></table>	B1	B2	B3	<table><tr><td>C1</td><td>C2</td></tr></table>	C1	C2
B1						
B2						
B3						
C1	C2					

2.8.13 Solución

```
<table border="1">
  <tbody>
    <tr>
      <td>A</td>
    </tr>
    <tr>
      <td>
        <table border="1">
          <tbody>
            <tr>
              <td>
                <table border="1">
                  <tbody>
                    <tr>
                      <td>B1</td>
                    </tr>
                    <tr>
                      <td>B2</td>
                    </tr>
                    <tr>
                      <td>B3</td>
                    </tr>
                  </tbody>
                </table>
              </td>
            </tr>
          </tbody>
        </table>
      </td>
      <td>

```

```
        </table>
      </td>
      <td>
        <table border="1">
          <tbody>
            <tr>
              <td>C1</td>
              <td>C2</td>
            </tr>
          </tbody>
        </table>
      </td>
    </tr>
  </tbody>
</table>
</td>
</tr>
</tbody>
</table>
```

2.8.14 Ejercicio sobre tablas (VIII)

Crea una tabla con la estructura siguiente

A1	A2	A3
B1		
B2		

2.8.15 Solución

```
<table border="1">
  <tr>
    <td>
      <table border="1">
        <tbody>
```

```

        <tr>
          <td>A1</td>
          <td>A2</td>
          <td>A3</td>
        </tr>
      </tbody>
    </table>
  </td>
</tr>
<tr>
  <td>
    <table border="1">
      <tbody>
        <tr>
          <td>B1</td>
        </tr>
        <tr>
          <td>B2</td>
        </tr>
      </tbody>
    </table>
  </td>
</tr>
</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
```

Si se desea que un control de tipo checkbox o radio aparezca marcado por defecto se debe añadir el atributo `multiple="multiple"`.

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`.

Si se desea que una opción (o varias si usamos el selector múltiple) aparezca marcada se debe usar `selected="selected"`.

2.9.5 Textareas

Permiten introducir textos muy largos:

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


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 4b
Celda 3b	Celda 4c
Celda 3c	

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

<table><tr><td>1a</td><td>1b</td></tr></table>	1a	1b	<table><tr><td>2a</td><td>2b</td></tr><tr><td>2c</td><td>2c</td></tr></table>	2a	2b	2c	2c
1a	1b						
2a	2b						
2c	2c						
3a	<table><tr><td>4a1</td><td>4a2</td><td>4a3</td></tr></table>	4a1	4a2	4a3			
4a1	4a2	4a3					
<table><tr><td>3b1</td><td>3b2</td></tr><tr><td>3b3</td><td>3b4</td></tr></table>	3b1	3b2	3b3	3b4	4b		
3b1	3b2						
3b3	3b4						

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: Más de 400 euros ▾
☐ Con enfermedad profesional
☐ Con padres a cargo
☐ Con hijos a cargo

2.10.8 Solución

El HTML siguiente produce el resultado que nos piden

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Formulario fiscal</title>
</head>
<body>

<form>
  <fieldset>
    <legend>
      Datos fiscales
    </legend>
    <input type="checkbox" id="enparo"
      name="sit_laboral">
    <label for="enparo">En paro</label>
    <br/>
    <input type="checkbox" id="autonomo"
      name="sit_laboral">
    <label for="autonomo">Autónomo</label>
    <br/>
    <input type="checkbox" id="c_ajena"
      name="sit_laboral">
    <label for="c_ajena">Por c.ajena</label>
    <br/>
  </fieldset>
  <fieldset>
    <legend>Datos personales</legend>
    <label for="nombre">Nombre</label>
    <input type="text" id="nombre">
    <br/>
    <label for="apellidos">Apellidos</label>
    <input type="text" id="apellidos">
    <br/>
    <label for="sueldo">Sueldo</label>
    <select id="sueldo"></select>
  </fieldset>
</form>

```

```

    <option>Más de 400 euros</option>
    <option>Menos de 400 euros</option>
    <option>Variable</option>
  </select>
<br/>
<input type="checkbox" id="con_ep">
<label for="con_ep">
  Con enfermedad profesional
</label> <br/>
<input type="checkbox" id="con_padres">
<label for="con_padres">
  Con padres a cargo
</label> <br/>
<input type="checkbox" id="con_hijos">
<label for="con_hijos">
  Con hijos a cargo
</label> <br/>
</fieldset>
</form>
</body>
</html>

```

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

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Formulario</title>
</head>
<body>
<form>
  <fieldset>
    <legend>Datos fiscales</legend>
    <select multiple="multiple">
      <option>Automoción</option>
      <option selected="selected">

```

```

        Metal
    </option>
    <option>Informática</option>
    <option selected="selected">
        Finanzas
    </option>
</select>
<br/>
<input type="checkbox" id="autonomo">
<label for="autonomo">
    Autónomo
</label>
<input type="checkbox" id="c ajena">
<label for="c ajena">
    Por c. ajena
</label>
<input type="checkbox" id="nosabe">
<label for="nosabe">
    No sabe
</label>
<br/>
Describa su función:<br/>
<textarea>Escriba aquí</textarea>
</fieldset>
</form>

</body>
</html>

```

2.10.10 Enunciado

Crea un formulario como este

2.10.11 Solución

```

<form>
    <fieldset>
        <legend>
            Laboral
        </legend>
        <input type="checkbox"
            name="contratos[]"
            id="ajena">
            Por cuenta ajena
    </fieldset>

```

```

<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>
```

2.11 Examen

- El grupo DAM-1 realizará el Jueves 10 de noviembre de 2016
- El grupo ASIR-1 realizará el examen el Jueves 17 de noviembre de 2016.

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 Sintaxis

CSS funciona mediante reglas, de las cuales se muestra un ejemplo a continuación.

```
h1, h3 {  
    background-color: blue;  
    color: white;  
}
```

En las reglas tenemos tres cosas:

1. Los selectores. En este caso queremos modificar como van a quedar todos los `<h1>` y `<h3>`
2. Las propiedades. En el ejemplo se pretende cambiar el color de fondo y el color de las letras.
3. Los valores. En este caso se pone el valor `blue` para la propiedad `background-color` y el valor `white` para la propiedad `color`

3.3 Los atributos `class` e `id`

A menudo tendremos que hacer en cambios en un grupo de elementos, pero a veces no serán todos los elementos de una misma clase. Por ejemplo, puede que queramos cambiar un elemento en concreto. Para poder hacer cambios *a un solo elemento* tendremos que haber puesto el atributo `id` como muestra el ejemplo siguiente:

```
<h1>Encabezamiento</h1>
.... texto ...

<h1 id="noticia_del_dia">Otro encabezamiento</h1>
.... texto ...

<h1>Y otro más</h1>
.... texto ...
```

Si luego desde CSS queremos modificar solo el encabezamiento con la noticia del día deberemos hacer esto

```
h1#noticia_del_dia{
    font-weight: bold;
}
```

Obsérvese que hemos usado la almohadillas (#) para decir «queremos seleccionar el h1 cuyo id es noticia_del_dia» y ponerlo en negrita. Debe señalarse que es importante que en el HTML **NO DEBE HABER DOS ELEMENTOS CON EL MISMO ID**

Si en vez de uno queremos aplicar un cambio a **un conjunto de elementos** deberemos ir al HTML y ponerles a todos ellos un atributo `class` con el mismo valor en todos ellos. Por ejemplo:

```
<h1 class="titular_economia">Encabezamiento</h1>
.... texto ...

<h1>Otro encabezamiento</h1>
.... texto ...

<h1 class="titular_economia">Y otro más</h1>
.... texto ...
```

Con este HTML podemos crear un CSS como este que cambia solo los h1 cuyo `class` tiene el valor `titular_economia`

```
h1.titular_economia{
    font-weight: bold;
}
```

Se puede poner el mismo `class` a distintos elementos, por ejemplo así

```
<h1 class="titular_economia">Encabezamiento</h1>
.... texto ...

<h1>Otro encabezamiento</h1>
.... texto ...

<h2 class="titular_economia">Y otro más</h2>
.... texto ...
```

Y luego usar un CSS como este:

```
h1.titular_economia, h2.titular_economia{
    font-weight: bold;
}
```

Este último CSS se puede resumir

```
.titular_economia{
  font-weight: bold;
}
```

Esta regla dice «seleccionar todos los elementos cuyo class sea titular_economia» y ponerlos en negrita. Estos mecanismos de resumen son muy útiles y facilitan mucho la tarea del diseñador CSS.

3.4 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, recordemos que 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

3.4.1 Ejercicio propuesto

Crea una página con la siguiente estructura.

- En la parte superior debe haber dos cajas. Una de ellas, a la izquierda, ocupa el 33 % y contiene el lema. La otra, a la derecha, contiene enlaces y ocupa el 66 %.
- En la parte central 3 cajas. La de la izquierda contiene publicidad y ocupa el 25 %. La central tiene el contenido y ocupa el 50 %, la de la derecha tiene más publicidad y ocupa el 25 %.
- En la parte de abajo hay una barra **que no se mueve nunca** y que ocupa el 100 %. Contiene el mensaje de copyright de la empresa.

Un posible HTML sería el siguiente:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet"
        type="text/css"
        href="estilo.css">
  <meta charset="utf-8">
  <title>Page Title</title>
</head>
<body>
<div id="lema">
  Lema...
</div>
<div id="enlaces">
  Enlaces....
</div>
<div id="publi1">
  Publicidad
</div>
<div id="contenido">
  Contenido...
</div>
<div id="publi2">
  Publicidad...
</div>
<div id="copyright">
  &copy; IES Maestre...
</div>
</body>
</html>
```

Y un posible CSS sería este:

```
h1#bienvenida{
  color: rgb(66, 66, 220);
  font-family: "Comic Sans MS";
}

p{
  background-color: rgb(240, 250, 230);
  margin-left:15%;
  margin-right: 15%;
  padding:5em;
  font-weight: bold;
  font-style: italic;
  text-align: justify;
}

tr.par{
  background-color: rgb(180, 180, 180);
}

tr.impar{
  background-color: rgb(220,220, 220);
}
```

3.4.2 Ejercicio propuesto (II)

Crear una página con la siguiente estructura:

- En la parte izquierda hay una barra de enlaces. Ocupa el 25 % y está **fija**.
- En la zona superior hay una capa con el lema de la empresa. Ocupa el 75 % y no se debe ver tapada por los enlaces.
- En la zona central hay dos capas. Una de ellas es el contenido y ocupa aproximadamente el 50 %. A su lado hay una capa con publicidad que ocupa el 20 %.
- En la zona inferior hay una capa con el copyright de la empresa. Ocupa el 50 % y se ve junto al margen derecho de la página.

Un posible HTML sería el siguiente:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet"
    type="text/css" href="estilo.css">
  <meta charset="utf-8">
  <title>Ejercicio</title>
</head>

<body>
<div id="enlaces">
Enlaces enlaces
</div>

<div id="lema">
Progresando con el tiempo,
mejorando con la experiencia
</div>

<div id="contenedor_central">
<div id="contenido">
Texto principal de la página
</div>
<div id="publi">
Publicidad publicidad
</div>
</div>

<div id="copyright">
&copy; IES Maestre 2016
</div>
</body>
</html>
```

Y un posible CSS sería este:

```
div#enlaces{
  position: fixed;
  top:0px;
  left:0px;
  width:20%;
}
```

```
div#lema{
  float:right;
  width:70%;
}
div#contenedor_central{
  float:right;
  width:75%;
  margin-right: 0px;
}
div#contenido{
  font-family: Helvetica;
  float:left;
  width: 62%;
}
div#publi{
  margin-right: 0px;
  float:right;
  width:32%;
}
div#copyright{
  float:right;
  width: 50%;
}
```

3.4.3 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.

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>
```

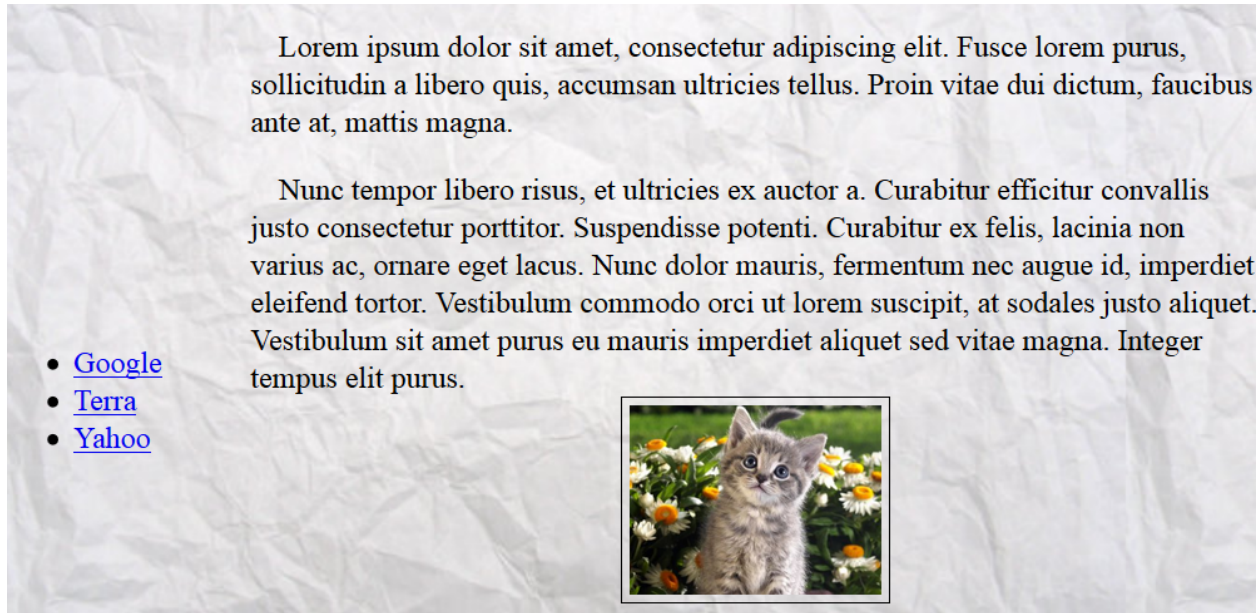


Figura 3.1: Resultado final

```

</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;
}

```

3.4.4 Ejercicio 2 de maquetación

Conseguir una página como esta



3.4.5 HTML

```

<div id="contenedorglobal">
  <div id="cabecera">
    <span id="marcacabecera">
      ACME
    </span>
  </div>
</div>

```



```

        <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>

```

3.4.6 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);
}
```

3.4.7 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>

```

3.4.8 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>

```

3.4.9 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>
</body>
</html>
```

3.5 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.

Deben recordarse algunas cosas:

- Un elemento `float` lo que hace es *dar permiso a otros elementos* para que ocupen el espacio que le ha sobrado.
- Cuando se usan varios elementos `float` es posible que otros elementos tengan que hacer `clear: both;` para asegurarnos de que dejen de aprovechar el espacio sobrante.
- Es muy frecuente que haya que «crear contenedores extra» para conseguir colocar cajas utilizando elementos `float`.

Por ejemplo, supongamos el siguiente HTML:

```
<div id="caja1">
  Texto...
</div>
<div id="caja2">
  Texto...
</div>
<div id="caja3">
  Texto...
</div>
```

```
<div id="caja4">
  Texto...
</div>
```

Y supongamos que se necesita que las cuatro cajas queden colocadas verticalmente en pantalla y siguiendo el orden «caja4», «caja3», «caja1» y «caja2». Una posible solución sería crear dos contenedores extra de la manera siguiente:

```
<div id="contenedor1">
  <div id="caja1">
    Texto...
  </div>
  <div id="caja2">
    Texto...
  </div>
</div> <!--Fin del contenedor1-->
<div id="contenedor2">
  <div id="caja3">
    Texto...
  </div>
  <div id="caja4">
    Texto...
  </div>
</div> <!--Fin del contenedor2-->
```

Y ahora haríamos lo siguiente:

- El contenedor 1 flotará hacia la derecha. Dentro de él la caja 1 flotará a la izquierda y la caja 2 flotará a la derecha.
- El contenedor 2 flotará hacia la izquierda. Dentro de él la caja 4 flotará hacia la izquierda y la caja 3 hacia la derecha.

Probemos el siguiente CSS:

```
#contenedor{
  float:right;
  width:48%;
}
#contenedor2{
  float:left;
  width:48%;
}
#caja1, #caja4{
  float:left;
  width:48%;
}
#caja2, #caja3{
  float:right;
  width:48%;
}
```

El resultado será:

3.5.1 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

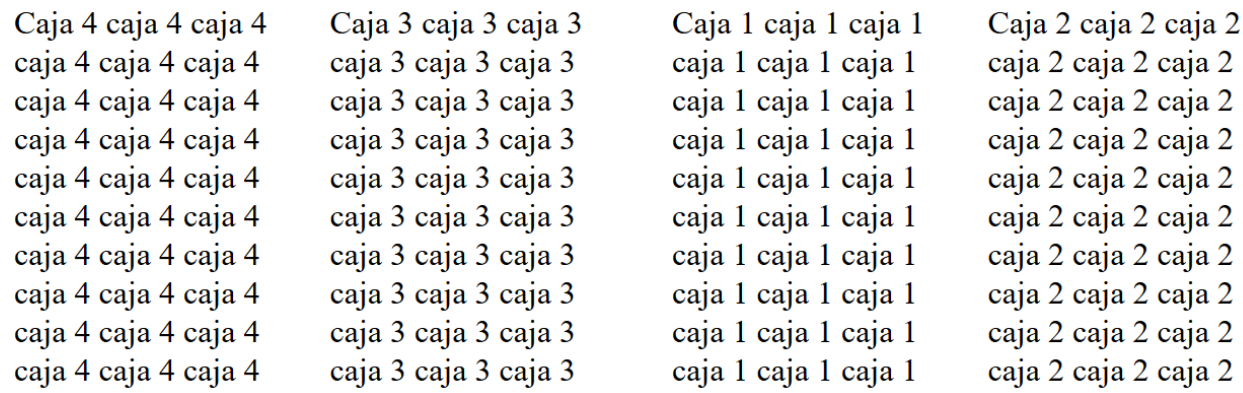


Figura 3.2: Cuatro cajas manipuladas con float

anchura del 100 %.

3.6 Maquetación avanzada con grid-layouts

La llegada de CSS 3 ha supuesto un gran cambio en la forma de maquetar páginas, ya que esta versión ha incluido una novedad llamada «grid-layouts», que nos permiten varias características interesantes:

1. Un elemento (por ejemplo un `div`) se puede dividir en forma de tabla
2. Dicha tabla puede tener columnas y/o filas de distinto tamaño.
3. Un elemento hijo puede ocupar la celda que le toque o llenar un área de varias celdas.

Para poder utilizarlo necesitaremos:

- Un contenedor principal (en concreto un `div`). Este contenedor se portará como una tabla en la que definiremos las filas y las columnas
- Un conjunto de elementos (normalmente otros `div`) que irán dentro del contenedor principal. Estos elementos se portarán como «celdas flexibles», ya que podremos colocar cada celda donde queramos y hacer que ocupe las filas y columnas que queramos.

Supongamos el siguiente HTML

```
<div id="contenedor">
  <div class="celda" id="a">Celda A</div>
  <div class="celda" id="b">Celda B</div>
  <div class="celda" id="c">Celda C</div>
  <div class="celda" id="d">Celda D</div>
  <div class="celda" id="e">Celda E</div>
  <div class="celda" id="f">Celda F</div>
</div>
```

Si no le hacemos nada se verá lo siguiente:

Vamos a añadir bordes a la celdas para que se vea mejor el efecto. Usemos el siguiente CSS

```
.celda{
    border: solid black 1px;
}
```

Celda A
Celda B
Celda C
Celda D
Celda E
Celda F

Figura 3.3: Grid sin maquetar

Celda A
Celda B
Celda C
Celda D
Celda E
Celda F

Figura 3.4: Celdas con borde

El resultado queda así:

Supongamos que queremos que el `div` cuyo `class` es `contenedor` se porte como una tabla de 5 por 5. ¡Recuérdese que solo tenemos 6 celdas! A pesar de eso, queremos una distribución como esta:

Figura 3.5: Tabla de 5 x 5

Y ahora supongamos que dentro de esa tabla queremos repartir los elementos de manera que quede más o menos como lo siguiente:

Analizando lo que se pide se observa que:

- La «rejilla invisible» ocupa todo el ancho de la pantalla y es más alta de lo normal (pondremos una medida vertical en píxeles que sea razonablemente grande para poder apreciar el efecto)
- La celda A empieza en la fila 1 y llega hasta la 3. Empieza en la columna 4 y llega hasta la 6 **que sabemos que no existe, sino que es el límite de la tabla**.
- La celda B empieza en la fila 3 y llega hasta la 4. En columnas va de la 2 a la 5.
- La celda C va de la fila 1 a la 6 y está solo en la columna 1.
- La D va de la fila 3 a la 6 y solo ocupa la 5.
- La E va de la fila 1 a la 3 y de la columna 2 a la 4.
- La celda F va de la fila 4 a la 6 y de la columna 2 a la 5.



Figura 3.6: Tabla de 5 x 5

Además añadiremos algún color a tales «celdas» para que podamos ver el área que ocupan.

```
#contenedor{
  display: grid;
  grid-template-rows: 20% 20% 20% 20% 20%;
  grid-template-columns: 20% 20% 20% 20% 20%;
  width:100%;
  height:640px;
}
.celda{
  border: solid black 1px;
}
#a{
  grid-row: 1/3;
  grid-column: 4/6;
  background-color: rgb(200, 200, 200);
}
#b{
  grid-row: 3/4;
  grid-column : 2/5 ;
  background-color: rgb(210, 240, 200);
}
#c{
  grid-row: 1/6;
  grid-column : 1 ;
  background-color: rgb(210, 220, 200);
}
#d{
  grid-row: 3/6 ;
  grid-column :5 ;
  background-color: rgb(210, 230, 230);
}
#e{
```



```

    grid-row: 1/3;
    grid-column : 2/4 ;
    background-color: rgb(210, 240, 240);
}
#f{
    grid-row: 4/6;
    grid-column : 2/5 ;
    background-color: rgb(240, 240, 240);
}

```

3.7 Media queries

Las *media queries* (algo así como «consultas sobre el tipo de medio en el que se va a mostrar/procesar el HTML») forman parte de CSS 3, por lo que solo deben utilizarse en navegadores relativamente modernos.

Las media queries permiten hacer diversas comprobaciones. Si se cumplen dichas comprobaciones se ejecutará un CSS u otro. Por ejemplo, supongamos que queremos tener dos estructuras diferentes de página en función de si se va a mostrar el HTML en pantalla o en papel.

Partamos del siguiente HTML:

```

<div id="caja1">
  Caja 1
</div>

<div id="caja2">
  Caja 2
</div>

<div id="caja3">
  Caja 3
</div>

```

Y ahora supongamos que cuando se muestra el HTML en pantalla queremos que tenga un color de fondo, pero que si se va a imprimir no tengan ningún fondo (para ahorrar tinta, por ejemplo). Sabiendo que existen dos tipos de medios llamados `screen` y `print` podemos usar un CSS como este:

```

@media screen {
    div{
        font-size: xx-large;
        border: double black 1px;
        background-color: grey;
        margin: 30px;
    }
}
@media print {
    div{
        font-size: small;
        margin: 15px;
    }
}

```

Este CSS puede hacer dos cosas distintas:

- Cuando el HTML se muestra en pantalla las cajas tendrán todas un borde, un fondo, un tipo de letra grande y mucho margen.
- Sin embargo, cuando se va a imprimir no hay fondos, el margen es mucho más pequeño y el tipo de letra también.

A continuación se muestra una captura de lo que muestra el navegador:



Figura 3.7: HTML mostrado en pantalla

Y también se muestra una captura de lo que muestra el navegador cuando vamos a imprimir:



Figura 3.8: HTML para imprimir

Uno de los usos más comunes de las *media queries* es la comprobación de la resolución en la que se está visualizando el HTML y en función de ello mostrar distintas estructuras al usuario. Por ejemplo, supongamos que deseamos mostrar nuestra página anterior de dos formas en pantalla.

- Cuando la resolución sea de 800 px o más haremos que `caja1` y `caja2` estén una al lado de la otra ocupando cada una la mitad de la pantalla aproximadamente.
- Cuando la resolución sea de 799px o menos `caja1`, `caja2` y `caja3` se mostrarán una encima de otra pero con un margen entre ellas de 40px.

Para conseguir esto hay predicados de utilidad que podemos combinar con los que acabamos de ver para conseguir lo que deseamos. Dos de los más útiles son `min-width` y `max-width`. Veamos como se usan para conseguir lo que nos piden:

```
/* Si estamos en una pantalla y la anchura mínima que podemos
 * usar es 800px...*/
@media screen and (min-width:800px) {
  div{
```

```

    border:solid black 1px;
}
/*... entonces dividir caja 1 y caja 2 en dos mitades*/
#caja1{
    float:left;
    width:48%;
}
#caja2{
    float:right;
    width:48%;
}
#caja3{
    clear:both;
}
}
/* Sin embargo, si estamos en una pantalla y como máximo
 * tenemos 799px (es decir, una pantalla más bien estrecha)...*/
@media screen and (max-width:799px) {
    div{
        border:solid black 1px;
        margin-top: 40px;
        text-align: center;
    }
}

```

El resultado de este CSS en una pantalla grande es:



Figura 3.9: Página para una resolución grande

Sin embargo, en una pantalla pequeña (se puede cambiar el tamaño de la ventana del navegador para simular el ejemplo):



Figura 3.10: Página para una resolución pequeña

Hay diversas cosas que podemos comprobar:

- `min-width: 100px` o `max-width: 900px` para ver las anchuras mínimas o máximas que nos ofrece un dispositivo.
- `@media screen`, `@media print` o `@media: handheld` para comprobar si el HTML se va a mostrar en una pantalla, se va a imprimir o se muestra un dispositivo portátil como móvil o tablet. Hay otros `@media` como `@media braille` o `@media tv`, pero se usan menos.

- `orientation: portrait` u `orientation: landscape` para saber si la pantalla está en horizontal o en vertical.

3.8 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`.

3.9 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.
- Desde hace poco se pueden indicar también con `hsl(num, num, num)`

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.

3.10 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.

3.11 Alineación del texto

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

3.12 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`

3.13 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».

3.14 Selectores

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>
```

3.14.1 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

3.14.2 Solución p.destacado

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

3.14.3 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.

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 3.11: 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 3.12: Resultado

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 3.13: Resultado

3.14.4 Solución `p.destacado > li.elemento_numeracion`

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

¿Por qué los cambios no afectan a ninguno?

3.14.5 Solución `p.destacado > .elemento_numeracion`

Ahora sí veremos que algo cambia, en concreto los últimos `` que llevan el `class=elemento_numeracion`, ya que *sí son hijos directos* de un elemento que lleva un `class=destacado`.

3.14.6 Solución `.destacado > #id1`

Ahora el resultado es este

¿Por qué ahora sí funciona?

3.15 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.

3.15.1 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.

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 3.14: Resultado

```
<!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

3.15.2 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 3.15: 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/small» (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)

3.15.3 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>
```

Se puede destacar texto usando lo siguiente:

Página con Bootstrap

Este párrafo es muy importante

Este párrafo es normal

Figura 3.16: Párrafo destacado

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

3.16 Ejercicio responsive I

Hacer una página cuyo diseño se adapte automáticamente en función de la resolución. Dicha página tendrá 3 cajas cuyos id serán A, B y C. El comportamiento de las cajas será el siguiente:

- Si la página se visualiza en una pantalla de 400px o menos las 3 cajas se limitarán a mostrarse una encima de la otra.
- Si la pantalla tiene un tamaño de entre 401px y 800px las cajas A y B se mostrarán al principio, cubriendo cada una una anchura del 50 % (quizá haya que ajustar a un porcentaje menor). La caja C se mostrará debajo de A y B.
- Si la pantalla tiene 801px o más A, B y C se mostrarán una al lado de la otra. A cubrirá un 20 %, B un 20 % y C un 58 %.

A continuación se muestra el resultado aproximado que se debe conseguir:

Caja A caja A caja A caja A caja A caja A
caja A caja A caja A caja A caja A caja A

Caja B caja B caja B caja B caja B caja B
caja B caja B caja B caja B caja B caja B

Caja C caja C caja C caja C caja C caja C
caja C caja C caja C caja C caja C caja C

Figura 3.17: Resultado para pantallas pequeñas

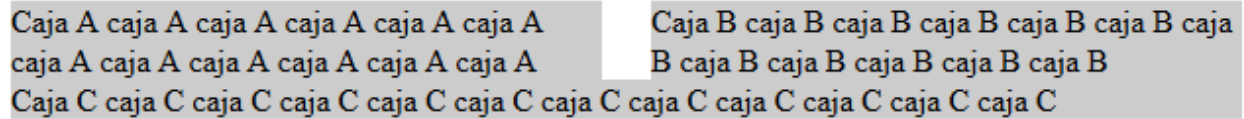


Figura 3.18: Resultado para pantallas medianas

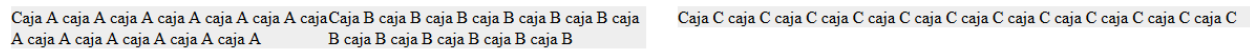


Figura 3.19: Resultado para pantallas grandes

Un posible HTML que resolviera esto sería el siguiente:

```
<body>
  <div id="A">
    Caja A caja A caja A caja A
    caja A caja A caja A caja A
    caja A caja A caja A caja A
  </div>
  <div id="B">
    Caja B caja B caja B caja B
    caja B caja B caja B caja B
    caja B caja B caja B caja B
  </div>
  <div id="C">
    Caja C caja C caja C caja C
    caja C caja C caja C caja C
    caja C caja C caja C caja C
  </div>
</body>
```

Y el CSS que lo acompaña sería este:

```
/* En pantallas pequeñas...*/
@media screen and (max-width:400px) {
  /* ...no hacemos nada, dejamos que el
   * navegador "apile" las cajas. Simplemente
   * cambiamos el margen y el color para ver
   * que nos funciona*/
  div {
    margin-top: 40px;
    background-color: #eeeeee;
  }
} /*Fin del media para ventanas pequeñas*/
/* En pantallas medianas...*/
@media screen and (min-width: 401px) and (max-width:800px) {
  /* ...haremos "flotar" a A y B...*/
  #A{
    float: left;
    width:48%;
  }
  #B{
    float:right;
  }
}
```

```

        width: 48%;
    }
    /* ...y haremos que C "limpie" el espacio sobrante*/
    #C{
        clear: both;
    }
    /* También ponemos un color distinto
    * pero solo para ver si lo hacemos bien*/
    div{
        background-color: #cccccc;
    }
} /*Fin del media para ventanas medianas*/
/* Si estamos en pantallas grandes...*/
@media screen and (min-width:801px) {
    /*...entonces A y B flotan hacia la izquierda...*/
    #A, #B{
        width:20%;
        float:left;
    }
    /* y C flota a la derecha llevándose el espacio
    * que sobre. No lo ajustamos al 60% para
    * evitar desbordamientos*/
    #C{
        width:58%;
        float: right;
    }
    /* También volvemos a cambiar el color para
    * las comprobaciones*/
    div{
        background-color: #eeeeee;
    }
}

```

3.17 Ejercicio responsive II

Hacer una página cuyo diseño se adapte automáticamente en función de la resolución. Dicha página tendrá 4 cajas cuyos id serán A, B, C y D. El comportamiento de las cajas será el siguiente:

- Para todos los casos hay una rejilla contenedora de 4 filas (todas de la misma altura) y 3 columnas (de anchos 20 %, 20 % y 60 %).
- Si la pantalla tiene menos de 800px se mostrará una distribución como la que se muestra en la figura II-1.
- Si la pantalla tiene menos de 800px se mostrará una distribución como la que se muestra en la figura II-2.

A continuación se muestra el HTML:

```

<div id="contenedor">
  <div id="A">
    Caja A
  </div>
  <div id="B">
    Caja B
  </div>
  <div id="C">
    Caja C
  </div>

```

[illegible]

Figura 3.20: Figura II-1. (Para pantallas estrechas)

[illegible]

Figura 3.21: Figura II-2 (Para pantallas anchas)

```

    <div id="D">
        Caja D
    </div>
</div>

```

Y un posible CSS:

```

/* Todas las cajas tienen borde siempre*/
div{
    border: solid 1px black;
}

div#contenedor{
    display: grid;
    grid-template-rows: 25% 25% 25% 25%;
    grid-template-columns: 20% 20% 60%;
}

@media screen and (min-width:800px){

    /* Esto no hacía falta, se usa
    * para comprobar que nos sale
    * bien al estrechar o ensanchar
    * la "pantalla"*/
    #A, #B, #C, #D{
        background-color: rgb(240, 240,220);
    }

    #A{
        grid-row: 1;
        grid-column:1/4 ;
    }
    #B{
        grid-row:2 ;
        grid-column:1/4 ;
    }
    #C{
        grid-row: 3/5;
        grid-column: 1/3;
    }
    #D{
        grid-row: 3/5;
        grid-column:3/4 ;
    }
} /* Fin del media para max-width 800px*/

@media screen and (max-width:799px){
    /* Esto no hacía falta, se usa
    * para comprobar que nos sale
    * bien al estrechar o ensanchar
    * la "pantalla"*/
    #A, #B, #C, #D{
        background-color: rgb(220, 240, 230);
    }

    #A{
        grid-row: 1;
        grid-column:1/3 ;
    }
    #B{
        grid-row: 1;

```

```
        grid-column: 3 ;
    }
    #C{
        grid-row: 2;
        grid-column: 1/5 ;
    }
    #D{
        grid-row: 3/4;
        grid-column: 1/5 ;
    }
} /* Fin del media para min-width 799px*/
```


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]
}
document.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
}

/* 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,
  fiijo: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>
<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.

4.14.1 HTML del configurador

4.14.2 JS del configurador (con JQuery (DAM))

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);
    }
  </style>
</head>
<body>
```

```

        #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=$( "#alerones" ).prop("checked")
    var marcado_monovolumen=$( "#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>
    .muestrarrojo{
      background-color:red;
    }
    .muestraverde{
      background-color: green;
    }
    .muestraazul{
      background-color: blue;
    }
    .muestragris{
      background-color: grey;
    }
    .muestrablanca{
      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", "muestrablanco",
                "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 ("muestrablanco")
}

```

4.16 Otro configurador de coches

Sin utilizar JQuery se desea crear un configurador de coches en JS que responda a las siguientes premisas:

- Hay dos modelos a elegir: el modelo A cuesta 7000 euros y el modelo B cuesta 9000.
- Se pueden elegir dos tipos de motor. El motor de gasolina cuesta 2000 y el diésel 5000 euros.
- Se pueden elegir 0, 1, muchos o todos los extras siguientes: Pintura metalizada por 1000 euros más, pack de sonido por 500 euros más y pack de seguridad por 1000 euros más

4.16.1 Configurador en HTML

```

<form>
    <h3>Elija un modelo</h3>
    <input type="radio" name="modelo"
        id="modelo_a">Modelo A<br/>
    <input type="radio" name="modelo"
        id="modelo_b">Modelo B<br/>
    <h3>Elija un tipo de motor</h3>
    <input type="radio" name="motor"
        id="gasolina">Gasolina<br/>
    <input type="radio" name="motor"
        id="diesel">Diésel<br/>
    <h3>Elija extras</h3>
    <input type="checkbox" name="extras"

```



```

        id="metalizada">Pintura metalizada<br/>
<input type="checkbox" name="extras"
        id="sonido">Extra sonido<br/>
<input type="checkbox" name="extras"
        id="seguridad">Extra seguridad<br/>
<input type="submit" value="Calcular"
        onclick="calcular();return false;">
</form>
<div id="zonaresultados"></div>

```

4.16.2 Configurador en JS

```

function esta_checked(id) {
    var control;
    control=document.getElementById(id);
    if (control.checked) {
        return true;
    } else {
        return false;
    }
}

function calcular() {
    var preciototal    =0;
    var preciodelo     =0;
    var preciomotor    =0;
    var precioextras   = 0;
    if ( (!esta_checked("modelo_a"))
        && (!esta_checked("modelo_b")) ) {
        alert("Debe marcar un modelo");
        return ;
    }
    if (esta_checked("modelo_a")) {
        preciodelo=7000;
    }
    if (esta_checked("modelo_b")) {
        preciodelo=9000;
    }
    if (esta_checked("gasolina")) {
        preciomotor=2000;
    }
    if (esta_checked("diesel")) {
        preciomotor=4000;
    }
    if (esta_checked("metalizada")) {
        precioextras=1000;
    }
    if (esta_checked("sonido")) {
        precioextras=precioextras+500;
    }
    if (esta_checked("seguridad")) {
        precioextras=precioextras+1000;
    }
    preciototal=preciodelo+preciomotor+precioextras;

    var zonaresultados;

```

```

    zonaresultados=document.getElementById(
        "zonaresultados");
    zonaresultados.innerHTML="Precio:"+preciototal;
}

```

4.17 Calculo de impuestos

Se desea crear una pequeña aplicación web que permita calcular los impuestos que se deben pagar a partir de unos datos, en concreto:

- Se debe introducir el salario anual en un campo de tipo `number`.
- Hay dos radios que permiten indicar si el contribuyente tiene hijos o no.
- Hay dos checkboxes que permiten saber si el contribuyente tiene derecho a algunas modificaciones llamadas B1 y B2.

Las reglas para el calculo son:

- Si el salario es menor de 20000 los impuestos son 0.
- Si el salario está entre 20001 y 30000 los impuestos son el 10 %.
- Si el salario está entre 30001 y 50000 los impuestos son el 20 %.
- Si el salario es mayor de 50000 se paga el 38 %
- Si se tienen hijos los impuestos se reducen en 180 euros.
- Si se tiene la bonificación B1 los impuestos se reducen en 355 euros.
- Si se tiene la bonificación B2 los impuestos se reducen en 560 euros.
- Se recuerda que las bonificaciones B1 y B2 son compatibles (de hecho hemos dicho que están en checkboxes).
- La cantidad de impuestos puede ser negativa (que significa que le sale «a devolver»)

4.17.1 HTML calculador

```

<!DOCTYPE html>

<html>
<head>
    <meta charset="utf-8">
    <link type="text/css" href="css/estilo.css"
        rel="stylesheet">
    <title>Plantilla de web</title>
    <script type="text/javascript"
        src="js/jquery.js"></script>
    <script type="text/javascript" src="js/programa.js"></script>
</head>

<body>
<!--Toda nuestra página irá aquí dentro-->
<div id="contenido">

<h1>Plantilla de proyecto web</h1>
<form>

```

```

Salario:
<input type="number" id="salario" min="0" max="1000000" value="0"> <br/>
<input type="radio" id="con_hijos" name="hijos">Con hijos a cargo <br/>
<input type="radio" id="sin_hijos" name="hijos" checked="checked">
    Sin hijos a cargo <br/>
<input type="checkbox" id="bonificacion_b1"> Con derecho a bonif. B1 <br/>
<input type="checkbox" id="bonificacion_b2"> Con derecho a bonif. B2 <br/>
<button id="calcular">Calcular impuestos</button>

</form>
<div id="mensajes">Aquí aparecerán sus resultados</div>
</body>
</html>

```

4.17.2 JS Calculador (con JQuery (DAM))

```

function getFloat ( identificador ){
    let objeto_control=$(identificador);
    let cadena_control=objeto_control.val();
    let cantidad=parseFloat(cadena_control);
    return cantidad;
}

function isChecked ( identificador ){
    let objeto_control=$(identificador);
    /*Comprobamos si el control
    tiene la propiedad checked*/
    if (objeto_control.prop("checked")){
        return true;
    }
    /* Si no tiene la propiedad checked
    es que no está marcado*/
    return false;
}

function calcular(){
    let salario_anual=getFloat("#salario");
    let impuestos=0;

    if ( (salario_anual>=20000) && (salario_anual<=30000) ){
        impuestos=salario_anual * 0.1;
    }

    if ( (salario_anual>=30001) && (salario_anual<=50000) ){
        impuestos=salario_anual * 0.2;
    }

    if (salario_anual>=50001){
        impuestos=salario_anual * 0.38;
    }

    if (isChecked("#con_hijos")){
        impuestos = impuestos - 180;
    }

    if (isChecked("#bonificacion_b1")){
        impuestos = impuestos - 245;
    }

    if (isChecked("#bonificacion_b2")){
        impuestos = impuestos - 535;
    }
}

```

```
    let div_mensajes=$("#mensajes");
    div_mensajes.html("Sus <b>impuestos</b> son:"+impuestos);
    return false;
}
function main(){
    let boton=$("#calcular");
    boton.click ( calcular );
}
let objeto_documento=$(document);
objeto_documento.ready(main);
```

4.17.3 JS Calculador (sin JQuery (ASIR))

```
function limpiar_errores() {
    let objeto_diesel=document.getElementById("diesel");
    let objeto_pot2300=document.getElementById("pot2300");
    if ((objeto_diesel.checked) && (objeto_pot2300.checked)) {
        alert("Incompatibilidad diesel+2300");
        objeto_pot2300.checked=false;
    }

    let objeto_pintura_normal=document.getElementById("normal");
    let objeto_color_gris =document.getElementById("gris");
    let objeto_color_azul =document.getElementById("azul");
    let objeto_color_verde=document.getElementById("verde");

    if ( (objeto_pintura_normal.checked)
        && (objeto_color_gris.checked) ){
        alert("Incompatibilidad normal+gris");
        objeto_color_gris.checked=false;
    }
    if ( (objeto_pintura_normal.checked)
        && (objeto_color_azul.checked) ){
        alert("Incompatibilidad normal+azul");
        objeto_color_azul.checked=false;
    }
    if ( (objeto_pintura_normal.checked)
        && (objeto_color_verde.checked) ){
        alert("Incompatibilidad normal+verde");
        objeto_color_verde.checked=false;
    }
}
function calcular() {

    limpiar_errores();

    let precio=0;

    let objeto_gasolina=document.getElementById("gasolina");
    if (objeto_gasolina.checked) {
        precio=precio + 7000;
    }
}
```

```

let objeto_diesel=document.getElementById("diesel");
if (objeto_diesel.checked) {
    precio=precio + 8200;
}

let objeto_pot1100=document.getElementById("pot1100");
if (objeto_pot1100.checked) {
    precio=precio + 800;
}
let objeto_pot1800=document.getElementById("pot1800");
if (objeto_pot1800.checked) {
    precio=precio + 1900;
}
let objeto_pot2300=document.getElementById("pot2300");
if (objeto_pot2300.checked) {
    precio=precio + 2500;
}

//Mostramos el precio final
let objeto_mensajes=document.getElementById("mensajes");
objeto_mensajes.innerHTML="Precio <u>final</u>"+precio;
}

```

4.18 Comparador de telefonía

Se desea crear una aplicación que permita al usuario saber qué compañía de telefonía le conviene más partiendo de los siguientes datos

- La empresa A ofrece una tarifa que cuesta 20 euros al mes con los 1000 primeros minutos gratis y despues cada minuto cuesta 8 céntimos.
- La empresa B ofrece una tarifa que cuesta 10 euros al mes con los 500 primeros minutos gratis y despues cada llamada cuesta 12 céntimos.

Hacer una aplicación que permita al usuario introducir la cantidad de minutos que llamará para tres meses distintos que llamaremos «Mes 1», «Mes 2» y «Mes 3» y que le diga qué compañía le interesa más.

4.18.1 HTML del comparador

```

<form>
Mes 1<input type="number" value="800"
           min="0" max="5000" id="mes1"> <br/>
Mes 2<input type="number" value="2000"
           min="0" max="5000" id="mes2"> <br/>
Mes 3<input type="number" value="600"
           min="0" max="5000" id="mes3"> <br/>
<input type="submit" value="¿Qué me conviene?"
       onclick="calcular();return false">
</form>

```

4.18.2 JS del comparador

```
function extraer_numero(id) {
    var control;
    control=document.getElementById(id);
    var valor=control.value;
    var numero=parseInt(valor);
    return numero;
}

function precio_mes_empresa_a(minutos) {
    //Primero vemos si se pasa
    if (minutos<1000){
        //Si no se pasa, el precio es 20
        return 20;
    }
    //Si llegamos a este punto, es porque se pasa

    //Calculamos cuantos minutos se pasa
    var exceso=minutos-1000;
    //Y ese exceso es a 8 centimos/minutos
    var precio_exceso=exceso*0.08;
    //El precio de ese mes es
    //20 euros más lo que cueste el exceso
    var precio_mes=20+precio_exceso;
    //Devolvemos el calculo a quien lo pidiese
    return precio_mes;
}

function precio_mes_empresa_b(minutos) {
    //Primero vemos si se pasa
    if (minutos<500){
        //Si no se pasa, el precio es 10
        return 10;
    }
    //Si llegamos a este punto, es porque se pasa

    //Calculamos cuantos minutos se pasa
    var exceso=minutos-500;
    //Y ese exceso es a 12 centimos/minutos
    var precio_exceso=exceso*0.12;
    //El precio de ese mes es
    //10 euros más lo que cueste el exceso
    var precio_mes=10+precio_exceso;
    //Devolvemos el calculo a quien lo pidiese
    return precio_mes;
}

function calcular() {
    var minutos_mes1;
    minutos_mes1=extraer_numero("mes1");
    var minutos_mes2;
    minutos_mes2=extraer_numero("mes2");
    var minutos_mes3;
    minutos_mes3=extraer_numero("mes3");

    var coste_mes1_empresa_a;
    coste_mes1_empresa_a=precio_mes_empresa_a(
```

```
        minutos_mes1);

coste_mes2_empresa_a=precio_mes_empresa_a(
    minutos_mes2);
coste_mes3_empresa_a=precio_mes_empresa_a(
    minutos_mes3);

//Calculamos el precio que costaria
//esos minutos en la empresa A
var coste_total_a;
coste_total_a=coste_mes1_empresa_a+coste_mes2_empresa_a+coste_mes3_empresa_a;

//Repetimos el proceso para la empresa B
var coste_mes1_empresa_b;
coste_mes1_empresa_b=precio_mes_empresa_b(
    minutos_mes1);
coste_mes2_empresa_b=precio_mes_empresa_b(
    minutos_mes2);
coste_mes3_empresa_b=precio_mes_empresa_b(
    minutos_mes3);
//Calculamos ahora el precio que costaria
//esos minutos en la empresa B
var coste_total_b;
coste_total_b=coste_mes1_empresa_b+coste_mes2_empresa_b+coste_mes3_empresa_b;
var zonaresultados;
zonaresultados=document.getElementById("zonaresultados");
var texto;
texto="Con la A el precio es:" + coste_total_a;
texto=texto+"<br/>";
texto=texto+"Con la B el precio es:" + coste_total_b;
zonaresultados.innerHTML=texto
}
```


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 (de hecho, el coloreador de sintaxis automático descubre que no es XML y el fichero se muestra de manera literal)

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 (usando ?)
- Se puede indicar si un elemento debe aparecer de forma obligatoria.
- Se puede indicar si algo aparece una o muchas veces (usando +).
- Se puede indicar si algo aparece cero o muchas veces (usando *).

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>
```

5.5.2 Ejemplo de DTD (productos)

Se pide un conjunto de reglas en forma de DTD para definir qué se permitirá en los archivos XML de datos de una empresa de fabricación:

- La raíz es <productos>
- Dentro de productos puede haber <raton> o <teclado> que pueden repetirse e ir en cualquier orden (RRTT, T, TR, TTRR)
- Todo <raton> tiene siempre un <codigo> y puede o no tener una <descripcion>.
- Todo <teclado> tiene siempre un <codigo>, debe llevar siempre una <descripcion> y puede o no tener un <peso>

Elaborar la DTD que formaliza estas reglas.

```
<!ELEMENT productos (raton|teclado)* >
<!ELEMENT raton (codigo, descripcion?) >
<!ELEMENT codigo (#PCDATA) >
<!ELEMENT descripcion (#PCDATA) >
<!ELEMENT teclado (codigo, descripcion, peso?) >
<!ELEMENT peso (#PCDATA) >
```

El siguiente fichero debe validarse correctamente:

```
<productos>
</productos>
```

Y el siguiente también

```
<productos>
  <teclado>
    <codigo>T1</codigo>
    <descripcion>Teclado inalamb.</descripcion>
  </teclado>
</productos>
```

Y este también (a pesar del flagrante error en el peso)

```

<productos>
  <raton>
    <codigo>R1</codigo>
  </raton>
  <teclado>
    <codigo>T1</codigo>
    <descripcion>Teclado inalamb.</descripcion>
    <peso>|@~|@~|@~</peso>
  </teclado>
</productos>

```

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>

```

```
</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>

```

```
        </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 completa

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

<!ELEMENT asignatura (nombre, profesor)>
<!ATTLIST asignatura codigo CDATA #REQUIRED>

<!ELEMENT profesor (nrp, nombre, ap1, ap2?)>

<!ELEMENT dni      (#PCDATA)>
<!ELEMENT nombre   (#PCDATA)>
<!ELEMENT ap1      (#PCDATA)>
<!ELEMENT ap2      (#PCDATA)>
<!ELEMENT nrp      (#PCDATA)>
```

Un ejemplo de fichero válido:

```
<listacursos>
  <curso>
    <alumno>
      <dni>44e</dni>
      <nombre>Juan</nombre>
      <ap1>Sanchez</ap1>
      <asignatura codigo="LM1">
        <nombre>Leng marcas</nombre>
        <profesor>
          <nrp>8</nrp>
          <nombre>Oscar</nombre>
          <ap1>Gomez</ap1>
        </profesor>
      </asignatura>
    </alumno>
  </curso>
</listacursos>
```

```

        </asignatura>
    </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 quisiéramos 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 ningun pais, 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

El objetivo final debe ser validar un fichero como este:

```
<operaciones>
  <operacion>
    <venta>
      <titulosvendidos>
        <titulo>Don Quijote</titulo>
        <titulo>Rimas y leyendas</titulo>
        <cantidadtotal>2000</cantidadtotal>
        <importe moneda="euros">4400</importe>
      </titulosvendidos>
    </venta>
    <venta>
      <titulosvendidos>
        <titulo>Rinconete y Cortadillo</titulo>
        <titulo>Sainetes</titulo>
        <cantidadtotal>1000</cantidadtotal>
        <entregado/>
        <importe moneda="libras">290</importe>
      </titulosvendidos>
    </venta>
  </operacion>
  <operacion>
    <compra>
      <tituloscomprados>
        <titulo>De la Tierra a la Luna</titulo>
        <titulo>Barbarroja</titulo>
        <proveedor>Editorial EDSA</proveedor>
        <fechacompra>
          <dia>10</dia>
          <mes>6</mes>
          <anio>2018</anio>
        </fechacompra>
      </tituloscomprados>
    </compra>
    <venta>
      <titulosvendidos>
        <titulo>Cinco semanas en globo</titulo>
        <titulo>Sainetes</titulo>
        <cantidadtotal>700</cantidadtotal>
        <entregado/>
        <importe moneda="euros">1490</importe>
      </titulosvendidos>
    </venta>
```

```

    <compra>
      <tituloscomprados>
        <titulo>De la Tierra a la Luna</titulo>
        <titulo>Barbarroja</titulo>
        <proveedor>Editorial Recopila</proveedor>
        <fechacompra>
          <dia>2</dia>
          <mes>12</mes>
          <anio>2017</anio>
        </fechacompra>
      </tituloscomprados>
    </compra>
  </operacion>
</operaciones>

```

5.12.1 Solución al mayorista de libros

La siguiente DTD valida el fichero arriba mostrado:

```

<!--El elemento raíz es operaciones y dentro de él hay uno o más elementos operación-->
<ELEMENT operaciones (operacion+)>
<!--Una operación puede ser ventas o compras, en cualquier orden y repetidas las_
veces que sea necesario-->
<ELEMENT operacion (venta|compra)+>
<ELEMENT venta (titulosvendidos)>
<!--Una venta tiene uno o más títulos, la cantidad de libros vendidos, puede haber un_
elemento entregado que indique si la entrega se ha realizado, y debe haber un_
elemento importe con un atributo obligatorio llamado moneda. -->
<ELEMENT titulosvendidos (titulo+, cantidadtotal, entregado?, importe)>
<!--Antes de que se nos olvide, fabricamos el elemento importe y su atributo moneda-->
<ELEMENT importe (#PCDATA)>
<ATTLIST importe moneda CDATA #REQUIRED>
<!--Fabricamos el título y la cantidad total-->
<ELEMENT titulo (#PCDATA)>
<ELEMENT cantidadtotal (#PCDATA)>
<!--El elemento entregado parece que es un vacío-->
<ELEMENT entregado EMPTY>
<!--Una compra tiene:

-Uno o más títulos comprados.
-Nombre de proveedor.
-Una fecha de compra, que debe desglosarse en elementos día, mes y año -->
<ELEMENT compra (tituloscomprados)>
<ELEMENT tituloscomprados (titulo+, proveedor, fechacompra)>
<ELEMENT proveedor (#PCDATA)>
<!--Desglosamos la fecha-->
<ELEMENT fechacompra (dia, mes, anio)>
<ELEMENT dia (#PCDATA)>
<ELEMENT mes (#PCDATA)>
<ELEMENT anio (#PCDATA)>

```


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 kmmaximos indicando que el componente debe sustituirse tras un cierto número de kilómetros.

Un posible fichero de ejemplo que podría validar sería este:

```
<pedido>
  <tractor>
    <componente nombrefabricante="Ebro">
      <fechaentrega>
        <mes>2018</mes> <anio>2018</anio>
      </fechaentrega>
      <fragil/>
      <peso unidad="kg">12</peso>
      <numserie>00A</numserie>
    </componente>
    <componente nombrefabricante="Avia">
      <fechaentrega>
        <dia>12</dia><mes>1</mes><anio>2019</anio>
      </fechaentrega>
      <nofragil/>
      <peso unidad="g">1450</peso>
      <numserie>00D</numserie>
      <kmmaximos>25000</kmmaximos>
    </componente>
  </tractor>
  <tractor>
    <componente nombrefabricante="John Deere">
      <fragil/>
      <peso unidad="g">770</peso>
      <numserie>43Z</numserie>
    </componente>
  </tractor>
</pedido>
```

5.13.1 Solución: DTD fabricante tractores

```
<!ELEMENT pedido      (tractor)+>
<!ELEMENT tractor     (componente)+>
<!ELEMENT componente  (fechaentrega?, (fragil|nofragil),
                        peso, numserie, kmmaximos?)>

<!ELEMENT fechaentrega (dia?, mes, anio)>
<!ELEMENT dia          (#PCDATA)>
<!ELEMENT mes          (#PCDATA)>
```

```

<!ELEMENT anio      (#PCDATA)>
<!ELEMENT fragil    EMPTY>
<!ELEMENT nofragil  EMPTY >
<!ELEMENT peso      (#PCDATA)>
<!ATTLIST peso      unidad CDATA #REQUIRED>
<!ELEMENT numserie  (#PCDATA)>
<!ELEMENT kmaximos  (#PCDATA)>
<!ATTLIST componente nombrefabricante CDATA #REQUIRED>

```

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>

```

5.15 Ejercicio: multinacional

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 (obligatorio), 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 Esquemas XML

Los esquemas XML son un mecanismo radicalmente distinto de crear reglas para validar ficheros XML. Se caracterizan por:

- Estar escritos en XML. Por lo tanto, las mismas bibliotecas que permiten procesar ficheros XML de datos permitirían procesar ficheros XML de reglas.
- Son mucho más potentes: ofrecen soporte a tipos de datos con comprobación de si el contenido de una etiqueta es de tipo `integer`, `date` o de otros tipos. También se permite añadir restricciones como indicar valores mínimo y máximo para un número o determinar el patrón que debe seguir una cadena válida
- Ofrecen la posibilidad de usar *espacios de nombres*. Los espacios de nombres son similares a los paquetes Java: permiten a personas distintas el definir etiquetas con el mismo nombre pudiendo luego distinguir etiquetas iguales en función del espacio de nombres que importemos.

5.17.1 Un ejemplo

Supongamos que deseamos tener ficheros XML con un solo elemento llamado `<cantidad>` que debe tener dentro un número.

```
<cantidad>20</cantidad>
```

Un posible esquema sería el siguiente:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cantidad" type="xsd:integer"/>
</xsd:schema>
```

¿Qué contiene este fichero?

1. En primer lugar se indica que este fichero va a usar unas etiquetas ya definidas en un espacio de nombres (o XML Namespace, de ahí `xmlns`). Esa definición se hace en el espacio de nombres que aparece en la URL. Nuestro validador no descargará nada, esa URL es oficial y todos los validadores la conocen. Las etiquetas de ese espacio de nombres van a usar un prefijo que en este caso será `xsd`. Nótese que el prefijo puede ser como queramos (podría ser «abcd» o «zztop»), pero la costumbre es usar `xsd`.
2. Se indica que habrá un solo elemento y que el tipo de ese elemento es `<xsd:integer>`. Es decir, un entero básico.

Si probamos el fichero de esquema con el fichero de datos que hemos indicado veremos que efectivamente el fichero XML de datos es válido. Sin embargo, si en lugar de una cantidad incluyésemos una cadena, veríamos que el fichero **no se validaría**

5.17.2 Tipos de datos básicos

Podemos usar los siguientes tipos de datos:

- `xsd:byte`: entero de 8 bits.
- `xsd:short`: entero de 16 bits
- `xsd:int`: número entero de 32 bits.
- `xsd:long`: entero de 64 bits.
- `xsd:integer`: número entero sin límite de capacidad.
- `xsd:unsignedByte`: entero de 8 bits sin signo.
- `xsd:unsignedShort`: entero de 16 bits sin signo.
- `xsd:unsignedInt`: entero de 32 bits sin signo.
- `xsd:unsignedLong`: entero de 64 bits sin signo.
- `xsd:string`: cadena de caracteres en la que los espacios en blanco se respetan.
- `xsd:normalizedString`: cadena de caracteres en la que los espacios en blanco no se respetan y se reemplazarán secuencias largas de espacios o fines de línea por un solo espacio.
- `xsd:date`: permite almacenar fechas que deben ir **obligatoriamente** en formato AAAA-MM-DD (4 dígitos para el año, seguidos de un guión, seguido de dos dígitos para el mes, seguidos de un guión, seguidos de dos dígitos para el día del mes)
- `xsd:time`: para almacenar horas en formato HH:MM:SS.C
- `xsd:datetime`: mezcla la fecha y la hora separando ambos campos con una T mayúscula. Esto permitiría almacenar 2020-09-22T10:40:22.6.
- `xsd:duration`. Para indicar períodos. Se debe empezar con «P» y luego indicar el número de años, meses, días, minutos o segundos. Por ejemplo «P1Y4M21DT8H» indica un período de 1 año, 4 meses, 21 días y 8 horas. Se aceptan períodos negativos poniendo -P en lugar de P.
- `xsd:boolean`: acepta solo valores «true» y «false».
- `xsd:anyURI`: acepta URIs.
- `xsd:anyType`: es como la clase `Object` en Java. Será el tipo del cual heredaremos cuando no vayamos a usar ningún tipo especial como tipo padre.

La figura siguiente (tomada de la web del W3C) ilustra todos los tipos así como sus relaciones de herencia:

5.17.3 Derivaciones

Prácticamente en cualquier esquema XML crearemos tipos nuevos (por establecer un símil es como si programásemos clases Java). Todos nuestros tipos tienen que heredar de otros tipos pero a la hora de «heredar» tenemos más posibilidades que en Java (donde solo tenemos el «extends»). En concreto podemos heredar de 4 formas:

1. Poniendo restricciones (*restriction*). Consiste en tomar un tipo y crear otro nuevo en el que no se puede poner cualquier valor.

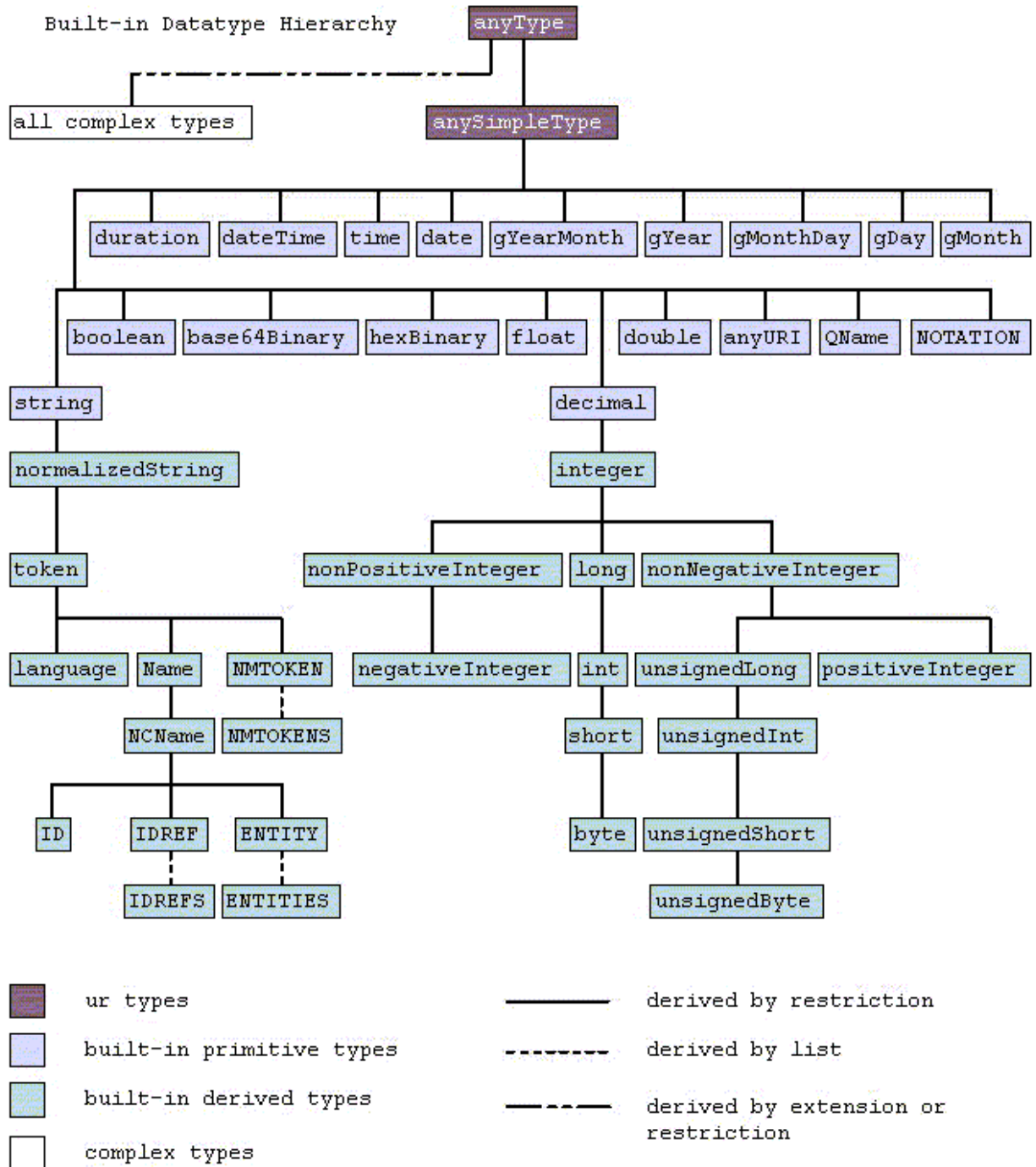


Figura 5.1: Tipos en los XML Schemas

2. Extendiendo un tipo (*extension*). Se toma un tipo y se crea uno nuevo añadiendo cosas a los posibles valores que pueda tomar el tipo inicial.
3. Haciendo listas (*lists*). Es como crear vectores en Java.
4. Juntando otros tipos para crear tipos complejos (*union*). Es como crear clases Java en las que añadimos atributos de tipo `int`, `String`, etc...

En general, las dos derivaciones más usadas con diferencia son las restricciones y las extensiones, que se comentan por separado en los puntos siguientes.

5.17.4 Tipos simples y complejos

Todo elemento de un esquema debe ser de uno de estos dos tipos.

- Un elemento es de tipo simple si no permite dentro ni elementos hijo ni atributos.
- Un elemento es tipo complejo si permite tener dentro otras cosas (que veremos en seguida). Un tipo complejo puede a su vez tener contenido simple o contenido complejo:
 - Los que son de contenido simple no permiten tener dentro elementos hijo pero sí permiten atributos.
 - Los que son de contenido complejo sí permiten tener dentro elementos hijo y atributos.

Así, por ejemplo un tipo simple que no lleve ninguna restricción se puede indicar con el campo `type` de un `element` como hacíamos antes:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cantidad" type="xsd:integer"/>
</xsd:schema>
```

Sin embargo, si queremos indicar alguna restricción adicional ya no podremos usar el atributo `type`. Debemos reescribir nuestro esquema así:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType>
    Aquí irán las restricciones, que hemos omitido por ahora.
  </xsd:simpleType>
</xsd:schema>
```

5.17.5 Ejercicio:edad de los trabajadores

Se desea crear un esquema que permita validar la edad de un trabajador, que debe tener un valor entero de entre 16 y 65.

Por ejemplo, este XML debería validarse:

```
<edad>28</edad>
```

Pero este no debería validarse:

```
<edad>-3</edad>
```

La solución podría ser algo así:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="edad"
```



```

        type="tipoEdad"/>
<xsd:simpleType name="tipoEdad">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="16"/>
    <xsd:maxInclusive value="65"/>
  </xsd:restriction>
</xsd:simpleType>
</xsd:schema>

```

5.17.6 Ejercicio: peso de productos

Se desea crear un esquema que permita validar un elemento peso, que puede tener un valor de entre 0 y 1000 pero aceptando valores con decimales, como por ejemplo 28.88

Una posible solución sería:

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="peso" type="tipoPeso"/>
  <xsd:simpleType name="tipoPeso">
    <xsd:restriction base="xsd:decimal">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="1000"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>

```

5.17.7 Ejercicio: pagos validados

Crear un esquema que permita validar un elemento pago en el cual puede haber cantidades enteras de entre 0 y 3000 euros.

```

<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="pago" type="tipoPago"/>
  <xsd:simpleType name="tipoPago">
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="3000"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>

```

5.17.8 Ejercicio: validación de DNIs

Crear un esquema que permita validar un único elemento dni que valide el patrón de 7-8 cifras + letra que suelen tener los DNI en España:

```

<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="dni" type="tipoDNI"/>
  <xsd:simpleType name="tipoDNI">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[0-9]{7,8}[A-Z]"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>

```

```
</xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

5.17.9 Uniendo la herencia y el sistema de tipos

Llegados a este punto ocurre lo siguiente:

- Por un lado tenemos que especificar si nuestros tipos serán simples o complejos (los cuales a su vez pueden ser complejos con contenido simple o complejos con contenido complejo).
- Por otro lado se puede hacer herencia ampliando cosas (extensión) o reduciendo cosas (restricciones a los valores).

Se deduce por tanto que no podemos aplicar todas las «herencias» a todos los tipos:

1. Los tipos simples no pueden tener atributos ni subelementos, por lo tanto **les podremos aplicar restricciones pero nunca la extensión.**
2. Los tipos complejos (independientemente del tipo de contenido) sí pueden tener otras cosas dentro por lo que **les podremos aplicar tanto restricciones como extensiones.**

5.17.10 Restricciones

Como se ha dicho anteriormente la forma más común de trabajar es crear tipos que en unos casos aplicarán modificaciones en los tipos ya sea añadiendo cosas o restringiendo posibilidades. En este apartado se verá como aplicar restricciones.

Si queremos aplicar restricciones para un tipo simple las posibles restricciones son:

- `minInclusive` para indicar el menor valor numérico permitido.
- `maxInclusive` para indicar el mayor valor numérico permitido.
- `minExclusive` para indicar el menor valor numérico que ya no estaría permitido.
- `maxExclusive` para indicar el mayor valor numérico que ya no estaría permitido.
- `totalDigits` para indicar cuantas posibles cifras se permiten.
- `fractionDigits` para indicar cuantas posibles cifras decimales se permiten.
- `length` para indicar la longitud exacta de una cadena.
- `minLength` para indicar la longitud mínima de una cadena.
- `maxLength` para indicar la longitud máxima de una cadena.
- `enumeration` para indicar los valores aceptados por una cadena.
- `pattern` para indicar la estructura aceptada por una cadena.

Si queremos aplicar restricciones para un tipo complejo con contenido las posibles restricciones son las mismas de antes, pero además podemos añadir el elemento `<attribute>` así como las siguientes.

- `sequence` para indicar una secuencia de elementos
- `choice` para indicar que se debe elegir un elemento de entre los que aparecen.

5.17.11 Atributos

En primer lugar es muy importante recordar que **si queremos que un elemento tenga atributos entonces ya no se puede considerar que sea de tipo simple. Se debe usar FORZOSAMENTE un complexType**. Por otro lado en los XML Schema todos los atributos **son siempre opcionales, si queremos hacerlos obligatorios habrá que añadir un «required»**.

Un atributo se define de la siguiente manera:

```
<xsd:attribute name="fechanacimiento" type="xsd:date" use="required"/>
```

Esto define un atributo llamado `nombre` que aceptará solo fechas como valores válidos y que además es obligatorio poner siempre.

5.18 Ejercicios de XML Schemas

5.18.1 Cantidades limitadas

Crear un esquema que permita verificar algo como lo siguiente:

```
<cantidad>20</cantidad>
```

Se necesita que la cantidad tenga solo valores aceptables entre -30 y +30.

5.18.2 Solución a las cantidades limitadas

La primera pregunta que debemos hacernos es ¿necesitamos crear un tipo simple o uno complejo?. Dado que nuestro único elemento no tiene subelementos ni atributos dentro podemos afirmar que solo necesitamos un tipo simple.

Como aparentemente nuestro tipo necesita usar solo valores numéricos y además son muy pequeños nos vamos a limitar a usar un `short`. Sobre ese `short` pondremos una restricción que permita indicar los valores mínimo y máximo.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="cantidad">
    <xs:simpleType>
      <xs:restriction base="xs:short">
        <xs:minInclusive value="-30"/>
        <xs:maxInclusive value="30"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```

Este esquema dice que el elemento raíz debe ser `cantidad`. Luego indica que es un tipo simple y dentro de él indica que se va a establecer una restricción teniendo en mente que se va a «heredar» del tipo `short`. En concreto se van a poner dos restricciones, una que el valor mínimo debe ser -30 y otra que el valor máximo debe ser 30.

Existe una alternativa más recomendable, que es separar los elementos de los tipos. De esa manera, se pueden «reutilizar» las definiciones de tipos.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="cantidad" type="tipoCantidades">
  </xs:element>
```

```

<xs:simpleType name="tipoCantidades">
  <xs:restriction base="xs:short">
    <xs:minInclusive value="-30"/>
    <xs:maxInclusive value="30"/>
  </xs:restriction>
</xs:simpleType>
</xs:schema>

```

Obsérvese que hemos puesto el tipo por separado y le hemos dado el nombre `tipoCantidades`. El elemento raíz tiene su nombre y su tipo en la misma línea.

5.18.3 Cantidades limitadas con atributo divisa

Se desea crear un esquema para validar XML en los que haya un solo elemento raíz llamado `cantidad` en el que se debe poner siempre un atributo «divisa» que indique en qué moneda está una cierta cantidad. El atributo divisa siempre será una cadena y la cantidad siempre será un tipo numérico que acepte decimales (por ejemplo `float`). El esquema debe validar los archivos siguientes:

```
<cantidad divisa="euro">20</cantidad>
```

```
<cantidad divisa="dolar">18.32</cantidad>
```

Pero no debe validar ninguno de los siguientes:

```
<cantidad>20</cantidad>
```

```
<cantidad divisa="dolar">abc</cantidad>
```

5.18.4 Solución a las cantidades limitadas con atributo divisa

Crearemos un tipo llamado «tipoCantidad». Dicho tipo *ya no puede ser un simpleType ya que necesitamos que haya atributos*. Como no necesitamos que tenga dentro subelementos entonces este `complexType` llevará dentro un `simpleContent` (y no un `complexContent`).

Aparte de eso, como queremos «ampliar» un elemento para que acepte tener dentro un atributo obligatorio «cantidad» usaremos una `<extension>`. Así, el posible esquema sería este:

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cantidad" type="tipoCantidad"/>
  <xsd:complexType name="tipoCantidad">
    <xsd:simpleContent>
      <xsd:extension base="xsd:float">
        <xsd:attribute name="divisa" type="xsd:string" use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:schema>

```

5.18.5 Cantidades limitadas con atributo divisa con solo ciertos valores

Queremos ampliar el ejercicio anterior para evitar que ocurran errores como el siguiente:

```
<cantidad divisa="aaaa">18.32</cantidad>
```

Vamos a indicar que el atributo solo puede tomar tres posibles valores: «euros», «dolares» y «yenes».

5.18.6 Solución al atributo con solo ciertos valores

Ahora tendremos que crear dos tipos. Uno para el elemento `cantidad` y otro para el atributo `divisa`. Llamaremos a estos tipos `tipoCantidad` y `tipoDivisa`.

La solución comentada puede encontrarse a continuación. Como puede verse, hemos incluido comentarios. Pueden insertarse etiquetas `annotation` que permiten incluir anotaciones de diversos tipos, siendo la más interesante la etiqueta `documentation` que nos permite incluir comentarios.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cantidad" type="tipoCantidad"/>
  <xsd:annotation>
    <xsd:documentation>
      A continuación creamos el tipo cantidad
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType name="tipoCantidad">
    <xsd:annotation>
      <xsd:documentation>
        Como solo va a llevar atributos debemos
        usar un simpleContent
      </xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
      <xsd:annotation>
        <xsd:documentation>
          Como queremos "ampliar" un tipo/clase
          para que lleve atributos usaremos
          una extension
        </xsd:documentation>
      </xsd:annotation>
      <xsd:extension base="xsd:float">
        <xsd:attribute name="divisa" type="tipoDivisa"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <xsd:annotation>
    <xsd:documentation>
      Ahora tenemos que fabricar el "tipoDivisa" que indica
      los posibles valores válidos para una divisa. Estas
      posibilidades se crean con una "enumeration". Nuestro
      tipo es un "string" y como vamos a restringir los posibles
      valores usaremos "restriction"
    </xsd:documentation>
  </xsd:annotation>
  <xsd:simpleType name="tipoDivisa">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="euros"/>
      <xsd:enumeration value="dolares"/>
      <xsd:enumeration value="yenes"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

5.18.7 Ejercicio: codigos y sedes

Se necesita tener un esquema que valide un fichero en el que hay un solo elemento llamado `codigo`

- Dentro de código hay una cadena con una estructura rígida: 2 letras mayúsculas, seguidas de 2 cifras, seguidas a su vez de 3 letras.
- El elemento `código` debe llevar un atributo `sede` que será de tipo cadena.

5.18.8 Solución a los códigos y sedes

Se nos piden dos cosas:

1. Restringir un tipo básico, en este caso el `string`
2. Extender una etiqueta para que tenga un atributo.

Como no se puede hacer a la vez, deberemos dar dos pasos. Primero crearemos un tipo con la restricción y después crearemos un segundo tipo con la extensión.

Cuando haya conflictos, siempre debemos crear primero la restricción y luego la extensión

Así, creamos primero esto:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="codigo" type="tipoCodigoRestringido"/>

  <xsd:simpleType name="tipoCodigoRestringido">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Z]{2}[0-9]{2}[A-Z]{3}"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Y después lo ampliamos para que se convierta en esto:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="codigo" type="tipoCodigo"/>

  <xsd:simpleType name="tipoCodigoRestringido">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Z]{2}[0-9]{2}[A-Z]{3}"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="tipoCodigo">
    <xsd:simpleContent>
      <xsd:extension base="tipoCodigoRestringido">
        <xsd:attribute name="sede"
                      type="xsd:string"
                      use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:schema>
```

5.18.9 Ejercicio: productos con atributos

Se desea crear un esquema que permita validar un elemento raíz llamado `producto` de tipo `xsd:string`. El producto tiene dos atributos:

- Un atributo se llamará `cantidad` y es obligatorio. Debe aceptar solo enteros positivos.
- También habrá un atributo llamado `unidad` que solo acepta los `xsd:string` «cajas» y «pales».

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="producto" type="tipoProducto"/>
  <xsd:complexType name="tipoProducto">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="cantidad"
          type="xsd:unsignedInt" use="required"/>
        <xsd:attribute name="unidad"
          type="tipoUnidad"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <xsd:simpleType name="tipoUnidad">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="caja"/>
      <xsd:enumeration value="pale"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

5.18.10 Ejercicio: clientes con información adicional

Se desea crear un esquema XML que permita validar un elemento llamado `cliente` que puede almacenar un `xsd:string`. El cliente contiene:

- Un atributo obligatorio llamado `codigo` que contiene el código del cliente, que siempre consta de tres letras mayúsculas de tres números.
- Un atributo optativo llamado `habitual` que se usará para saber si es un cliente habitual o no. Acepta valores «true» y «false».
- Un atributo optativo llamado `cantidad` que indica su compra. Es un entero con valores de entre 0 y 1000.

5.18.11 Lista de clientes como XML Schemas

En este apartado volveremos a ver un problema que ya resolvíamos con DTD: 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 plazo. Como no todo el mundo usa plazos el `<diasentrega>` es optativo.

Vayamos paso a paso. Primero decimos como se llama el elemento raíz y de qué tipo es:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="listaclientes" type="tipoListaClientes"/>
</xsd:schema>
```

Ahora queda definir el tipo `tipoListaClientes`. Este tipo va a contener un elemento (por lo que ya sabemos que es un `complexType` con `complexContent` dentro), y en concreto queremos que sea un solo elemento llamado `cliente`, es decir **queremos imponer una restricción**. Aunque queramos un solo elemento tendremos que indicar una restricción. Como queremos permitir que el elemento pueda aparecer muchas veces utilizaremos un `maxOccurs` con el valor `unbounded`.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="listaclientes" type="tipoListaClientes"/>
  <xsd:complexType name="tipoListaClientes">
    <xsd:complexContent>
      <xsd:restriction>
        <xsd:element name="cliente" type="tipoCliente"
          maxOccurs="unbounded"/>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```

Definamos ahora el tipo `tipoCliente`. Dicho tipo **necesita tener subelementos dentro** así que evidentemente va a ser de tipo complejo. La pregunta es ¿es «tipo complejo con contenido simple» o «tipo complejo con contenido complejo»? Si lo hiciéramos de «tipo complejo con contenido simple» podríamos tener atributos pero no subelementos, así que forzosamente tendrá que ser de un «tipo complejo con contenido complejo». Igual que antes impondremos una restricción que es permitir solo que aparezcan ciertos elementos en cierto orden. El elemento `plazo` lo haremos optativo.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="listaclientes" type="tipoListaClientes"/>
  <xsd:complexType name="tipoListaClientes">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="cliente" type="tipoCliente"
            maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tipoCliente">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="cif" type="xsd:string"/>
          <xsd:element name="nombre" type="xsd:string"/>
          <xsd:element name="plazo" type="xsd:string"
            minOccurs="0"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```

Si ahora probamos este XML veremos que el fichero se valida perfectamente a pesar de que es evidente que tiene errores. Es lógico, dado que no hemos aprovechado a fondo el sistema de tipos de XML para evitar que nadie suministre datos incorrectos en un XML. Dicha mejora la dejaremos para el siguiente ejercicio.

```
<listaclientes>
  <cliente>
    <cif>dd</cif>
```



```

    <nombre>20</nombre>
  </cliente>
  <cliente>
    <cif>dd</cif>
    <nombre>20</nombre>
    <plazo>ABCD</plazo>
  </cliente>
</listaclientes>

```

5.18.12 Ampliación del esquema para clientes

Ahora ampliaremos el XML Schema del fichero anterior para que nadie suministre información incorrecta.

En concreto tenemos tres datos:

1. El CIF, que vamos a presuponer que siempre tiene 8 cifras y al final una letra mayúsculas. Si alguna empresa tiene 7 cifras deberá incluir un 0 extra.
2. El nombre, que puede ser una cadena cualquiera.
3. El plazo, que debería ser un número positivo válido.

Ahora, el fichero anterior no debería ser validado por el validador, pero sí debería serlo un fichero como este.

```

<listaclientes>
  <cliente>
    <cif>01234567D</cif>
    <nombre>Juan Sanchez</nombre>
  </cliente>
  <cliente>
    <cif>05676554A</cif>
    <nombre>Pedro Diaz</nombre>
    <plazo>45</plazo>
  </cliente>
</listaclientes>

```

La solución a los tres problemas indicados antes sería la siguiente:

1. El nombre puede ser una cadena cualquiera, por lo que tendrá que seguir siendo de tipo `xsd:string`. Eso significa que si alguien introdujese un número en el nombre el fichero seguiría validándose. Por desgracia dicho problema no se puede resolver.
2. El plazo debería ser un número. Le asignaremos un tipo `xsd:unsignedInt`.
3. El CIF es más complejo. Debemos crear un tipo nuevo y establecer una restricción a los posibles valores que puede tomar.

Así, una posible solución sería esta:

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="listaclientes" type="tipoListaClientes"/>
  <xsd:complexType name="tipoListaClientes">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="cliente" type="tipoCliente"
            maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tipoCliente">
    <xsd:sequence>
      <xsd:element name="cif" type="cif" maxOccurs="1"/>
      <xsd:element name="nombre" type="string" maxOccurs="1"/>
      <xsd:element name="plazo" type="unsignedInt" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

```

```

    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tipoCliente">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="cif" type="tipoCif"/>
          <xsd:element name="nombre" type="xsd:string"/>
          <xsd:element name="plazo" type="xsd:unsignedInt" minOccurs="0"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:simpleType name="tipoCif">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[0-9]{8}[A-Z]"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="tipoPlazo">
    <xsd:restriction base="xsd:unsignedInt"/>
  </xsd:simpleType>
</xsd:schema>

```

5.18.13 Ejercicio: lista de códigos

Se nos pide crear un esquema que permita validar un fichero como el siguiente:

```

<listacodigos>
  <codigo>AAA2DD</codigo>
  <codigo>BBB2EE</codigo>
  <codigo>BBB2EE</codigo>
</listacodigos>

```

En concreto, todo código tiene la estructura siguiente:

1. Primero van tres mayúsculas
2. Despues va exactamente un dígito.
3. Por último hay exactamente dos mayúsculas.

Un posible esquema XML sería el siguiente (obsérvese como usamos `maxOccurs` para indicar que el elemento puede repetirse un máximo de «infinitas veces»:

```

<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="listacodigos"
    type="tipoLista"/>
  <xsd:complexType name="tipoLista">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="codigo"
            type="tipoCodigo"
            maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>

```

```

    </xsd:complexContent>
  </xsd:complexType>
  <xsd:simpleType name="tipoCodigo">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Z]{3}[0-9][A-Z]{2}" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>

```

5.18.14 Ejercicio: otra lista de clientes

Ahora se nos pide crear un esquema que permita validar un fichero como el siguiente, en el que hay una lista de clientes y el nombre es optativo, aunque los apellidos son obligatorios:

```

<listaclientes>
  <cliente>
    <nombre>Juan</nombre>
    <apellidos>Sanchez</apellidos>
  </cliente>
  <cliente>
    <nombre>Jose</nombre>
    <apellidos>Diaz</apellidos>
  </cliente>
</listaclientes>

```

La solución puede ser algo así:

```

<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="listaclientes"
    type="tipoLista"/>
  <xsd:complexType name="tipoLista">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="cliente"
            type="tipoCliente"
            maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="tipoCliente">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="nombre"
            type="xsd:string"
            minOccurs="0"/>
          <xsd:element name="apellidos"
            type="xsd:string"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>

```

```
</xsd:schema>
```

5.18.15 Ejercicio: lista de alumnos

Se desea construir un esquema para validar listas de alumnos en las que:

- La raíz es listaalumnos.
- Dentro de ella hay uno o más alumno. Todo alumno tiene siempre un DNI que es obligatorio y que tiene una estructura formada por 7 u 8 cifras seguidas de una mayúscula.
- Todo alumno tiene un elemento nombre y un ap1 obligatorios.
- Todo alumno puede tener despues del ap1 un elemento ap2 y uno edad, ambos son optativos.
- El elemento edad debe ser entero y positivo.

Un ejemplo de fichero:

```
<listaalumnos>
  <!--DNI atributo obligatorio-->
  <alumno dni="5667545Z">
    <!--Nombre y ap1 obligatorios-->
    <nombre>Jose</nombre>
    <ap1>Sanchez</ap1>
  </alumno>
  <alumno dni="5778221D">
    <nombre>Andres</nombre>
    <ap1>Ruiz</ap1>
    <!--Ap2 y edad son optativos-->
    <ap2>Ruiz</ap2>
    <!--La edad debe ser positiva-->
    <edad>25</edad>
  </alumno>
</listaalumnos>
```

Y a continuación una posible solución:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="listaalumnos" type="tipoListaAlumnos"/>
  <xsd:complexType name="tipoListaAlumnos">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="alumno"
            type="tipoAlumno"
            maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tipoAlumno">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="nombre"
            type="xsd:string"/>
          <xsd:element name="ap1"
```

```

                                type="xsd:string"/>
        <xsd:element name="ap2"
                    type="xsd:string"
                    minOccurs="0"/>
        <xsd:element name="edad"
                    type="xsd:positiveInteger"
                    minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="dni" type="tipoDNI"/>
</xsd:restriction>
</xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="tipoDNI">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="[0-9]{7,8}[A-Z]"/>
    </xsd:restriction>
</xsd:simpleType>
</xsd:schema>

```

5.18.16 Ejercicio: lista de artículos (con atributos optativos)

Supongamos el fichero siguiente con las reglas que se explicitan en los comentarios:

```

<listaproductos>
  <articulo>
    <!--Estructura 2 letras, 2 cifras-->
    <codigo>CD12</codigo>
    <!--Descripcion es optativo y su atributo autor tb-->
    <descripcion autor="Pepe">Monitor</descripcion>
  </articulo>
  <articulo>
    <codigo>CA12</codigo>
  </articulo>
  <articulo>
    <codigo>AA99</codigo>
    <descripcion>Teclado</descripcion>
  </articulo>
</listaproductos>

```

A continuación se muestra una solución con un esquema que valida ficheros como el indicado:

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="listaproductos" type="tipoListaProductos"/>
  <xsd:complexType name="tipoListaProductos">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="articulo"
                      type="tipoArticulo"
                      maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType> <!--Fin de listaarticulos-->
  <xsd:complexType name="tipoArticulo">

```

```

<xsd:complexContent>
  <xsd:restriction base="xsd:anyType">
    <xsd:sequence>
      <xsd:element name="codigo" type="tipoCodigo"/>
      <xsd:element name="descripcion"
        type="tipoDescripcion"
        minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:restriction>
</xsd:complexContent>
</xsd:complexType> <!--Fin de  articulo-->

<xsd:simpleType name="tipoCodigo">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[A-Z]{2}[0-9]{2}"/>
  </xsd:restriction>
</xsd:simpleType> <!--Fin de codigo-->

<xsd:complexType name="tipoDescripcion">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="autor" type="xsd:string"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
</xsd:schema>

```

5.18.17 Ejercicio: lista de componentes (un enfoque distinto)

Dado un archivo como el siguiente en el cual aparecen las reglas incluidas como comentarios, crear el esquema que valide la estructura de tales ficheros:

```

<listacomponentes>
  <!--Obligatoria fecha entrega-->
  <componente entrega="2018-03-15">
    <fabricante>
      <!--Posibles fabricantes FAB1, FAB2 y FAB3-->
      <nombre>FAB1</nombre>
      <!--Calificacion es un string y es optativa-->
      <calificacion>Positiva</calificacion>
    </fabricante>
    <!--Atributo unidad es cadena. Dentro de peso
    solo puede haber numeros con decimales y mayores de 0-->
    <peso unidad="kg">40.5</peso>
  </componente>
  <componente entrega="2018-12-31">
    <fabricante>
      <nombre>FAB2</nombre>
    </fabricante>
    <peso unidad="miligramos">260.5</peso>
  </componente>
</listacomponentes>

```

Ahora en lugar de ir definiendo tipos empezando por el elemento raíz vamos a ir definiendo primero los tipos de los elementos más básicos que encontremos, e iremos construyendo los tipos más complejos a partir de los tipos fáciles que ya hayamos construido. Como veremos después, el resultado va a ser el mismo.

La solución:

```

n<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="tipoNombre">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="FAB1"/>
      <xsd:enumeration value="FAB2"/>
      <xsd:enumeration value="FAB3"/>
    </xsd:restriction>
  </xsd:simpleType>
  <!--Tipo auxiliar, el atributo lo incluimos despues-->
  <xsd:simpleType name="tipoPesoRestringido">
    <xsd:restriction base="xsd:float">
      <xsd:minExclusive value="0"/>
    </xsd:restriction>
  </xsd:simpleType>

  <!--En este tipo peso incluimos ya el atributo-->
  <xsd:complexType name="tipoPeso">
    <xsd:simpleContent>
      <xsd:extension base="tipoPesoRestringido">
        <xsd:attribute name="unidad"
          type="xsd:string"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

  <xsd:complexType name="tipoFabricante">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="nombre"
            type="tipoNombre"/>
          <xsd:element name="calificacion"
            type="xsd:string"
            minOccurs="0"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tipoComponente">
    <xsd:sequence>
      <xsd:element name="fabricante"
        type="tipoFabricante"/>
      <xsd:element name="peso" type="tipoPeso"/>
    </xsd:sequence>
    <xsd:attribute name="entrega" type="xsd:date"/>
  </xsd:complexType>

  <xsd:complexType name="tipoListaComponentes">
    <xsd:sequence>
      <xsd:element name="componente"
        type="tipoComponente"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <!--Aunque el elemento aparezca al final,

```

```

no pasa nada-->
<xsd:element name="listacomponentes"
              type="tipoListaComponentes"/>

</xsd:schema>

```

5.18.18 Ejercicio: listas con choice

Se pide elaborar un esquema que valide un fichero con las restricciones siguientes:

- El elemento raíz es `articulos`. Dicho elemento raíz debe llevar siempre un atributo `fechaGeneración`.
- Dentro de la raíz puede haber uno o varios de cualquiera de los siguientes elementos: `monitor`, `teclado` o `raton`. Cualquiera de los tres elementos puede llevar un atributo `codigo` que tiene siempre la estructura «tres letras, guión, tres letras, guión, tres cifras». Además, cualquiera de los tres debe llevar dentro y en primer lugar un elemento `descripcion` que contiene texto.
- Un `monitor` debe llevar (aparte de la descripción que va en primer lugar) un elemento `resolución` que a su vez debe llevar dentro dos elementos y en este orden `ancho` y `alto`. Tanto `ancho`, como `alto` deben llevar siempre dentro un entero positivo.
- Un `ratón` debe llevar (aparte de la descripción que va en primer lugar) un elemento `peso` que siempre lleva dentro un entero positivo. Además, el `peso` lleva siempre dentro un atributo `unidad` que solo puede valer «g» o «cg».

En el fichero siguiente se muestra un ejemplo

```

<!--Obligatorio el tener fechaGeneracion-->
<articulos fechaGeneracion="2018-03-01">
  <!--El atributo codigo es optativo siempre-->
  <monitor codigo="AAA-DDD-222">
    <!--Descripcion obligatoria-->
    <descripcion>Monitor de x pulgadas...</descripcion>
    <resolucion>
      <ancho>1920</ancho>
      <alto>1400</alto>
    </resolucion>
  </monitor>
  <raton>
    <!--Descripcion obligatoria-->
    <descripcion>Raton ergonómico...</descripcion>
    <!--La unidad es g o cg-->
    <peso unidad="g">100</peso>
  </raton>
  <teclado codigo="DDD-XXX-111">
    <!--Descripcion obligatoria-->
    <descripcion>Teclado estándar</descripcion>
  </teclado>
  <monitor codigo="CCC-GGG-666">
    <!--Descripcion obligatoria-->
    <descripcion>Monitor de x pulgadas...</descripcion>
    <resolucion>
      <ancho>1400</ancho>
      <alto>1000</alto>
    </resolucion>
  </monitor>
</articulos>

```


5.18.19 Ejercicio: listas de productos

Se pide validar correctamente algo como esto:

```
<listaproductos>
  <producto codigo="DX-22"><!--Codigo obligatorio-->
    <descripcion>Ordenador</descripcion><!--Optativa-->
    <peso>23.44</peso><!--Positivo con decimales-->
  </producto>
  <producto codigo="CX-124">
    <peso>17.50</peso>
  </producto>
  <producto codigo="CX-124">
    <peso>17.50</peso>
  </producto>
</listaproductos>
```

Las reglas son:

1. Una lista de productos puede tener dentro muchos productos.
2. Todo producto tiene un «codigo» cuya estructura *dos mayúsculas seguidas de un guión seguido de dos o tres cifras*
3. Todo producto *puede tener (optativo)* un elemento descripción que es de tipo texto.
4. Todo producto **debe tener** un elemento peso que debe aceptar decimales pero que nunca puede ser negativo, es decir su valor mínimo es 0

La solución se muestra a continuación:

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="listaproductos" type="tipoLista"/>
  <xsd:complexType name="tipoLista">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="producto"
            type="tipoProducto"
            maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tipoProducto">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="descripcion"
            type="xsd:string"
            minOccurs="0"/>
          <xsd:element name="peso"
            type="tipoPeso"/>
        </xsd:sequence>
        <xsd:attribute name="codigo"
          type="tipoCodigo"
          use="required"/>
      </xsd:restriction>
```

```

    </xsd:complexContent>
  </xsd:complexType>
  <xsd:simpleType name="tipoPeso">
    <xsd:restriction base="xsd:decimal">
      <xsd:minInclusive value="0"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="tipoCodigo">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Z]{2}-[0-9]{2,3}"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>

```

5.18.20 Ejercicio: validación de componentes

Validar un fichero como este:

```

<listacomponentes>
  <componente>
    <tarjetagrafica>
      <memoria>2GB</memoria>
      <precio moneda="euros">190</precio>
    </tarjetagrafica>
  </componente>
  <componente codigo="123456">
    <monitor>
      <tamano>14</tamano>
      <precio moneda="euros">99.49</precio>
    </monitor>
  </componente>
</listacomponentes>

```

Las reglas son las siguientes:

1. El elemento raíz se llama `listacomponentes`.
2. Dentro de él puede haber uno o más elementos `componente`.
3. Un componente puede ser una `tarjetagrafica` o un `monitor`.
4. Un componente puede tener un atributo llamado `codigo` cuya estructura es siempre un dígito de 6 cifras.
5. Una tarjeta gráfica siempre tiene dos elementos llamados `memoria` y `precio`.
6. La memoria siempre es una cifra seguido de GB o TB.
7. El tamaño del monitor siempre es un entero positivo.
8. El precio siempre es una cantidad positiva con decimales. El precio siempre lleva un atributo `moneda` que solo puede valer «euros» o «dolares» y que se utiliza para saber en qué moneda está el precio.

La solución se muestra a continuación:

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="listacomponentes" type="tipoLista"/>
  <xsd:complexType name="tipoLista">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">

```

```

        <xsd:sequence>
            <xsd:element name="componente"
                type="tipoComponente"
                maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:restriction>
</xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="tipoComponente">
    <xsd:complexContent>
        <xsd:restriction base="xsd:anyType">
            <xsd:choice>
                <xsd:element name="tarjetagrafica" type="tipoTarjeta"/>
                <xsd:element name="monitor" type="tipoMonitor"/>
            </xsd:choice>
            <xsd:attribute name="codigo" type="tipoCodigo"/>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="tipoCodigo">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="[1-9][0-9]{5}"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="tipoTarjeta">
    <xsd:complexContent>
        <xsd:restriction base="xsd:anyType">
            <xsd:sequence>
                <xsd:element name="memoria" type="tipoMemoria"/>
                <xsd:element name="precio" type="tipoPrecio"/>
            </xsd:sequence>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="tipoMemoria">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="[0-9]+[GT]B"/>
    </xsd:restriction>
</xsd:simpleType>
<!--Aqui definimos un precio con restriccion del cual
heredaremos despues para añadir el atributo a
la cantidad-->
<xsd:simpleType name="tipoPrecioRestringido">
    <xsd:restriction base="xsd:decimal">
        <xsd:minInclusive value="0"/>
    </xsd:restriction>
</xsd:simpleType>
<!--Aqui heredamos del tipo anterior y añadimos
el atributo-->
<xsd:complexType name="tipoPrecio">
    <xsd:simpleContent>
        <xsd:extension base="tipoPrecioRestringido">
            <xsd:attribute name="moneda" type="tipoMoneda"/>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>

```

```

<xsd:simpleType name="tipoMoneda">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="euros"/>
    <xsd:enumeration value="dolares"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="tipoMonitor">
  <xsd:complexContent>
    <xsd:restriction base="xsd:anyType">
      <xsd:sequence>
        <xsd:element name="tamaño" type="xsd:integer"/>
        <xsd:element name="precio" type="tipoPrecio"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
</xsd:schema>

```

5.18.21 Ejercicio: inventario

Varios administradores necesitan intercambiar información sobre inventario de material de oficina. Para ello, han llegado a un acuerdo sobre lo que se permite en un fichero XML de inventario. La idea básica es permitir ficheros como este:

```

<inventario>
  <objeto>
    <mesa>
      <peso>4.55</peso>
      <superficie unidad="cm2">100</superficie>
    </mesa>
  </objeto>
  <objeto>
    <silla>
      <peso>3.50</peso>
    </silla>
  </objeto>
</inventario>

```

Las reglas concretas son estas:

1. Dentro de <objeto> puede haber uno de estos dos elementos hijo: un elemento <mesa> o un elemento <silla>.
2. Toda mesa tiene un elemento hijo <peso>. El peso siempre es un decimal positivo con dos cifras decimales.
3. Toda mesa tiene una <superficie>. La superficie es un unsignedInt. La superficie siempre tiene un atributo que puede ser solo una de estas dos cadenas: m2 o cm2.
4. Toda silla tiene siempre un <peso> y las reglas de ese peso son exactamente las mismas que las reglas de <peso> del elemento <mesa>

La solución podría descomponerse de la forma siguiente:

Resolvamos primero el problema de crear un tipo para el elemento <peso>.

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="peso"
    type="tipoPeso"></xsd:element>

```

```

<xsd:simpleType name="tipoPeso">
  <xsd:restriction base="xsd:decimal">
    <xsd:minInclusive value="0"/>
    <xsd:fractionDigits value="2"/>
  </xsd:restriction>
</xsd:simpleType>
</xsd:schema>

```

Ahora resolvamos el problema del elemento <silla>. Para resolverlo, podemos aprovechar el tipo tipoPeso que acabamos de crear:

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="silla"
    type="tipoSilla"></xsd:element>
  <xsd:simpleType name="tipoPeso">
    <xsd:restriction base="xsd:decimal">
      <xsd:minInclusive value="0"/>
      <xsd:fractionDigits value="2"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="tipoSilla">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="peso"
            type="tipoPeso"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>

```

Ahora resolveremos el problema de la superficie:

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="superficie"
    type="tipoSuperficie"></xsd:element>
  <xsd:simpleType name="tipoPeso">
    <xsd:restriction base="xsd:decimal">
      <xsd:minInclusive value="0"/>
      <xsd:fractionDigits value="2"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="tipoSilla">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="peso"
            type="tipoPeso"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tipoSuperficie">
    <xsd:simpleContent>
      <xsd:extension base="xsd:unsignedInt">
        <xsd:attribute name="unidad"
          type="tipoUnidad"

```

```

        use="required"/>
    </xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
<xsd:simpleType name="tipoUnidad">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="m2"/>
        <xsd:enumeration value="cm2"/>
    </xsd:restriction>
</xsd:simpleType>
</xsd:schema>

```

Y apoyándonos en eso haremos la mesa:

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="mesa"
        type="tipoMesa"></xsd:element>
    <xsd:simpleType name="tipoPeso">
        <xsd:restriction base="xsd:decimal">
            <xsd:minInclusive value="0"/>
            <xsd:fractionDigits value="2"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="tipoSilla">
        <xsd:complexContent>
            <xsd:restriction base="xsd:anyType">
                <xsd:sequence>
                    <xsd:element name="peso"
                        type="tipoPeso"/>
                </xsd:sequence>
            </xsd:restriction>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="tipoSuperficie">
        <xsd:simpleContent>
            <xsd:extension base="xsd:unsignedInt">
                <xsd:attribute name="unidad"
                    type="tipoUnidad"
                    use="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
    <xsd:simpleType name="tipoUnidad">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="m2"/>
            <xsd:enumeration value="cm2"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="tipoMesa">
        <xsd:complexContent>
            <xsd:restriction base="xsd:anyType">
                <xsd:sequence>
                    <xsd:element name="peso"
                        type="tipoPeso"/>
                    <xsd:element name="superficie"
                        type="tipoSuperficie"/>
                </xsd:sequence>
            </xsd:restriction>
        </xsd:complexContent>
    </xsd:complexType>

```

```

    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>

```

Y ya solo queda indicar que un inventario es una lista de objetos (pondremos el maxOccurs a unbounded) e indicaremos que un objeto puede ser una elección (<xsd:choice>) entre dos tipos de objetos.

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="inventario"
    type="tipoInventario"/>
  <xsd:simpleType name="tipoPeso">
    <xsd:restriction base="xsd:decimal">
      <xsd:minInclusive value="0"/>
      <xsd:fractionDigits value="2"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="tipoSilla">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="peso"
            type="tipoPeso"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tipoSuperficie">
    <xsd:simpleContent>
      <xsd:extension base="xsd:unsignedInt">
        <xsd:attribute name="unidad"
          type="tipoUnidad"
          use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <xsd:simpleType name="tipoUnidad">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="m2"/>
      <xsd:enumeration value="cm2"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="tipoMesa">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="peso"
            type="tipoPeso"/>
          <xsd:element name="superficie"
            type="tipoSuperficie"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tipoInventario">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="objeto"

```

```

                                type="tipoObjeto"
                                maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:restriction>
</xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="tipoObjeto">
    <xsd:complexContent>
        <xsd:restriction base="xsd:anyType">
            <xsd:choice>
                <xsd:element name="mesa" type="tipoMesa"/>
                <xsd:element name="silla" type="tipoSilla"/>
            </xsd:choice>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
</xsd:schema>

```

5.19 Ejercicio tipo examen

Se necesita crear un esquema que controle la correcta sintaxis de ficheros con este estilo:

```

<productosfinancieros>
  <producto>
    <bono>
      <valoractual moneda="yenes">2.212</valoractual>
      <beneficio>-2.83</beneficio>
    </bono>
  </producto>
  <producto>
    <futuro>
      <elemento idioma="espanol">Petroleo</elemento>
      <beneficio>-3.83</beneficio>
    </futuro>
  </producto>
  <producto>
    <acciones>
      <empresa pais="usa">ENRON</empresa>
      <beneficio>2.91</beneficio>
    </acciones>
  </producto>
</productosfinancieros>

```

Las reglas concretas son las siguientes:

1. El elemento raíz es <productosfinancieros>. Dentro de él debe haber uno o más elementos <producto>.
2. Un <producto> puede ser de tres tipos: <bono>, <futuro> y <acciones>.
3. Todos los productos tienen siempre un elemento hijo llamado <beneficio> que puede ser un número con dos decimales (puede ser positivo o negativo).
4. Todo <bono> puede tener dentro un elemento llamado <valoractual> que contiene un valor decimal que puede ser positivo o negativo y tener o no decimales. El elemento <valoractual> deberá llevar dentro un atributo llamado moneda que solo puede tomar los valores dolares, euros o yenes.

5. Todo <futuro> tiene un hijo llamado <elemento> que puede contener dentro cadenas de cualquier tipo. Para saber en qué idioma está la cadena se usa un atributo llamado idioma que indica el idioma en el que está escrita la cadena.
6. Las acciones siempre tienen un elemento <empresa> que indica el nombre de la empresa y un atributo llamado país que indica de donde es la empresa. De momento queremos limitarnos a los países usa, alemania, japon y espana.

Recuérdese que siempre que no nos digan nada, se supone que un elemento o atributo es **obligatorio**. Si algo es optativo nos dirán «puede tener dentro», «puede contener», «puede aparecer», etc...

Una posible solución sería esta:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="productosfinancieros"
    type="tipoProductosFinancieros"/>
  <xsd:complexType name="tipoProductosFinancieros">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="producto"
            type="tipoProducto"
            maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tipoProducto">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:choice>
          <xsd:element name="bono"
            type="tipoBono"/>
          <xsd:element name="futuro"
            type="tipoFuturo"/>
          <xsd:element name="acciones"
            type="tipoAcciones"/>
        </xsd:choice>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tipoBono">
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:sequence>
          <xsd:element name="valoractual"
            type="tipoValorActual"/>
          <xsd:element name="beneficio"
            type="tipoBeneficio"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="tipoValorActual">
    <xsd:simpleContent>
      <xsd:extension base="xsd:decimal">
        <xsd:attribute name="moneda" type="tipoMoneda"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:schema>
```

```

</xsd:complexType>
<xsd:simpleType name="tipoMoneda">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="dolares"/>
    <xsd:enumeration value="euros"/>
    <xsd:enumeration value="yenes"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="tipoBeneficio">
  <xsd:restriction base="xsd:decimal">
    <xsd:fractionDigits value="2"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="tipoFuturo">
  <xsd:complexContent>
    <xsd:restriction base="xsd:anyType">
      <xsd:sequence>
        <xsd:element name="elemento"
                      type="tipoElemento"/>
        <xsd:element name="beneficio"
                      type="tipoBeneficio"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

```

```

        <xsd:enumeration value="japon"/>
        <xsd:enumeration value="espania"/>
        <xsd:enumeration value="usa"/>
    </xsd:restriction>
</xsd:simpleType>
</xsd:schema>

```

5.20 Ejercicio tipo examen (II)

Crear una DTD que permita validar un fichero como el siguiente:

```

<inventario>
  <objeto codigo="MM2809">
    <mesa>
      <tipo>Oficina</tipo>
      <localizacion>B09</localizacion>
    </mesa>
  </objeto>
  <objeto>
    <ordenador>
      <procesador fabricante="Intel">
        i3
      </procesador>
      <memoria unidad="GB">2</memoria>
      <discoduro>520</discoduro>
    </ordenador>
  </objeto>
</inventario>

```

Las reglas son las siguientes:

- El elemento raíz es <inventario>.
- Dentro de <inventario> debe haber una <mesa> o un <ordenador>.
- Dentro de mesa puede haber (o no) un primer elemento <tipo>. Despues debe haber un elemento <localizacion>.
- Dentro de ordenador puede haber 3 elementos optativos pero que de aparecer lo hacen en el siguiente orden.
- Primero un elemento <procesador> que puede llevar un atributo fabricante.
- Despues un elemento <memoria> que debe llevar obligatoriamente un atributo unidad.
- Despues un elemento <discoduro>.

5.21 Examen

El examen de este tema tendrá lugar el miércoles 22 de marzo de 2017.

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 comparar 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 getNodeName() 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.getNodeName();
        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.getNodeName();
        if (tipo==Node.ELEMENT_NODE){
            System.out.println("Hijo:"+nodo.getNodeName());
        } //Fin del if
    } //Fin del for
} //Fin del método

```

6.5.4 Ejercicio: extracción de información financiera

Dado el archivo XML siguiente...

```

<listado>
  <futuro precio="11.28">
    <producto>Cafe</producto>
  </futuro>
</listado>

```

```

        <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>

```

... crear un programa XML que:

- Busque todos los elementos cuya ciudad de procedencia sea «Madrid».
- Si el elemento es un futuro mostrará el contenido de la etiqueta «producto».
- Si el elemento es una divisa se mostrará el contenido de la etiqueta «nombre».
- Si el elemento es una letra se mostrará el contenido de la etiqueta «pais_emisor».
- Si el elemento es un bono se mostrará el contenido de la etiqueta «pais_de_procedencia».

Una posibilidad (incompleta) sería esta:

```

public class ProcesadorFinanzas {

    public static Element getRaiz (String rutaFichero){

        Element raiz=null;
        DocumentBuilderFactory fabrica;
        fabrica=DocumentBuilderFactory.newInstance();
        DocumentBuilder constructor;
        try {
            constructor=fabrica.newDocumentBuilder();
            FileInputStream fichero;

```

```

        fichero=new FileInputStream(rutaFichero);
        Document documento;
        documento=constructor.parse(fichero);
        raiz=documento.getDocumentElement();
    } catch (ParserConfigurationException e) {
        e.printStackTrace();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (SAXException e) {
        // TODO Auto-generated catch block
    } catch (IOException e) {
        e.printStackTrace();
    }
    return raiz;
}

public static void mostrarMadrid(Element raiz){
    NodeList hijos=raiz.getChildNodes();
    for (int i=0; i<hijos.getLength(); i++){
        Node nodoVisitado=hijos.item(i);
        if (nodoVisitado.getNodeType()
            == Node.ELEMENT_NODE) {
            //El nodo sí es un elemento y no
            //un nodo con texto \n
            Element elemVisitado;
            elemVisitado=(Element) nodoVisitado;
            if (elemVisitado.getTagName().equals("futuro")){
                procesarFuturo(elemVisitado);
            }
            if (elemVisitado.getTagName().equals("bono")){
                procesarBono(elemVisitado);
            }
            if (elemVisitado.getTagName().equals("letra")){
                procesarLetra(elemVisitado);
            }
            if (elemVisitado.getTagName().equals("divisa")){
                procesarDivisa(elemVisitado);
            }
        }
    }
}

private static void procesarDivisa(Element elemVisitado) {
    // TODO Auto-generated method stub
}

private static void procesarLetra(Element elemVisitado) {
    // TODO Auto-generated method stub
}

private static void procesarBono(Element elemVisitado) {
    // TODO Auto-generated method stub
}

private static void procesarFuturo(Element elemVisitado) {
    NodeList hijosCiudad;
    hijosCiudad=elemVisitado.getElementsByTagName(
        "ciudad_procedencia");
    Node unicaCiudad=hijosCiudad.item(0);
    NodeList ciudades=unicaCiudad.getChildNodes();

```

```

Node ciudadProcedencia=ciudades.item(1);
Element elemCiudad=(Element) ciudadProcedencia;
if (elemCiudad.getTagName().equals("madrid")){
    NodeList productos;
    productos=elemVisitado.getElementsByTagName(
        "producto");
    Element elemProducto;
    elemProducto=(Element) productos.item(0);
    System.out.println(
        elemProducto.getTextContent() );
}
}
}
public static void main(String[] args) {
    Element raiz=getRaiz(
        "c:/users/ogomez/documents/finanzas.xml");
    mostrarMadrid(raiz);
}
}

```

6.5.5 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.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

```

6.5.6 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);
        }
    }
}

```

6.5.7 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);
                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

```

6.5.8 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

```

6.5.9 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("País extraído:"+paisExtraído);
        if (paisExtraído.equals(nombrePaís)) {
            cuantos++;
        }
    }
    return cuantos;
}

```

6.5.10 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 nombrePaís){
    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;
}

```

6.5.11 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);
    }
}

```

6.5.12 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

```

6.5.13 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++;
                } //Fin del if para tokió
            } //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 tokió
            } //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 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>
    <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>
</listado>

```

```

        </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 Procesamiento con SAX

Simple Api for XML (o SAX) es un conjunto de clases para procesar XML de una forma muchísimo más eficiente (pero también más incómoda). Consiste en un *parser* que va leyendo etiqueta por etiqueta. Cada vez que el parser encuentra una etiqueta nueva nos lo comunicará mediante un evento y tendremos que incluir código de gestión de eventos que decidan que hacer.

La forma más sencilla de trabajar es hacer que nuestra clase herede de `DefaultHandler` y sobrecargar el código de los métodos `startElement` y `endElement`.

Cuando se procesa XML podemos encontrarnos con que se usen o no espacios de nombres. Si usamos espacios de nombres SAX nos devolverá un argumento pero si no los usamos SAX nos devolverá otro argumento. Observemos el siguiente código:

```
public class ProcesadorSAX extends DefaultHandler{

    @Override
    public void startElement(
        String ns, String nombreCuandoHayNS,
        String nombreCuandoNoHayNS,
        Attributes atributos)
        throws SAXException
    {
        System.out.println(nombreCuandoNoHayNS);
    }
}
```

En este código SAX avisará a nuestro método `startElement` (el nombre debe ser así), cada vez que encuentre una etiqueta. Como en nuestros documentos no estamos usando espacios de nombres nos interesa imprimir el tercer parámetro.

Para hacer que Java procese un fichero mediante SAX usando nuestra clase como procesador de etiquetas haremos lo siguiente:

```
public class ProcesadorSAX extends DefaultHandler{

    @Override
    public void startElement(
        String ns, String nombreCuandoHayNS,
        String nombreCuandoNoHayNS,
        Attributes atributos)
        throws SAXException
    {
        System.out.println(nombreCuandoNoHayNS);
    }

    public static void main(String[] args)
        throws ParserConfigurationException,
        SAXException, IOException {

        SAXParserFactory fabrica;
        fabrica=SAXParserFactory.newInstance();
        SAXParser parser=fabrica.newSAXParser();
        XMLReader lector=parser.getXMLReader();
        lector.setContentHandler(new ProcesadorSAX());
        lector.parse(
            "c:/users/ogomez/documents/finanzas.xml");
    }
}
```

Ahora Java irá leyendo el fichero etiqueta por etiqueta y mostrándonos los nombres de todas. No importará que el fichero ocupe varios GB, ya que SAX no cargará el fichero completo en memoria.

6.7.1 Ejercicio: encontrar producto

Encontrar todos los productos cuyo nombre sea «Cafe» y su mercado «América Latina».


```
public void characters(  
    char[] letras, int ini, int longitud){  
    if ((mercadoEncontrado) && (cafeEncontrado)){  
        String cadena=new String(letras, ini, longitud);  
        if (cadena.equals("América Latina")){  
            System.out.println("Encontrado Cafe de AL");  
            cafeEncontrado=false;  
            mercadoEncontrado=false;  
            productoEncontrado=false;  
            futuroEncontrado=false;  
        }  
    }  
    if ((productoEncontrado) && (futuroEncontrado)){  
        String cadena=new String(letras, ini, longitud);  
        if (cadena.equals("Cafe")){  
            cafeEncontrado=true;  
        } //Fin del if interno  
    } //Fin del if externo  
} //Fin del método characters
```

6.7.2 Ejercicio: precios

Encontrar todos las divisas cuya ciudad de procedencia sea «Madrid».

6.8 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.iesmaestredecatalrava.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 XPath

Según el W3C, XPath (que ya va por su versión 3.0) es un lenguaje diseñado para acceder a las distintas partes de un archivo XML. En nuestro caso nos va a resultar de mucha utilidad combinado con XSLT, que se verá un poco después.

XPath se basa en expresiones. Así, dado un archivo XML y una expresión XPath se dice que la expresión «se evalúa» y se obtiene un resultado que puede ser:

- Una lista de nodos.
- Un boolean (true o false)
- Un float.

- Una cadena.

XPath también ofrece algunas funciones de utilidad que se asemejan a las de algunos lenguajes de programación.

7.3.1 Acceso a elementos

El mecanismo de acceso en XPath es muy similar al acceso a directorios que ofrecen algunos sistemas operativos. Para los ejemplos siguientes se usará el siguiente archivo XML

```
<inventario>
  <producto codigo="AAA-111">
    <nombre>Teclado</nombre>
    <peso unidad="g">480</peso>
  </producto>
  <producto codigo="ACD-981">
    <nombre>Monitor</nombre>
    <peso unidad="kg">1.8</peso>
  </producto>
  <producto codigo="DEZ-138">
    <nombre>Raton</nombre>
    <peso unidad="g">50</peso>
  </producto>
</inventario>
```

Así dado este archivo tenemos las expresiones siguientes:

Si usamos la expresión `/inventario` se selecciona *el nodo inventario que cuelga de la raíz*. Como puede verse la raíz en XPath es un elemento conceptual, no existe como elemento. Además, dado como es XML solo puede haber un elemento en la raíz. Así, el resultado de evaluar la expresión `/inventario` para el archivo de ejemplo produce el resultado siguiente:

```
<inventario>
  <producto codigo="AAA-111">
    <nombre>Teclado</nombre>
    <peso unidad="g">480</peso>
  </producto>
  <producto codigo="ACD-981">
    <nombre>Monitor</nombre>
    <peso unidad="kg">1.8</peso>
  </producto>
  <producto codigo="DEZ-138">
    <nombre>Raton</nombre>
    <peso unidad="g">50</peso>
  </producto>
</inventario>
```

Como puede verse, obtenemos el propio archivo original. Sin embargo, podemos movernos a través del árbol XML de forma similar a un árbol de directorios. Y obsérvese que decimos «similar». Observemos por ejemplo que dentro de `<inventario>` hay 3 elementos `<producto>`. Si pensamos en la expresión XPath `/inventario/producto` puede que pensemos que obtendremos el primer producto (el que tiene el código AAA-111), sin embargo **una expresión XPath se parece a una consulta SQL**, y lo que obtiene la expresión es «todo elemento `<producto>` que sea hijo de `<inventario>`». Es decir, el fichero siguiente (que no es XML, sino una lista de nodos):

```
<producto codigo="AAA-111">
  <nombre>Teclado</nombre>
  <peso unidad="g">480</peso>
</producto>
```

```
<producto codigo="ACD-981">
  <nombre>Monitor</nombre>
  <peso unidad="kg">1.8</peso>
</producto>

<producto codigo="DEZ-138">
  <nombre>Raton</nombre>
  <peso unidad="g">50</peso>
</producto>
```

En cualquier lista podemos acceder a sus elementos como si fuese un vector. Sin embargo en XPath **los vectores empiezan por 1**. Por lo cual la expresión `/inventario/producto[1]` produce este resultado:

```
<producto codigo="AAA-111">
  <nombre>Teclado</nombre>
  <peso unidad="g">480</peso>
</producto>
```

Y la expresión `/inventario/producto[3]` produce este:

```
<producto codigo="DEZ-138">
  <nombre>Raton</nombre>
  <peso unidad="g">50</peso>
</producto>
```

Obsérvese que no existe el elemento 4 y que por tanto la expresión `/inventario/producto[4]` producirá un error. Otro aspecto relevante es que no deben confundirse los vectores con las condiciones (que el W3C llama «predicados»), y con las cuales podremos seleccionar nodos que cumplan ciertas condiciones. De hecho, una buena forma de verlos es asumir que en los corchetes **siempre se ponen condiciones y que si hay un número como por ejemplo el 2 nos referimos a la condición «extraer el elemento cuya posición es igual a 2»**.

Dado un elemento, también podemos extraer un cierto atributo usando la arroba @. Así, la expresión `/inventario/producto[3]/@codigo` devuelve como resultado `ACD-981`, que es el atributo código del tercer elemento `producto` que está dentro de `inventario` el cual cuelga de la raíz.

Supongamos que deseamos extraer el producto cuyo código sea «AAA-111». Si usamos `/inventario/producto` extraemos todos los elementos `producto` hijos de `inventario`, pero recordemos que entre corchetes podemos poner condiciones. Dado que queremos comprobar si `@codigo = «AAA-111»`, la expresión correcta será `/inventario/producto[@codigo="AAA-111"]`, la cual nos devuelve lo siguiente:

```
<producto codigo="AAA-111">
  <nombre>Teclado</nombre>
  <peso unidad="g">480</peso>
</producto>
```

De hecho se puede profundizar aún más y usar la expresión `/inventario/producto[@codigo="AAA-111"]/nombre` que extrae los nombres de los elementos `producto` cuyo código sea «AAA-111». Y aún más para extraer solo el texto de los elementos `nombre` usando la expresión `/inventario/producto[@codigo="AAA-111"]/nombre/text()`. Como vemos en esta última expresión ya hemos usado una función, en concreto `text()`.

En una condición podemos referirnos a cualquier hijo de un nodo, así por ejemplo, podemos extraer los productos cuyo peso sea mayor de 50 usando `/inventario/producto[peso>50]`. Sin embargo, sabemos que la unidad es importante, por lo que en realidad podemos extraer los que pesen más de 50 gramos usando esto `/inventario/producto[peso>50 and peso/@unidad="g"]`.

Si se observa despacio el fichero, se observará que en realidad el tercer producto debería aparecer también. Para ello debemos ampliar la expresión convirtiendo los 50 g a kg, es decir comparando con 0.005 kg y la expresión siguiente `/inventario/producto[(peso>=50 and peso/@unidad="g") or (peso>=0.005 and peso/@unidad="kg")]`.

Utilizando XPath y XSLT veremos que podemos transformar un XML en casi cualquier otro XML utilizando la potencia combinada de ambos lenguajes.

7.4 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>
```

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)

7.4.1 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
<u>Cervantes</u>
Poeta en Nueva York
<u>Lorca</u>

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);
}
```

7.4.2 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.

7.4.3 Ejercicio (carga de estilos)

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>

```

7.4.4 Ejercicio (conversion entre XMLs)

Dado el fichero de información del catálogo, transformar dicho XML en otro fichero en el que la etiqueta `title` vaya en español, es decir, que el resultado quede así:

```

<catalogo>
  <libro>
    <title>Don Quijote</title>
    <autor>Cervantes</autor>
  </libro>
  <libro>
    <title>
      Poeta en Nueva York
    </title>
    <autor>Lorca</autor>
  </libro>
</catalogo>

```

La solución podría ser algo así:

```

<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <catalogo>
      <xsl:for-each select="/catalogo/libro">

```

```

    <libro>
      <titulo>
        <xsl:value-of select="title"/>
      </titulo>
      <autor>
        <xsl:value-of select="autor"/>
      </autor>
    </libro>
  </xsl:for-each>
</catalogo>
</xsl:template>
</xsl:stylesheet>

```

7.4.5 Ejercicio (generación de atributos)

Dado el archivo XML del catálogo generar un XML en el que el autor vaya como un atributo del título, es decir, que quede algo así:

```

<catalogo>
  <libro>
    <titulo escritor="Cervantes">Don Quijote</titulo>
  </libro>
  <libro>
    <titulo escritor="Lorca">
      Poeta en Nueva York
    </titulo>
  </libro>
</catalogo>

```

La solución:

```

<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <catalogo>
      <xsl:for-each select="/catalogo/libro">
        <libro>
          <titulo>
            <xsl:attribute name="escritor">
              <xsl:value-of select="autor"/>
            </xsl:attribute>
            <xsl:value-of select="title"/>
          </titulo>
        </libro>
      </xsl:for-each>
    </catalogo>
  </xsl:template>
</xsl:stylesheet>

```

7.4.6 Ejercicio (tabla HTML)

Convertir el catalogo XML en una tabla HTML

Solución:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>Catalogo de libros</title>
      </head>
      <body>
        <h1>Listado de libros</h1>
        <table border="1">
          <xsl:for-each select="catalogo/libro">
            <tr>
              <td>
                <xsl:value-of select="title"/>
              </td>
              <td>
                <xsl:value-of select="autor"/>
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

7.4.7 Ejercicio (generacion)

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>
```

7.4.8 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>
```

7.4.9 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="hoja1.xsl" 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>

```

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

```

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/
↪autor">
                  <xsl:sort order="descending"/>
                  <li>

```



```

<xsl:select="."/>
<xsl:value-of
</li>
</xsl:for-each>
</ol>
</li>
</xsl:for-each> <!--Fin del recorrido de libro-->
</ol>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

7.4.11 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"/>
              </li>
            </xsl:if>
          </xsl:for-each>
        </ol>
      </body>
    </html>
  </template>
</xsl:stylesheet>

```

```

<autor">
<select="."/>
<xsl:for-each select="autores/
<xsl:sort order="descending"/>
<li>
<xsl:value-of
</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;

7.4.12 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>

```

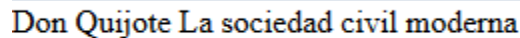
Hagámoslo paso a paso. En primer lugar tendremos que crear el fichero `ejercicio1.xsl` y crear la estructura básica:

```
<?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 navegado 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>
```



Don Quijote La sociedad civil moderna

Figura 7.1: Paso inicial del XSL

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="/">
<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>
```

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>
```



Filtrado con XSLT

1. Don Quijote
2. La sociedad civil moderna

Figura 7.2: Extrayendo los títulos con XSL

```
<head>
  <title>Filtrado con XSLT</title>
</head>
<body>
<h1>Filtrado con XSLT</h1>
<ol>
  <xsl:for-each select="catalogo/libro">

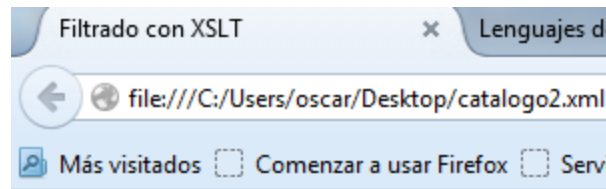
    <xsl:if test="@fechaedicion > 2000">

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

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

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">
```



Filtrado con XSLT

1. La sociedad civil moderna

Figura 7.3: Procesando los que son > 2000

```

<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

7.4.13 Ejercicio: condiciones complejas

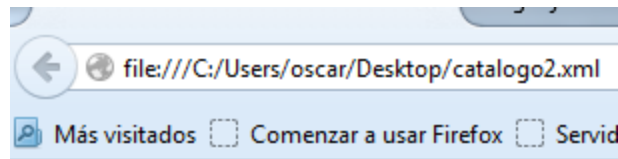
Supongamos que nos dan el siguiente fichero de inventario:

Y supongamos que nos dicen que se necesita extraer la información relativa a los productos que pesan más de 5. Una primera aproximación equivocada sería esta:

```

<xsl:template match="/">
  <inventario>
    <xsl:for-each select="inventario/elemento">
      <xsl:if test="peso > 5">
        <nombre>
          <xsl:value-of select="nombre"/>
        </nombre>
      </xsl:if>
    </xsl:for-each>
  </inventario>
</template>

```



Filtrado con XSLT

1. La sociedad civil moderna

1. Luis Diaz
2. Pedro Campos

Figura 7.4: Mostrando también los autores

```

    </xsl:if>
  </xsl:for-each>
</inventario>
</xsl:template>
</xsl:stylesheet>

```

Esta solución está equivocada porque de entrada *la pregunta está mal*. Si se refieren a 5kg solo debería mostrarse el ordenador y si se refieren a 5g solo debería mostrarse el altavoz.

Una solución correcta sería esta. Obsérvese como se meten unos if dentro de otros para extraer la información deseada.

```

<xsl:template match="/">
  <inventario>
    <xsl:for-each select="inventario/elemento">
      <xsl:if test="./peso/@unidad = 'kg'">
        <xsl:if test="peso > 5">
          <nombre>
            <xsl:value-of select="nombre"/>
          </nombre>
        </xsl:if>
      </xsl:if>
      <xsl:if test="peso/@unidad = 'g'">
        <xsl:if test="peso > 5000">
          <nombre>
            <xsl:value-of select="nombre"/>
          </nombre>
        </xsl:if>
      </xsl:if>
    </xsl:for-each>
  </inventario>
</xsl:template>
</xsl:stylesheet>

```

7.4.14 Transformación en tabla

Se nos pide convertir el inventario de antes en la tabla siguiente donde el peso debe estar normalizado y aparecer siempre en gramos:

Ordenador	10000
Altavoz	450

Una posible solución sería:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
  <head><title>Tabla de inventario</title></head>
  <body>
    <table border='1'>
      <xsl:for-each select="inventario/elemento">
        <tr>
          <td><xsl:value-of select="nombre"/></td>
          <td>
            <xsl:if test="peso/@unidad='kg'">
              <xsl:value-of select="peso * 1000"/>
            </xsl:if>
            <xsl:if test="peso/@unidad='g'">
              <xsl:value-of select="peso"/>
            </xsl:if>
          </td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

7.4.15 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>
    <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>
        <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>

```

7.4.16 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>

```

7.4.17 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:

```
<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»

Anexo: ejercicios sobre tablas

Las tablas HTML muestran cierta complejidad cuando se anidan. En los ejercicios siguientes se muestran algunas tablas junto con su resolución en HTML. Los ejercicios no se muestran con ningún orden de dificultad.

9.1 Tabla 1

Generar la tabla siguiente

Celda	Celda				Celda
Celda	Celda	Celda	Celda	Celda	Celda
	Celda	Celda	Celda	Celda	
	Celda	Celda	Celda	Celda	
	Celda	Celda	Celda	Celda	
Celda	Celda	Celda	Celda		Celda
	Celda	Celda	Celda		
	Celda	Celda	Celda		

Solución:

```
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
  <table border='1'>
    <tr>
```

```

        <td> Celda </td>
        <td> Celda </td>
        <td> Celda </td>
    </tr>
    <tr>
        <td> Celda </td>
        <td>
            <table border='1'>
                <tr>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                </tr>
                <tr>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                </tr>
                <tr>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                </tr>
                <tr>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                </tr>
            </table>
        </td>
        <td> Celda </td>
    </tr>
    <tr>
        <td> Celda </td>
        <td>
            <table border='1'>
                <tr>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                </tr>
                <tr>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                </tr>
                <tr>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                </tr>
            </table>
        </td>
        <td> Celda </td>
    </tr>

```



```

        </tr>
    </table>
</body>
</html>

```

9.2 Tabla 2

Generar la tabla siguiente

Celda	Celda	Celda	Celda
Celda	Celda	Celda	Celda
Celda	Celda	Celda	Celda
Celda	Celda	Celda	Celda
Celda	Celda	Celda	Celda
Celda	Celda	Celda	Celda
Celda	Celda	Celda	Celda
Celda	Celda	Celda	Celda
Celda	Celda	Celda	Celda

Solución:

```

<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
    <table border='1'>
        <tr>
            <td>
                <table border='1'>
                    <tr>
                        <td> Celda </td>
                        <td> Celda </td>
                    </tr>
                    <tr>
                        <td> Celda </td>
                        <td> Celda </td>
                    </tr>
                    <tr>
                        <td> Celda </td>
                        <td> Celda </td>
                    </tr>
                    <tr>
                        <td> Celda </td>
                        <td> Celda </td>
                    </tr>
                </table>
            </td>
        </tr>
    </table>

```

```

        <td>
        <table border='1'>
            <tr>
                <td> Celda </td>
                <td> Celda </td>
            </tr>
            <tr>
                <td> Celda </td>
                <td> Celda </td>
            </tr>
            <tr>
                <td> Celda </td>
                <td> Celda </td>
            </tr>
        </table>
    </td>
</tr>
<tr>
    <td> Celda </td>
    <td> Celda </td>
</tr>
<tr>
    <td> Celda </td>
    <td> Celda </td>
</tr>
<tr>
    <td> Celda </td>
    <td> Celda </td>
</tr>
</table>
</body>
</html>

```

9.3 Tabla 3

Generar la tabla siguiente

Celda	Celda		Celda	Celda						
Celda	Celda		Celda	Celda						
Celda	Celda		Celda	Celda						
Celda	<table><tr><td>Celda</td><td>Celda</td></tr><tr><td>Celda</td><td>Celda</td></tr><tr><td>Celda</td><td>Celda</td></tr></table>		Celda	Celda	Celda	Celda	Celda	Celda	Celda	Celda
	Celda	Celda								
	Celda	Celda								
Celda	Celda									

Solución:

```

<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
  <table border='1'>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td>
        <table border='1'>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
        </table>
      </td>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
  </table>
</body>
</html>

```

9.4 Tabla 4

Generar la tabla siguiente

Celda				Celda	Celda	Celda	Celda	C
				Celda	Celda	Celda	Celda	
				Celda	Celda	Celda	Celda	
Celda	Celda	Celda	Celda	Celda				C
Celda	Celda	Celda	Celda					
Celda	Celda	Celda	Celda					
Celda	Celda			Celda				C
Celda	Celda							

Solución:

```

<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
  <table border='1'>
    <tr>
      <td> Celda </td>
      <td>
        <table border='1'>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
        </table>
      </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td>
        <table border='1'>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
        </table>
      </td>
      <td> Celda </td>
    </tr>
  </table>

```

```

        <td> Celda </td>
        <td> Celda </td>

    </tr>
    <tr>

        <td> Celda </td>
        <td> Celda </td>
        <td> Celda </td>

    </tr>
    <tr>

        <td> Celda </td>
        <td> Celda </td>
        <td> Celda </td>

    </tr>
</table>
</td>
</tr>
<tr>
    <td>
        <table border='1'>
            <tr>

                <td> Celda </td>
                <td> Celda </td>
                <td> Celda </td>
                <td> Celda </td>

            </tr>
            <tr>

                <td> Celda </td>
                <td> Celda </td>
                <td> Celda </td>
                <td> Celda </td>

            </tr>
            <tr>

                <td> Celda </td>
                <td> Celda </td>
                <td> Celda </td>
                <td> Celda </td>

            </tr>
        </table>
    </td>
    <td> Celda </td>
    <td> Celda </td>
    <td> Celda </td>
</tr>
<tr>
    <td>
        <table border='1'>
            <tr>

                <td> Celda </td>
                <td> Celda </td>

            </tr>
            <tr>

                <td> Celda </td>
                <td> Celda </td>

            </tr>
        </table>
    </td>
    <td> Celda </td>
    <td> Celda </td>

```

```

        <td> Celda </td>
    </tr>
    <tr>
        <td> Celda </td>
        <td> Celda </td>
        <td> Celda </td>
        <td> Celda </td>
    </tr>
</table>
</body>
</html>

```

9.5 Tabla 5

Generar la tabla siguiente

Celda	Celda	Celda	Celda	Celda												
Celda	Celda	Celda	Celda													
Celda	Celda	Celda	Celda													
Celda				<table><tr><td>Celda</td><td>Celda</td><td>Celda</td><td>Celda</td></tr><tr><td>Celda</td><td>Celda</td><td>Celda</td><td>Celda</td></tr><tr><td>Celda</td><td>Celda</td><td>Celda</td><td>Celda</td></tr></table>	Celda	Celda	Celda	Celda	Celda	Celda	Celda	Celda	Celda	Celda	Celda	Celda
Celda	Celda	Celda	Celda													
Celda	Celda	Celda	Celda													
Celda	Celda	Celda	Celda													
Celda				Celda												
Celda	Celda			Celda												
Celda	Celda															

Solución:

```

<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
    <table border='1'>
        <tr>
            <td>
                <table border='1'>
                    <tr>
                        <td> Celda </td>
                        <td> Celda </td>
                        <td> Celda </td>
                        <td> Celda </td>
                    </tr>
                    <tr>
                        <td> Celda </td>
                        <td> Celda </td>
                        <td> Celda </td>
                        <td> Celda </td>
                    </tr>
                </table>
            </td>
            <td data-cs="3" data-kind="parent" data-rs="2"></td>
            <td data-kind="ghost"></td>
            <td data-kind="ghost"></td>
            <td data-kind="parent" data-rs="2">Celda</td>
        </tr>
        <tr>
            <td data-cs="4" data-kind="parent">Celda</td>
            <td data-kind="ghost"></td>
            <td data-kind="ghost"></td>
            <td data-kind="ghost"></td>
        </tr>
    </table>

```

```

        </tr>
        <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
        </tr>
    </table>
</td>
<td> Celda </td>
</tr>
<tr>
    <td> Celda </td>
    <td>
        <table border='1'>
            <tr>
                <td> Celda </td>
                <td> Celda </td>
                <td> Celda </td>
                <td> Celda </td>
            </tr>
            <tr>
                <td> Celda </td>
                <td> Celda </td>
                <td> Celda </td>
                <td> Celda </td>
            </tr>
            <tr>
                <td> Celda </td>
                <td> Celda </td>
                <td> Celda </td>
                <td> Celda </td>
            </tr>
        </table>
    </td>
</tr>
<tr>
    <td> Celda </td>
    <td> Celda </td>
</tr>
<tr>
    <td>
        <table border='1'>
            <tr>
                <td> Celda </td>
                <td> Celda </td>
            </tr>
            <tr>
                <td> Celda </td>
                <td> Celda </td>
            </tr>
            <tr>
                <td> Celda </td>
                <td> Celda </td>
            </tr>
        </table>
    </td>
    <td> Celda </td>

```

```

        </tr>
    </table>
</body>
</html>

```

9.6 Tabla 6

Generar la tabla siguiente

Celda	Celda	Celda	Celda		Celda
Celda	Celda	Celda	Celda		Celda
Celda	Celda	Celda	Celda	Celda	Celda
Celda	Celda	Celda	Celda	Celda	Celda

Solución:

```

<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
    <table border='1'>
        <tr>
            <td> Celda </td>
            <td>
                <table border='1'>
                    <tr>
                        <td> Celda </td>
                        <td> Celda </td>
                        <td> Celda </td>
                    </tr>
                    <tr>
                        <td> Celda </td>
                        <td> Celda </td>
                        <td> Celda </td>
                    </tr>
                </table>
            </td>
            <td> Celda </td>
        </tr>
        <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
        </tr>
    </table>

```



```

        <td> Celda </td>
    </tr>
    <tr>
        <td> Celda </td>
        <td>
            <table border='1'>
                <tr>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                </tr>
                <tr>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                </tr>
            </table>
        </td>
        <td> Celda </td>
    </tr>
</table>
</body>
</html>

```

9.7 Tabla 7

Generar la tabla siguiente

Celda	Celda	Celda								
Celda	Celda	Celda								
<table><tr><td>Celda</td><td>Celda</td><td>Celda</td><td>Celda</td></tr><tr><td>Celda</td><td>Celda</td><td>Celda</td><td>Celda</td></tr></table>	Celda	Celda	Celda	Celda	Celda	Celda	Celda	Celda	Celda	Celda
Celda	Celda	Celda	Celda							
Celda	Celda	Celda	Celda							
Celda	<table><tr><td>Celda</td><td>Celda</td></tr><tr><td>Celda</td><td>Celda</td></tr></table>	Celda	Celda	Celda	Celda	Celda				
Celda	Celda									
Celda	Celda									

Solución:

```
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
    <table border='1'>
```

```
<tr>
  <td> Celda </td>
  <td> Celda </td>
  <td> Celda </td>
</tr>
<tr>
  <td> Celda </td>
  <td> Celda </td>
  <td> Celda </td>
</tr>
<tr>
  <td>
    <table border='1'>
      <tr>
        <td> Celda </td>
        <td> Celda </td>
        <td> Celda </td>
        <td> Celda </td>
      </tr>
      <tr>
        <td> Celda </td>
        <td> Celda </td>
        <td> Celda </td>
        <td> Celda </td>
      </tr>
    </table>
  </td>
  <td> Celda </td>
  <td> Celda </td>
</tr>
<tr>
  <td> Celda </td>
  <td>
    <table border='1'>
      <tr>
        <td> Celda </td>
        <td> Celda </td>
      </tr>
      <tr>
        <td> Celda </td>
        <td> Celda </td>
      </tr>
    </table>
  </td>
  <td> Celda </td>
</tr>
</table>
</body>
</html>
```

9.8 Tabla 8

Generar la tabla siguiente

				Celda	Celda	Celda	Celda	Ce
Celda	Celda	Celda	Celda			Celda	Celda	Ce
Celda	Celda	Celda	Celda			Celda	Celda	Ce
						Celda	Celda	Ce
Celda				Celda	Celda	Celda		
Celda				Celda	Celda	Celda		
Celda				Celda	Celda	Celda		

Solución:

```
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
  <table border='1'>
    <tr>
      <td>
        <table border='1'>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
        </table>
      </td>
      <td> Celda </td>
      <td> Celda </td>
      <td>
        <table border='1'>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
        </table>
      </td>
    </tr>
  </table>
```

```

        <td> Celda </td>
      </tr>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
  </table>
</td>
</tr>
<tr>
  <td> Celda </td>
  <td> Celda </td>
  <td> Celda </td>
  <td> Celda </td>
</tr>
<tr>
  <td> Celda </td>
  <td> Celda </td>
  <td> Celda </td>
  <td> Celda </td>
</tr>
<tr>
  <td> Celda </td>
  <td> Celda </td>
  <td> Celda </td>
  <td> Celda </td>
</tr>
</table>
</body>
</html>

```

9.9 Tabla 9

Generar la tabla siguiente

Celda	Celda	Celda
	Celda	Celda
Celda	Celda	
Celda	Celda	

Solución:

```

<!DOCTYPE html>
<html>

```

```

<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
  <table border='1'>
    <tr>
      <td> Celda </td>
      <td>
        <table border='1'>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
        </table>
      </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
  </table>
</body>
</html>

```

9.10 Tabla 10

Generar la tabla siguiente

Celda	Celda	Celda	Celda	Celda	
	Celda	Celda			
	Celda	Celda			
	Celda	Celda			
Celda	Celda	Celda	Celda	Celda	Celda
		Celda	Celda	Celda	
		Celda	Celda	Celda	
		Celda	Celda	Celda	
Celda	Celda	Celda	Celda	Celda	

Solución:

```
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
  <table border='1'>
    <tr>
      <td> Celda </td>
      <td>
        <table border='1'>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
        </table>
      </td>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
      <td>
        <table border='1'>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
        </table>
      </td>
    </tr>
  </table>
```

```

        </table>
      </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td>
        <table border='1'>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
        </table>
      </td>
      <td> Celda </td>
      <td> Celda </td>
      <td>
        <table border='1'>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
        </table>
      </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
  </table>
</body>
</html>

```

9.11 Tabla 11

Generar la tabla siguiente

Celda	Celda	Celda	Celda	Celda
Celda	Celda	Celda	Celda	
Celda	Celda	Celda	Celda	
Celda	Celda	Celda	Celda	
Celda	Celda			Celda
Celda	Celda			
Celda	Celda			
Celda				Celda
Celda				Celda

Solución:

```
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
  <table border='1'>
    <tr>
      <td>
        <table border='1'>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
        </table>
      </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td>
        <table border='1'>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
        </table>
      </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td data-cs="4" data-kind="parent">Celda</td>
      <td>Celda</td>
    </tr>
    <tr>
      <td data-cs="4" data-kind="parent">Celda</td>
      <td>Celda</td>
    </tr>
  </table>

```



```

        <tr>
            <td> Celda </td>
            <td> Celda </td>
        </tr>
        <tr>
            <td> Celda </td>
            <td> Celda </td>
        </tr>
        <tr>
            <td> Celda </td>
            <td> Celda </td>
        </tr>
        </table>
    </td>
    <td> Celda </td>
</tr>
<tr>
    <td> Celda </td>
    <td> Celda </td>
</tr>
<tr>
    <td> Celda </td>
    <td> Celda </td>
</tr>
</table>
</body>
</html>

```

9.12 Tabla 12

Generar la tabla siguiente

Celda	Celda	Celda	Celda
Celda	Celda	Celda	Celda

Solución:

```

<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>

```

```

<body>
  <table border='1'>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
  </table>
</body>
</html>

```

9.13 Tabla 13

Generar la tabla siguiente

Celda	Celda			Celda
Celda	Celda			Celda
Celda	Celda			Celda
Celda	Celda	Celda	Celda	Celda
	Celda	Celda	Celda	Celda
	Celda	Celda	Celda	Celda
	Celda	Celda	Celda	Celda
Celda				

Solución:

```

<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
  <table border='1'>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
  </table>

```

```

        <td> Celda </td>
    </tr>
    <tr>
        <td> Celda </td>
        <td> Celda </td>
        <td> Celda </td>
    </tr>
    <tr>
        <td> Celda </td>
        <td>
            <table border='1'>
                <tr>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                </tr>
                <tr>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                </tr>
                <tr>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                </tr>
                <tr>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                    <td> Celda </td>
                </tr>
            </table>
        </td>
        <td> Celda </td>
    </tr>
</table>
</body>
</html>

```

9.14 Tabla 14

Generar la tabla siguiente

			Celda	Celda
Celda	Celda	Celda	Celda	Celda
			Celda	Celda
			Celda	Celda
Celda	Celda	Celda	Celda	

Solución:

```

<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
  <table border='1'>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
      <td>
        <table border='1'>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
        </table>
      </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
  </table>
</body>
</html>

```

9.15 Tabla 15

Generar la tabla siguiente

Celda	Celda	Celda
Celda	Celda	Celda

Solución:

```
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
  <table border='1'>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
  </table>
</body>
</html>
```

9.16 Tabla 16

Generar la tabla siguiente

Celda				Celda
Celda	Celda	Celda	Celda	Celda
Celda	Celda	Celda	Celda	
Celda	Celda	Celda	Celda	
Celda				Celda
Celda				Celda

Solución:

```
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
  <table border='1'>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td>
        <table border='1'>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
          <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
          </tr>
        </table>
      </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
  </table>

```

```

        <td> Celda </td>
        <td> Celda </td>
    </tr>
</table>
</body>
</html>

```

9.17 Tabla 17

Generar la tabla siguiente

Celda	Celda	Celda	Celda	Celda	Celda
	Celda	Celda	Celda		
Celda	Celda			Celda	Celda

Solución:

```

<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
    <table border='1'>
        <tr>
            <td> Celda </td>
            <td>
                <table border='1'>
                    <tr>
                        <td> Celda </td>
                        <td> Celda </td>
                        <td> Celda </td>
                    </tr>
                    <tr>
                        <td> Celda </td>
                        <td> Celda </td>
                        <td> Celda </td>
                    </tr>
                </table>
            </td>
            <td> Celda </td>
            <td> Celda </td>
        </tr>
    </table>

```

```
        <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
        </tr>
    </table>
</body>
</html>
```

9.18 Tabla 18

Generar la tabla siguiente

Celda	Celda	Celda
Celda	Celda	Celda
Celda	Celda	Celda

Solución:

```
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
    <table border='1'>
        <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
        </tr>
        <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
        </tr>
        <tr>
            <td> Celda </td>
            <td> Celda </td>
            <td> Celda </td>
        </tr>
    </table>
```



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

9.19 Tabla 19

Generar la tabla siguiente

Celda	Celda
Celda	Celda
Celda	Celda

Solución:

```
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Ejercicio</title>
</head>
<body>
  <table border='1'>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
    <tr>
      <td> Celda </td>
      <td> Celda </td>
    </tr>
  </table>
</body>
</html>
```

Anexo: ejercicios sobre formularios

En los ejercicios siguientes se han muestra el diseño básico de algunos formularios junto con el HTML que los resuelve.

10.1 Formulario 1

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Contiene los siguientes checkboxes:checkbox con el name «escritorio» , value «escritoriokde» y el texto «KDE», checkbox con el name «escritorio» , value «escritoriognome» y el texto «GNOME», checkbox con el name «escritorio» , value «escritoriunity» y el texto «Unity».

Complete, por favor

☐ KDE

☐ GNOME

☐ Unity

Solución:

```
<form>
<fieldset>
  <legend>Complete, por favor</legend>
  <input type='checkbox' name='escritorio' value='escritoriokde'> KDE    <br/>
  <input type='checkbox' name='escritorio' value='escritoriognome'> GNOME  <br/>
  <input type='checkbox' name='escritorio' value='escritoriounity'> Unity  <br/>
  <br/>
</fieldset>
</form>
```

10.2 Formulario 2

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Hay los siguientes cuadros de texto:cuadro de texto con el texto «Instituto» y el name instituto, cuadro de texto con el texto «Estudios elegidos» y el name estudios
- Hay una lista desplegable múltiple con el name «sexo» y con las siguientes opciones: opción «Mujer» con el value mujer, opción «Hombre» con el value hombre.

Solución:

```
<form>
<fieldset>
  <legend>Indique</legend>
  Instituto<input type='text' name='instituto'>
  Estudios elegidos<input type='text' name='estudios'>
  <br/>
  <select name='sexo' multiple='multiple'>
    <option value='mujer'>Mujer</option>
    <option value='hombre'>Hombre</option>
  </select>
  <br/>
</fieldset>
</form>
```

10.3 Formulario 3

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Contiene los siguientes checkboxes: checkbox con el name «procesador» , value «procesadorintel» y el texto «Intel», checkbox con el name «procesador» , value «procesadoramd» y el texto «AMD».

Opciones

☐ Intel

☐ AMD

Solución:

```
<form>
<fieldset>
  <legend>Opciones</legend>
  <input type='checkbox' name='procesador' value='procesadorintel'> Intel  <br/>
  <input type='checkbox' name='procesador' value='procesadoramd'> AMD    <br/>
  <br/>
</fieldset>
</form>
```

10.4 Formulario 4

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Hay los siguientes cuadros de texto: cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos

Complete, por favor

Nombre Apellidos

Solución:

```
<form>
<fieldset>
  <legend>Complete, por favor</legend>
  Nombre<input type='text' name='nombre'>
  Apellidos<input type='text' name='apellidos'>
  <br/>
</fieldset>
</form>
```

10.5 Formulario 5

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Hay un `textarea` que mide 6 filas y 54 columnas que lleva dentro el texto «Inserte aquí el texto»
- Contiene los siguientes `checkboxes`:checkbox con el name «idioma» , value «idiomaespanol» y el texto «Español», checkbox con el name «idioma» , value «idiomaingles» y el texto «Inglés», checkbox con el name «idioma» , value «idiomaaleman» y el texto «Alemán», checkbox con el name «idioma» , value «idiomafrances» y el texto «Francés».

- Hay un textarea que mide 7 filas y 55 columnas que lleva dentro el texto «Escriba aquí, por favor»
- Contiene los siguientes checkboxes:checkbox con el name «ciclo» , value «ciclosmir» y el texto «SMIR», checkbox con el name «ciclo» , value «cicloasir» y el texto «ASIR», checkbox con el name «ciclo» , value «ciclodam» y el texto «DAM», checkbox con el name «ciclo» , value «ciclodaw» y el texto «DAW».
- Hay los siguientes cuadros de texto:cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos, cuadro de texto con el texto «Direccion» y el name direccion

Completar estas opciones

Inserte aqui el texto

☐ Español
☐ Inglés
☐ Alemán
☐ Francés

Escriba aquí, por favor

Rellene las opciones siguientes

☐ SMIR ☐ ASIR ☐ DAM ☐ DAW
Nombre
Apellidos
Direccion

Solución:

```

<form>
<fieldset>
  <legend>Completar estas opciones</legend>
  <textarea rows='6' cols='54'>
    Inserte aqui el texto
  </textarea> <br/>
  <input type='checkbox' name='idioma' value='idiomaespanol'> Español <br/>
  <input type='checkbox' name='idioma' value='idiomaingles'> Inglés <br/>
  <input type='checkbox' name='idioma' value='idiomaaleman'> Alemán <br/>
  <input type='checkbox' name='idioma' value='idiomafrances'> Francés <br/>
  <br/>
  <textarea rows='7' cols='55'>
    Escriba aquí, por favor
  </textarea> <br/>

```



```

</fieldset>
<fieldset>
  <legend>Rellene las opciones siguientes</legend>
  <input type='checkbox' name='ciclo' value='ciclosmir'> SMIR
  <input type='checkbox' name='ciclo' value='cicloasir'> ASIR
  <input type='checkbox' name='ciclo' value='ciclodam'> DAM
  <input type='checkbox' name='ciclo' value='ciclodaw'> DAW
  <br/>
  Nombre<input type='text' name='nombre'> <br/>
  Apellidos<input type='text' name='apellidos'> <br/>
  Direccion<input type='text' name='direccion'> <br/>
  <br/>
</fieldset>
</form>

```

10.6 Formulario 6

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Hay un textarea que mide 4 filas y 48 columnas que lleva dentro el texto «Escriba aquí, por favor»
- Contiene los siguientes radiobuttons: radio con el name «red», value «red2g» y el texto «2G», radio con el name «red», value «red3g» y el texto «3G», radio con el name «red», value «red4g» y el texto «4G».
- Hay los siguientes cuadros de texto: cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos
- Hay una lista desplegable múltiple con el name «red» y con las siguientes opciones: opción «2G» con el value 2g, opción «3G» con el value 3g, opción «4G» con el value 4g.

Indique

Escriba aquí, por favor

☐ 2G

☐ 3G

☐ 4G

Indique sus preferencias por favor

Nombre Apellidos

2G

3G

4G

Solución:

```
<form>
<fieldset>
  <legend>Indique</legend>
  <textarea rows='4' cols='48'>
    Escriba aquí, por favor
  </textarea> <br/>
  <input type='radio' name='red' value='red2g'> 2G <br/>
  <input type='radio' name='red' value='red3g'> 3G <br/>
  <input type='radio' name='red' value='red4g'> 4G <br/>
</fieldset>
<fieldset>
  <legend>Indique sus preferencias por favor</legend>
  Nombre<input type='text' name='nombre'>
  Apellidos<input type='text' name='apellidos'>
  <br/>
  <select name='red' multiple='multiple'>
    <option value='2g'>2G</option>
    <option value='3g'>3G</option>
    <option value='4g'>4G</option>
  </select>
  <br/>
</fieldset>
</form>
```

```
</fieldset>  
</form>
```

10.7 Formulario 7

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Hay los siguientes cuadros de texto: cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos, cuadro de texto con el texto «Direccion» y el name direccion
- Hay los siguientes cuadros de texto: cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos, cuadro de texto con el texto «Direccion» y el name direccion
- Contiene los siguientes checkboxes: checkbox con el name «lenguaje» , value «lenguajejava» y el texto «Java», checkbox con el name «lenguaje» , value «lenguajepython» y el texto «Python», checkbox con el name «lenguaje» , value «lenguajehtml» y el texto «HTML», checkbox con el name «lenguaje» , value «lenguajevisual_basic» y el texto «Visual Basic», checkbox con el name «lenguaje» , value «lenguajecss» y el texto «CSS».
- Hay una lista desplegable múltiple con el name «asignatura» y con las siguientes opciones: opción «Geografía» con el value geografia, opción «Lengua» con el value lengua, opción «Matemáticas» con el value matematicas, opción «Historia» con el value historia.
- Hay los siguientes cuadros de texto: cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos

Completar

Nombre Apellidos

Nombre

Apellidos

Direccion

☐ Java ☐ Python ☐ HTML ☐ Visual Basic ☐ CSS

Geografía
Lengua
Matemáticas
Historia

Rellenar

Nombre

Apellidos

Solución:

```
<form>
<fieldset>
  <legend>Completar</legend>
  Nombre<input type='text' name='nombre'>
  Apellidos<input type='text' name='apellidos'>
  Direccion<input type='text' name='direccion'>
  <br/>
  Nombre<input type='text' name='nombre'> <br/>
  Apellidos<input type='text' name='apellidos'> <br/>
  Direccion<input type='text' name='direccion'> <br/>
  <br/>
  <input type='checkbox' name='lenguaje' value='lenguajejava'> Java
  <input type='checkbox' name='lenguaje' value='lenguajepython'> Python
  <input type='checkbox' name='lenguaje' value='lenguajehtml'> HTML
  <input type='checkbox' name='lenguaje' value='lenguajevisual_basic'> Visual Basic
  <input type='checkbox' name='lenguaje' value='lenguajecss'> CSS
  <br/>
  <select name='asignatura' multiple='multiple'>
    <option value='geografia'>Geografía</option>
    <option value='lengua'>Lengua</option>
    <option value='matematicas'>Matemáticas</option>
    <option value='historia'>Historia</option>
```

```
</select>
<br/>
</fieldset>
<fieldset>
  <legend>Rellenar</legend>
  Nombre<input type='text' name='nombre'> <br/>
  Apellidos<input type='text' name='apellidos'> <br/>
</fieldset>
</form>
```

10.8 Formulario 8

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Contiene los siguientes radiobuttons:radio con el name «conector» , value «conectorusb» y el texto «USB», radio con el name «conector» , value «conectorparalelo» y el texto «Paralelo», radio con el name «conector» , value «conectorps2» y el texto «PS2».
- Contiene los siguientes radiobuttons:radio con el name «velocidad» , value «velocidad100_mbits» y el texto «100 Mbits», radio con el name «velocidad» , value «velocidad1000_mbits» y el texto «1000 Mbits».
- Hay los siguientes cuadros de texto:cuadro de texto con el texto «Instituto» y el name instituto, cuadro de texto con el texto «Estudios elegidos» y el name estudios

Completar estas opciones

☐ USB
☐ Paralelo
☐ PS2

☐ 100 Mbits
☐ 1000 Mbits

Completar

Instituto

Estudios elegidos

Solución:

```
<form>
<fieldset>
  <legend>Completar estas opciones</legend>
  <input type='radio' name='conector' value='conectorusb'> USB    <br/>
  <input type='radio' name='conector' value='conectorparalelo'> Paralelo    <br/>
  <input type='radio' name='conector' value='conectorps2'> PS2    <br/>
  <br/>
  <input type='radio' name='velocidad' value='velocidad100_mbits'> 100 Mbits    <br/>
  <input type='radio' name='velocidad' value='velocidad1000_mbits'> 1000 Mbits    <br/>
  <br/>
</fieldset>
<fieldset>
  <legend>Completar</legend>
  Instituto<input type='text' name='instituto'> <br/>
  Estudios elegidos<input type='text' name='estudios'> <br/>
  <br/>
</fieldset>
</form>
```

10.9 Formulario 9

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Contiene los siguientes radiobuttons:radio con el name «aula» , value «aulaa01» y el texto «A01», radio con el name «aula» , value «aulaa02» y el texto «A02», radio con el name «aula» , value «aulaa03» y el texto «A03».
- Contiene los siguientes radiobuttons:radio con el name «idioma» , value «idiomaespanol» y el texto «Español», radio con el name «idioma» , value «idiomaingles» y el texto «Inglés», radio con el name «idioma» , value «idiomaaleman» y el texto «Alemán», radio con el name «idioma» , value «idiomafrances» y el texto «Francés».
- Hay los siguientes cuadros de texto:cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos

Formulario de opciones

☐ A01
☐ A02
☐ A03

☐ Español
☐ Inglés
☐ Alemán
☐ Francés

Nombre

Apellidos

Solución:

```
<form>
<fieldset>
  <legend>Formulario de opciones</legend>
  <input type='radio' name='aula' value='aulaa01'> A01
  <input type='radio' name='aula' value='aulaa02'> A02
  <input type='radio' name='aula' value='aulaa03'> A03
```

```
<br/>
<input type='radio' name='idioma' value='idiomaespanol'> Español <br/>
<input type='radio' name='idioma' value='idiomaingles'> Inglés <br/>
<input type='radio' name='idioma' value='idiomaaleman'> Alemán <br/>
<input type='radio' name='idioma' value='idiomafrances'> Francés <br/>
<br/>
Nombre<input type='text' name='nombre'> <br/>
Apellidos<input type='text' name='apellidos'> <br/>
<br/>
</fieldset>
</form>
```

10.10 Formulario 10

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Hay los siguientes cuadros de texto:cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos, cuadro de texto con el texto «Direccion» y el name direccion
- Hay los siguientes cuadros de texto:cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos, cuadro de texto con el texto «Direccion» y el name direccion
- Hay una lista desplegable múltiple con el name «asignatura» y con las siguientes opciones: opción «Matemáticas» con el value matematicas, opción «Historia» con el value historia, opción «Geografía» con el value geografia, opción «Lengua» con el value lengua.
- Hay un textarea que mide 8 filas y 49 columnas que lleva dentro el texto «Utilice este recuadro por favor»
- Contiene los siguientes radiobuttons:radio con el name «idioma» , value «idiomaespanol» y el texto «Español», radio con el name «idioma» , value «idiomaingles» y el texto «Inglés», radio con el name «idioma» , value «idiomaaleman» y el texto «Alemán», radio con el name «idioma» , value «idiomafrances» y el texto «Francés».
- Contiene los siguientes radiobuttons:radio con el name «navegador» , value «navegadorfirefox» y el texto «Firefox», radio con el name «navegador» , value «navegadorchrome» y el texto «Chrome», radio con el name «navegador» , value «navegadoropera» y el texto «Opera», radio con el name «navegador» , value «navegadorie» y el texto «IE».
- Hay los siguientes cuadros de texto:cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos, cuadro de texto con el texto «Direccion» y el name direccion

Rellenar

Nombre Apellidos

Nombre

Apellidos

Direccion

Matemáticas ▾

Complete, por favor

Utilice este recuadro por favor

☐ Español

☐ Inglés

☐ Alemán

☐ Francés

☐ Firefox ☐ Chrome ☐ Opera ☐ IE

Nombre Apellidos

Solución:

```
<form>
<fieldset>
  <legend>Rellenar</legend>
  Nombre<input type='text' name='nombre'>
  Apellidos<input type='text' name='apellidos'>
  Direccion<input type='text' name='direccion'>
  <br/>
  Nombre<input type='text' name='nombre'> <br/>
  Apellidos<input type='text' name='apellidos'> <br/>
  Direccion<input type='text' name='direccion'> <br/>
  <br/>
  <select name='asignatura' >
    <option value='matematicas'>Matemáticas</option>
    <option value='historia'>Historia</option>
    <option value='geografia'>Geografía</option>
    <option value='lengua'>Lengua</option>
  </select>
  <br/>
</fieldset>
<fieldset>
  <legend>Complete, por favor</legend>
  <textarea rows='8' cols='49'>
```

```
Utilice este recuadro por favor
</textarea> <br/>
<input type='radio' name='idioma' value='idiomaespanol'> Español <br/>
<input type='radio' name='idioma' value='idiomaingles'> Inglés <br/>
<input type='radio' name='idioma' value='idiomaaleman'> Alemán <br/>
<input type='radio' name='idioma' value='idiomafrances'> Francés <br/>
<br/>
<input type='radio' name='navegador' value='navegadorfirefox'> Firefox
<input type='radio' name='navegador' value='navegadorchrome'> Chrome
<input type='radio' name='navegador' value='navegadoropera'> Opera
<input type='radio' name='navegador' value='navegadorie'> IE
<br/>
Nombre<input type='text' name='nombre'>
Apellidos<input type='text' name='apellidos'>
Direccion<input type='text' name='direccion'>
<br/>
</fieldset>
</form>
```

10.11 Formulario 11

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Hay una lista desplegable múltiple con el name «lenguaje» y con las siguientes opciones: opción «Python» con el value python, opción «HTML» con el value html, opción «Visual Basic» con el value visual_basic, opción «Java» con el value java.
- Hay una lista desplegable múltiple con el name «lenguaje» y con las siguientes opciones: opción «Java» con el value java, opción «Python» con el value python, opción «HTML» con el value html, opción «Visual Basic» con el value visual_basic.
- Hay los siguientes cuadros de texto: cuadro de texto con el texto «Instituto» y el name instituto, cuadro de texto con el texto «Estudios elegidos» y el name estudios
- Contiene los siguientes radiobuttons: radio con el name «idioma» , value «idiomaingles» y el texto «Inglés», radio con el name «idioma» , value «idiomaaleman» y el texto «Alemán», radio con el name «idioma» , value «idiomafrances» y el texto «Francés».
- Hay un textarea que mide 8 filas y 60 columnas que lleva dentro el texto «Inserte aquí el texto»

Rellenar

Python ▼

Java
Python
HTML
Visual Basic

Instituto

Estudios elegidos

☐ Inglés
☐ Alemán
☐ Francés

Por favor, complete estas opciones

Inserte aquí el texto

Solución:

```
<form>
<fieldset>
  <legend>Rellenar</legend>
  <select name='lenguaje' >
    <option value='python'>Python</option>
    <option value='html'>HTML</option>
    <option value='visual_basic'>Visual Basic</option>
    <option value='java'>Java</option>
  </select>
  <br/>
  <select name='lenguaje' multiple='multiple'>
    <option value='java'>Java</option>
    <option value='python'>Python</option>
    <option value='html'>HTML</option>
    <option value='visual_basic'>Visual Basic</option>
  </select>
  <br/>
  Instituto<input type='text' name='instituto'> <br/>
  Estudios elegidos<input type='text' name='estudios'> <br/>
  <br/>
  <input type='radio' name='idioma' value='idiomaingles'> Inglés <br/>
  <input type='radio' name='idioma' value='idiomaaleman'> Alemán <br/>
```

```
<input type='radio' name='idioma' value='idiomafrances'> Francés <br/>
<br/>
</fieldset>
<fieldset>
  <legend>Por favor, complete estas opciones</legend>
  <textarea rows='8' cols='60'>
    Inserte aqui el texto
  </textarea> <br/>
</fieldset>
</form>
```

10.12 Formulario 12

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Contiene los siguientes radiobuttons:radio con el name «procesador», value «procesadoramd» y el texto «AMD», radio con el name «procesador», value «procesadorintel_i5» y el texto «Intel i5», radio con el name «procesador», value «procesadorintel_i7» y el texto «Intel i7».
- Contiene los siguientes radiobuttons:radio con el name «idioma», value «idiomaespanol» y el texto «Español», radio con el name «idioma», value «idiomaingles» y el texto «Inglés», radio con el name «idioma», value «idiomaaleman» y el texto «Alemán», radio con el name «idioma», value «idiomafrances» y el texto «Francés».
- Hay una lista desplegable múltiple con el name «medio_transporte» y con las siguientes opciones: opción «Automóvil» con el value automovil, opción «Moto» con el value moto, opción «Autobús» con el value autobus.
- Hay los siguientes cuadros de texto:cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos, cuadro de texto con el texto «Direccion» y el name direccion
- Contiene los siguientes radiobuttons:radio con el name «lenguaje», value «lenguajejava» y el texto «Java», radio con el name «lenguaje», value «lenguajepython» y el texto «Python», radio con el name «lenguaje», value «lenguajehtml» y el texto «HTML», radio con el name «lenguaje», value «lenguajevisual_basic» y el texto «Visual Basic».
- Hay un textarea que mide 4 filas y 49 columnas que lleva dentro el texto «Inserte aqui el texto»
- Contiene los siguientes radiobuttons:radio con el name «ide», value «ideeclipse» y el texto «Eclipse», radio con el name «ide», value «idenetbeans» y el texto «Netbeans», radio con el name «ide», value «idenotepad» y el texto «Notepad», radio con el name «ide», value «ideidea» y el texto «IDEA».
- Hay una lista desplegable múltiple con el name «ciclo» y con las siguientes opciones: opción «ASIR» con el value asir, opción «DAM» con el value dam, opción «DAW» con el value daw.

Rellene las opciones siguientes

☐ AMD
☐ Intel i5
☐ Intel i7

☐ Español ☐ Inglés ☐ Alemán ☐ Francés

Automóvil
 Moto
 Autobús

Nombre Apellidos

Rellenar

☐ Java ☐ Python ☐ HTML ☐ Visual Basic

Inserte aquí el texto

☐ Eclipse
☐ Netbeans
☐ Notepad
☐ IDEA

ASIR ▼

Solución:

```
<form>
<fieldset>
  <legend>Rellene las opciones siguientes</legend>
  <input type='radio' name='procesador' value='procesadoramd'> AMD   <br/>
  <input type='radio' name='procesador' value='procesadorintel_i5'> Intel i5   <br/>
  <input type='radio' name='procesador' value='procesadorintel_i7'> Intel i7   <br/>
  <br/>
  <input type='radio' name='idioma' value='idiomaespanol'> Español
  <input type='radio' name='idioma' value='idiomaingles'> Inglés
  <input type='radio' name='idioma' value='idiomaaleman'> Alemán
  <input type='radio' name='idioma' value='idiomafrances'> Francés
  <br/>
  <select name='medio_transporte' multiple='multiple'>
    <option value='automovil'>Automóvil</option>
    <option value='moto'>Moto</option>
    <option value='autobus'>Autobús</option>
  </select>
  <br/>
  Nombre<input type='text' name='nombre'>
  Apellidos<input type='text' name='apellidos'>
  Direccion<input type='text' name='direccion'>
  <br/>
```

```

</fieldset>
<fieldset>
  <legend>Rellenar</legend>
  <input type='radio' name='lenguaje' value='lenguajejava'> Java
  <input type='radio' name='lenguaje' value='lenguajepython'> Python
  <input type='radio' name='lenguaje' value='lenguajehtml'> HTML
  <input type='radio' name='lenguaje' value='lenguajevisual_basic'> Visual Basic
  <br/>
  <textarea rows='4' cols='49'>
    Inserte aqui el texto
  </textarea> <br/>
  <input type='radio' name='ide' value='ideeclipse'> Eclipse <br/>
  <input type='radio' name='ide' value='idenetbeans'> Netbeans <br/>
  <input type='radio' name='ide' value='idenotepad'> Notepad <br/>
  <input type='radio' name='ide' value='ideidea'> IDEA <br/>
  <br/>
  <select name='ciclo' >
    <option value='asir'>ASIR</option>
    <option value='dam'>DAM</option>
    <option value='daw'>DAW</option>
  </select>
  <br/>
</fieldset>
</form>

```

10.13 Formulario 13

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Hay un `textarea` que mide 4 filas y 56 columnas que lleva dentro el texto «Utilice este cuadro para escribir»
- Hay los siguientes cuadros de texto: cuadro de texto con el texto «Nombre» y el `name` `nombre`, cuadro de texto con el texto «Apellidos» y el `name` `apellidos`
- Contiene los siguientes `checkboxes`: `checkbox` con el `name` «`dia`», `value` «`dialunes`» y el texto «Lunes», `checkbox` con el `name` «`dia`», `value` «`diamartes`» y el texto «Martes», `checkbox` con el `name` «`dia`», `value` «`diamiercoles`» y el texto «Miércoles», `checkbox` con el `name` «`dia`», `value` «`diajueves`» y el texto «Jueves», `checkbox` con el `name` «`dia`», `value` «`diasabado`» y el texto «Sabado».
- Contiene los siguientes `checkboxes`: `checkbox` con el `name` «`idioma`», `value` «`idiomaingles`» y el texto «Inglés», `checkbox` con el `name` «`idioma`», `value` «`idiomaaleman`» y el texto «Alemán», `checkbox` con el `name` «`idioma`», `value` «`idiomafrances`» y el texto «Francés».
- Contiene los siguientes `checkboxes`: `checkbox` con el `name` «`red`», `value` «`red2g`» y el texto «2G», `checkbox` con el `name` «`red`», `value` «`red3g`» y el texto «3G», `checkbox` con el `name` «`red`», `value` «`red4g`» y el texto «4G».
- Hay una lista desplegable múltiple con el `name` «`idioma`» y con las siguientes opciones: opción «Inglés» con el `value` `ingles`, opción «Alemán» con el `value` `aleman`, opción «Francés» con el `value` `frances`.
- Contiene los siguientes `radiobuttons`: `radio` con el `name` «`procesador`», `value` «`procesadorintel`» y el texto «Intel», `radio` con el `name` «`procesador`», `value` «`procesadoramd`» y el texto «AMD».

Indique

Utilice este cuadro para escribir

Nombre

Apellidos

☐ Lunes
☐ Martes
☐ Miércoles
☐ Jueves
☐ Sabado

Rellene las opciones siguientes

☐ Inglés ☐ Alemán ☐ Francés
☐ 2G
☐ 3G
☐ 4G

Inglés
 Alemán
 Francés

☐ Intel ☐ AMD

Solución:

```
<form>
<fieldset>
  <legend>Indique</legend>
  <textarea rows='4' cols='56'>
    Utilice este cuadro para escribir
  </textarea> <br/>
  Nombre<input type='text' name='nombre'> <br/>
  Apellidos<input type='text' name='apellidos'> <br/>
  <br/>
  <input type='checkbox' name='dia' value='dialunes'> Lunes <br/>
  <input type='checkbox' name='dia' value='diamartes'> Martes <br/>
  <input type='checkbox' name='dia' value='diamiercoles'> Miércoles <br/>
  <input type='checkbox' name='dia' value='diajueves'> Jueves <br/>
  <input type='checkbox' name='dia' value='diasabado'> Sabado <br/>
  <br/>
</fieldset>
<fieldset>
  <legend>Rellene las opciones siguientes</legend>
  <input type='checkbox' name='idioma' value='idiomaingles'> Inglés
  <input type='checkbox' name='idioma' value='idiomaaleman'> Alemán
  <input type='checkbox' name='idioma' value='idiomafrances'> Francés
  <br/>
  <input type='radio' name='procesador' value='intel'> Intel
  <input type='radio' name='procesador' value='amd'> AMD
</fieldset>
</form>
```

```
<input type='checkbox' name='red' value='red2g'> 2G <br/>
<input type='checkbox' name='red' value='red3g'> 3G <br/>
<input type='checkbox' name='red' value='red4g'> 4G <br/>
<br/>
<select name='idioma' multiple='multiple'>
  <option value='ingles'>Inglés</option>
  <option value='aleman'>Alemán</option>
  <option value='frances'>Francés</option>
</select>
<br/>
<input type='radio' name='procesador' value='procesadorintel'> Intel
<input type='radio' name='procesador' value='procesadoramd'> AMD
<br/>
</fieldset>
</form>
```

10.14 Formulario 14

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Contiene los siguientes radiobuttons:radio con el name «medio_transporte» , value «medio_transporteautomovil» y el texto «Automóvil», radio con el name «medio_transporte» , value «medio_transportemoto» y el texto «Moto», radio con el name «medio_transporte» , value «medio_transporteautobus» y el texto «Autobús».
- Hay un textarea que mide 6 filas y 47 columnas que lleva dentro el texto «Utilice este recuadro por favor»
- Contiene los siguientes checkboxes:checkbox con el name «formato» , value «formatojpg» y el texto «JPG», checkbox con el name «formato» , value «formatopng» y el texto «PNG».
- Contiene los siguientes radiobuttons:radio con el name «idioma» , value «idiomaespanol» y el texto «Español», radio con el name «idioma» , value «idiomaingles» y el texto «Inglés», radio con el name «idioma» , value «idiomaaleman» y el texto «Alemán», radio con el name «idioma» , value «idiomafrances» y el texto «Francés».

Preferencias

☐ Automóvil
☐ Moto
☐ Autobús

Utilice este recuadro por favor

☐ JPG
☐ PNG

☐ Español ☐ Inglés ☐ Alemán ☐ Francés

Solución:

```
<form>
<fieldset>
  <legend>Preferencias</legend>
  <input type='radio' name='medio_transporte' value='medio_transporteautomovil'>
↪Automóvil    <br/>
  <input type='radio' name='medio_transporte' value='medio_transportemoto'> Moto
↪<br/>
  <input type='radio' name='medio_transporte' value='medio_transporteautobus'>
↪Autobús    <br/>
  <br/>
  <textarea rows='6' cols='47'>
    Utilice este recuadro por favor
  </textarea> <br/>
  <input type='checkbox' name='formato' value='formatojpg'> JPG    <br/>
  <input type='checkbox' name='formato' value='formatopng'> PNG    <br/>
  <br/>
  <input type='radio' name='idioma' value='idiomaespanol'> Español
  <input type='radio' name='idioma' value='idiomaingles'> Inglés
  <input type='radio' name='idioma' value='idiomaaleman'> Alemán
  <input type='radio' name='idioma' value='idiomafrances'> Francés
  <br/>
</fieldset>
```

```
</form>
```

10.15 Formulario 15

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Hay los siguientes cuadros de texto:cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos, cuadro de texto con el texto «Direccion» y el name direccion
- Contiene los siguientes radiobuttons:radio con el name «conexion» , value «conexionwifi» y el texto «Wifi», radio con el name «conexion» , value «conexioncable» y el texto «Cable», radio con el name «conexion» , value «conexionfibra» y el texto «Fibra».
- Hay los siguientes cuadros de texto:cuadro de texto con el texto «Instituto» y el name instituto, cuadro de texto con el texto «Estudios elegidos» y el name estudios
- Hay una lista desplegable múltiple con el name «dia» y con las siguientes opciones: opción «Lunes» con el value lunes, opción «Martes» con el value martes, opción «Miércoles» con el value miercoles, opción «Jueves» con el value jueves, opción «Sabado» con el value sabado.
- Contiene los siguientes checkboxes:checkbox con el name «asignatura» , value «asignaturahistoria» y el texto «Historia», checkbox con el name «asignatura» , value «asignaturageografia» y el texto «Geografía», checkbox con el name «asignatura» , value «asignaturalengua» y el texto «Lengua», checkbox con el name «asignatura» , value «asignaturamatematicas» y el texto «Matemáticas».
- Hay un textarea que mide 6 filas y 50 columnas que lleva dentro el texto «Inserte aqui el texto»

Formulario de opciones

Nombre

Apellidos

Direccion

☐ Wifi ☐ Cable ☐ Fibra

Instituto

Estudios elegidos

☐ Lunes
☐ Martes
☐ Miércoles
☐ Jueves

Preferencias

☐ Historia ☐ Geografía ☐ Lengua ☐ Matemáticas

Inserte aqui el texto

Solución:

```
<form>
<fieldset>
  <legend>Formulario de opciones</legend>
  Nombre<input type='text' name='nombre'> <br/>
  Apellidos<input type='text' name='apellidos'> <br/>
  Direccion<input type='text' name='direccion'> <br/>
  <br/>
  <input type='radio' name='conexion' value='conexionwifi'> Wifi
  <input type='radio' name='conexion' value='conexioncable'> Cable
  <input type='radio' name='conexion' value='conexionfibra'> Fibra
  <br/>
  Instituto<input type='text' name='instituto'> <br/>
  Estudios elegidos<input type='text' name='estudios'> <br/>
  <br/>
  <select name='dia' multiple='multiple'>
    <option value='lunes'>Lunes</option>
    <option value='martes'>Martes</option>
    <option value='miercoles'>Miércoles</option>
    <option value='jueves'>Jueves</option>
    <option value='sabado'>Sabado</option>
  </select>
  <br/>
</fieldset>
```

```
</fieldset>
<fieldset>
  <legend>Preferencias</legend>
  <input type='checkbox' name='asignatura' value='asignaturahistoria'> Historia
  <input type='checkbox' name='asignatura' value='asignaturageografia'> Geografía
  <input type='checkbox' name='asignatura' value='asignaturalengua'> Lengua
  <input type='checkbox' name='asignatura' value='asignaturamatematicas'> Matemáticas
  <br/>
  <textarea rows='6' cols='50'>
    Inserte aqui el texto
  </textarea>  <br/>
</fieldset>
</form>
```

10.16 Formulario 16

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Contiene los siguientes radiobuttons:radio con el name «formato» , value «formatojpg» y el texto «JPG», radio con el name «formato» , value «formatopng» y el texto «PNG».
- Contiene los siguientes checkboxes:checkbox con el name «escritorio» , value «escritoriokde» y el texto «KDE», checkbox con el name «escritorio» , value «escritoriognome» y el texto «GNOME», checkbox con el name «escritorio» , value «escritoriounity» y el texto «Unity».

Preferencias

☐ JPG
 ☐ PNG

☐ KDE

☐ GNOME

☐ Unity

Solución:

```
<form>
<fieldset>
  <legend>Preferencias</legend>
  <input type='radio' name='formato' value='formatojpg'> JPG
  <input type='radio' name='formato' value='formatopng'> PNG
  <br/>
  <input type='checkbox' name='escritorio' value='escritoriokde'> KDE    <br/>
  <input type='checkbox' name='escritorio' value='escritoriognome'> GNOME  <br/>
  <input type='checkbox' name='escritorio' value='escritoriounity'> Unity  <br/>
  <br/>
</fieldset>
</form>
```

10.17 Formulario 17

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Hay un `textarea` que mide 7 filas y 46 columnas que lleva dentro el texto «Por favor, escriba aquí»
- Hay un `textarea` que mide 6 filas y 60 columnas que lleva dentro el texto «Inserte aquí el texto»

Por favor, complete estas opciones

Por favor, escriba aquí

Inserte aquí el texto

Solución:

```
<form>
<fieldset>
  <legend>Por favor, complete estas opciones</legend>
  <textarea rows='7' cols='46'>
    Por favor, escriba aquí
  </textarea> <br/>
  <textarea rows='6' cols='60'>
    Inserte aquí el texto
  </textarea> <br/>
</fieldset>
</form>
```

10.18 Formulario 18

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Contiene los siguientes `radiobuttons`: radio con el name «procesador», value «procesadoramd» y el texto «AMD», radio con el name «procesador», value «procesadorintel_i5» y el texto «Intel i5», radio con el name «procesador», value «procesadorintel_i7» y el texto «Intel i7».

- Hay los siguientes cuadros de texto:cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos, cuadro de texto con el texto «Direccion» y el name direccion
- Contiene los siguientes checkboxes:checkbox con el name «sexo» , value «sexomujer» y el texto «Mujer», checkbox con el name «sexo» , value «sexohombre» y el texto «Hombre».
- Contiene los siguientes checkboxes:checkbox con el name «ciclo» , value «ciclosmir» y el texto «SMIR», checkbox con el name «ciclo» , value «cicloasir» y el texto «ASIR», checkbox con el name «ciclo» , value «ciclodam» y el texto «DAM», checkbox con el name «ciclo» , value «ciclodaw» y el texto «DAW».
- Contiene los siguientes radiobuttons:radio con el name «navegador» , value «navegadorfirefox» y el texto «Firefox», radio con el name «navegador» , value «navegadorchrome» y el texto «Chrome», radio con el name «navegador» , value «navegadoropera» y el texto «Opera», radio con el name «navegador» , value «navegadorie» y el texto «IE».

Opciones

☐ AMD
☐ Intel i5
☐ Intel i7

Rellene las opciones siguientes

Nombre

Apellidos

Direccion

☐ Mujer
☐ Hombre

☐ SMIR
☐ ASIR
☐ DAM
☐ DAW

☐ Firefox
☐ Chrome
☐ Opera
☐ IE

Solución:

```

<form>
<fieldset>
  <legend>Opciones</legend>
  <input type='radio' name='procesador' value='procesadoramd'> AMD
  <input type='radio' name='procesador' value='procesadorintel_i5'> Intel i5
  <input type='radio' name='procesador' value='procesadorintel_i7'> Intel i7
  <br/>
</fieldset>

```

```

<fieldset>
  <legend>Rellene las opciones siguientes</legend>
  Nombre<input type='text' name='nombre'> <br/>
  Apellidos<input type='text' name='apellidos'> <br/>
  Direccion<input type='text' name='direccion'> <br/>
  <br/>
  <input type='checkbox' name='sexo' value='sexomujer'> Mujer
  <input type='checkbox' name='sexo' value='sexohombre'> Hombre
  <br/>
  <input type='checkbox' name='ciclo' value='ciclosmir'> SMIR
  <input type='checkbox' name='ciclo' value='cicloasir'> ASIR
  <input type='checkbox' name='ciclo' value='ciclodam'> DAM
  <input type='checkbox' name='ciclo' value='ciclodaw'> DAW
  <br/>
  <input type='radio' name='navegador' value='navegadorfirefox'> Firefox
  <input type='radio' name='navegador' value='navegadorchrome'> Chrome
  <input type='radio' name='navegador' value='navegadoropera'> Opera
  <input type='radio' name='navegador' value='navegadorie'> IE
  <br/>
</fieldset>
</form>

```

10.19 Formulario 19

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Hay los siguientes cuadros de texto:cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos
- Contiene los siguientes checkboxes:checkbox con el name «preferencia» , value «preferenciaciencias» y el texto «Ciencias», checkbox con el name «preferencia» , value «preferencialetras» y el texto «Letras».
- Hay una lista desplegable múltiple con el name «asignatura» y con las siguientes opciones: opción «Historia» con el value historia, opción «Geografía» con el value geografia, opción «Lengua» con el value lengua, opción «Matemáticas» con el value matematicas.
- Hay una lista desplegable múltiple con el name «asignatura» y con las siguientes opciones: opción «Geografía» con el value geografia, opción «Lengua» con el value lengua, opción «Matemáticas» con el value matematicas, opción «Historia» con el value historia.
- Hay un textarea que mide 4 filas y 49 columnas que lleva dentro el texto «Escriba aquí, por favor»
- Hay los siguientes cuadros de texto:cuadro de texto con el texto «Nombre» y el name nombre, cuadro de texto con el texto «Apellidos» y el name apellidos, cuadro de texto con el texto «Direccion» y el name direccion
- Contiene los siguientes radiobuttons:radio con el name «asignatura» , value «asignaturalengua» y el texto «Lengua», radio con el name «asignatura» , value «asignaturamatematicas» y el texto «Matemáticas», radio con el name «asignatura» , value «asignaturahistoria» y el texto «Historia», radio con el name «asignatura» , value «asignaturageografia» y el texto «Geografía».

Complete, por favor

Nombre Apellidos

☐ Ciencias
☐ Letras

☐ Historia
☐ Geografía
☐ Lengua
☐ Matemáticas

Indique

Geografía

Escriba aquí, por favor

Nombre Apellidos

☐ Lengua
☐ Matemáticas
☐ Historia
☐ Geografía

Solución:

```
<form>
<fieldset>
  <legend>Complete, por favor</legend>
  Nombre<input type='text' name='nombre'>
  Apellidos<input type='text' name='apellidos'>
  <br/>
  <input type='checkbox' name='preferencia' value='preferenciaciencias'> Ciencias
  <br/>
  <input type='checkbox' name='preferencia' value='preferencialetras'> Letras  <br/>
  <br/>
  <select name='asignatura' multiple='multiple'>
    <option value='historia'>Historia</option>
    <option value='geografia'>Geografía</option>
    <option value='lengua'>Lengua</option>
    <option value='matematicas'>Matemáticas</option>
  </select>
  <br/>
</fieldset>
<fieldset>
  <legend>Indique</legend>
  <select name='asignatura' >
    <option value='geografia'>Geografía</option>
```

```
<option value='lengua'>Lengua</option>
<option value='matematicas'>Matemáticas</option>
<option value='historia'>Historia</option>
</select>
<br/>
<textarea rows='4' cols='49'>
  Escriba aquí, por favor
</textarea> <br/>
Nombre<input type='text' name='nombre'>
Apellidos<input type='text' name='apellidos'>
Direccion<input type='text' name='direccion'>
<br/>
<input type='radio' name='asignatura' value='asignaturalengua'> Lengua
<input type='radio' name='asignatura' value='asignaturamatematicas'> Matemáticas
<input type='radio' name='asignatura' value='asignaturahistoria'> Historia
<input type='radio' name='asignatura' value='asignaturageografia'> Geografía
<br/>
</fieldset>
</form>
```

10.20 Formulario 20

Generar el formulario siguiente de acuerdo a los siguientes requisitos

- Contiene los siguientes checkboxes:checkbox con el name «ciclo» , value «cicloasir» y el texto «ASIR», checkbox con el name «ciclo» , value «ciclodam» y el texto «DAM», checkbox con el name «ciclo» , value «ciclodaw» y el texto «DAW».

Rellenar

☐ ASIR
☐ DAM
☐ DAW

Solución:

```
<form>
<fieldset>
  <legend>Rellenar</legend>
  <input type='checkbox' name='ciclo' value='cicloasir'> ASIR  <br/>
  <input type='checkbox' name='ciclo' value='ciclodam'> DAM  <br/>
  <input type='checkbox' name='ciclo' value='ciclodaw'> DAW  <br/>
  <br/>
</fieldset>
</form>
```


11.1 Fichero origen

Para los ejercicios siguiente supondremos que se va a trabajar con el fichero que se muestra a continuación:

```
<inventario>
  <producto codigo="P1">
    <peso unidad="kg">10</peso>
    <nombre>Ordenador</nombre>
    <lugar edificio="B">
      <aula>10</aula>
    </lugar>
  </producto>
  <producto codigo="P2">
    <peso unidad='g'>500</peso>
    <nombre>Switch</nombre>
    <lugar edificio="A">
      <aula>6</aula>
    </lugar>
  </producto>
</inventario>
```

11.2 Generación de lista con puntos

Convertir el fichero origen en una lista punteada similar a la que se muestra en la figura:

- Elemento P1
 - Nombre: Ordenador
 - Peso:10kg
- Elemento P2
 - Nombre: Switch
 - Peso:500g

11.3 Recuperación de elementos pesados

Se pide un XSLT que muestre exactamente la misma información del fichero origen pero sin mostrar los elementos cuyo peso sea menor de 7.

Una posible solución sería esta:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <inventario>
    <xsl:for-each select="inventario/producto">
      <xsl:if test="peso < 7">
        <producto>
          <peso>
            <xsl:value-of select="peso"/>
          </peso>
          <nombre>
            <xsl:value-of select="nombre"/>
          </nombre>
          <lugar>
            <xsl:attribute name="edificio">
              <xsl:value-of
                select="lugar/@edificio"/>
            </xsl:attribute>
            <aula>
              <xsl:value-of
                select="lugar/aula"/>
            </aula>
          </lugar>
        </producto>
      </xsl:if>
    </xsl:for-each>
  </inventario>
</xsl:template>
</xsl:stylesheet>
```

11.4 Productos del edificio B

Se pide ahora mostrar en el resultado la misma información del fichero origen pero solo en los casos en que el lugar del producto sea el edificio B

La solución es muy parecida, necesitando solamente modificar la condición.

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <inventario>
    <xsl:for-each select="inventario/producto">
      <xsl:if test="lugar/@edificio='B'">
        <producto>
          <peso>
            <xsl:value-of select="peso"/>
          </peso>
          <nombre>
            <xsl:value-of select="nombre"/>
          </nombre>
          <lugar>
            <xsl:attribute name="edificio">
              <xsl:value-of
                select="lugar/@edificio"/>
            </xsl:attribute>
            <aula>
              <xsl:value-of
                select="lugar/aula"/>
            </aula>
          </lugar>
        </producto>
      </xsl:if>
    </xsl:for-each>
  </inventario>
</xsl:template>
</xsl:stylesheet>
```

11.5 Tabla de localizaciones

Generar una tabla HTML que muestre la información del fichero origen de la manera siguiente:

Resultados

Ordenador	10	B10
Switch	5	A6
Mesa	15	B6

```

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <inventario>
    <xsl:for-each select="inventario/producto">
      <xsl:if test="lugar/@edificio='B'">
        <producto>
          <peso>
            <xsl:value-of select="peso"/>
          </peso>
          <nombre>
            <xsl:value-of select="nombre"/>
          </nombre>
          <lugar>
            <xsl:attribute name="edificio">
              <xsl:value-of
                select="lugar/@edificio"/>
            </xsl:attribute>
            <aula>
              <xsl:value-of
                select="lugar/aula"/>
            </aula>
          </lugar>
        </producto>
      </xsl:if>
    </xsl:for-each>
  </inventario>
</xsl:template>
</xsl:stylesheet>

```

11.6 Tablas con edificios separados

Hacer una plantilla que fabrique una tabla con los datos de los productos del edificio A y otra tabla separada para los productos del edificio B

Edificio A

Switch	5	A6
--------	---	----

Edificio B

Ordenador	10	B10
Mesa	15	B6

Una posible solución sería esta:

```

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

```



```

<xsl:template match="/">
  <html>
    <head><title>Datos por edificio</title></head>
    <body>
      <h1>Edificio A</h1>
      <table border='1'>
        <xsl:for-each select="inventario/producto">
          <xsl:if test="lugar/@edificio='A'">
            <tr>
              <td>
                <xsl:value-of
                  select="nombre"/>
              </td>
              <td>
                <xsl:value-of
                  select="peso"/>
              </td>
              <td>
                <xsl:value-of
                  select="lugar/@edificio"/>
                <xsl:value-of
                  select="lugar/aula"/>
              </td>
            </tr>
          </xsl:if>
        </xsl:for-each>
      </table>
      <h1>Edificio B</h1>
      <table border='1'>
        <xsl:for-each select="inventario/producto">
          <xsl:if test="lugar/@edificio='B'">
            <tr>
              <td>
                <xsl:value-of
                  select="nombre"/>
              </td>
              <td>
                <xsl:value-of
                  select="peso"/>
              </td>
              <td>
                <xsl:value-of
                  select="lugar/@edificio"/>
                <xsl:value-of
                  select="lugar/aula"/>
              </td>
            </tr>
          </xsl:if>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>

```

11.7 Productos del aula 6

Se pide generar un inventario en el que aparezcan solo los nombres de productos que estén en el aula 6.

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <inventario>
    <xsl:for-each select="/inventario/producto">
      <xsl:if test=" lugar/aula= '6' ">
        <nombre>
          <xsl:value-of select="nombre"/>
        </nombre>
      </xsl:if>
    </xsl:for-each>
  </inventario>
</xsl:template>
</xsl:stylesheet>
```

11.8 Productos del edificio B

El siguiente ejercicio es muy parecido al anterior, con la salvedad de que ahora solo nos piden los nombres de los productos ubicados en el edificio B.

```
<inventario>
  <xsl:for-each select="/inventario/producto">
    <xsl:if test=" lugar/@edificio = 'B' ">
      <nombre>
        <xsl:value-of select="nombre"/>
      </nombre>
    </xsl:if>
  </xsl:for-each>
</inventario>
</xsl:template>
</xsl:stylesheet>
```

11.9 Condiciones múltiples: peso ligero y edificio A

Se pide ahora generar un fichero HTML con una tabla pero en la que solo aparezcan los productos cuyo edificio sea el A y además pesen menos de 7kg.

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <head><title>Resultados</title></head>
    <body>
      <xsl:for-each select="inventario/producto">
        <xsl:if test="lugar/@edificio = 'A'">
          <xsl:if test="peso/@unidad = 'g'">
            <xsl:if test="peso < 7000">
              <tr>
```

```

        <td>
          <xsl:value-of
            select="nombre"/>
        </td>
      </tr>
    </xsl:if>
  </xsl:if>
  <xsl:if test="peso/@unidad = 'Kg'">
    <xsl:if test="peso < 7">
      <tr>
        <td>
          <xsl:value-of
            select="nombre"/>
        </td>
      </tr>
    </xsl:if>
  </xsl:if>
</xsl:if>
</xsl:for-each>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

11.10 Ejercicio de examen XSLT

Dado el siguiente fichero XML

```

<catalogo>
  <libro isbn="i1">
    <titulo>Don Quijote</titulo>
    <autores>
      <autor nacimiento="1547">Cervantes</autor>
    </autores>
  </libro>
  <libro isbn="i2">
    <titulo>Antologia</titulo>
    <autores>
      <autor nacimiento="1898">Lorca</autor>
      <autor nacimiento="1910">Miguel Hernandez</autor>
    </autores>
  </libro>
</catalogo>

```

Conseguir lo siguiente:

1. Mostrar en un HTML con lista numerada los títulos de los libros con algún autor nacido despues de 1900.

```

<xsl:stylesheet>
  No se da la solución de este
  ejercicio
</xsl:stylesheet>

```

2. Mostrar en un HTML la lista de los autores ordenada por orden alfabético inverso.

```
<xsl:stylesheet>
  No se da la solución de este
  ejercicio
</xsl:stylesheet>
```

3. Mostrar el nombre de los autores nacidos despues del año 1700.

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <head><title>Resultado</title></head>
    <body>
      <table border='1'>
        <tr>
          <td>Nombre</td>
          <td>Año nacimiento</td>
        </tr>
        <xsl:for-each
select="catalogo/libro/autores/autor">
          <xsl:if test="@nacimiento > 1700">
            <td>
              <xsl:value-of select="."/>
            </td>
            <td>
              <xsl:value-of select="@nacimiento"/>
            </td>
          </xsl:if>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

11.11 Transformación de un XML bancario

Una empresa utiliza el siguiente XML para intercambiar información entre bases de datos de distintos proveedores. Sin embargo han comprado un nuevo sistema que necesita que la información tenga una estructura siguiente. Los dos listados que se ven a continuación ilustran la estructura original y la nueva estructura que deben tener los datos. Crear el XSLT que permita convertir la información original en un formato que pueda entender el nuevo sistema.

```
<!--Estructura original de la información-->
<listado>
  <cuenta>
    <titular dni="5671001D">Ramon Perez</titular>
    <saldoactual moneda="euros">12000</saldoactual>
    <fechacreacion>13-abril-2012</fechacreacion>
  </cuenta>
  <fondo>
    <cuentaasociada>20-A</cuentaasociada>
    <datos>
      <cantidaddepositada>20000</cantidaddepositada>
      <moneda>Euros</moneda>
    </datos>
```

```

</fondo>
<fondo>
  <cuentaasociada>21-DX</cuentaasociada>
  <datos>
    <cantidaddepositada>4800</cantidaddepositada>
    <moneda>Dolares</moneda>
  </datos>
</fondo>
<cuenta>
  <titular dni="39812341C">Carmen Diaz</titular>
  <saldoactual moneda="euros">1900</saldoactual>
  <fechacreacion>15-febrero-2011</fechacreacion>
</cuenta>
</listado>

```

```

<!--Estructura final que debemos conseguir-->
<datos>
  <cuentas>
    <cuenta dnititular="5671001D">
      <creacion>13-abril-2012</creacion>
      <titular>Ramon Perez</titular>
      <saldoactual>12000 euros</saldoactual>

    </cuenta>
    <cuenta dnititular="39812341C">
      <creacion>15-febrero-2011</creacion>
      <titular>Carmen Diaz</titular>
      <saldoactual>1900 euros</saldoactual>

    </cuenta>
  </cuentas>
  <fondos>
    <fondo cuentaasociada="20-A">
      <cantidaddepositada>20000</cantidaddepositada>
      <moneda>Euros</moneda>
    </fondo>
    <fondo cuentaasociada="21-DX">
      <cantidaddepositada>4800</cantidaddepositada>
      <moneda>Dolares</moneda>
    </fondo>
  </fondos>
</datos>

```

11.11.1 Análisis del problema

Es necesario hacer varios cambios:

1. Se ha cambiado el nombre de elemento raíz de listado a datos.
2. Ahora todos los elementos cuenta van dentro de un nuevo elemento cuentas y todos los elementos fondo van dentro de un nuevo elemento fondos.
3. El dni se ha movido del elemento titular al elemento cuenta.
4. La fechacreación se ha movido y se ha renombrado a creacion.
5. El elemento moneda desaparece y su texto se ha puesto al lado de la cantidad que hay en saldoactual.

6. En el elemento `fondo` se ha quitado el elemento `datos`.
7. El elemento `cuentaasociada` ha pasado a ser un atributo.

11.11.2 Solución paso a paso

Empecemos por crear una hoja muy básica, que busque el elemento raíz y devuelva como salida el elemento `datos` (que va a ser la nueva raíz)

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <datos>
  </datos>
</xsl:template>
</xsl:stylesheet>
```

Si probamos dicho XSLT aplicándolo al XML original obtendremos esto:

```
<datos/>
```

No pasa nada porque se obtenga el elemento raíz vacío, el programa de transformación lo hace para ahorrar tiempo y bytes.

Una vez que hemos cambiado el elemento raíz tenemos que generar dos elementos más que agrupen los elementos `cuenta` y los elementos `fondo`. Para ello, basta con escribirlos como muestra la siguiente hoja de estilo.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <datos>
    <cuentas></cuentas>
    <fondos></fondos>
  </datos>
</xsl:template>
</xsl:stylesheet>
```

Ahora tenemos que ir buscando todos los elementos `cuenta` y meterlos dentro de `cuentas`. Después resolveremos el problema de los fondos. Para recorrer elementos necesitamos un bucle `for-each`. Como la plantilla ya nos ha situado en la raíz necesitaremos que el bucle no vaya dando cada uno de los elementos `listado/cuenta`. Es decir, le pedimos al bucle que se meta en el elemento hijo `listado` y nos vaya dando cada uno de los elementos `cuenta` que hay dentro. Un posible bucle sería este:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <datos>
    <cuentas>
      <xsl:for-each select="listado/cuenta">
        <cuenta></cuenta>
      </xsl:for-each>
    </cuentas>
    <fondos></fondos>
  </datos>
</xsl:template>
</xsl:stylesheet>
```

Que al pasárselo a nuestros datos nos da esto:

```
<datos>
  <cuentas>
    <cuenta/>
    <cuenta/>
  </cuentas>
  <fondos/>
</datos>
```

Como puede verse, la plantilla genera dos elementos cuenta, uno por cada cuenta que nos da el bucle. Obsérvese que podríamos haber hecho esto para tener un nombre de elemento distinto, **y este es el «truco» para poder cambiar de nombre un elemento** :

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <datos>
    <cuentas>
      <xsl:for-each select="listado/cuenta">
        <otroelemento></otroelemento>
      </xsl:for-each>
    </cuentas>
    <fondos></fondos>
  </datos>
</xsl:template>
</xsl:stylesheet>
```

Sigamos con el problema original: ya hemos creado un elemento cuentas que lleva dentro un elemento cuenta para cada una de las cuentas originales. Ahora en dicho elemento cuenta vamos a meter dentro un atributo llamado dnititular usando la etiqueta xsl:attribute que debe ir **dentro del elemento al que le queramos poner el atributo y además al principio**. Si queremos varios atributos no pasa nada podemos ponerlos todos dentro del elemento pero recordando ponerlos al principio.

Así, el código siguiente nos fabrica el atributo.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <datos>
    <cuentas>
      <xsl:for-each select="listado/cuenta">
        <cuenta>
          <!--Esto añade el atributo dnititular a cuenta-->
          <xsl:attribute name="dnititular">10</xsl:attribute>
        </cuenta>
      </xsl:for-each>
    </cuentas>
    <fondos></fondos>
  </datos>
</xsl:template>
</xsl:stylesheet>
```

Pero hay un problema, todas las cuentas tienen el dnititular a 10. Necesitamos la etiqueta value-of que nos permite **extraer el contenido de un elemento o atributo**, en este caso queremos extraer el valor del atributo dni que está dentro del elemento titular. Esto se hace con titular/@dni que significa «extraer el atributo dni que debe estar dentro del elemento titular».

Así, el código siguiente:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
```

```

<datos>
  <cuentas>
    <xsl:for-each select="listado/cuenta">
      <cuenta>
        <xsl:attribute name="dnititular">
          <xsl:value-of select="titular/@dni"/>
        </xsl:attribute>
      </cuenta>
    </xsl:for-each>
  </cuentas>
  <fondos></fondos>
</datos>
</xsl:template>
</xsl:stylesheet>

```

Nos devuelve como resultado:

```

<datos>
  <cuentas>
    <cuenta dnititular="5671001D"/>
    <cuenta dnititular="39812341C"/>
  </cuentas>
  <fondos/>
</datos>

```

El paso siguiente va a ser crear el elemento titular que también se llama titular en el archivo original. Para ello lo metemos dentro de cuenta y permitiendo que la creación del atributo dnititular se quede al principio.

El código XSLT es este

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <datos>
    <cuentas>
      <xsl:for-each select="listado/cuenta">
        <cuenta>
          <xsl:attribute name="dnititular">
            <xsl:value-of select="titular/@dni"/>
          </xsl:attribute>
          <!--Creamos el elemento titular y metemos
dentro del valor original del titular-->
          <titular>
            <xsl:value-of select="titular"/>
          </titular>
        </cuenta>
      </xsl:for-each>
    </cuentas>
    <fondos></fondos>
  </datos>
</xsl:template>
</xsl:stylesheet>

```

Que genera el siguiente resultado:

```

<datos>
  <cuentas>
    <cuenta dnititular="5671001D">
      <titular>Ramon Perez</titular>
    </cuenta>
  </cuentas>
  <fondos/>
</datos>

```



```

</cuenta>
<cuenta dnititular="39812341C">
  <titular>Carmen Diaz</titular>
</cuenta>
</cuentas>
<fondos/>
</datos>

```

Ahora vamos a crear el elemento `saldoactual`. Dentro de ese elemento debemos escribir el texto que ya tuviese el `saldoactual` antiguo y escribiendo al lado el atributo `moneda`.

El código necesario es este

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <datos>
    <cuentas>
      <xsl:for-each select="listado/cuenta">
        <cuenta>
          <xsl:attribute name="dnititular">
            <xsl:value-of select="titular/@dni"/>
          </xsl:attribute>
          <!--Creamos el elemento titular y metemos
dentro del valor original del titular-->
          <titular>
            <xsl:value-of select="titular"/>
          </titular>
          <!--Creamos el saldo actual-->
          <saldoactual>
            <!--Y metemos dentro la cantidad que tuviese
el fichero original...-->
            <xsl:value-of select="saldoactual"/>
            <!--Y extraemos la moneda...-->
            <xsl:value-of select="saldoactual/@moneda"/>
          </saldoactual>
        </cuenta>
      </xsl:for-each>
    </cuentas>
    <fondos></fondos>
  </datos>
</xsl:template>
</xsl:stylesheet>

```

Y el resultado es este:

```

<datos>
  <cuentas>
    <cuenta dnititular="5671001D">
      <titular>Ramon Perez</titular>
      <saldoactual>12000euros</saldoactual>
    </cuenta>
    <cuenta dnititular="39812341C">
      <titular>Carmen Diaz</titular>
      <saldoactual>1900euros</saldoactual>
    </cuenta>
  </cuentas>
  <fondos/>
</datos>

```

Por último, añadamos la fecha de creación. En el fichero original se llama `fechacreación` y en el fichero final se llama `creación` y además va como primer elemento. El XSLT sería este:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <datos>
    <cuentas>
      <xsl:for-each select="listado/cuenta">
        <cuenta>
          <xsl:attribute name="dnititular">
            <xsl:value-of select="titular/@dni"/>
          </xsl:attribute>
          <!--Creamos el elemento "creación" que en
              realidad contiene el texto del elemento
              "fechacreación" original-->
          <creacion>
            <xsl:value-of select="fechacreacion"/>
          </creacion>
          <!--Creamos el elemento titular y metemos
              dentro del valor original del titular-->
          <titular>
            <xsl:value-of select="titular"/>
          </titular>
          <!--Creamos el saldo actual-->
          <saldoactual>
            <!--Y metemos dentro la cantidad que tuviese
              el fichero original...-->
            <xsl:value-of select="saldoactual"/>
            <!--Y extraemos la moneda...-->
            <xsl:value-of select="saldoactual/@moneda"/>
          </saldoactual>
        </cuenta>
      </xsl:for-each>
    </cuentas>
    <fondos></fondos>
  </datos>
</xsl:template>
</xsl:stylesheet>
```

Que genera el resultado siguiente:

```
<datos>
  <cuentas>
    <cuenta dnititular="5671001D">
      <creacion>13-abril-2012</creacion>
      <titular>Ramon Perez</titular>
      <saldoactual>12000euros</saldoactual>
    </cuenta>
    <cuenta dnititular="39812341C">
      <creacion>15-febrero-2011</creacion>
      <titular>Carmen Diaz</titular>
      <saldoactual>1900euros</saldoactual>
    </cuenta>
  </cuentas>
  <fondos/>
</datos>
```

Con esto, la parte de las cuentas ya está hecha. Ahora queda lo siguiente

1. Hacer un elemento `fondos` con un bucle que vaya generando elementos `fondo`.
2. Poner en el fondo el atributo `cuentaasociada`.
3. Crear el elemento `cantidaddepositada`.
4. Crear el elemento `moneda`.

Vamos con el paso 1 «*crear el elemento fondos*»

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <datos>
    <cuentas>
      <xsl:for-each select="listado/cuenta">
        <cuenta>
          <xsl:attribute name="dnititular">
            <xsl:value-of select="titular/@dni"/>
          </xsl:attribute>
          <!--Creamos el elemento "creación" que en
          realidad contiene el texto del elemento
          "fechacreación" original-->
          <creacion>
            <xsl:value-of select="fechacreacion"/>
          </creacion>
          <!--Creamos el elemento titular y metemos
          dentro del valor original del titular-->
          <titular>
            <xsl:value-of select="titular"/>
          </titular>
          <!--Creamos el saldo actual-->
          <saldoactual>
            <!--Y metemos dentro la cantidad que tuviese
            el fichero original...-->
            <xsl:value-of select="saldoactual"/>
            <!--Y extraemos la moneda...-->
            <xsl:value-of select="saldoactual/@moneda"/>
          </saldoactual>
        </cuenta>
      </xsl:for-each>
    </cuentas>
    <fondos>
      <xsl:for-each select="listado/fondo">
        <!--Paso 1: crear un fondo por cada fondo original-->
        <fondo>

          </fondo>
        </xsl:for-each>
      </fondos>
    </datos>
  </xsl:template>
</xsl:stylesheet>
```

Ahora el paso 2: *poner el atributo «cuentaasociada»*. El XSLT sería así:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <datos>
    <cuentas>
      <xsl:for-each select="listado/cuenta">
```

```

        <creacion>
            <xsl:value-of select="fechacreacion"/>
        </creacion>
        <!--Creamos el elemento titular y metemos
dentro del valor original del titular-->
        <titular>
            <xsl:value-of select="titular"/>
        </titular>
        <!--Creamos el saldo actual-->
        <saldoactual>
            <!--Y metemos dentro la cantidad que tuviese
el fichero original...-->
            <xsl:value-of select="saldoactual"/>
            <!--Y extraemos la moneda...-->
            <xsl:value-of select="saldoactual/@moneda"/>
        </saldoactual>
    </cuenta>
</xsl:for-each>
</cuentas>
<fondos>
    <xsl:for-each select="listado/fondo">
        <!--Paso 1: crear un fondo por cada fondo original-->
        <fondo>
            <!--Paso 2, crear el atributo cuentaasociada-->
            <xsl:attribute name="cuentaasociada">
                <xsl:value-of select="cuentaasociada"/>
            </xsl:attribute>

            </fondo>
        </xsl:for-each>
    </fondos>
</datos>
</xsl:template>
</xsl:stylesheet>

```

Con el XSLT siguiente conseguimos el paso 3: «*crear el elemento cantidaddepositada*»

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <datos>
        <cuentas>
            <xsl:for-each select="listado/cuenta">
                <cuenta>
                    <xsl:attribute name="dnititular">
                        <xsl:value-of select="titular/@dni"/>
                    </xsl:attribute>
                    <!--Creamos el elemento "creación" que en
realidad contiene el texto del elemento
"fechacreación" original-->
                    <creacion>
                        <xsl:value-of select="fechacreacion"/>
                    </creacion>
                </cuenta>
            </xsl:for-each>
        </cuentas>
    </datos>
</template>
</stylesheet>

```

```

        </creacion>
        <!--Creamos el elemento titular y metemos
        dentro del valor original del titular-->
        <titular>
            <xsl:value-of select="titular"/>
        </titular>
        <!--Creamos el saldo actual-->
        <saldoactual>
            <!--Y metemos dentro la cantidad que tuviese
            el fichero original...-->
            <xsl:value-of select="saldoactual"/>
            <!--Y extraemos la moneda...-->
            <xsl:value-of select="saldoactual/@moneda"/>
        </saldoactual>
    </cuenta>
</xsl:for-each>
</cuentas>
<fondos>
    <xsl:for-each select="listado/fondo">
        <!--Paso 1: crear un fondo por cada fondo original-->
        <fondo>
            <!--Paso 2, crear el atributo cuentaasociada-->
            <xsl:attribute name="cuentaasociada">
                <xsl:value-of select="cuentaasociada"/>
            </xsl:attribute>
            <!--Paso 3, crear el elemento cantidaddepositada-->
            <cantidaddepositada>
                <xsl:value-of select="datos/cantidaddepositada"/>
            </cantidaddepositada>
        </fondo>
    </xsl:for-each>
</fondos>
</datos>
</xsl:template>
</xsl:stylesheet>

```

Y por último el paso 4 «crear el elemento moneda». El XSLT sería algo así:

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <datos>
        <cuentas>
            <xsl:for-each select="listado/cuenta">
                <cuenta>
                    <xsl:attribute name="dnititular">
                        <xsl:value-of select="titular/@dni"/>
                    </xsl:attribute>
                    <!--Creamos el elemento "creación" que en
                    realidad contiene el texto del elemento
                    "fechacreación" original-->
                    <creacion>
                        <xsl:value-of select="fechacreacion"/>
                    </creacion>
                    <!--Creamos el elemento titular y metemos
                    dentro del valor original del titular-->
                    <titular>
                        <xsl:value-of select="titular"/>
                    </titular>
                </cuenta>
            </xsl:for-each>
        </cuentas>
    </datos>
</template>
</stylesheet>

```

```

        <!--Creamos el saldo actual-->
        <saldoactual>
            <!--Y metemos dentro la cantidad que tuviese
            el fichero original...-->
            <xsl:value-of select="saldoactual"/>
            <!--Y extraemos la moneda...-->
            <xsl:value-of select="saldoactual/@moneda"/>
        </saldoactual>
    </cuenta>
</xsl:for-each>
</cuentas>
<fondos>
    <xsl:for-each select="listado/fondo">
        <!--Paso 1: crear un fondo por cada fondo original-->
        <fondo>
            <!--Paso 2, crear el atributo cuentaasociada-->
            <xsl:attribute name="cuentaasociada">
                <xsl:value-of select="cuentaasociada"/>
            </xsl:attribute>
            <!--Paso 3, crear el elemento cantidaddepositada-->
            <cantidaddepositada>
                <xsl:value-of select="datos/cantidaddepositada"/>
            </cantidaddepositada>
            <!--Paso 4: Crear el elemento moneda-->
            <moneda>
                <xsl:value-of select="datos/moneda"/>
            </moneda>
        </fondo>
    </xsl:for-each>
</fondos>
</datos>
</xsl:template>
</xsl:stylesheet>

```

Si probamos el XSLT anterior veremos que efectivamente conseguimos transformar el fichero original en el fichero resultado que nos piden, que es el siguiente:

```

<datos>
  <cuentas>
    <cuenta dnititular="5671001D">
      <creacion>13-abril-2012</creacion>
      <titular>Ramon Perez</titular>
      <saldoactual>12000euros</saldoactual>
    </cuenta>
    <cuenta dnititular="39812341C">
      <creacion>15-febrero-2011</creacion>
      <titular>Carmen Diaz</titular>
      <saldoactual>1900euros</saldoactual>
    </cuenta>
  </cuentas>
  <fondos>
    <fondo cuentaasociada="20-A">
      <cantidaddepositada>20000</cantidaddepositada>
      <moneda>Euros</moneda>
    </fondo>
    <fondo cuentaasociada="21-DX">
      <cantidaddepositada>4800</cantidaddepositada>
      <moneda>Dolares</moneda>
    </fondo>
  </fondos>
</datos>

```

```
</fondo>  
</fondos>  
</datos>
```