DWEC - Examen Práctico RA4

Contenido

Instrucciones

Lee atentamente las siguientes indicaciones y pregunta cualquier duda que tengas.

Te **recomiendo encarecidamente** que leas el ejercicio completo antes de hacer los distintos apartados indicados, ya que te dará ideas de cómo plantear la solución.

Se te ha facilitado una estructura de carpetas con todos los documentos, salvo los de las clases de los ejercicios 1 y 2.

Los documentos **HTML** solo podrás modificarlos para indicar tu nombre y apellidos donde se indica (borra el texto "**Nombre y apellidos**" y pon tus datos) y para añadir las referencias a los archivos JS que corresponda. **Todos los HTML han de tener su referencia al JS sino se considerará que el ejercicio no está hecho**.

NO AÑADAS LIBRERÍAS JS. Si quieres usar alguna función extra añádela en el JS que corresponda.

En ningún ejercicio es necesario **ningún tipo de formulario**, todo se probará por consola o se ejecuta directamente.

Para la realización de este examen dispones de **las tres horas de clase**, y el método de entrega es a través de Moodle. Modifica el nombre de la carpeta a:

DWEC.ExRA4.apellido.nombre

Y comprímela a un archivo ZIP, RAR o 7z con el mismo nombre

Por ejemplo, para Antonio Sierra, el archivo comprimido (y el de la carpeta) será:

DWEC.Ex.RA4.sierra.antonio

ES IMPORTANTE QUE, a la hora de entregar, guardéis los cambios, le deis a **Enviar tarea** luego confirméis que queréis enviar el trabajo para su evaluación y **confirmar que sale el estado de entrega como enviado para calificar:**

Estado de la entrega	Enviado para calificar

Para la corrección de este examen será **obligatorio** usar los nombres de funciones y parámetros que se indican.

Ejercicios

Ejercicio 1

Usando lo visto en clase. Deberás implementar mediante ***prototype*** las dos clases siguientes, cada una en su propio archivo JS, dentro de la carpeta *ejercicio1/js*.

La clase *Telefono* heredará de la clase *Dispositivo*.

NOTA: Deberás asignar a las propiedades y métodos el nombre aquí indicado. Al igual que el nombre de las clases tendrán la primera letra en mayúscula y el resto en minúscula, tal como aparece en las tablas. Observa que los setters y getters son funciones, no son puros.

CLASE	Dispositivo	[Definida mediante prototipos]	
Propiedades			
Nombre	**Tipo**	**Visibilidad**	**Descripción**
_id	**Texto**	**privado**	Valor por defecto: “NOID”
_autonomia	**número**	**privado**	Valor por defecto: 0
_carga	**número**	**privado**	Valor por defecto: 0
Setters y Getters			
Nombre	**Entrada**	**Salida**	**Descripción**
setID	**id:texto**		Convertirá la entrada *id* a mayúsculas y lo verificará con el método verificarID definido más adelante. De no ser verificado correctamente no modificará nada. De verificarlo modificará el valor de la propiedad _id.
getID		**texto**	Devolverá el valor de la propiedad _id
setAutonomia	**valor:número**		Asignará el valor recibido a _autonomia.
getAutonomia		**texto**	Devolverá el valor de la propiedad _autonomia
setCarga	**valor:número**		Comprobará que es un valor recibido es entero entre 0 y 100, ambos inclusives. De ser así asignará el valor recibido a _carga. De no verificarse el valor no se modificará nada.

Métodos			
Nombre	**Entrada**	**Salida**	**Descripción**
toString		**texto**	Devuelve la cadena de texto: *"DISP: _identificador; _modelo; _consumo;"* [Siendo las palabras en azul las propiedades de la instancia]
verificarID	**id:texto**	**bool**	La entrada *id* deberá tener al menos 10 caracteres. Devolverá TRUE de ser así, y FALSE en caso contrario.
horasRestantes	**uso:número**	**número**	Devolverá el resultado del siguiente cálculo: uso*_autonomia*_carga/100

CLASE Telefono		**[Definida mediante prototipos]**	
Hereda de *Electrodomestico*			
Propiedades			
Nombre	**Tipo**	**Visibilidad**	**Descripción**
_pulgadas	**número**	**privado**	Valor por defecto: 0
_tipoPanel	**int**	**privado**	Valor por defecto: 0
_tiposPanelArr			Array que no se podrá modificar y que tendrá por valor: *["OLED","AMOLED","QLED","NanoCell"]*
Setters y Getters			
Nombre	**Entrada**	**Salida**	**Descripción**
setPulgaddas	**valor:número**		Asignará el valor recibido a _pulgadas.
getPulgadas		**número**	Devolverá el valor de _pulgadas
setTipoPanel	**tipo:entero**		Si el valor de tipo recibido es menor que 0 o mayor a 3 asignará el valor 0 a la propiedad _tipoPanel. De lo contrario asignará el valor recibido a _tipoPanel.

getTipoPanel		**entero**	Devolverá el valor de la propiedad _tipoPanel
getTipoPanelTexto		**texto**	Devolverá el tipo de panel en formato texto, siendo ésta la correspondiente al usar la propiedad _tipoPanel como índice en el array _tiposPanelArr.
Métodos			
Nombre	**Entrada**	**Salida**	**Descripción**
toString		**texto**	Llamando al método de la clase madre, devuelve la cadena de texto de ésta añadiendo al final: *_pulgadas; _tipoPanel;* [Siendo las palabras en azul las propiedades de la instancia]
verificarID	**id:texto**	**bool**	La entrada *id* deberá comenzar con "TELF - " y deberá tener al menos 10 caracteres y un máximo de 20 en total. Devolverá TRUE de ser así, y FALSE en caso contrario.

Ejercicio 2

Usando lo visto en clase. Desarrolla las dos mismas clases de antes, pero esta vez mediante ***Class***, todos los **setters y getters** deberán ser puros, y no funciones que lo simulen como en los prototipos de antes.

Ejercicio 3

Usando lo visto en clase. En **ejercicio3.js** se te facilita la colección ***vehiculos*** ya predefinida. Básicamente es un array de objetos literales con una estructura tal que:

```
{
    tipo: "Lancha",
    marca: "Regina",
    modelo: "La que no",
    matricula: "2659SQB",
    kilometros: 69400
}
```

1. Crea una función sort que permita ordenar los vehículos según el número de kilómetros de forma ascendente (de menor a mayor). Posteriormente crea un bucle que recorra todos los elementos de la colección y saque en la división ***salida1*** una línea por cada uno

de ellos tal que:

tipo; marca; modelo; matricula; kilometros;

usando el ejemplo de antes sería:

Lancha; Regina; La que no; 2659SQB, 69400;

2. Crea una función sort que permita ordenar los vehículos según su modelo de forma ascendente (de la A a la Z). Posteriormente haz otro bucle que saque los mismos datos que antes pero en la división ***salida2***.

Ejercicio 4

Crea en el **ejercicio5.js** una función que pueda recibir un número indeterminado de parámetros de entrada con el nombre **calculosMultiples**. Deberá cumplir lo siguiente:

1. Si no recibe ningún parámetro sacará por consola el mensaje "**No se introdujeron parámetros**".
2. El primer parámetro solo podrá tomar los valores **E** o **L**, en cualquier otro caso indicará por consola el mensaje "**El cálculo escogido no es válido**"
3. En caso de escoger **E**, se esperarán solo otros dos argumentos más, que serán números y no será necesario verificar que lo son, devolviendo el resultado de elevar el primero al segundo. Por ejemplo, los parámetros (E, 2, 4) serían para hacer el cálculo 2^4, o lo que es lo mismo 2x2x2x2 = 16.

   De introducirse menos o más parámetros que los indicados sacará por consola el mensaje "**Se ha introducido un número erróneo de parámetros**".
4. En caso de introducir una **L** partir del segundo parámetro de entrada **se considerará que el usuario solo introduce números**, enteros o decimales, por lo que no será necesario verificar estos, y podrán ser tantos como quiera el usuario, y devolverá mediante **return** el listado de números ordenados de mayor a menor.

No es necesario hacer ningún prompt, menú o botón, solo la función para ejecutarla desde consola.