JAVASCRIPT ES6 | CLASSES

Objetivos de la clase

- ✓ Comprender la diferencia en la utilización de clases entre lenguajes de programación orientados a objetos y los basados en prototipos.
- ✓ Entender la utilidad que nos proporcionan las clases en ES6 y entender las diferencias entre esta sintaxis y la de ES5.
- ✓ Aprender a declarar clases, utilizar herencia, constructores, getters y setters.

¿Qué es una clase?

En la programación orientada a objetos:

Se utiliza la abstracción para crear modelos basados en el mundo real a través de clases. *Las clases serían entonces, "los moldes" a partir de los cuales podemos crear objetos*.
Así, la clase define las características que tendrá el objeto (propiedades y métodos), el cual llamamos instancia de esta clase.
Ej: Java, Python, C++.

¿Qué es una clase?

En la programación basada en prototipos:

Las clases no están presentes y la reutilización de comportamiento (conocido como herencia en lenguajes basados en clases) se lleva a cabo a través de un proceso de "decoración" de objetos existentes que sirven de prototipos.
Ej: JavaScript, Cecil, NewtonScript.

- ✓ En JavaScript, antes de ES6, han existido muchos patrones y librerías para emular el desarrollo clásico orientado a clases. Ej: JSClass, Classify.js.
- ✓ Muchos desarrolladores tradicionales que provenían de lenguajes, como C++, Java o Python, se sentían confundidos porque las **clases y la herencia** en JavaScript **no son el método principal** para crear objetos similares o relacionados.
- ✓ El patrón de clase fue ampliamente debatido y discutido a través del proceso de diseño de ES6.

Las **clases** de JavaScript fueron introducidas en ES6 como una **mejora sintáctica sobre la herencia basada en prototipos**.
No es más que "*syntactical sugar*" sobre objetos y prototipos.
Es decir, **no introduce un nuevo modelo de herencia orientada a objetos, sino que provee una sintaxis mucho más clara y simple**, pero **la lógica detrás de la herencia / prototipos sigue siendo la misma**.

La keyword *class* en ES6 admite:

- ✓ herencia basada en prototipos
- ✓ constructores
- ✓ llamadas super()
- ✓ instancias
- ✓ métodos estáticos

Pregunta

¿ Introduce ES6 un nuevo modelo de herencia orientado a objetos en JS?

Con ES5 podíamos crear "clases" así:

```
var BankAccount =
    function(clientName, currency) {
        this.clientName = clientName;
        this.currency = currency;
        this.balance = 0.0;
    };

BankAccount.prototype.showBalance = function() {
    return this.currency + " " + this.balance;
};

BankAccount.prototype.withdrawMoney = function(amount) {
    if (amount <= this.balance) {
        this.balance -= amount;
    } else {
        throw new Error('not enough funds');
    }
};

BankAccount.prototype.depositMoney = function(amount) {
    this.balance += amount
};

var account1 = new BankAccount('mike',
    '$');
account1.depositMoney(100);
account1.withdrawMoney(25);
account1.showBalance();
// $ 75
```

Ahora podemos crear "clases" así:

```
class BankAccount {
    constructor (clientName, currency) {
        this.clientName = clientName;
        this.currency = currency;
        this.balance = 0.0;
    }

    showBalance() {
        return `${this.currency} ${this.balance}`;
    }

    withdrawMoney(amount) {
        if (amount <= this.balance) {
            this.balance -= amount;
        } else {
            throw new Error('not enough funds');
        }
    }

    depositMoney(amount) {
        this.balance += amount
    }
}

let account1 = new BankAccount('mike',
    '$');
account1.depositMoney(100);
account1.withdrawMoney(25);
account1.showBalance();
// $ 75
```

Definiendo una clase en ES6

Al igual que en el caso de las funciones, hay dos formas de definir clases en ES6:

Declaración de clases:

Para declarar una clase, se utiliza la palabra *class* y un nombre para la clase.

```
class Car {
    constructor(brand) {
        this.brand = brand;
    }
}
```

Primero necesitas declarar la clase y luego acceder a ella, de otro modo dará error. (hoisting)

Expresiones de clases:

Pueden tener nombre o ser anónimas.

```
// Anónima
var Poligono = class {
    constructor(height, width) {
        this.height = height;
        this.width = width;
    }
};

// Nombrada
var Poligono = class Poligono {
    constructor(height, width) {
        this.height = height;
        this.width = width;
    }
};
```

Las **expresiones de clase** también se pueden usar como argumentos pasados a funciones.

Por ejemplo:

```
function carFactory(car) {
    return new car();
}
const Toyota  = carFactory(class {
    start() {
        console.log("Your car is ready to roll");
    }
    stop() {
        console.log("Shutting down the engines");
    }
});
Toyota.start(); // Your car is ready to roll
Toyota.stop(); // Shutting down the engines
```

La función "factory" toma la definición de clase como argumento y devuelve el objeto de la clase pasada como argumento.

Pregunta

¿Qué keyword utilizamos para declarar una clase en ES6?

¿Cuáles son las dos formas de definir una clase en ES6?

Constructor

- ✓ es un método especial, el cual es llamado en el momento de la creación de la instancia (cuando se crea el objeto).
- ✓ es un método de la clase.
- ✓ se usa para establecer las propiedades del objeto o para llamar a los métodos para preparar el objeto para su uso.
- ✓ solo puede haber un "constructor" por clase.
- ✓ para llamar al constructor de la clase padre, utilizamos la keyword *super().*

Herencia

Antes de ES6:

Podíamos heredar de otra clase usando la keyword *call*, pasando el contexto del objeto y los atributos que queríamos a otro objeto.

Ya sea que tengas experiencia en otro lenguaje de programación orientado a objetos o no, esta sintaxis es un poco confusa.

```
var Product =
    function(name, price) {
        this.name = name;
        this.price = price;
    };

Product.prototype.nameAndPrice = function() {
    console.log(
        "The product's name is: " + this.name,
        "and the product's price is: " +
        this.price
    );
};

var Electronic = function(name, price, brand) {
    Product.call(this, name, price);
    this.brand = brand;
};

Electronic.prototype = Object.create(Product.prototype);
Electronic.prototype.constructor = Electronic;

// creating a new electronic product
var newElectroProd = new Electronic('speaker', 100,
    "Sony");
newElectroProd.nameAndPrice();

// The product's name is: speaker and the product's price is: 100
```

Herencia

Con ES6:

extends es una keyword que nos permite la creación de una clase como hija de otra clase. La herencia nos permite incorporar el estado y el comportamiento de otra clase. Extender una clase desde su clase padre evita duplicar código.

Si hay un constructor presente en la subclase, primero debes llamar a *super()* antes de usar la keyword *this*.

```
class Product {
    constructor(name, price) {
        this.name = name;
        this.price = price;
    }

    nameAndPrice() {
        console.log(
            "The product's name is: " + this.name,
            "and the product's price is: " +
            this.price
        );
    }
}

class Electronic extends Product {
    constructor(name, price, brand) {
        super(name, price);
        this.brand = brand;
    }
}

let banana = new Product("Banana", 2);
banana.nameAndPrice();

let mac = new Electronic("Mac", 800,
    "Apple");
mac.nameAndPrice();

// The product's name is: Banana and the product's price is: 2
// The product's name is: Mac and the product's price is: 800
```

Pregunta

¿Qué es el constructor en una clase de ES6?

¿Para qué lo utilizamos?

¿Podemos tener más de uno en la misma clase?

Getters y Setters

- ✓ Getter: función que obtiene un valor de una propiedad.
- ✓ Setter: función que establece el valor de una propiedad.

Esto nos facilita establecer u obtener el valor de una propiedad. Se los conoce como *accessor properties.*

Getters y Setters ES5

```
var person = {
    firstName: 'Jimmy',
    lastName: 'Smith',
    get fullName() {
        return this.firstName + ' ' + this.lastName;
    },
    set fullName (name) {
        var words = name.toString().split(' ');
        this.firstName = words[0] || '';
        this.lastName = words[1] || '';
    }
}

person.fullName = 'Jack Franklin';
console.log(person.firstName); // Jack
console.log(person.lastName) // Franklin
```

Para crearlos usamos las keywords *get* y *set*.

Getters y Setters ES6

```
class AeroPlane {
    constructor(model, capacity) {
        this._model = model;
        this._capacity = capacity;
    }
    get model() {
        return this._model;
    }
    get capacity() {
        return this._capacity;
    }
    set model(model) {
        this._model = model;
    }
    set capacity(capacity) {
        this._capacity = capacity;
    }
}
const jet = new AeroPlane("Jet", 100);

console.log(jet.capacity);
//100
console.log(jet.model);
//Jet
jet.model = "avion";
console.log(jet.model);
//avion
```

Para crearlos usamos las keywords *get* y *set*.

Los **métodos** de la clase padre pueden ser **sobrescritos** utilizando el mismo nombre en la clase hija.

```
class AeroPlane {
    constructor(capacity) {
        this.capacity = capacity;
    }
    showCapacity() {
        console.log(`Capacity of this plane: ${this.capacity}`);
    }
    fly() {
        console.log("Engines on, and the plane will take off soon");
    }
}
class FighterPlane extends AeroPlane {
    fly() {
        console.log("Engines on, and the plane is gone");
    }
    fire() {
        console.log("Loading weapons and firing");
    }
}
const phantom = new FighterPlane(2);

phantom.fly(); // Engines on, and the plane is gone
phantom.showCapacity(); // Capacity of this plane: 2
```

Además de los métodos, también podemos **sobrescribir constructores.**

```
class AeroPlane {
    constructor(capacity, color) {
        this.capacity = capacity;
        this.color = color;
    }
}
class FighterPlane extends AeroPlane {
    constructor(color) {
        // This fighterplane is 2-seater
        super(2, color);
    }
}
const phantom = new FighterPlane("grey");

console.log(phantom.capacity);      // 2
```

Actividad

En el siguiente ejemplo, la propiedad *capacity* y el método *showCapacity* se repiten en ambas clases. Simplificar estas clases, utilizando herencia.

```
class AeroPlane {
    constructor(capacity) {
        this.capacity = capacity;
    }
    showCapacity() {
        console.log(`Capacity of this plane: ${this.capacity}`);
    }
}
class FighterPlane {
    constructor(capacity) {
        this.capacity = capacity;
    }
    showCapacity() {
        console.log(`Capacity of this plane: ${this.capacity}`);
    }
    fire() {
        console.log("Loading weapons and firing");
    }
}
```

Resolución actividad

```
class AeroPlane {
    constructor(capacity) {
        this.capacity = capacity;
    }
    showCapacity() {
        console.log(`Capacity of this plane: ${this.capacity}`);
    }
}
class FighterPlane extends AeroPlane {
    fire() {
        console.log("Loading weapons and firing");
    }
}
const phantom = new FighterPlane(2);
phantom.showCapacity(); // Capacity of this plane: 2
```

Recursos adicionales

- ✓ You Don't Know JS: ES6 & Beyond – Kyle Simpson – Chapter 3 – Classes:

https://github.com/getify/You-Dont-Know-JS/blob/master/es6%20%26%20beyond/ch3.md#classes

- ✓ MDN:

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Classes

Resumiendo: Las clases en ES6 nos permiten…

- ✓ imitar una sintaxis similar a la usada en otros lenguajes basados en clases.
- ✓ simplificar la herencia utilizando la keyword *extends*.
- ✓ declarar objetos de una manera mucho más clara.