6. POO

C. Clases en JavaScript Moderno

1. Class

- Con JavaScript ES5 aparece la palabra reservada **class**
- Esta class siempre va a llevar asociado un método **constructor**, que será el que defina e inicialice los atributos.
- Igual que antes, es conveniente utilizar valores por defecto para que nos sirva para crear objetos con o sin parámetros.

```
class Punto {
    constructor(ejex = 0, ejey = 0) {
        // constructor
        this.x = ejex;
        this.y = ejey;
    }
```

```
var punto2=new Punto();
var punto1=new Punto(2,4);
```

1. Class

- Los métodos ahora ya no necesitan ni this ni function, tan solo con el nombre es suficiente (y los parámetros, por supuesto).

```
mostrar () {
        document.write("(");
        document.write(this.x);
        document.write(",");
        document.write(this.y);
        document.write(")");
}
```

2. Getters y Setters

- Los get y los set de Javascript se denominan técnicamente accessors o también "Object Accessors".
- Otra manera de denominar este tipo de miembros de objetos es con el nombre de propiedades computadas o "computed properties «
- Los get y los set son útiles básicamente para realizar el acceso a propiedades de los objetos que son resultado de computar otras propiedades existentes.

```
get valorx() {
    return this.x;
}

set valorx(parametro) {
    this.x=parametro;
}

get valory() {
    return this.y;
}

set valory(parametro) {
    this.y=parametro;
}
```

2. Getters y Setters

- Un ejemplo más claro de getters en Javascript lo tenemos con el siguiente objeto persona. Este objeto podría tener un par de propiedades como el "nombre" y los "apellidos". Además podríamos desear conocer el nombre completo, pero realmente éste no sería realmente una propiedad nueva del objeto, sino el resultado de un cómputo de concatenación entre el nombre y los apellidos que tenemos en las dos propiedades anteriores. Los gets no pueden recibir parámetros y se usan como propiedades

- Ejemplo de set (Setters) en este caso lo que vamos a hacer es recibir un parámetro con aquello que se tenga que asignar y realizamos los cambios en las propiedades de los objetos que sea pertinente. Se invoca como una asignación.

```
class  Persona = {
   constructor(nombre,apellidos){
       this.nombre=nombre;
       this.apellidos=apellidos;
       this.tipo= " ";
   }
   get nombreCompleto() {
     return this.nombre + ' ' +this.apellidos;
   }
   set edad(años){
     this.tipo= "Adulto";
     if (años<18){
       this.tipo= "Adolescente";
     }
   }
 }
// uso
let persona = new Persona("Jose" , "García")
console.log(persona.nombreCompleto);
persona.edad=14; // asigna a tipo Adolescente
```

3. toString

- En lugar del método mostrar, se suele implementar un método llamado **toString** que devolverá los valores del objeto que queremos que se representen.

```
toString () {
    var salida="(";
    salida=salida+this.valorx;
    salida=salida+",";
    salida=salida+this.valory;
    salida=salida+")";
    return salida;
}
```

4. Herencia

- Podemos hacer otra clase de herede de la clase padre
- Para ello usaremos en la definición la palabra reservada **extends** y el nombre de la clase padre.
- En este caso el constructor debe incluir una sentencia super(); indicando que estamos utilizando el constructor del padre.

```
class Vector extends Punto {
    constructor() { //
        super(); //utiliza el constructor del padre


    }
```

4. Herencia

- En el caso de clases hijas, podemos definir también una serie de métodos que estarán disponibles para objetos de esta clase

```
class Vector extends Punto {
    constructor() { //
        super(); //utiliza el constructor del padre

    }

    sumaVector(a,b) {   // a es un objeto de tipo punto, b es otro objeto de tipo punto
        this.valorx=(a.valorx+b.valorx);
        this.valory=(a.valory+b.valory);

    }
}
```

5. Métodos privados

- Que queremos que un **método** sea **privado**, es decir, solo pueda usarse dentro de la propia clase, pero fuera no (incluso en la herencia se pierde), debemos usar el modificador **static** antes del nombre del método.

```
static sumaVector(a,b) {
    this.valorx=(a.valorx+b.valorx);
    this.valory=(a.valory+b.valory);

}
```

- No se invoca a través de un objeto de la clase donde está definido, si no, desde la propia clase. Si en la clase Vector tenemos este método sumaVector, lo usamos como **Vector.sumaVector(a,b)**