Tras mucho tiempo emulando el comportamiento de clases en ECMAScript, la versión *ES2015* incluye la nueva sintaxis para su definición.

```
class MyClass {
  constructor() {
    this.value = '';
  }
}
// Instanciamos la clase
let myClass = new MyClass();
```

Clases en ECMAScript

Es importante tener en cuenta que las clases en ECMAScript son la forma de estandarizar la creación y el manejo de objetos. **No se pretende convertir el lenguaje a programación orientada a objetos**. Las características incluidas son:

- **Herencia**: Podemos extender clases partiendo de otras.
- **Constructor**: Inicialización del objeto mediante su constructor.
- **Métodos de instancia**.
- **Métodos estáticos**: Estos métodos se llaman desde la propia clase, sin necesidad de instanciar el objeto.
- **Sobrescritura de métodos**: Las clases que heredan pueden sobrescribir métodos del padre.
- **super**: Los métodos padre que han sido sobrescritos se encuentran disponibles a través de `super`.

En el siguiente ejemplo encontraréis todas las propiedades nombradas:

```
// Definimos la clase MyString
class MyString {
  // Cada vez que instanciemos un objeto se llamará a este método.
  // La cadena se la pasamos al instanciarlo
  constructor(str) {
    // Esta es una propiedad privada. Los métodos de instancia pueden acceder
    // a ella.
    this.str = str;
  }

  // Este es un método de instancia
  getString() {
    return this.str;
  }

  // Este es un método estático
  static helloString(str) {
    return 'Hello ' + str;
```

```
  }
}

// Esta clase hereda de MyString, es decir, por defecto define todos sus métodos.
class MyUppercaseString extends MyString {
  // Sobrescribimos el método de instancia
  getString() {
    // Primero obtenemos la String a partir del método padre y luego
    // transformamos el texto
    return super.getString().toUpperCase();
  }

  // Sobrescribimos el método estático
  static helloString(str) {
    // Del mismo modo obtenemos el resultad del padre, pero esta vez
    // le pasamos el parámetro del método. Si no incluimos el parámetro
    // este no se pasa al método padre.
    return super.helloString(str).toUpperCase();
  }
}

// Instanciamos los objetos
let myString = new MyString('test');
let myUppecaseString = new MyUppercaseString('test');

// Comprobamos su valor
console.log(myString.getString()) // test
console.log(myUppecaseString.getString()) // TEST

// Ahora probemos los métodos estáticos
console.log(MyString.helloString('world')) // Hello world
console.log(MyUppercaseString.helloString('world')) // HELLO WORD
```

Código en Babel REPL.

Scope en clases

Al igual que con las funciones, **las clases de ECMAScript también trabajan de manera peculiar con el scope de `this`**. Veámoslo con un ejemplo:

```
class Test {
  constructor() {
    this.test = 0;
  }

  myMethod() {
    return this.test;
  }
}

const exec = method => {
```

```
  return method();
}

let test = new Test();
console.log(exec(test.myMethod)); // Cannot read property 'test' of undefined
```

Código en Babel REPL.

Al pasar `myMethod` como parámetro a una función el scope de `this` cambia al de la función en sí. **Esto provoca el error del ejemplo, ya que en el scope de la función exec la propiedad this no está definida**. Una posible manera de solventar este problema es utilizar el método `bind` . Este método retorna una nueva función en la que `this` se asigna al valor que recibe como parámetro:

```
class Test {
  constructor() {
    this.test = 0;
    this.myMethod = this.myMethod.bind(this);
  }

  myMethod() {
    return this.test;
  }
}

const exec = method => {
  return method();
}

let test = new Test();
console.log(exec(test.myMethod)); // 0
```

Código en Babel REPL.

Otra posible solución que no requiere redefinir la función es **utilizar la nueva notación de funciones con el operador** `=>` . El inconveniente de utilizar este método es que aún es una propuesta que se encuentra en el `stage-2` .

> Lo más importante de este operador es que asigna el valor de la variable `this` al contexto actual.

```
class Test {
  constructor() {
    this.test = 0;
  }

  myMethod = () => {
    return this.test;
  }
}
```

```
const exec = method => {
  return method();
}

let test = new Test();
console.log(exec(test.myMethod)); // 0
```

Código en Babel REPL;