TEMA 5 (Parte C). Clases.(POO).

Desarrollo Web en Entorno Cliente.

Profesor: Juan José Gallego García

Índice :

Constructores en JS. Prototype. POO basado en prototipos.

En JS un objeto (Objeto, Array, Function ….) hereda propiedades de otro objeto general que posee, llamado **prototype**. Cuando hacemos referencia a una propiedad de las definidas de un objeto que no tiene, la busca en su prototipo. De esta forma un objeto “hereda propiedades” de ese prototipo.

Ej: `array.length ;`

Vimos en la parte anterior del tema cómo crear objetos con propiedades y métodos definiendo individualmente cada uno. Pero si necesitamos crear un gran número de objetos del mismo tipo habría que repetir la definición de las mismas propiedades y métodos una y otra vez.

Por ello es más fácil hacer uso de una función constructora que haga de plantilla para crear esos objetos. La forma es la siguiente :

```
function Coche (marca,modelo,color){
  this.marca=marca;
  this.modelo=modelo;
  this.color=color;
  this.nombre=function ( ) {return this.marca +" "+this.modelo}
}
```

- La función constructora ‘Coche’ contiene las propiedades y métodos que serán copiadas a cada objeto instanciado.
- La clave ‘this’ ,en este caso, hace referencia a cada objeto instanciado.
- Por convenio se nombra la función con la primera letra en mayúscula.

- Para instanciar los nuevos objetos se usa ‘new’:

```
var coche1 = new Coche ("Audi","A4","Gris");
var coche2 = new Coche ("Seat","León","Negro");
```

- Para acceder a cada propiedad o método :

```
alert (coche1.nombre()); // Devuelve 'Audi A4'
alert (coche2.nombre()); // Devuelve 'Seat León'
alert (coche1.marca);    // Devuelve 'Audi'
```

- Para añadir una nueva propiedad a un objeto, NO al constructor :

```
coche1.familiar=false; // Añade una nueva propiedad.
alert(coche1.familiar); // Devuelve 'false'.
alert(coche2.familiar); // Devuelve 'undefined',esta propiedad sólo existe en el objeto 'coche1'
```

- Para añadir una nueva propiedad o método al constructor :

```
Coche.prototype.familiar = false;
Coche.prototype.getColor=function (){return this.color};
alert(coche1.getColor()); // Devuelve 'Gris'.
alert(coche2.familiar); // Devuelve 'false'.
```

Métodos para propiedades de objetos.ES5

```
// En el ejemplo anterior:
 console.log( Object.getOwnPropertyNames(coche1)
```

```
// Crea o modifica la propiedad de un objeto.

Object.defineProperty(object, property, descriptor)

// Crea o modifica varias propiedades de un objeto

Object.defineProperties(object, descriptors)

// Accede a los atributos de una propiedad

Object.getOwnPropertyDescriptor(object, property)

// Devuelve un array con todas las propiedades.

Object.getOwnPropertyNames(object)

// Devuelve un array con las propiedades enumerables.

Object.keys(object)

// Devuelve el prototype de un objeto.

Object.getPrototypeOf(object)
```

```
// Evita que se creen nuevas propiedades a un objeto.

Object.preventExtensions(object)

// Devuelve True si se permite añadir propiedades al obj

Object.isExtensible(object)

// Evita modificar atributos de propiedades, sí valores

Object.seal(object)

// Consulta si un objeto es modificable según lo anterior.

Object.isSealed(object)

// Evita cualquier cambio en un objeto.

Object.freeze(object)

// Devuelve True si un objeto está congelado (ver anterior)

Object.isFrozen(object)
```

Variables privadas en funciones constructoras con prototipos.

En los anteriores ejemplos hemos instanciado objetos creados a partir de un constructor, y nos permitía acceder a variables o propiedades del objeto directamente, sin pasar por un método que las devolviera.
Sabemos que para poder llevar un buen control del código, y que no aparezcan errores de codificación no deseados, es aconsejable mantener de forma privada las variables de estado de los objetos, es lo que llamamos encapsulación. En los primeros estándares de JS no existen modificadores para conseguir hacer privadas la variables dentro de la función constructora, pero se puede declarar las variables sin el this, consiguiendo así que solo se puedan hacer llamadas desde métodos internos:

```
function Coche (marca){
  var _marca=marca;
  this.nombre=function ( ) {return _marca};
}
```

```
var coche1 = new Coche ("Ford");
alert (coche1.nombre()); // Devuelve 'Ford'
alert (coche1._marca); // Devuelve 'undefined'
```

Puede ser necesario llamar a un método privado, y éste tener que acceder a una variable pública dentro de la clase, una de las formas :

```
function Coche (marca){
  var _marca=marca;
  this.modelo="Mustang";
  var model= function(){return this.modelo};
  this.nombre=function ( ) {return model.call(this)};
}
var coche1 = new Coche ("Ford");
alert (coche1.nombre()); // Devuelve 'Mustang'
alert (coche1.model()); // Devuelve 'error is not function'
```

Por qué model.call(this)?, si usáramos la forma model ( ) en la llamada al método, y al no ser reconocido en las instancias, this (**this.modelo**) hace referencia al objeto windows (objeto global) y no al propio objeto donde está definido.

Herencia (prototipos)

La herencia es una de las características más importante en que se basa la POO. Para ello, una clase puede heredar propiedades y métodos de otra superior o padre. En POO por prototipos la forma de herencia se especifica creando una llamada (call) de la clase padre con los parámetros correspondientes:

```
function Coche (marca,modelo,color){
  this.marca=marca;
  this.modelo=modelo;
  this.color=color;
  this.nombre=function ( ) {return this.marca+" "+this.modelo}
}
```

```
function CocheFurgoneta (marca,modelo,color,volumen) {
   Coche.call(this,marca,modelo,color);
   this.volumen=volumen;
   this.getVolumen=function () {return this.volumen};
}
```

Podemos instanciar el nuevo objeto :

```
var coche1 = new CocheFurgoneta ("Renault","Express","Blanco",3);
alert(coche1.nombre()+" con volumen "+coche1.getVolumen()+"m3");
// Salida: "Renault Express con volumen 3m3"
```

Podemos mostrar todas las propiedades y métodos del objeto instanciado.

```
alert(Object.keys(coche1));
```

Definición de clases en ES6. Class

Anteriormente hemos visto la forma de cómo a partir de un constructor de una función podemos crear ‘plantillas’ de objetos para luego instanciarlos, y aunque no sigue características estrictas de programación orientada a objetos como ya conocemos en otros lenguajes (java, C++, php , etc…) ,vamos a poder realizar programas basados en este paradigma.

No obstante, a partir de la versión ES6 se ha introducido el concepto de clase (mediante la palabra clave class) , que aproxima y formaliza el uso de clases para programación orientada a objetos en JS.

Para definir una clase usamos class y añadimos un constructor a la clase.

```
class Coche {
   constructor(marca,modelo,color){
         this.marca=marca;
         this.modelo=modelo;
         this.color=color;
         this.nombre=function ( ) {return this.marca+" "+this.modelo}
    }

     muestra () {return this.marca+" "+this.modelo};
}

var coche1= new Coche ("Renault","Express","Blanco");
alert (coche1.nombre()); // Salida: Renault Express
alert (coche1.muestra());//Salida : Renault Express
```

Se ve en el ejemplo anterior, que se pueden declarar métodos dentro del constructor con this creándose una copia para cada instancia, o fuera, común a las instancias.

Variables y métodos privados en clases. Estándares antiguos

Al igual que por prototipos, declarar variables o métodos privados lo hacemos sin el this, en este caso no es necesario call (this), sólo llamar al método.

```
class Coche {
    constructor(marca,modelo,color){
        var _varPrivada = 5;
        var _muestraPrivado=function () {return _varPrivada};
        this.marca=marca;
        this.modelo=modelo;
        this.color=color;
        this.nombre=function ( ) {return _muestraPrivado( )}
    }

}
 var coche1= new Coche ("Renault","Express","Blanco");
 alert (coche1.nombre()); // Muestra 5
 alert (coche1._varPrivada); // Error
 alert (coche1._muestraPrivado()); // Error
```

IMPORTANTE: Para la forma que se expone en el ejemplo sólo se reconocen las variables privadas en métodos definidos dentro del **constructor**.

Variables y métodos privados en clases. Últimos estándares

Otra forma relativamente nueva para hacer privada una variable y un método es anteponiendo '#' :

```
class Coche {
     #marca;  //Variable privada
     #modelo; //Variable privada
    constructor(marca,modelo){
       this.#marca=marca;
       this.#modelo=modelo;

     }
     #nombre ( ) {return this.#marca+" "+this.#modelo}; //Privado
     nombre1 () {return this.#nombre ()}; //Público ...
 }
 var coche1= new Coche("Renault","Express"); //Instanciamos ...
 alert(coche1.nombre1()); //Renault Express;
```

Variables y métodos estáticos.

Las variables y métodos estáticos son llamados a partir del nombre de la clase, no se instancian, normalmente son usados para crear funciones en las clases cuando no es necesario instanciar. Se crean con el modificador **static**. También pueden ser privados:

```
class Valores {
    static x=5;
    static y=4
    constructor(){  }
    static nombre (n) {return (this.x+this.y)*n};
}
alert (Valores.x); //Devuelve 5
alert(Valores.nombre(2))  //Devuelve 18
```

Herencia (clases)

Para crear una herencia usamos la palabra clave extends, y con super llamamos las propiedades y métodos de la clase padre.

```
class Coche {
     #marca; //Variable privada
     #modelo;
    constructor(marca,modelo){
       this.#marca=marca;
       this.#modelo=modelo;

     }
     nombre ( ) {return this.#marca+" "+this.#modelo}; //Privado

 }
class CocheFurgoneta extends Coche {
    #volumen;
    constructor(marca,modelo,volumen){
        super(marca,modelo) ;
        this.#volumen=volumen;
     }
     getVolumen () {return this.#volumen};
 }
 var coche1= new CocheFurgoneta ("Renault","Express",3);
 alert(coche1.nombre()+" con volumen "+coche1.getVolumen()+"m3");
```

Por último,

- Saber que para comprobar si un objeto está instanciado correctamente de una clase, usamos instanceof y devuelve ‘true’.

```
alert (coche1 instanceof Coche) ; // Devuelve true
alert (coche1 instanceof CocheFurgoneta) ; // Devuelve true
```

- Para usar una clase antes hay que declararla. (**modo estricto)**.
- En **JS**, las clases deben tener un solo método **constructor**.
- Se pueden usar los métodos set ( ) y get ( ) haciendo las llamadas para leer o establecer los valores como propiedades.

```
get lee ( ) {return this.dato)}; // Declarado en la clase fuera del constructor.
set escribe (x) {this.dato=x};
….
objeto1.lee; // Se llama como propiedad.
objeto1.escribe= 4; // Se establece como propiedad
```

Bibliografía

- LibrosWeb
- W3Schools
- Developer Mozilla Docs