Desarrollo Web en Entorno Cliente

UD 04. Clases y objetos en JavaScript

Actualizado Octubre 2021

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

> 🕮 **Importante**

> 🌡 **Atención**

> 💬 **Interesante**

ÍNDICE DE CONTENIDO

UD04. CLASES Y OBJETOS EN JAVASCRIPT

1. INTRODUCCIÓN: CLASES Y OBJETOS

JavaScript es un lenguaje que permite el uso de objetos. Muchas veces el término clase y objeto se confunden. Una definición podría ser que una clase define "como es un objeto" y que un objeto es la plasmación efectiva de ese objeto. A partir de una clase se pueden crear si se desea muchos objetos.

Para entenderlo un ejemplo: supongamos tenemos la clase "casa". Esa clase define que atributos tiene una casa. Un ejemplo de esos atributos podría ser: dirección, numero habitaciones, metros cuadrados.

Ahora bien, cada objeto es una casa existente. Podemos tener por ejemplo dos objetos que surgen a partir de la clase "casa". Uno sería una casa con dirección "Avenida del puerto 1, Valencia", 3 habitaciones y 100m2 y otro una casa con dirección "Calle Colón 1, Valencia", 5 habitaciones y 200m2.

La clase definía como podían ser los objetos y los objetos en si son clases contextualizadas en algo concreto.

2. DEFINIENDO CLASES

Las clases en JavaScript se introdujeron en esta versión (ECMAScript2015 o ES6) y supusieron una mejora significativa en la herencia basada en prototipos de JavaScript a la que estábamos acostumbrados. Estas clases nos dan una sintaxis más clara y simple para crear objetos y trabajar con herencia.

A diferencia de la versión anterior de JavaScript en la que utilizábamos la palabra reservada *function* para definir una «clase» (entre comillas), en esta versión se utiliza *class*; además, las propiedades se deben indicar dentro de un método constructor.

La sintaxis sería la siguiente:

Ejemplo JS5:

```
function coche(marca, modelo, anyo, matricula) {
      // Definimos una funcion para la clase
      function mostrarCoche() {
            var resultado = "Marca " + this.marca+ " modelo "
+this.modelo;
            alert(resultado);
      }
```

```
      // Definimos e inicializamos los atributos
      this.marca=marca;
      this.modelo=modelo;
      this.anyo=anyo;
      this.matricula=matricula;
      // Asociamos la funcion definido antes al objeto
      this.mostrarCoche=mostrarCoche;
}
```

Ejemplo 1 JS6:

```
class TarjetaFelicitacion {
  // Constructor que se inicializa con un mensaje asociado
  // a una tarjeta de felicitacioón
  constructor(mensaje) {
    this.mensajeTarjeta = mensaje;
  }
  // Recibe el nombre del destinatario y devuelve un mensaje personalizado
  mensaje(nombre) {
    return x + ", tengo este mensaje para ti " + this.mensajeTarjeta ;
  }
}
// Instanciamos la tarjeta con un mensaje
miTarjeta = new TarjetaFelicitacion("Feliz cumple!");
// Obtenemos el mensaje generado y lo metemos en el HTML de miDiv
document.getElementById("miDiv").innerHTML =miTarjeta.present("Carlos");
```

Ejemplo 2 JS6:

```
class Telefono {
  constructor (marca, modelo){
    this.marca = marca;
    this.modelo = modelo;
  }
}

let miTelefono = new Telefono ("Google", "Pixel");
console.log(miTelefono.marca+" "+miTelefono.modelo);
```

Mas información en https://www.w3schools.com/js/js_classes.asp

3. HERENCIA

Al igual que en otros lenguajes de programación, definimos la relación entre una clase padre y una clase hijo utilizando la palabra reservada "*extends*", que definiría la relación de esta segunda clase con la primera. Además, para hacer referencia a los atributos de la clase padre utilizamos la palabra reservada *"super"*.

Brevemente, la sintaxis para definir una cabecera de una clase hijo sería la siguiente:

```
class ClaseHijo extends ClasePadre
```

Continuando con el ejemplo mostrado en el apartado anterior vamos a crear una clase Modelo que hereda de Teléfono y también cómo hacer referencia a las propiedades y a los métodos de la clase padre desde la clase hija.

Ejemplo:

```
class Telefono {
        constructor (marca){
            this.marca = marca;
        }
        anuncio () {
            return "Ha llegado el nuevo teléfono de "+this.marca;
        }
    }
class Modelo extends Telefono {
        constructor (marca, modelo){
            super(marca);
            this.modelo = modelo;
        }
        anuncioCompleto(){
            return this.anuncio() + ": el modelo " + this.modelo;
        }
    }
let miTelefono = new Modelo ("Google", "Pixel");
mensaje.innerHTML = miTelefono.anuncioCompleto();
```

4. USO DE MÉTODOS GET Y SET

Estos métodos nos permiten extraer (get) y modificar (set) las propiedades de un objeto. De este modo, nosotros podemos elegir exactamente, mediante estos métodos, qué propiedades pueden ser accedidas y modificadas y cuáles no.

De hecho, los *"getters"* y *"setters"* determinan el fundamento del principio de encapsulación de la programación orientada a objetos. Lo habitual en otros lenguajes de programación es definir los *"getters"* y *"setters"* con la palabra **get** o **set** seguida del nombre de la propiedad. Pero JavaScript es un caso especial, y los getters y setters se escriben con la palabra get o set, separadas por un espacio del nombre de la propiedad, con una particularidad

❗ Atención: ¡no podemos poner el mismo nombre al método que a la propiedad porque entraríamos en un bucle y se produciría un error!

Por eso muchos desarrolladores utilizan el guion bajo para nombrar la propiedad.

Ejemplo:

```
class Telefono {
    constructor(marca){
      this._marca = marca;
    }
    get marca() {
      return this._marca;
    }
    set marca (mar) {
      this._marca = mar;
    }
  }
  let miTelefono = new Telefono ("Google");
  miTelefono.marca = "iPhone";
  mensaje.innerHTML = "Mi telefono es un "+miTelefono.marca;
```

5. USO DE MÉTODOS ESTÁTICOS (STATIC)

Al igual que ocurre en otros lenguajes de programación, un método estático se llama directamente sin instanciar la clase (es decir, sin necesidad de crear un objeto). De hecho, si tratáramos de llamar a un método estático a partir de un objeto obtendríamos un error.

Este tipo de métodos se utilizan sobre todo para crear funciones de utilidad en una aplicación.

Para crear un método estático no hay más que utilizar la siguiente sintaxis:

```
static nombreMetodo (parametros) {//código}
```

Vamos a ver un ejemplo calculando el área y el perímetro de un rectángulo.

Ejemplo:

```
class Rectangulo {
    constructor(base, altura){
      this.base = base;
      this.altura = altura;
    }
    static area (b, a) {
      return b * a;
    }
    static perimetro (b, a) {
      return 2*b + 2*a;
    }
  }
let rectangulo1 = new Rectangulo (2,3);
//Si utilizamos directamente los elementos del objeto
//instanciado daría un error
//console.log(rectangulo1.area(2,3));
//console.log(rectangulo1.perimetro(2,3));
mensaje.innerHTML = Rectangulo.perimetro(2,3);
```

6. PASANDO VARIABLES, ARRAYS Y OBJETOS A CADENAS: JSON

JSON es una notación para convertir variables, arrays y objetos en cadenas de texto y así poder facilitar la comunicación entre distintos programas (enviándose el contenido de un objeto como cadena de texto.

Más información sobre el formato en http://www.w3schools.com/js/js_json.asp

En este tema solo explicaremos su uso, con un fin introductorio.

JSON es uno de los formatos más utilizados para intercambiar información entre programas (cliente, servidor, API...).

En Javascript podemos usar JSON de la siguiente forma:

A) Para convertir un objeto a texto siguiendo el formato JSON

```
textoJSON=JSON.stringify(objeto);
```

B) Para convertir una cadena de textoen JSON a un objeto usamos:

```
let objeto = JSON.parse(textoJSON);
```

Ejemplo:

```
//Construimos los objetos a partir de la clase
let coche1=new coche("Seat","Ibiza",2000,"1234ABC");
textoJSON=JSON.stringify(coche1);
alert(textoJSON);
//Reconstruimos el objeto y usamos un metodo para probarlo
let cocheReconstruido=JSON.parse(textoJSON);
cocheReconstruido.mostrarCoche();
```

7. MATERIAL ADICIONAL

[1] Curso de Javascript en Udacity https://www.udacity.com/course/javascript-basics--ud804

[2] Trabajando con objetos en Javascript https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Trabajando_con_objectos

8. BIBLIOGRAFÍA

[1] Javascript Mozilla Developer https://developer.mozilla.org/es/docs/Web/JavaScript

[2] Javascript ES6 W3C https://www.w3schools.com/js/js_es6.asp