JS: POO

Antonio Espín Herranz

Terminología

- Clase
  - Define las características del Objeto.
- Objeto
  - Una instancia de una Clase.
- Constructor
  - Un método llamado en el momento de la instanciación. Permite inicializar el objeto que se crea.
- Propiedad
  - Una característica del Objeto, como el color.
- Método
  - Una capacidad / funcionalidad del Objeto, como caminar.
- Herencia
  - Una Clase puede heredar características de otra Clase. Extiende, especializa el comportamiento de otra clase.
- Encapsulamiento
  - Una Clase sólo define las características del Objeto, un Método sólo define cómo se ejecuta el Método.
- Abstracción
  - La conjunción de herencia compleja, métodos, propiedades de un objeto debe ser capaz de simular un modelo de la realidad.
- Polimorfismo
  - Diferentes Clases podrían definir el mismo método o propiedad.

Objetos en JavaScript

- JavaScript permite la programación orientada a objetos:
  - Está basado en prototipos: las nuevas clases se generan clonando las clases base (prototipos) y extendiendo su funcionalidad
- Un objeto JavaScript tiene propiedades y métodos
  - Ejemplo: propiedades y métodos de un objeto `String`

```
<script type="text/javascript">
   var txt="Hola mundo!"
   document.write(txt.length)          //propiedad
   document.write(txt.toUpperCase())   //metodo
</script>
```

Objeto JavaScript VS Objeto Java

- Similitudes
  - Ambos tienen propiedades y métodos
- Diferencias
  - Un objeto JavaScript tiene tipado dinámico mientras que un objeto Java tiene tipado estático
  - En JavaScript, se pueden añadir propiedades y métodos a un objeto dinámicamente

Creación de un objeto JavaScript

- Formas diferentes de crear clases / objetos:
  1. Crear una instancia directa de un objeto usando el constructor predefinido de la clase **`Object.`**
  2. Crear primero una plantilla (**constructor**) para crear la instancia del objeto a partir de ésta.
  3. Crear la instancia de un objeto usando el formato **JSON**. Objetos en línea.

Opción 1: crear una instancia de `Object` (1)

- Invocar el constructor de la clase Object

```
personaObj = new Object(); //inicialmente vacio
```

- Añadir propiedades al objeto

```
personaObj.nombre = "Pau";
personaObj.apellidos = "Gasol";
personaObj.edad = 27;
```

- Añadir una función anónima al objeto

```
personaObj.displayEdad = function() {
        alert("La edad es " + this.edad);
}
```

Opción 1: crear una instancia de `Object` (y 2)

- Añadir una función predefinida

```
function displayEdad() {
        alert("La edad es " + this.edad);
}

personaObj.displayEdad = displayEdad;
```

Opción 2: crear una plantilla del objeto

- La **plantilla** es una función que define la estructura del objeto
- Puede pensarse en ella como el **constructor** del objeto

```
function Persona(nombre, apellidos, edad) {
    this.nombre = nombre;
    this.apellidos = apellidos;
    this.edad = edad;
    this.displayEdad = function() {
        alert("La edad es " + this.edad);
    }
}
```

- this hace referencia al propio objeto.
- A partir de la plantilla se pueden crear instancias del objeto

```
pauGasol = new Persona("Pau", "Gasol", 27);
```

- Al objeto se le pueden añadir nuevas propiedades o métodos

```
pauGasol.equipo = "Lakers";
```

Opción 2a: constructor

- ```
  class ClassName {
    constructor() { ... }
    method_1() { ... }
    method_2() { ... }
    method_3() { ... }
  }
  ```

- Dentro del constructor se reciben parámetros y dentro se definen las propiedades de la clase y se le asignan valores.

- A los métodos no hay que poner function.
  ```

Opción 3: Objetos en línea

- Se pueden definir objetos utilizando la sintaxis de **JSON**:

```
var miObjeto =
{
    propiedad1: valor1,
    propiedad2: valor2,
    metodo1: function()
    {
        // código del método1.
    }
}
```

Utilizar

- Utilizaremos el formato json para trabajar con los frameworks de JS y para crear nuestros objetos.
  - Opción 3

- Además también la forma más moderna definiendo el constructor.
  - Opción 2a

Ejemplo: Objetos en línea

```
timObject = {
    property1 : "Hello",
    property2 : "MmmMMm",
    property3 : ["mmm", 2, 3, 6, "kkk"],
    method1 : function(){alert("Method had been called" + this.property1)}
};

timObject.method1();
alert(timObject.property3[2]); // Imprime un 3.

var circle = { x : 0, y : 0, radius: 2 }

// Permite anidar objetos.
var rectangle = {
    upperLeft : { x : 2, y : 2 },
    lowerRight : { x : 4, y : 4}
}

alert(rectangle.upperLeft.x) // Imprime un 2.
```

Objetos JavaScript como *arrays* asociativos

- Un objeto JavaScript es básicamente un array asociativo con propiedades y métodos, que son indexadas por el nombre.

- Las siguientes líneas de código son semánticamente equivalentes
  - `miObjeto.miPropiedad = "loquesea";`
  - `miObjeto['miPropiedad'] = "loquesea";`
  - **Esto nos permite recorrer todas las propiedades y métodos del objeto con un bucle `for in`.**

Herencia: extends

```
class Padre {
    constructor(param1){
        this.param1 = param1;
    }
    metodo1(){}
}
class Hija extends Padre {
    constructor(param1, param2){
        super(param1);
        this.param2 = this.param2;
    }
    metodo2();
}
```

Prototype

- Un **`prototype`** es una propiedad de todo objeto JavaScript
- Permite añadir propiedades y métodos a todos los objetos instanciados con la función **`new`**

```
// Constructor de MyObject
function Persona(nombre, altura){
    this.nombre=nombre;
    this.altura=altura;
}

// Añade una función al prototipo
Persona.prototype.displayAltura = function {
   alert("La altura es " + this.altura);
}

// Crea una instancia del objeto
var pau=new Persona("Pau", "215 cms");
pau.displayAltura();
```

Herencia con prototype

- Otra posibilidad de prototype es utilizarlo para la herencia:

```
function Padre() {
   this.atributoPadre=3;
}

function Hijo() {
   this.atributoHijo=4;
}
Hijo.prototype = new Padre;

a = new Hijo;
alert (a.atributoPadre);
alert (a.atributoHijo);
```

Herencia con prototype

- Cuando los constructores tienen argumentos: utilizar call desde la clase hija:

```
function Padre(attPadre) {
   this.atributoPadre=attPadre;
}
 function Hijo(attHijo, attPadre) {
   Padre.call(this,attPadre);
   this.atributoHijo=attHijo;
}
Hijo.prototype = new Padre;

a = new Hijo(2,3);
alert (a.atributoPadre);
alert (a.atributoHijo);
```

Ejemplos: constructor (otra notación)

- Dos formas de crear el constructor:

```
function Gato( parametros ) {
// Codigo
};

var Gato = function (parametros) {
/* Codigo*/
}
```

Ejemplos: con propiedades

```
var Gato = function (nombre, color, edad) {
        this.nombre = nombre;
        this.color = color;
        this.edad = edad;
        }

// Podemos asignar tipos:
var Gato = function (nombre, color, edad) {
                this.nombre = new String(nombre);
                this.color = new String(color);
                this.edad = new Number(edad);
                //Por ejemplo
                if (isNaN(this.edad)) {
                                alert("Error en el data-typing, edad no es un numero");
                }
}
var Michi = new Gato(("Michifu", "azul", 2);
```

Ejemplos: funciones externas

- También se pueden asignar funciones externas a propiedades de la clase:

```
Dentro de la clase …
function Gato(){
        …
        this.comer = comeExterna;
        …
}
// No es necesario que esté por encima de la clase:
function comeExterna(){
    console.log("…");
}
var o =  new Gato("","",9);
o.comer();
```

Ejemplo: Herencia

```
var Gato = function () {
    this.ojos = 2;
    this.piernas = 4;
}
var Siames = function () {
    this.color = "blanco";
    this.color_ojos = "azul";
}
Siames.prototype = new Gato();
//Eso hace que se copie el prototipo de Gato y se añada al de Siames.
var Catboy = new Siames();
alert(Catboy.ojos);
alert(Catboy.color);
```

Ejemplo 2: Herencia

```
var Saludator = function(nom){
    this.nombre = nom
    this.saluda = function() {
        alert("hola "+this.nombre)
    }
}
// Añade un objeto al prototipo
Saludator.prototype = {
    apellido: "cruel",
    despide: function() {
        alert("adios "+this.nombre+" "+this.apellido)
    }
}
var obj1 = new Saludator("mundo")
obj1.despide()
// "adios mundo cruel"
```

Ejemplo: var privadas

- Dentro de la clase se pueden definir variables privadas (basta con hacerlas con var).
- Se pueden añadir métodos getters / setters.

Ejemplo: setters / getters

```
var Gato = function (nombre, color, edad) {
    var alias;      // Ojo, no ponemos this.
        …
    …
    this.getAlias = function (){
     return alias;
    }

    this.setAlias = function(nuevoAlias){
        alias = nuevoAlias;
    }
}
```

Clases ES6

```
class Persona {
    // Atributos privados
    #nombre;
    #edad;
    #altura;

    // Variable estática privada
    static #numInstancias = 0;

    constructor(nombre='', edad=0, altura=0.0){
        this.#nombre = nombre
        this.#edad = edad
        this.#altura = altura

        Persona.#numInstancias++
    }

    static getNumInstancias(){
        return  Persona.#numInstancias
    }

    getNombre(){
        return this.#nombre
    }

    setNombre(nombre){
        this.#nombre = nombre
    }

}
```

- Se pueden definir atributos y métodos privados, así como atributos y métodos static que sean públicos o privados.

Agrupando clases: paquetes

- Las clases las podemos agrupar en paquetes / namespace para evitar problemas en proyectos grandes:

```
paquete = {

   Padre : function (attPadre) {
      this.atributoPadre=attPadre;
   },

   Hijo : function (attHijo, attPadre) {
        paquete.Padre.call(this,attPadre);
        this.atributoHijo=attHijo;
   }
}
paquete.Hijo.prototype = new paquete.Padre;

a = new paquete.Hijo(2,3);
alert (a.atributoPadre);
alert (a.atributoHijo);
```

Agrupando clases: subpaquetes

```
paquete = {

   Padre : function (attPadre) {
      this.atributoPadre=attPadre;
   },

   subpaquete : {
      Hijo : function (attHijo, attPadre) {
         paquete.Padre.call(this,attPadre);
         this.atributoHijo=attHijo;
      }
   }
}
paquete.subpaquete.Hijo.prototype = new paquete.Padre;

a = new paquete.subpaquete.Hijo(2,3);
alert (a.atributoPadre);
alert (a.atributoHijo);
```

instanceof

- Para comprobar si un objeto pertenece a una determinada clase:

- Objeto **instanceof** Clase
  - Devuelve true o false.