TEMA 7

CLASES

INTRODUCCIÓN

Las clases de javascript son una mejora sintáctica sobre la herencia basada en prototipos. La sintaxis de las clases no introduce un nuevo modelo de herencia orientada a objetos en JavaScript. Las clases proveen una sintaxis mucho más clara y simple para crear objetos y lidiar con la herencia. Podemos decir que es una forma de organizar código de forma entendible con el objetivo de simplificar el funcionamiento de nuestro programa.

1. LA SINTAXIS CLASS

La sintaxis básica para declarar una clase es la siguiente:

```
class MyClass {
    constructor() {…}
     method1() { …}
method2() { …}
     method3() { …}
      ….
}
```

Una vez que tenemos una clase creada, llamamos instanciar una clase o crear una instancia a la acción de crear un nuevo objeto basado en una clase particular. Esta acción la realizamos a través de la palabra new, seguida del nombre de la clase, la cual puede tener parámetros, en cuyo caso se controlarían desde un constructor.

En JavaScript, para instanciar una clase, se usa una sintaxis muy similar a otros lenguajes, como Java. Es tan sencillo como escribir lo siguiente:

```
class Animal   {} //Una vez declarada una clase (aún vacía)
 const pato = new Animal();
```

Algo importante que tenemos que saber es que la declaración de clases no son alojadas, es decir, no podemos declarar la clase antes de generar un objeto asociado a esa clase, algo que sí podemos hacer con las funciones. El siguiente código nos generaría un error de referencia, es decir, todavía no se sabe que es la clase correspondiente.

```
const p = new Rectangle();
class Rectangle  { }
```

1.1 EXPRESIONES DE CLASES

Una expresión de clase es otra forma de definir una clase. Las expresiones de clase pueden ser nombradas o anónimas. El nombre dado a la expresión de clase nombrada es local dentro del cuerpo de la misma. La expresión asociada una clase anónima será la siguiente.

```
let Rectangulo = class {
    constructor (alto, ancho) {
        this.alto = alto;
        this.ancho = ancho;
    }
};
```

Ahora para acceder a las propiedades del objeto lo haremos añadiendo un punto con el nombre que será **Rectangulo.alto**.

```
let Rectangulo = class  Rectangulo2{
    constructor (alto, ancho) {
        this.alto = alto;
        this.ancho = ancho;
  }
};
```

1.2 CUERPO DE CLASE Y DEFINICIÓN DE MÉTODOS

El contenido de una clase es la parte que se encuentra entre llaves {}. Este es el lugar donde se definen los miembros de clase, los métodos o constructores.

Los **constructores** aunque son funciones, no se usan igual que éstas. El método constructor se utiliza para crear e inicializar un objeto creado con una clase. Es importante recordar que en JavaScript solo puede haber un método especial con el nombre "**constructor**" en una clase. Si definimos dos, se generará un error.

```
class Persona {
      constructor(nombre, apellido, edad){
        this.nombre = nombre
        this.apellido = apellido
        this.edad = edad
}
}
```

En el ejemplo anterior, definimos la clase Persona. Cuando queremos crear un objeto asociado a la clase Persona, podemos pasar hasta tres parámetros, pero si no pasamos ninguno o pasamos más, solo se tendrán en cuenta los tres primeros.

Como en otros lenguajes de programación, se recomienda respetar los estándares y escribir el nombre de la clase con su primera letra mayúscula. Para asignar los valores pasados como parámetros se usa la palabra clave this, que hace referencia al objeto que estamos creando. Se pueden asignar propiedades que no haya en los parámetros, pero siempre usando this para referenciar al objeto. Así por ejemplo, si usamos la siguiente instrucción:

```
this.datos = `${nombre} ${apellido} ${edad}`
```

de esta forma, tras esta instrucción el objeto Persona tendrá un nuevo atributo denominado datos.

Los objetos pueden tener funciones asociadas a él, en estos casos se les denomina métodos. Estos métodos se tienen que declarar dentro de la clase, pero fuera del constructor, y no se puede usar el formato flechan En este caso no necesitamos indicar que es una function, solo escribir el nombre de la función y los paréntesis con o sin parámetros, según sea el caso. Veamos la función como se define:

```
saludar (){
    return `Hola, me llamo ${this.nombre} y tengo ${this.edad} años`;
}
```

Ejercicio clases: Crea una clase libro donde tendremos cuatro atributos: título, autor, género y año de publicación. Crear un método que devuelva toda la información del libro. Pide tres libros y guárdalos en un array, mediante la sentencia prompt.

Comprobaremos que los campos no se guardan vacíos, y además los géneros aceptados serán "aventuras", "terror" o "policiacos". Además, el año debe tener 4 dígitos, obligatoriamente.

Crear una función que nos muestre todos los libros que tenemos guardados. Crear una función que muestre los autores ordenados alfabéticamente.

Crea una función que pida un género y nos muestre la información de los libros que pertenezcan a este género, usando el método que devuelve la información definido en la clase.

1.3 GETTERS/SETTERS

Los métodos get y set se usan para asignar y extraer atributos de un objeto. Es importante tener en cuenta que el nombre de los getters/setters no pueden ser el mismo que la propiedad porque se produciría un bucle, ya que al acceder a la propiedad invocaríamos al método que a su ve accede a la propiedad que invoca al método. En muchas ocasiones los desarrolladores usan el guion bajo para nombrar la propiedad.

```
class Telefono{
      constructor(marca){
         this.marca = marca
}
get obtenMarca(){
   return  this.marca;
}
set fijarMarca(marca){
   this.marca=marca;
}
```

Aunque estos métodos son funciones, JavaScript los considera atributos, así que para utilizarlos debemos hacerlo sin los paréntesis. Veamos de qué forma, una vez instanciado el objeto.

```
let miTelefono = new Telefono("Xiaomi");
let marcaTelefono = miTelefono.obtenMarca;
miTelefono.fijarMarca = "Apple";
```

1.4 METODOS ESTÁTICOS

Como ya hemos visto con la clase Math anteriormente, en javascript también podemos utilizar y declarar métodos estáticos. Recordar que los métodos estáticos no necesitan instanciar una clase para ser utilizados. Cuando queremos indicar que un método sea estático, lo que hacemos es utilizar en su declaración la palabra clave static. Veamos un ejemplo, con la clase rectángulo:

```
class Rectangulo{
      constructor(base, altura){
```

```
            this.base = base;
            this.altura = altura;
    }
     static area ( x,y){
         return   x*y;
    }
     static perimetro ( x,y){
         return   2* x + 2*y;
    }
    }
```

Cuando nosotros queremos utilizar el método área o perímetro bastará escribir lo siguiente:

```
let arearect = Rectangulo.area(3,5);
let perimetrorect = Rectangulo.perimetro(3,5);
```

2. HERENCIA DE CLASES

En JavaScript igual que en todos los lenguajes de programación una vez definida una clase, podemos crear una nueva clase que herede de ella. Para ello utilizaremos la palabra clave extends. Veamos cómo utilizarlo en un ejemplo sobre la clase inicial teléfono

```
class Teléfono{
    constructor(marca){
        this.marca = marca
    }
    }
```

Si queremos generar una nueva clase modelo, donde guardemos la información del modelo de nuestro teléfono, lo que haremos será escribir los siguiente:

```
class Modelo extends Teléfono{
    constructor(marca,modelo){
        super(marca)
        this.modelo=modelo
    }
    }
```

Como vemos en el constructor debemos llamar al constructor de la superclase y para ello utilizaremos la palabra clave super, tal como vemos en el ejemplo anterior. Notar que el número de parámetros que debemos pasar en la función super es el mismo que tenga el constructor de la clase principal. Recordar que en este caso nuestras clases solo pueden tener un constructor.

3. MANEJO DE MÉTODOS EN CLASES HEREDADAS

Veamos qué tenemos que hacer para utilizar un método que hemos definido en la superclase, desde un método definido en la subclase. Vamos a modificar las clases anteriores

```
class Teléfono{
      constructor(marca){
         this.marca = marca
}
mostrarInformacion(){
   return "Ha llegado el nuevo teléfono" + this.marca;
}
}
```

En la superclase hemos definido el método mostrarInformacion() que simplemente nos devuelve la información del objeto creado. Si nosotros queremos añadir a esta información la que obtenemos en la subclase Modelo, y obtenerla desde la superclase, debemos hacer lo siguiente:

```
class Modelo extends Teléfono{

      constructor(marca,modelo){

         super(marca)

        this.modelo=modelo

}

mostrarModelo(){

     return this.mostrarInformacion()+ " y el modelo " + this.modelo;

}
```

Como vemos simplemente debemos incluir la sentencia **this.nombrefunción()** para acceder a ese método y que se ejecute.

4. CLASES EN FICHEROS EXTERNOS

Generalmente, para tener el código lo más organizado posible, las clases se suelen almacenar en ficheros individuales, de forma que cada clase que creamos, debería estar en un fichero con su mismo nombre:

```
//Animal.js
export class Animal {
   /* Contenido de la clase */
}
```

Luego, si queremos crear objetos basados en esta clase, lo habitual suele ser improtar el fichero de la clase en cuestión y crear el objeto a prtir de la clase. Algo similar al siguiente fragmento de código.

```
// index.js
import { Animal } form  "./Animal.js";
const pato = new Animal();
```

Si nuestra aplicación se complica mucho, podríamos comenzar a crear carpetas para organizar mejor aún nuestros ficheros de clases, y por ejemplo, tener la clase Animal.js dentro de una carpeta clases (o algo similar). Esto nos brindaría una mejor experiencia de desarrollo, pero el nombre de las carpetas o su organización ya dependería del desarrollador o del equipo de desarrollo.

.