Objetos en JavaScript

Como ya sabemos, en JavaScript un objeto es un tipo de dato representado por llaves (**{}**) donde podemos guardar otros datos. Estos datos se almacenan designándoles una propiedad y a esa propiedad un valor.

Los objetos se pueden construir de dos maneras: de forma **literal** y a partir de un **constructor** (generando nuevas **instancias**).

Para crear un objeto literal solo nos basta declarar una nueva variable y asignarle a ella un objeto.

Para crear objetos a partir de un constructor, primero hay que crear el constructor. Un constructor funcionará como una plantilla con la cual podremos generar nuevos objetos, asociándole todo lo que contenga el constructor. En esto, podemos asignarle dos cosas: **propiedades** (donde asignaremos las “cualidades” del objeto) y **métodos** (las funciones asignadas como acciones del objeto)

Hasta este momento, lo que sabemos es que para crear un constructor se debe crear una función constructora con palabra reservada *function* y dentro de ella asignarle las propiedades y métodos que queremos que nuestro objeto tenga. Acá unos ejemplos en código:

```
// Objeto literal:
var estudianteSusana = {
  nombre: 'Susana',
  conocimientos: ['variables', 'ciclo for', 'funciones',
'estructuras condicionales'],
  estudiar: function(aprendizaje){
    this.conocimientos.push(aprendizaje);
  }
}

// Constructor:
function Estudiante(name){
  this.nombre = name;
  this.conocimientos = [];
  this.estudiar = function(aprendizaje){
    this.conocimientos.push(aprendizaje);
  }
}

// Instancia:
var susana = new Estudiante();
susana.estudiar('objetos');
```

Una de las mejoras que propone ES6 para el trabajo con objetos en JavaScript, es el uso del concepto de **clase**, proveniente de la **Programación Orientada a Objetos**. Básicamente: El concepto de lo que hacíamos anteriormente creando una función constructora, en POO se le llama **clase**. La clase es el modelo (o plantilla, como le habíamos llamado), que define las variables y comportamientos que tendrán los objetos creados a partir de ella.

Ahora, para crear un constructor debemos utilizar una nueva palabra reservada: class. Dentro de ella ahora todo será métodos, incluido el constructor, que se llamará al momento de crear una nueva instancia. Veamos el mismo ejemplo anterior, ahora escrito con ES6:

```
class Estudiante{
        constructor(name){
          this.nombre = name
          this.conocimientos = []
        }
        estudiar(aprendizaje){
          this.conocimientos.push(aprendizaje)
          return this.conocimientos
        }
      }

      let susana = new Estudiante('Susana')
      susana.estudiar('Objetos con ES6')
```

Clases en ECMAScript 2015 (ES6)

Sintáxis básica

Para crear una clase en ES6 vamos a utilizar la palabra reservada *class*, en el modo "*use strict*", de esta forma:

```
Clase básica
"use strict";

class Person {
    constructor(firstname, lastname, birthday) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.birthday = new Date(birthday);
    }
    getAge() {
        var ageDifMs = Date.now() - this.birthday.getTime();
        var ageDate = new Date(ageDifMs);
        return Math.abs(ageDate.getUTCFullYear() - 1970);
    }
}
```

typeof de una clase

Si comprobamos el tipo de *Person* nos vamos a encontrar que *typeof* devuelve *function*, pero a diferencia de las funciones, no vamos a poder llamar directamente a *Person* y siempre vamos a tener que utilizar *new*:

```
// Comprobamos el tipo de Person
// Devuelve function
console.log(typeof Person);

// Intentamos llamar a Person como una función
// TypeError: Class constructors cannot be invoked without 'new'
Person('Pablo', 'Almunia', '07-08-1966');

// Creamos un objeto con new
var p = new Person('Pablo', 'Almunia', '07-08-1966');
```

Definición y uso

A diferencia de las funciones, las clases no pueden ser utilizadas antes de ser definidas. Las funciones en Javascript sufren un proceso denominado *hoisting* por el que las variables declaradas con *var* y las funciones son desplazadas al principio del alcance donde se encuentra y como consecuencia pueden ser llamadas antes de su declaración. En el caso de las clases esto no funciona así y si intentamos utilizar una clase antes de definirla obtendremos un error:

```
"use strict";

// Producirá un error, ya que estamos usando una clase antes de declararla
// ReferenceError: Person is not defined
var p = new Person('Pablo', 'Almunia', '07-08-1966');

class Person {
    constructor(firstname, lastname, birthday) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.birthday = new Date(birthday);
    }
    get age() {
        var ageDifMs = Date.now() - this.birthday.getTime();
        var ageDate = new Date(ageDifMs);
        return Math.abs(ageDate.getUTCFullYear() - 1970);
    }
}
```

Expresiones de clases

De forma similar a las funciones, es posible asignar la definición de una clase a una variable. A diferencia de las funciones, no se aceptan clases anónimas, pero al igual que en caso de las funciones, el nombre de la clase queda oculto y sólo es posible utilizar el nombre de la variable que se ha utilizado en la asignación:

```
"use strict";

const People = class Person {
    constructor(firstname, lastname, birthday) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.birthday = new Date(birthday);
    }
    getClassName() {
        return Person.name;
    }
};
 var p = new People();
console.log(p.getClassName());
 var p2 = new Person();
```

Constructor

Dentro de la definición de la clase no es posible escribir código directamente, ya que como tal no se comporta como una función, lo que vamos a encontrar es la definición de métodos. Uno de estos métodos se comporta de una manera especial: *constructor*. Esta función será llamada cuando se instancie un objeto de esta clase por medio de *new*. Este pseudométodo tiene algunas características singulares, como que representa la función de la propia clase:

Constructor y clase
// Es verdadero
console.log(Person === Person.prototype.constructor);

El constructor no tiene por qué devolver un objeto, de forma implícita devuelve el objeto *this*, pero podemos devolver lo que queramos y por lo tanto podemos hacer que haga cosas bastante curiosas sobrescribiendo el resultado de llamar a *new*:

El constructor devuelve un objeto diferente al previsto

```
"use strict";

class Person {
   constructor(firstname, lastname, birthday) {
      return {
         lastname,
         firstname,
         birthday: new Date(birthday)
      };
   }
   getAge() {
      var ageDifMs = Date.now() - this.birthday.getTime();
      var ageDate = new Date(ageDifMs);
      return Math.abs(ageDate.getUTCFullYear() - 1970);
   }
}

// Creamos un objeto
var p = new Person('Pablo', 'Almunia', '07-08-1966');

// Podemos llamar a las propiedades del objeto devuelto por el constructor
console.log(p.firstname, p.lastname);

// No podemos llamar el método getAge(), ya que no está disponible en el objeto
// devuelto por el constructor, aunque exista en la definición de la clase
console.log(p.getAge());
```

Si no se define ningún método *constructor* se utilizará un constructor por defecto.

Métodos

El resto de métodos se definen en el cuerpo de la clase sin necesidad de utilizar *function*, basta con poner el nombre del método y paréntesis con los parámetros que recibirá (o ninguno) y escribir el cuerpo de la función.

Definición de métodos

```
"use strict";

class Person {
    constructor(firstname, lastname, birthday) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.birthday = new Date(birthday);
    }
    getAge() {
        var ageDifMs = Date.now() - this.birthday.getTime();
        var ageDate = new Date(ageDifMs);
        return Math.abs(ageDate.getUTCFullYear() - 1970);
    }
    toString() {
        return this.firstname + ' ' +  this.lastname + ' (' + this.getAge() + ')';
    }
}

var p = new Person('Pablo', 'Almunia', '07-08-1966');

// Produce una llamada implícita a toString()
console.log('Person: ' + p);
```

Métodos estáticos

Si lo necesitamos podemos escribir métodos estáticos, es decir, que pueden ser invocados desde la clase sin necesidad de que se cree una instancia de la misma. Para ello tenemos que poner la palabra *static* antes del nombre del método:

Método estático

```
"use strict";

class Person {
    constructor(firstname, lastname, birthday) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.birthday = new Date(birthday);
    }
    static getClassName() {
        return 'Person class';
    }
}

// Devuelve "Person class"
console.log(Person.getClassName());
```

Propiedades

No es posible definir propiedades en el cuerpo de una clase. Para poderlas definir podemos utilizar el constructor (o desde cualquier otro método), tal y como hemos visto en el primer ejemplo, donde se definen las propiedades *firstname*, *lastname* y *birthday* por medio de *this*.
Una alternativa a esta forma de crear las propiedades es utilizar métodos *get* y/o *set* para definir dos funciones que nos sirvan para gestionar su acceso:

```
"use strict";

class Person {
    constructor(firstname, lastname, birthday) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.birthday = new Date(birthday);
    }
    get age() {
        var ageDifMs = Date.now() - this.birthday.getTime();
        var ageDate = new Date(ageDifMs);
        return Math.abs(ageDate.getUTCFullYear() - 1970);
    }
}
 var p = new Person('Pablo', 'Almunia', '07-08-1966');
console.log(p.firstname, p.lastname);
console.log(p.age);
```

Herencia

Aunque en ES5 podemos utilizar la cadena de prototipos para realizar una herencia, con la sintaxis de clases de ES6 esto es mucho más sencillo y claro. Para ello debemos utilizar *extends* en la declaración de la clase:

Herencia

```
"use strict";

class Person {
    constructor(firstname, lastname, birthday) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.birthday = new Date(birthday);
    }
    get age() {
        var ageDifMs = Date.now() - this.birthday.getTime();
        var ageDate = new Date(ageDifMs);
        return Math.abs(ageDate.getUTCFullYear() - 1970);
    }
}

class Employee extends Person {
    constructor(firstname, lastname, birthday, position) {
        super(firstname, lastname, birthday);
        this.position = position;
    }
}

// Creamos un objeto
var p = new Employee('Pablo', 'Almunia', '07-08-1966', 'Architect');

// Consultamos sus propiedades
console.log(p.firstname, p.lastname, p.position);
console.log(p.age);
```

super

Para que un constructor o un método puedan llamar a miembros de la clase padre debe utilizar *super* que es una referencia a la clase superior. En el caso del constructor se le llamará simplemente con los parámetros, en el caso de querer llamara métodos o propiedades concretas usaremos *super*.nombredelmiembro.

Uso de super

```
"use strict";

class Person {
    constructor(firstname, lastname, birthday) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.birthday = new Date(birthday);
    }
    get age() {
        var ageDifMs = Date.now() - this.birthday.getTime();
        var ageDate = new Date(ageDifMs);
        return Math.abs(ageDate.getUTCFullYear() - 1970);
    }
}

class Employee extends Person {
    constructor(firstname, lastname, birthday, position) {
        super(firstname, lastname, birthday);
        this.position = position;
    }
    get yearsUntilRetirement() {
        return 67 - super.age;
    }
}

// Creamos un objeto
var p = new Employee('Pablo', 'Almunia', '07-08-1966', 'Architect');

// Consultamos sus propiedades
console.log(p.firstname, p.lastname, p.position);
console.log(p.yearsUntilRetirement);
```

Constructor por defecto

Si la clase hija no dispone de constructor, se incluye un constructor por defecto que llama al constructor de la clase padre con los parámetros que se hayan pasado al invocar a *new*:

```
Constructor por defecto
"use strict";

class Person {
    constructor(firstname, lastname, birthday) {
        this.firstname = firstname;
        this.lastname = lastname;
        this.birthday = new Date(birthday);
    }
    get age() {
        var ageDifMs = Date.now() - this.birthday.getTime();
        var ageDate = new Date(ageDifMs);
        return Math.abs(ageDate.getUTCFullYear() - 1970);
    }
}

class Unemployed extends Person {
    get yearsUntilRetirement() {
        return 67 - super.age;
    }
}

// Creamos un objeto
var p = new Unemployed('Pablo', 'Almunia', '07-08-1966');

// Consultamos sus propiedades
console.log(p.firstname, p.lastname, p.position);
console.log(p.yearsUntilRetirement);
```