# Parte 3
# Aproximación a la POO en JavaScript

## Capítulo 3-1
## Enfoque orientado a "objetos" en JavaScript

### 1. Introducción

Aunque la implementación del modelo de programación orientada a objetos (POO) no esté tan completa en JavaScript como en C++ o Java, JavaScript ofrece los mecanismos principales gestionados por estos lenguajes.

Recordemos los conceptos más importantes de la POO:

- Encapsulación: reunión de un conjunto de propiedades (parte de tratamiento de datos) y de funciones, también llamadas métodos (parte de procesamientos), dentro de un objeto tipo (quizás es más correcto hablar de clase), con la posibilidad de crear (instanciar) objetos a partir de esta clase.
- Herencia: posibilidad de "fabricar" una nueva clase a partir de una clase existente; esta nueva clase hereda las propiedades y métodos de la clase padre (se pueden añadir nuevas propiedades/métodos a la nueva clase).
- Polimorfismo: un método del mismo nombre asociado a varias clases puede tener comportamientos diferentes para algunas de estas clases.

## 2. Programación orientada a objetos a través de ejemplos

JavaScript siempre ha sido una pieza esencial en los desarrollos web, principalmente para la programación del lado "cliente", es decir, del lado del navegador. Habitualmente, sin sumergirse de lleno en el lenguaje, los desarrolladores producen código JavaScript de calidad mediocre, contentándose con adaptar el código fuente recuperado de sitios web y manipulando los conceptos POO lo menos posible.

En paralelo, han aparecido un gran número de librerías JavaScript y su uso permite producir aplicaciones de mejor calidad. El dominio de estas librerías supone tener conocimientos básicos de POO en JavaScript.

Por tanto, el objetivo de la exposición que sigue es presentarle lo que hay que saber sobre este asunto. Los conceptos se van a explicar a través de una serie de ejemplos.

Algunos lectores que ya tengan una importante experiencia en otros lenguajes POO (PHP 5, Java, C++...) al principio se pueden sentir incómodos con los aspectos específicos de la POO en JavaScript, la POO por prototipado.

### 2.1 Secuencia 1: Declaración de los objetos JavaScript de manera "Inline"

Se trata de la manera más sencilla de declarar un objeto en JavaScript.

```
/* Declaración inline de un objeto JavaScript */
/* NB: Esta técnica no permite la herencia a partir del objeto
más adelante */
var Adicion = {
    x: 5,
    y: 10,
    calculo: function()
    {
        return this.x + this.y;
    }
};

/* Uso del objeto Adicion */
document.write("Suma: " + Adicion.calculo());
```

El resultado obtenido de la ejecución será el siguiente:

```
Suma = 15
```

En este tipo de declaración de objeto, es posible prever la especificación de atributos (propiedades) y también de métodos.

La palabra clave `this` sirve para indicar que se está haciendo referencia a los atributos del objeto en sí mismo.

Está claro que con este tipo de declaración no será posible reutilizar este tipo de definición para crear un objeto de las mismas características (o parecidas). Por tanto, este método se utilizará poco (o nada) porque no permite la herencia. No se preocupe, porque volveremos sobre esto más en detalle, a lo largo de esta exposición.

## 2.2 Secuencia 2: Creación de objetos JavaScript con un constructor

También es posible crear nuestros objetos JavaScript con un constructor (concepto bien conocido en los lenguajes POO). En JavaScript, será suficiente con escribir una función y llamarla posteriormente con la palabra clave `new`. Por tanto, la función jugará el papel de clase sin serlo realmente.

Veamos con un ejemplo el desarrollo que es necesario seguir:

```
/* Definición de una función constructor, de nombre Coche */
var Coche = function()
{
    /* Atributo(s) del objeto */
    this.tieneMotor = true;
    /* Método(s) del objeto */
    this.avanzar = function()
    {
        document.write("avanza");
    }
}

/* Instanciación de un objeto simca1100 a través del constructor Coche */
var simca1100 = new Coche();

/* Visualización del atributo tieneMotor del objeto simca1100 */
```

```
if (simca1100.tieneMotor)
{
    document.write("El coche simca1100 tiene un motor<br />");
}
else
{
    document.write("El coche simca1100 no tiene motor !<br />");
}

/* Llamada al método avanzar del objeto simca1100 */
document.write("El coche simca1100 ");
simca1100.avanzar();
```

En nuestro ejemplo, se define en primer lugar una función `Coche`. Integra un atributo booleano que indica que los coches tienen un motor y un método (función) de nombre `avanzar`, que mostrará "`avanza`" cuando se pida, a partir de un objeto de tipo `Coche` (se entenderá rápidamente).

Posteriormente, se construye un objeto de nombre `simca1100` a partir del constructor `Coche` (siento que el término "constructor" pueda ser confuso en un ejemplo basado en coches):

```
/* Instanciación de un objeto simca1100 a través del constructor Coche */
var simca1100 = new Coche();
```

Ahora, la propiedad (atributo) `tieneMotor` se puede consultar para el objeto `simca1100` instanciado y también se puede ejecutar el método `avanzar`. Cuando se ejecute, tendremos:

```
El coche simca1100 tiene un motor
El coche simca1100 avanza
```

## 2.3 Secuencia 3: Variables privadas en una instancia de objeto

En el ejemplo de la secuencia anterior, habrá observado que las propiedades (atributos) llevan como prefijo la palabra clave `this`. Esto es lo que las hace usables desde el exterior (caso de la propiedad `tieneMotor`). Por el contrario, por necesidades locales del constructor (cálculo interno), puede declarar variables no expuestas, usando como prefijo la palabra clave `var`.

Veamos un ejemplo concreto:

```
/* Definición de una función constructor de nombre Coche */
var Coche = function()
{
    /* Variable(s) local(s) no accesible(s) desde el exterior del objeto */
    var numeroRuedas = 4;
    /* Método(s) del objeto */
    this.avanzar = function()
    {
    document.write("avanza");
    }
}

/* Instanciación de un objeto simca1100 a través del constructor Coche */
var simca1100 = new Coche();

/* Llamada al método avanzar del objeto simca1100 */
document.write("El coche simca1100 ");
simca1100.avanzar();

/* Intento de visualización de la variable local del constructor Coche */
document.write("<br />");
document.write("El coche simca1100 tiene " + simca1100.numeroRuedas +
" ruedas");
```

Se usa una variable local `numeroRuedas` en el constructor con el prefijo `var`. Esto solo es accesible, como estaba previsto, desde dentro del constructor, como se muestra en la ejecución de este script:

```
El coche simca1100 avanza
El coche simca1100 tiene undefined ruedas
```

## 2.4 Secuencia 4: Paso de argumento(s) a un constructor

En el ejemplo siguiente, vamos ver que es posible pasar uno o varios argumentos (lo habíamos visto ya para las funciones clásicas) a un constructor:

```
/* Definición de una función constructor de nombre Coche */
var Coche = function(modelo)
{
    /* Atributo(s) del objeto informado durante la instanciación */
    this.modelo = modelo;
}
```

```
/* Instanciación de un objeto simca1100 a través del constructor Coche
con paso de argumento */
var miCoche = new Coche("simca1100");

/* Visualización del atributo modelo del objeto miCoche */
document.write("Tengo un " + miCoche.modelo);
```

El argumento `modelo` está en los paréntesis que siguen al nombre del constructor:

```
var Coche = function(modelo)
```

y está disponible en el cuerpo del constructor por:

```
    this.modelo = modelo;
```

A continuación, es suficiente a nivel de la instanciación del objeto `miCoche` con pasar como argumento un valor ("`simca1100`" en nuestro caso):

```
var miCoche = new Coche("simca1100");
```

La propiedad (atributo) `modelo` del objeto se mostrará por:

```
document.write("Tengo un " + miCoche.modelo);
```

## 2.5 Secuencia 5: No compartición de los métodos por las instancias de objetos

Dado que los métodos se declaran durante la instanciación de los objetos, sus definiciones están duplicadas en memoria.

En un ejemplo pequeño, el impacto es bajo.

Por el contrario, si su aplicación manipula muchos objetos con métodos múltiples y complejos en los constructores, esto se convierte en inmanejable.

El siguiente ejemplo destaca el problema que acabamos de comentar:

```
/* Definición de una función constructor de nombre Coche */
var Coche = function()
{
    /* Atributo(s) del objeto */
    this.tieneMotor = true;
    /* Método(s) del objeto */
    this.avanzar = function()
```

```
    {
        document.write("avanza");
    }
    this.retroceder = function()
    {
        document.write("retrocede");
    }
}

/* Instanciación de un objeto simca1100 a través del constructor Coche */
var simca1100 = new Coche();

/* Instanciación de un objeto renault12 a través del constructor Coche */
var renault12 = new Coche();

/* Comprobación de la igualdad de métodos avanzar de los objetos simca1100
y renault12 */
if (simca1100.avanzar == renault.avanzar)
{
   document.write("Método avanzar compartido por los objetos simca1100 y
Renault12<br />");
}
else
{
    document.write("Método avanzar no compartido por los objetos simca1100
y Renault12<br />");
}
```

La ejecución confirma que el método `avanzar` no está factorizado:

```
Método avanzar no compartido por los objetos simca1100 y Renault12
```

La noción de prototipo que vamos a descubrir a continuación va a resolver este problema.

## 2.6 Secuencia 6: Noción de prototipo

Un prototipo es un conjunto de elementos (atributos/propiedades y métodos) que se va a asociar a un constructor (sin "almacenamiento" en el constructor en sí mismo). Durante la ejecución, cuando una propiedad de objeto solicitada en el código no se encuentre en el constructor del objeto en cuestión, se realizará una búsqueda en esta lista "adicional".

# Capítulo 3
# Entender los fundamentos de Vue.js

## 1. Instalación

### 1.1 Una versión por entorno

En este capítulo veremos que es posible instalar Vue.js de diferentes formas, dependiendo de las herramientas de las que disponga un proyecto existente o si inicia un nuevo proyecto de aplicación SPA.

En cualquiera de los casos hay dos versiones de la librería:

– una versión de desarrollo: incluye un modo de depuración que le permite mostrar advertencias en la consola e interactuar con varias herramientas de ayuda al desarrollo (extensiones del navegador);

– una versión de producción: es una versión minificada que no cuenta con el modo de depuración ni con el código de las herramientas de ayuda al desarrollo. Se utiliza para que cuando su sitio web está en línea se cargue lo más rápido posible en el navegador.

**Observación**

*Vue.js no es compatible con IE 8 ni sus versiones anteriores. Solo se puede utilizar en navegadores compatibles con ES 5 y versiones posteriores de JavaScript.*

## 1.2 Mediante descarga manual

- Para comenzar a usar Vue.js, descargue la última versión de la librería desde el sitio oficial: https://vuejs.org/v2/guide/installation.html


- Descargue la versión de desarrollo. Una vez que se haya descargado el archivo vue.js, colóquelo en el directorio de su sitio web y cárguelo en su página HTML con la etiqueta `<script>`.

```
<script src="vue.js"></script>
```

## 1.3 Mediante la inclusión de un CDN (lo más sencillo)

- Si no desea aumentar el peso de su proyecto, puede cargar este archivo desde un servidor remoto usando el siguiente CDN (*Content Delivery Network*):

```
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

Este es el método de instalación más sencillo para comenzar a usar las funciones de Vue.js.

**Observación**

*Como la librería está alojada en un servidor remoto, no podrá trabajar en su proyecto sin conexión.*

- Para una versión de producción, use el siguiente CDN para disponer de la última versión estable (v2.6.11):

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.6.11"></script>
```

Si está utilizando módulos nativos de JavaScript ES, también hay una compilación compatible a partir de la versión 2.6 de Vue.js:

```
<script type="module">
  import Vue from 'https://cdn.jsdelivr.net/npm/vue@2.6.0/dist/
vue.esm.browser.js'
</script>
```

## 1.4 Mediante npm o yarn para proyectos grandes

### 1.4.1 Descarga del paquete vue

npm (*Node Package Manager)* es el registro de librerías JavaScript más grande del mundo (más de 800 000). Cada librería de este registro se carga como un módulo CommonJS o ES, que sigue la sintaxis del módulo en el que se desarrolló.

También es un administrador de dependencias que se instala en su máquina con el framework Node.js.

Para instalar de manera optimizada varias librerías de JavaScript que vayan a utilizarse en proyectos de Vue.js de tamaño mediano, se recomienda usar un administrador de dependencias como npm o yarn (supercapa de `npm`). En este caso, para usarlas es necesario utilizar un empaquetador de módulos como Webpack o Browserify.

- Puede descargar Node.js desde la siguiente dirección:
  https://nodejs.org/es/download/
- Una vez instalado, abra el terminal de su máquina, escriba el siguiente comando y siga las instrucciones para crear un nuevo proyecto:

```
npm init
```

A continuación, se crea un archivo package.json, que servirá como un registro para que npm conozca los diferentes paquetes que utiliza su proyecto.

- Escriba el siguiente comando en la raíz de su proyecto web para descargar el paquete `vue`:

```
npm install vue
```

Este comando crea un directorio vue en `node_modules` dentro de su proyecto y añade la referencia a ese módulo en el archivo package.json.

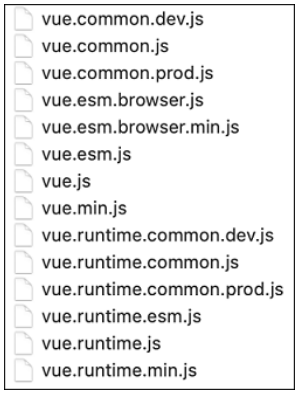
Si más adelante desea compartir su proyecto, puede compartir solo el código fuente sin el directorio `node_modules`.

Con el comando `npm install`, npm vuelve a instalar de manera automática todas las librerías que su proyecto necesita, leyendo el archivo package.json.

### 1.4.2 Explicación de las dierentes builds

Para entender las siguientes explicaciones, debe saber que la codificación de módulos JavaScript ha evolucionado mucho en los últimos años. Además, no es posible utilizarlos directamente en el navegador (excepto los módulos nativos de ES). Por este motivo es necesario utilizar un empaquetador capaz de analizar su sintaxis, para cargar estos archivos en un navegador (consulte el capítulo Nociones esenciales de JavaScript, sección Utilización de los módulos JavaScript).

Encontrará las diferentes compilaciones de la librería en el directorio `/node_modules/vue/dist`:




Aquí se resumen sus diferencias:

| | UMD | CommonJS Module | ES Module (para empaquetadores) | ES Module (para navegadores) |
|---|---|---|---|---|
| **Full** | vue.js | vue.common.js | vue.esm.js | vue.esm.browser.js |
| **Runtime-only** | vue.runtime.js | vue.runtime.common.js | vue.runtime.esm.js | - |
| **Full (production)** | vue.min.js | - | - | vue.esm.browser.min.js |
| **Runtime-only (production)** | vue.runtime.min.js | - | - | - |

A continuación se listan las definiciones de los diferentes términos.

– **Full**: este término designa las builds que contienen la parte Compilador y la parte Runtime.

– **Compiler**: se trata del código que se utiliza para compilar las cadenas de caracteres de la propiedad `template` de un componente Vue.js, dentro de una función de renderizado. Después de la compliación, el siguiente código:

```
// esto necesita un compilador
new Vue({
  template: '<div>{{ hello }}</div>'
})
```

se convertirá en:

```
// esto no lo necesita
new Vue({
  render (h) {
    return h('div', this.hello)
  }
})
```

– **Runtime**: este es el código que se encarga de crear la instancia y los componentes de Vue, manipular el DOM virtual y renderizar el DOM final, que representa a todo lo demás. Si los componentes Vue.js de la aplicación se escriben en un único archivo con la extensión .vue, la parte Compiler no es útil en la build. Esto hace necesario instalar en su proyecto el módulo vue-loader con npm o yarm, si está usando una compilación escrita con la sintaxis del módulo ES, o el módulo vueify si está usando una build escrita con la sintaxis CommonJS.

– **UMD (*Universal Module Definition*)**: es una sintaxis de módulo compatible con los módulos CommonJS y AMD. Por lo tanto, estos archivos no requieren un empaquetador y el navegador puede leerlos cuando se cargan con una etiqueta `<script>`. De forma predeterminada, la versión que se puede descargar manualmente desde el sitio oficial de Vue.js y a través de los CDN incluye las partes Compiler y Runtime.

– **CommonJS Module**: esta es la sintaxis antigua que se utiliza para escribir módulos en JavaScript, antes de la aparición de los módulos nativos de ES con la versión ES 6 de JavaScript. Estos archivos se utilizan si su proyecto usa el empaquetador de módulos Browserify o la primera versión de Webpack.

– **ES Module**: desde la versión ES 6, esta es la nueva sintaxis que se utiliza para escribir módulos de forma nativa en el lenguaje JavaScript.

  – Existe una versión para empaquetadores modernos como Webpack o Rollup. Esta versión está optimizada para que el empaquetador la pueda analizar más rápidamente, de modo que solo conserve las partes de la librería que se utilizan en su aplicación.

– La versión "browser" permite que los archivos se carguen directamente en el navegador usando la etiqueta `<script>`, con el atributo `type` con el valor `module`.

**Observación**

*La versión ES Module Browser le permite cargar la librería directamente en su página. Sin embargo, para evitar en su navegador problemas de compartición de recursos de origen cruzado (CORS-Cross-Origin Resource Sharing), es necesario utilizar un servidor local en su máquina. No podrá utilizar la librería Vue.js en una página index.html cargada en el navegador desde su explorador de archivos (URL en file:///).*

Sin la parte Compiler, la librería es un 30 % más ligera. Por lo tanto, para proyectos grandes se recomienda usar componentes de un solo archivo con la extensión .vue y un empaquetador em moderno como Webpack, Rollup o Parcel, con el módulo vue-loader.

Si no está usando vue-loader o vueify (utiliza el empaquetador Browserify más antiguo) y desea usar una compilación con la parte Compiler, será necesario crear un alias para la build adecuada en la configuración de su empaquetador.

## 1.5 Mediante Vue-CLI

Si no desea pasar uno o dos días configurando un empaquetador para su proyecto de aplicación SPA, Vue.js proporciona un CLI (*Command Line Interface*) oficial, para iniciar un proyecto de desarrollo de aplicación SPA de manera rápida y sin configuración.

Vue-CLI es un paquete npm construido sobre el empaquetador Webpack y proporciona un comando `vue` en su terminal.

Vue-CLI es un sistema completo para desarrollar sus aplicaciones Vue rápidamente y en particular ofrece:

– una interfaz gráfica para crear y administrar un proyecto Vue.js;

– la recarga en caliente (*hot reloading*), lo que le permite probar su código sin esperar la compilación de una build por Webpack, durante el desarrollo;

– una colección de complementos oficiales que integran las mejores herramientas frontend: transpiler, linter, preprocesador CSS, herramientas de prueba unitaria, etc.

Para obtener más información, puede consultar el capítulo Crear y desplegar una aplicación con Vue CLI.

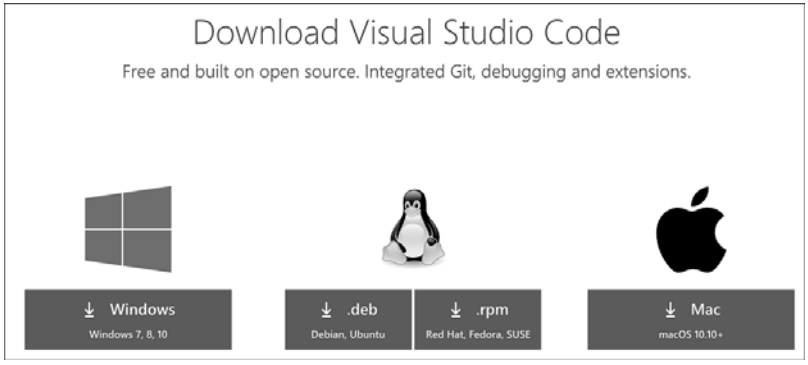
# 2. Herramientas de desarrollo

## 2.1 VS Code y sus plugins

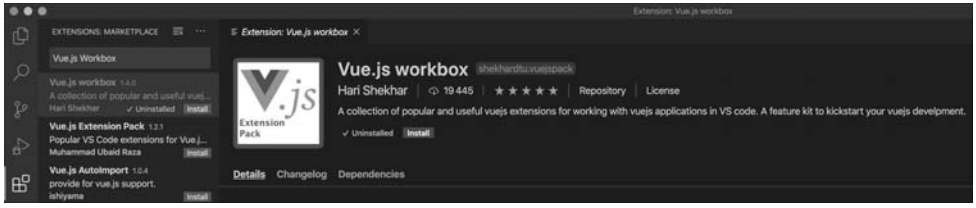
### 2.1.1 Instalar y configurar VS Code

Visual Studio Code es un editor de código gratuito, publicado por Microsoft y uno de los más utilizados en la actualidad por los desarrolladores de Vue.js. Es muy ligero y potente al mismo tiempo. También es muy extensible, gracias a sus miles de plugins.

VS Code está disponible para descargar en Windows, Mac y Linux en: https://code.visualstudio.com/Download




- Una vez instalado, ábralo.
- Pulse el botón  **Extensions**, a la izquierda de la ventana.
- Escriba "**Vue.js Workbox**":




- Ahora pulse el botón verde **Install**.