TAREA UT5

MODELO DE OBJETOS DEL DOCUMENTO

La tarea consiste en la implementación de varios scripts en lenguaje Javascript. Habrá que tener en cuenta las siguientes recomendaciones generales:

- Se recomienda estructurar el código lo más posible facilitando su mantenimiento y reutilización.
- Para tratar los eventos se usará el modelo W3C.
- Está prohibido el uso de variables globales salvo que se indique explícitamente lo contrario.
- En los ejercicios se proporciona código CSS y/o HTML. Está prohibida la modificación de los mismos, a excepción del fichero HTML para agregar las llamadas a los archivos JS correspondientes.

1. [1,5 puntos] Vamos a implementar un visor de imágenes que permite iterar por una serie de viñetas.

Se proporciona la página HTML y la hoja de estilos que hay que usar obligatoriamente. Ninguna puede ser modificada, excepto en lo relativo a las llamadas a scripts Javascript. Las imágenes se cargarán como imagen de fondo de la capa con id `imagen`

Se definirá en un archivo Javascript una clase **`Galeria`** con las siguientes características:

- Propiedades que almacenen la siguiente información:
  - Array que contiene los nombres de las imágenes que se van a mostrar.
  - Cantidad de imágenes que va a manejar la galería.
  - Índice de la imagen que se está mostrando actualmente (cursor)
- Métodos:
  - El constructor de la clase recibe como único parámetro la cantidad de imágenes que va a manejar la galería. Dado que todas las imágenes tienen el mismo nombre (sólo se

diferencian en el número), el constructor generará de manera automática los nombres de todas las imágenes que se van a manejar.
    - Un método que establezca el cursor en una posición aleatoria del array, devolviendo el nombre de la imagen que hay en la misma.
    - Un método que establezca el cursor en la última posición del array, devolviendo el nombre de la imagen que hay en la misma.
    - Un método que establezca el cursor en la primera posición del array, devolviendo el nombre de la imagen que hay en la misma.
    - Un método que establezca el cursor en posición anterior a la actual, devolviendo el nombre de la imagen que hay en la misma.
    - Un método que establezca el cursor en posición siguiente a la actual, devolviendo el nombre de la imagen que hay en la misma.

[NOTA1] Al llamar a las imágenes de manera igual, no sería estrictamente necesario el uso del array, pero vamos a mantenerlo, dando a nuestra aplicación la posibilidad de manejar imágenes con distintos nombres.

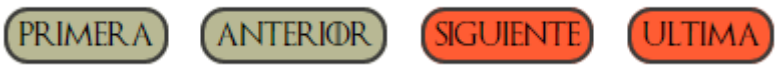
[NOTA2] Para automatizar completamente el proceso, lo ideal sería que accedieramos a la carpeta de imágenes y extrajéramos de la misma el nombre de las imágenes y cantidad. Sin embargo, esto es algo que no podemos hacer desde Javascript, necesitamos el uso de le lenguajes de servidor como PHP.

[NOTA2] Nótese que cuando referenciamos una carpeta desde la hoja de estilos CSS la ruta es relativa a la ubicación de la hoja de estilos, pero cuando accedemos a la propiedad style desde Javascript realmente estamos modificando el estilo in-line (perteneciente al HTML), luego la ruta de acceso será relativa a la ubicación de la página HTML.

Por otra parte, habrá que manejar los eventos que permitan que se muestre la galería, usando los métodos de la misma para movernos por las imágenes de la misma. Hay que tener en cuenta que:

- Inicialmente se cargará una imagen al azar.
- Si nos encontramos en la primera imagen, los botones *Primera* y *Anterior* se deshabilitarán:

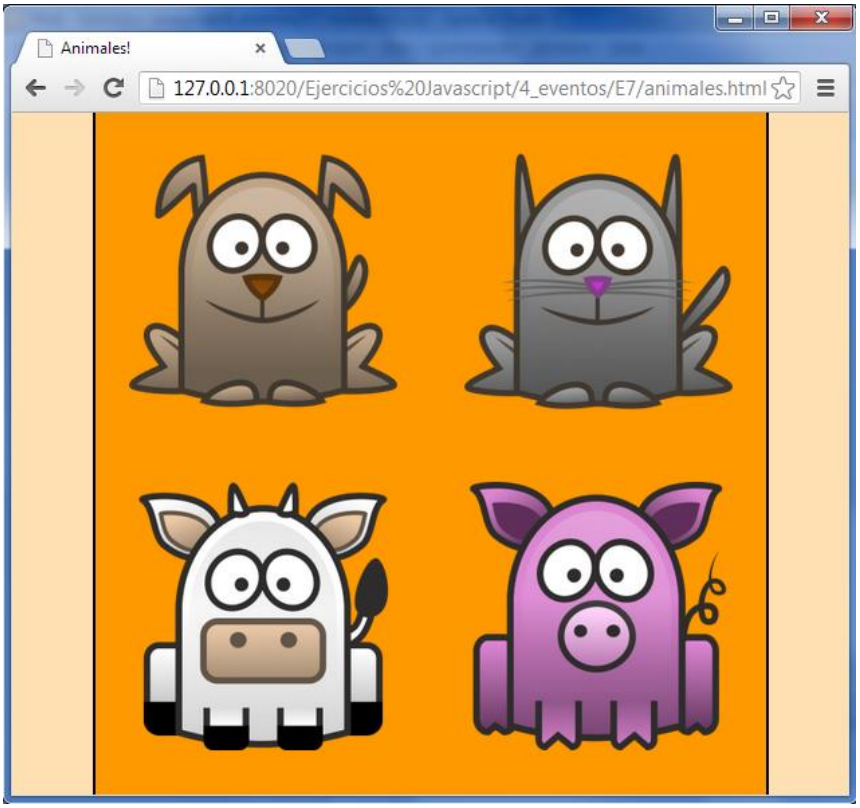
  Este comportamiento dejará de producirse cuando cambiamos de imagen.
- Si nos encontramos en la última imagen, los botones *Siguiente* y *Última* se deshabilitarán:



  Este comportamiento dejará de producirse cuando cambiamos de imagen.

Para los dos últimos puntos habrá que utilizar los estilos `.habilitado` y `.deshabilitado` que se adjuntan en la hoja de estilos.

2. [1,5 puntos] Vamos a realizar un pequeño juego. El juego debería mostrar varios iconos de animales. Cuando pulsamos en uno de los iconos, debería de reproducirse el sonido correspondiente al animal correspondiente, así como ejecutarse un movimiento aleatorio, que puede ser un giro de 360 grados, un cambio de tamaño al doble o un cambio de tamaño a la mitad.



Utiliza la hoja de estilos y la página HTML proporcionada, donde no vas a realizar ningún cambio a excepción de las referencias a los archivos Javascript.

Además se incluyen varios archivos de imagen y de sonido, cuyo nombre coincide con el del animal para el cual va a usarse.

- Desde Javascript vas a generar los siguientes elementos:
    - Un array donde están definidos los nombres de los animales (inicialmente sólo tenemos 4).
    - Un div con id animales que funcionará como contenedor de los animales de la página.
    - Una serie de divs con clase animal que se generarán dinámicamente y se añadirán al contenedor. Cada div tendrá **`clase animal`** (lo que le da las características CSS comunes a todos los animales), el **`id`** correspondiente al animal que estamos generando (lo que permitirá acceder a él de manera individualizada) y la imagen de fondo con el nombre del animal.
    - Una serie de etiquetas `<audio>` que nos va a permitir reproducir sonidos. Por otra parte se emitirá un sonido correspondiente al animal pulsado. Para informar a la etiqueta `audio` acerca de qué archivo tiene que reproducir, es necesario añadir a la misma un nodo nuevo (etiqueta `source`) cuyo atributo `src` indicará la ruta de acceso al archivo que se quiere reproducir.

- Al pulsar sobre cada animal:
    - Por una parte se generará un efecto de movimiento. La hoja de estilos proporciona 3 efectos (`rota`, `grande` y `pequenio`) definidos como clases CSS. La manera de agregar un efecto a un elemento es añadir a la clase actual la clase con el efecto deseado. Por ejemplo, si un div tiene definida la clase `animal` sólo se le aplicarán las reglas CSS correspondientes a `.animal`, pero si tiene definida la clase "animal rota" se le aplicarán las reglas correspondientes a `.animal` y `.rota` (esta última produce el giro).

      Posteriormente accederemos al nodo DOM correspondiente a la etiqueta de audio, invocando a su método `play()`.

A continuación modifica el programa para:

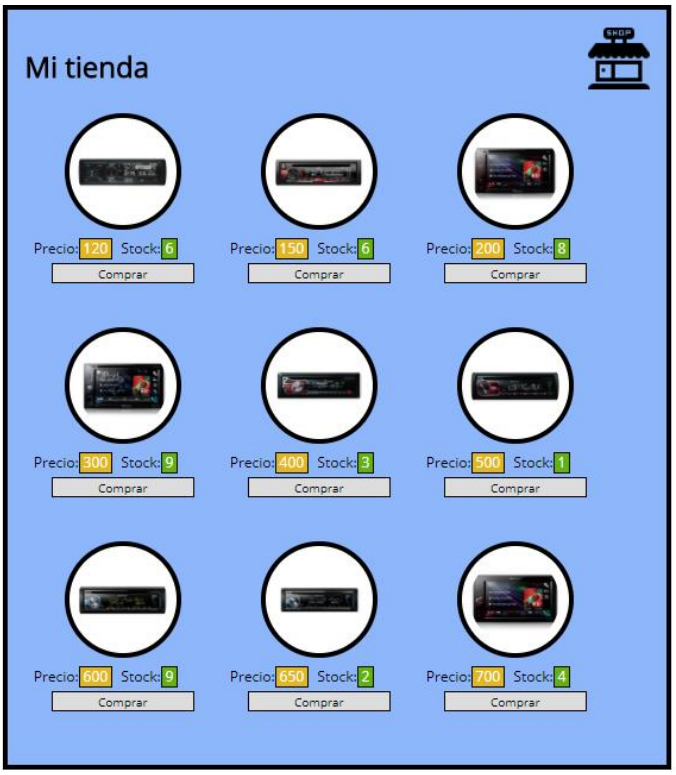
- Añadir 4 nuevos animales. Puedes descargar los iconos desde aquí y los sonidos desde aquí.
- Añadir 2 nuevos efectos.

Utiliza las funciones y/o objetos que creas oportunas. Se valorará positivamente el uso de programación estructurada y orientada a objetos.

Información adicional:

- Etiqueta audio de HTML5.
- API del DOM para elementos de audio.
- Transformaciones, animaciones y transiciones en CSS3

3. [3,5 puntos] En este ejercicio vas a implementar un carrito de la compra que funciona con Javascript. Se proporciona el archivo HTML (no se puede modificar) y la hoja de estilos asociada al ejercicio, que se recomienda reutilizar en la medida de lo posible:



Para ello vas a implementar las siguientes clases;

- La clase **`Producto`** se encarga de almacenar información acerca de un producto que hemos comprado

Clase **`Producto`**			[Definida mediante objetos literales]
Propiedades			
Nombre	Tipo	Visibilidad	Descripción
`id`	`cadena`	Pública	Id del producto comprado
`precio`	`real`	Pública	Precio del producto comprado
`imagen`	`cadena`	Pública	Nombre de la imagen que se usa para visualizar el producto

Se elimina la propiedad cantidad del producto

- La clase **Carrito** se encarga de almacenar los productos que vamos adquiriendo:

Clase **Carrito**	[Definida mediante prototipos]		
Propiedades			
Nombre	**Tipo**	**Visibilidad**	**Descripción**
compras	[]	Privada	Array bidimensional que contiene los productos y su stock (ver estructura de la tabla más adelante)
Métodos			
Nombre	**Parámetros**	**Devuelve**	**Descripción**
Carrito()	-	-	Constructor sin parámetros. Inicializa el array.
numeroCompras()		entero	Devuelve el número de productos que contiene el carrito
getCompraFromId(id)	id:cadena	Producto	Devuelve el producto con determinado **id** [**null** si no existe].
getCompraFromPosition(i)	i:entero	Producto	Devuelve el producto con determinado **id** [**null** si no existe].
insertaProducto(p)	p:producto	booleano	Añade un producto a la compra. Si el producto no existía previamente en el carrito, se inserta con el valor de la cantidad 1. En este caso se devolverá el valor **true** que indica que el producto no existía. Si el producto ya existía en el carrito, se incrementa la cantidad en una unidad y se devuelve **false**. En este caso se devolverá el valor false que indica que el producto ya existía en el carrito.
eliminaProductoFromId(id)	id:cadena	booleano	Elimina el producto con determinado id, desplazando todos los productos una posición. Devuelve verdadero si la eliminación se ha realizado correctamente.
getTotalCompra()	-	entero	Devuelve el coste total de las compras del carrito.
getCantidadFromId(id)	id:cadena	entero	Devuelve la cantidad correspondiente al producto con id id, o -1 si no existe el producto con dicho id.
incrementaCantidad(id)	id:cadena	entero	Incrementa la cantidad correspondiente al producto con id id, o -1 si no existe el producto con dicho id. Devuelve la cantidad incrementada.
decrementaCantidad(id)	id:cadena	entero	Decrementa la cantidad correspondiente al producto con id id, o -1 si no existe el producto con dicho id. Devuelve la cantidad decrementada.

En cuanto a la propiedad **compras** de Tienda, ésta debería consistir en un array bidimensional de 2 columnas. La primera columna contiene objetos de tipo Producto, mientras que la segunda columna contiene la cantidad de productos en el carrito (número entero

	0	1
0	{Producto}	1
1	{Producto}	1
2	{Producto}	2

La clase se implementará en el fichero **carrito.js**. En el fichero **carrito_prueba.js** se comprobará que la clase funciona correctamente imprimiendo información acerca del carrito por consola.

- La clase **Tienda** se encarga de almacenar los productos que muestra la tienda:

Clase **Tienda**			[Definida mediante prototipos]
Propiedades			
Nombre	**Tipo**	**Visibilidad**	**Descripción**
productos	[][]	Privada	Array bidimensional que contiene los productos y su stock (ver estructura de la tabla más adelante)
Métodos			
Nombre	**Parámetros**	**Devuelve**	**Descripción**
Tienda()	-	-	Constructor sin parámetros. Inicializa el array.
cargaProductos()	-	-	Carga los productos de la tienda a partir de la información contenida en el array de **productos.js**.
numeroProductos()	-	entero	Devuelve el número de productos que contiene la tienda
getProducto(id)	id:cadena	Producto	Devuelve el producto con determinado **id** [**null** si no existe].
getProductoFromPosition(pos)	pos:entero	Producto	Devuelve el producto que está en la posición **pos** [**null** si no existe].
getStock(id)	id:cadena	entero	Devuelve el stock del producto con determinado id. Si el id no existe, devuelve -1
decrementaStock(id)	id:cadena	entero	Si el stock es mayor que 0, decrementa el stock del producto con determinado id. Devuelve el stock actualizado. Si el id no existe, devuelve -1

En cuanto a la propiedad **productos** de Tienda, ésta debería consistir en un array bidimensional de 2 columnas, cuya información se obtendrá a partir del array del archivo productos.js . La primera columna contiene objetos de tipo Producto, mientras que la segunda columna contiene el stock del producto correspondiente (es por tanto un entero).

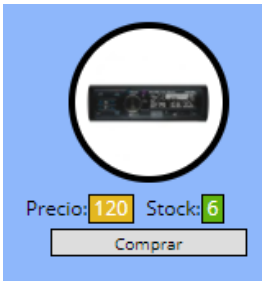
	0	1
0	{Producto}	4
1	{Producto}	6
2	{Producto}	2
3	{Producto}	8
4	{Producto}	0

Nótese que los objetos almacenados en el array del archivo productos.js no son de tipo Producto. Nótese además que habrá que implementar un mecanismo que permita mantener los ids de producto de manera unívoca.

La clase se implementará en el fichero **tienda.js**. En el fichero **tienda_prueba.js** se comprobará que la clase funciona correctamente cargando los productos correspondientes e imprimiendo información acerca de los mismos por consola.

Además se creará un fichero Javascript llamado **`ejercicio3.js`** que tenga las siguientes construcciones:

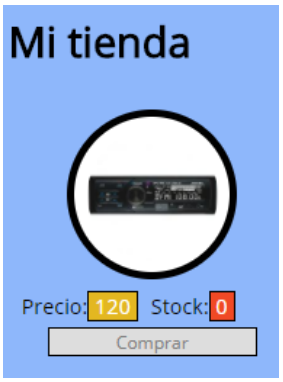
- **`tienda`**: Variable global que consiste en un objeto de tipo `Tienda`.
- **`carrito`**: Variable global que consiste en un objeto de tipo `Carrito`.
- Función **`creaCarrito()`**
  - Se llamará al cargar la página:
    - Instancia el objeto **`carrito`**.
    - Crea una capa destinada a almacenar el carrito de la compra.
- **`creaTienda()`**.
  - Se llamará al cargar la página.
    - Instancia el objeto **`tienda`**.
    - Muestra los distintos productos que contiene la tienda. Para cada producto se mostrará:
      - La imagen asociada al producto.
      - El precio del mismo.
      - Su stock.
      - Un botón que nos permitirá añadir el producto al carrito.



NO utilices una tabla para maquetar los productos, utiliza capas. Es recomendable mirar previamente el estilo CSS para comprobar la clase que habrá que aplicar a cada elemento.

Al pulsar el botón asociado a un producto se realizarán las siguientes acciones:

- Se actualizará el stock del producto en la tienda (tanto el objeto como en la vista asociada). En caso de que el stock pase a ser 0, se mostrará la cantidad en un formato diferenciado y se deshabilitará el botón de comprar.



- Se añadirá el producto correspondiente al objeto **`carrito`**, actualizando la capa que muestra el contenido del carrito. Para cada producto del carrito se mostrará:
  - La imagen con el producto.
  - El precio del mismo.
  - Un botón destinado a incrementar la cantidad del producto. Si dicho producto no tiene stock, el botón estará deshabilitado.

- Un botón destinado a decrementar la cantidad del producto. Si la cantidad pasa a ser 0, el producto se eliminará de la cesta de la compra previa confirmación por parte del usuario (puede usarse un alert() para preguntar al usuario si está seguro de querer eliminar el producto del carrito).

En todo momento se mostrará el coste total del carrito. Además, los cambios en la cantidad de cada producto del carrito deberán reflejarse en la información que muestra la tienda.

Por otra parte, si el usuario pulsa *Comprar* sobre un producto que ya está en el carrito, se actualizará la cantidad correspondiente al mismo.

Consideraciones generales:

- Tendrás que pensar un mecanismo para averiguar qué producto estamos manejando al comprar/eliminar. Para ello, se recomienda usar un atributo personalizado que contenga el id del producto usando datasets.

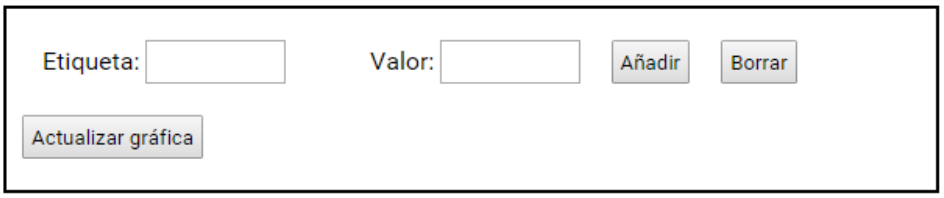
5. [1,5 puntos] Modifica el programa anterior para que la funcionalidad del carrito se realice usando *Drag and Drop*. Esto implica que:
   - Se eliminarán los botones de *Comprar* asociados a cada producto.
   - Al arrastrar un producto desde *Mi Tienda* al carro se añadirá al mismo.
   - Se mantendrán los botones + y – del carrito.
   - Al arrastrar un producto fuera del carro se eliminará del mismo (previa confirmación por parte del usuario.

6. [2 puntos] Vamos a implementar un formulario que nos va a permitir almacenar y visualizar datos en una gráfica de barras.

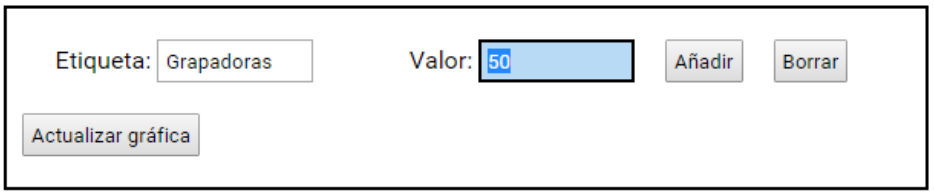
   Se implementarán las siguientes clases:
   - Clase **`Dato`** (Objetos literales): Contiene la lógica necesaria para almacenar un dato de una gráfica. Cada dato está compuesto por una etiqueta y un valor. La propia etiqueta funcionará como identificador. (es decir, supondremos que en nuestra aplicación no puede haber dos etiquetas iguales, aunque para simplificar la lógica de funcionamiento no vamos a realizar ningún tipo de validación).
   - Clase **`Grafica`** (Prototipo): Contiene la lógica necesaria para gestionar un listado de datos (objetos de tipo **`Dato`**) de una gráfica. Contiene métodos para la gestión del listado (añadir, borrar, buscar, número de elementos...).

Nuestra aplicación tendrá que instanciar un objeto de tipo **`Grafica`** para la gestión de los datos que vamos añadiendo.

Además se proporciona un formulario HTML con el siguiente aspecto:



El formulario permite introducir nuevos pares etiqueta/valor:



Al pulsar el botón *Añadir* se añadirá al objeto que gestiona la gráfica. Una vez, se borrarán las cajas de texto correspondientes y se podrán añadir nuevos datos a la gráfica. Además, se informará al usuario informando de que el dato ha sido añadido con un mensaje en la capa de mensajes (deberá tener el formato adecuado). Se comprobará la existencia de datos y validez de tipos (valor entero), y la existencia de etiquetas repetidas. En caso de error, se mostrarán los mismos en la capa de mensajes con el formato adecuado.

Al pulsar el botón *Actualizar gráfica* se mostrarán en la capa con id **`grafica`** (que inicialmente está oculta) todos los datos que contiene la gráfica actualmente.

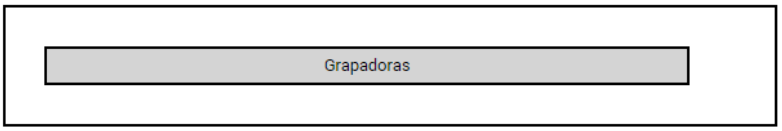
Cada dato de la gráfica se mostrará usando dos capas:
- Una capa con clase **`contenedor`** (externa)
- Otra capa con clase **`progreso`**, que está dentro de la anterior y que simboliza una barra de progreso, siendo el máximo valor de la barra (el 100%) el máximo valor de un dato introducido hasta el momento.

  A su vez la capa contendrá de progreso contendrá la etiqueta introducida.

Por ejemplo, imaginemos que introducimos el siguiente dado:

- Etiqueta **Grapadoras,** valor **50**

Si visualizamos la gráfica deberíamos El estado de la gráfica será el siguiente:

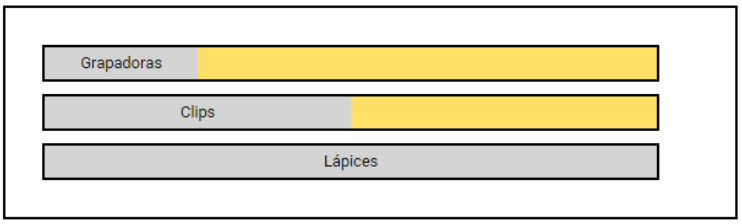


La capa interna ocupa el 100% de la capa contenedor, dado que tenemos un único dato hasta el momento.

Si a continuación introducimos dos nuevos datos:

- Etiqueta Clips, valor 100.
- Etiqueta Lápices, valor 200.

Y a continuación pulsamos el botón *Actualizar gráfica* la gráfica que se mostrará será la siguiente:



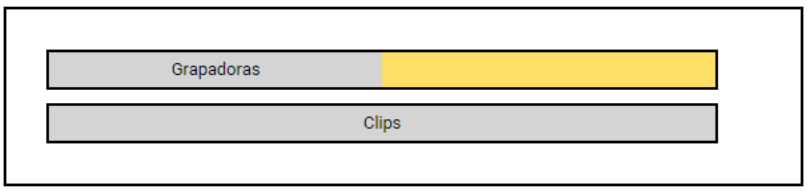
Se tomará el máximo valor introducido como el 100% (por eso la barra de progreso de *Lápices*, que tiene el mayor valor, ocupa el total), y para el resto de valores se calculará el porcentaje respecto al anterior, de modo que la capa interna ocupe dicho porcentaje respecto del ancho de la capa contenedora. Por ejemplo, la cantidad de clips (100) supone un 50% respecto de la cantidad de lápices (200), y la cantidad de grapadoras (50) supone un 25%.

Tendremos que establecer un método para establecer/determinar la anchura de un elemento. Recuérdese que desde el DOM no podemos acceder a la hoja de estilos CSS

En cualquier momento podemos borrar un elemento de la gráfica. Para ello tendremos que introducir el nombre de la etiqueta en la caja correspondiente y pulsar el botón *Borrar:*

Nótese que en este caso la caja de texto *Valor* no se está utilizando. **Es necesario** validar que la etiqueta introducida exista, mostrando el mensaje de error correspondiente en caso de que no sea así.

Una vez borrado el elemento anterior, al pulsar el botón *Actualizar gráfica* se mostrarán de nuevo los datos:



Nótese que han variado los valores de la barra de progreso, pues ahora el máximo valor es el correspondiente a Clips (valor 100), y la cantidad de grapadoras (50) supone ahora un 50% respecto del valor de Clips. Habrá que tener en cuenta la posibilidad de que se borre la última etiqueta para ocultar de nuevo la gráfica en este caso.