JavaScript

UD07. Clases y Objetos

Fundamentos de los Objetos. Ámbito.

- Los programadores de cualquier lenguaje comprenden el concepto de **ámbito**, la zona de acceso de determinadas variables.

- En ECMAScript el único ámbito que existe es el público.
- Todas las propiedades y métodos de todos los objetos de ECMAScript son públicos.

Estático o no estático.

- Desde un punto de vista estricto, ECMAScript carece de un ámbito estático. No obstante puede proporcionar propiedades y métodos de constructores que son simplemente funciones.
- Las funciones son objetos y los objetos pueden tener propiedades y métodos.

```
function diHola(){
  alert(“hi”);
}
 diHola.alternativo = function() {
  alert(“hola”); };
diHola(); //devuelve “hi”
diHola.alternativo(); //devuelve “hola”
```

En este caso, alternativo() es un método de la función diHola().

La palabra clave this.

- El uso de la palabra clave this, utilizada en métodos de objetos, siempre apunta al **objeto** que **invoca** el **método**. Se utiliza en la definición de métodos cuando se quiere hacer referencia al objeto que lo llama de manera genérica sin tener que preocuparse del nombre de la instancia.

```
function showColor() {
  alert(this.color);}

var oCar1 = new Object;
oCar1.color = "red";
oCar1.showColor = showColor;

var oCar2 =new Object;
oCar2.color ="blue";
oCar2.showColor =showColor;
```

```
oCar1.showColor(); //devuelve“red”
oCar2.showColor(); //devuelve“blue”
```

Definir clases y objetos.

- La posibilidad de utilizar objetos predefinidos es sólo una parte de cualquier lenguaje orientado a objetos.
- Las verdaderas prestaciones se obtienen al crear clases y objetos propios para usos específicos.
- La definición de objetos se puede llevar a cabo de diferentes formas:

Definir clases y objetos. Array asociativo.

- Técnicamente, un objeto JavaScript es un array asociativo formado por las propiedades y los métodos del objeto.
- La forma más directa de definir las propiedades y métodos es la notación de puntos de los array asociativos.

```
var elArray = new Array();
elArray.primero = 1;
elArray.segundo = 2;

alert(elArray['primero']);
alert(elArray.primero);
```

Definir clases y objetos. Propiedades y métodos.

- Propiedades:

```
elObjeto.id = "10";

elObjeto.nombre = "Objeto de
prueba";
```

- Métodos:

```
elObjeto.muestraId = function() {

  alert("El ID del objeto es " +
this.id);

}
```

De esta forma, en el ejemplo anterior:

```
var elObjeto = new Object();
elObjeto.id = "10";
elObjeto.muestraId = function() {
  alert("El ID del objeto es " + this.id);
}
```

Definir clases y objetos. Propiedades y métodos.

- También podemos asignar a los métodos de un objeto funciones definidas con anterioridad.

```
var elObjeto = new Object();
elObjeto.id = "10";
elObjeto.muestraId = function() {
    alert("El ID del objeto es " + this.id);
}
function obtieneId() {
    return this.id;
}
elObjeto.obtieneId = obtieneId;
//Fijaros que omitimos los paréntesis
```

Objetos que contienen otros objetos.

```
var Aplicacion = new Object();
Aplicacion.Modulos = new Array();
Aplicacion.Modulos[0] = new Object();
Aplicacion.Modulos[0].titulo = "Lector RSS";

var inicial = new Object();
inicial.estado = 1;
inicial.publico = 0;
inicial.nombre = "Modulo_RSS";
inicial.datos = new Object();

Aplicacion.Modulos[0].objetoInicial = inicial;
```

Definir clases y objetos. La notación JSON.

- JSON (JavaScript Object Notation) es un formato sencillo para el intercambio de información.
- El formato JSON permite representar estructuras de datos (arrays) y objetos (arrays asociativos) en forma de texto.
- JSON se ha convertido en una alternativa al formato XML, ya que es más fácil de leer y escribir, además de ser mucho más conciso.
- La notación JSON permite definir los arrays asociativos de una forma mucho más concisa.

Definir clases y objetos. La notación JSON.

- La notación JSON para los arrays asociativos se compone de tres partes:
  - Los contenidos del array asociativo se encierran entre llaves ({ y })
  - Los elementos del array se separan mediante una coma (,)
  - La clave y el valor de cada elemento se separan mediante dos puntos (:).

```
var titulosModulos = {"Lector RSS": "rss",
  "Gestor de email":  "email", "Agenda": "agenda"};
```

Definir clases y objetos. La notación JSON.

- Como JavaScript ignora los espacios en blanco sobrantes, es posible reordenar las claves y valores para que se muestren más claramente en el código fuente de la aplicación.

```
var títulos = {
  rss:    "Lector RSS",
  email:  "Gestor de email",
  agenda: "Agenda"
};
```

Definir clases y objetos. La notación JSON.

- La forma habitual para definir los objetos en JavaScript se basa en el siguiente modelo creado con la notación JSON:

```
var objeto = {
  "propiedad1": valor_simple_1,
  "propiedad2": valor_simple_2,
  "propiedad3": [array1_valor1, array1_valor2],
  "propiedad4": { "propiedad anidada": valor },
  "metodo1": nombre_funcion_externa,
  "metodo2": function() { ... },
  "metodo3": function() { ... },
  "metodo4": function() { ... }
};
```

Definir clases y objetos. La notación JSON.

- Notación tradicional.

```
var libro = new Object;
libro.numeroPaginas = 150;
libro.autores       = new Array();
libro.autores[0]    = new  Object();
libro.autores[0].id = 50;
libro.autores[1]    = new Object();
libro.autores[1].id  = 67;
```

- Notación JSON

```
var libro = {
 numeroPaginas: 150,
 autores: [
 {id: 50},
 {id: 67}
 ]
 };
```

Definir clases y objetos. Definición de Clase.

- La forma habitual de trabajo consiste en definir clases a partir de las cuales se crean los objetos.
- JavaScript no permite crear clases similares a las de lenguajes como Java o C++.
- De hecho, la palabra class sólo está reservada para su uso en futuras versiones hata que apareció el estándar ECMASCRIPT 6.
- Es posible utilizar en JavaScript unos elementos parecidos a las clases y que se denominan pseudoclases.
- Los conceptos que se utilizan para simular las clases son las funciones constructoras y el **prototype** de los objetos.

Clases en ECMAScript6

- Con la llegada de la nueva versión del estándar de JavaScript (ECMAScript 6 o ECMAScript 2015) la definición de una función como clase ha cambiado.
- ES6 aporta un azúcar sintáctico para declarar una clase como en la mayoría de los lenguajes de programación orientados a objetos, pero por debajo sigue siendo una función prototipal.

Declaración de clases (ES2015)

- Una manera de definir una clase es mediante una declaración de clase. Para declarar una clase, se utiliza la palabra reservada **class** y un nombre para la clase "Rectangulo".

```
class Rectangulo {
  constructor(alto, ancho) {
    this.alto = alto;
    this.ancho = ancho;
  }
}
```

Método Constructor. Métodos. (ES2015)

- El método constructor es un método especial para crear e inicializar un objeto creado con una clase.
- Solo puede haber un método especial con el nombre "constructor" en una clase.
  - Utilizando la palabra reservada **class** creamos una clase
  - El método especial **constructor** definirá la función constructora.
  - Los métodos add, borrar, cantidad y getNombre estarán dentro de la clase.

```
class Inventario {
    constructor(nombre) {
      this.nombre = nombre;
      this.articulos = [];
    }
    add (nombre, cantidad) {
      this.articulos[nombre] = cantidad;}
    borrar (nombre) {
      delete this.articulos[nombre]}
    cantidad (nombre) {
    return this.articulos[nombre]
    }
    getNombre () {
      return this.nombre;}
  }
```

Método Constructor. Métodos. (ES2015). Utilización.

- Su utilización sería de la siguiente forma:

```
var libros = new Inventario("Libros");
libros.add("AngularJS", 3);
libros.add("JavaScript", 10);
libros.add("NodeJS", 5);
libros.cantidad("AngularJS"); // 3
libros.cantidad("JavaScript"); // 10
libros.borrar("JavaScript");
libros.cantidad("JavaScript"); // undefined
```

Herencia (ES2015).

- Con esta nueva sintáxis podemos implementar la herencia utilizando las palabras reservadas **extend** para heredar de una clase más general y **super()** para llamar al constructor de la clase padre.

```
  //Creamos la clase Vehiculo
class Vehiculo {
  //Definimos el constructor
  constructor (tipo, nombre, ru
  this.tipo = tipo;
  this.nombre = nombre;
  this.ruedas = ruedas
  }
  //Definimos los métodos de la
  getRuedas () {
    return this.ruedas;}
  arrancar () {
    console.log("Arrancando el"+ this.nombre);}
  aparcar () {
    console.log("Aparcando el"+ this.nombre);}
  }
```

```
//Creamos la clase Coche que hereda de Vehiculo
class Coche extends Vehiculo {
  constructor (nombre) {
  //Llamamos al constructor de la clase
  //padre(Vehiculo) con super
  super('coche', nombre, 4);}
  }
  let fordFocus = new Coche('Ford Focus')
  fordFocus.getRuedas() // 4
  fordFocus.arrancar()
  // Arrancando el Ford Focus
```

Para saber más

- Herencia en javascript utilizando encadenamiento de prototipos.
  - https://www.arkaitzgarro.com/javascript/capitulo-9.html
- Clases y herencia utilizando ECMASCRIPT 6.
  - https://www.w3schools.com/js/js_classes.asp