cedia | TMO Training Management Office

info@cedia.org.ec
cedia.edu.ec

CURSO DE JAVASCRIPT

Corporación Ecuatoriana para el Desarrollo de la Investigación y la Academia

Ing. María Fernanda Granda J., PhD

25-11-2025

"Conectando ideas, transformando sociedades"

Módulo 3: Programación Orientada a Objetos

- Abstracción
- Modificador Static

Abstracción

- En JavaScript, una clase con métodos abstractos (conceptualmente) funciona como una "plantilla" o "contrato" que no se puede instanciar directamente, obligando a las clases que la heredan a implementar sus métodos.
- Como JavaScript no tiene clases abstractas nativas, se simulan usando funciones constructoras para establecer una estructura base que las clases hijas deben completar.

Ejemplo de uso de la Abstracción

- Programar clases para implementar. Máquinas de hacer café expreso y capuchino.
- Métodos para:
  - Seleccionar Bebida: se indica la bebida que ha seleccionado
  - Preparar bebida: indicar que está preparando la bebida. En el caso del capuchino indicar que se está realizando la espuma.
  - Servir bebida: Indicar que se está sirviendo la bebida en el vaso.
  - Hacer café: llama a los 3 métodos anteriores.

Abstracción

```
class MaquinaDeCafe {
    constructor(){
        if (new.target===MaquinaDeCafe){
            throw new Error("No se puede instanciar la clase Máquina de Café porque es Abstracta");
        }
    }
    //Método Abstracto
    seleccionarBebida(){
        throw new Error("Este método debe ser implementado");
    }
    prepararBebida(){
        console.log("Preparando la bebida ...");
    }
    servirBebida(){
        console.log("Sirviendo el café en un vaso ...");
    }
    hacerCafe(){
        this.seleccionarBebida();
        this.prepararBebida();
        this.servirBebida();
    }
}
```

Abstracción

```
class ExpressoMachine extends MaquinaDeCafe{
    seleccionarBebida(){
        console.log("Has seleccionado un expreso")
    }
}

class CappuccinoMachine extends MaquinaDeCafe {
    seleccionarBebida(){
        console.log("Has seleccionado un Cappuccino")
    }

    prepararBebida(){
        console.log("Realizando espuma")
        super.prepararBebida()
    }
}
```

Abstracción

```
const expresso = new ExpressoMachine()
const cappuccino = new CappuccinoMachine()

expresso.hacerCafe()
console.log("----------------------")
cappuccino.hacerCafe()
```

- Se puede probar tratando de instanciar la maquinaDeCafe

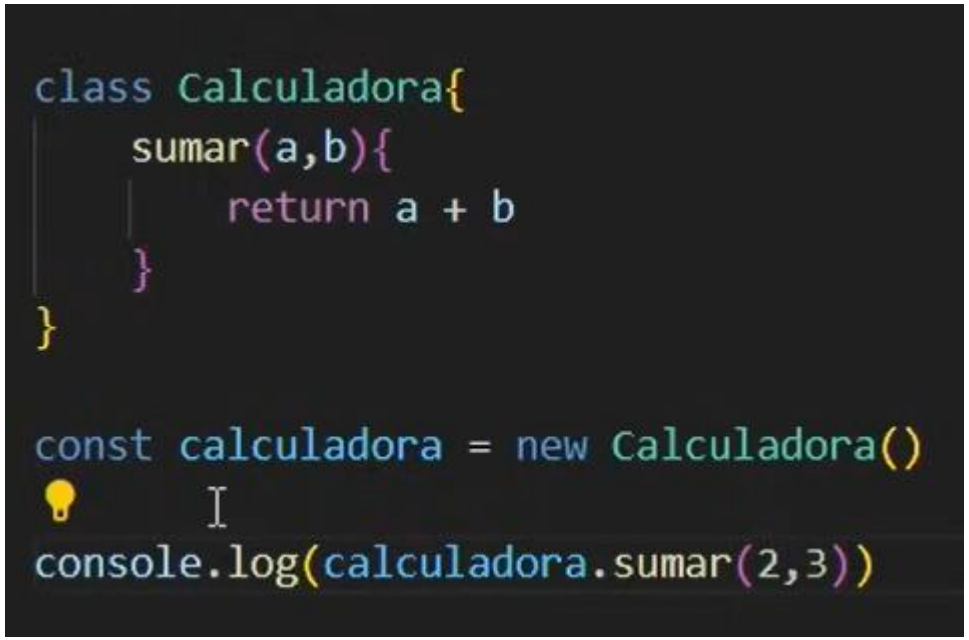
```
const maquinacafe = new MaquinaDeCafe()
```

Probar

- Qué pasa si documentamos los métodos abstractos implementados en ExpressoMachine y CapuchinoMachine?

Modificador Static

Si se quiere hacer la clase calculadora con las 4 operaciones básicas empezaríamos algo así:



Modificador Static

Si se utiliza **static** para que una propiedad y método sean de la clase y no de las instancias.

Podemos acceder directo a ellas sin instanciar.

```
class Calculadora {
    static sumar(a, b) {
        return a + b
    }
    static restar(a, b) {
        return a - b
    }
    static multiplicar(a, b) {
        return a * b
    }
    static dividir(a, b) {
        if (b == 0) {
            throw new Error("No se puede dividir por cero")
        }
        return a / b
    }
}


console.log(Calculadora.sumar(2,3))
console.log(Calculadora.restar(6,3))
console.log(Calculadora.multiplicar(2,3))
console.log(Calculadora.dividir(10,5))
```

Probar

- Subir los códigos de los dos ejemplos que se han revisado en la clase de hoy.

Características privadas (encapsular)

```
class Mascota {
    // ES2022 se agregó la posibilidad de hacer privadas las propiedas utilizando el #
    #nombre
    #especie
    #energia
    #hambre
    constructor(nombre, especie){
        this.#nombre = nombre
        this.#especie = especie
        this.#energia = 100
        this.#hambre = 0
    }
    // GETTER (obtener)
    obtenerNombre(){
        return this.#nombre
    }
    // SETTER (establecer o configurar)
    establecerNombre(nombre){
        this.#nombre = nombre
    }
}

const manchita = new Mascota("Manchita", "Perro")

 console.log(manchita.obtenerNombre())
 manchita.establecerNombre("")
 console.log(manchita.obtenerNombre())
```

Uso de métodos (encapsular)

```
// SETTER (establecer o configurar)
establecerNombre(nombre){
    if(nombre.length > 1){
        this.#nombre = nombre
    }
    console.log("El nombre debe ser mayor a 1 letra")
}

jugar(){
    if(this.#energia > 0){
        console.log(`${this.obtenerNombre()} está jugando y divirtiéndose`)
        this.#energia -= 20
        this.#hambre += 10
    } else {
        console.log(`${this.obtenerNombre()} está demasiado cansado para jugar`)
    }
}
```

Uso de métodos (encapsular)

```
jugar(){
    if(this.#energia > 0){
        console.log(`${this.obtenerNombre()} está jugando y divirtiéndose`)
        this.#energia -= 20
        this.#hambre += 10
    } else {
        console.log(`${this.obtenerNombre()} está demasiado cansado para jugar`)
    }
}

alimentar(){
    if(this.#hambre > 0){
        console.log(`${this.obtenerNombre()} está comiendo una sabrosa comida`)
        this.#hambre -= 20
        this.#energia = 10
    } else {
        console.log(`${this.obtenerNombre()} no tiene hambre`)
    }
}
```

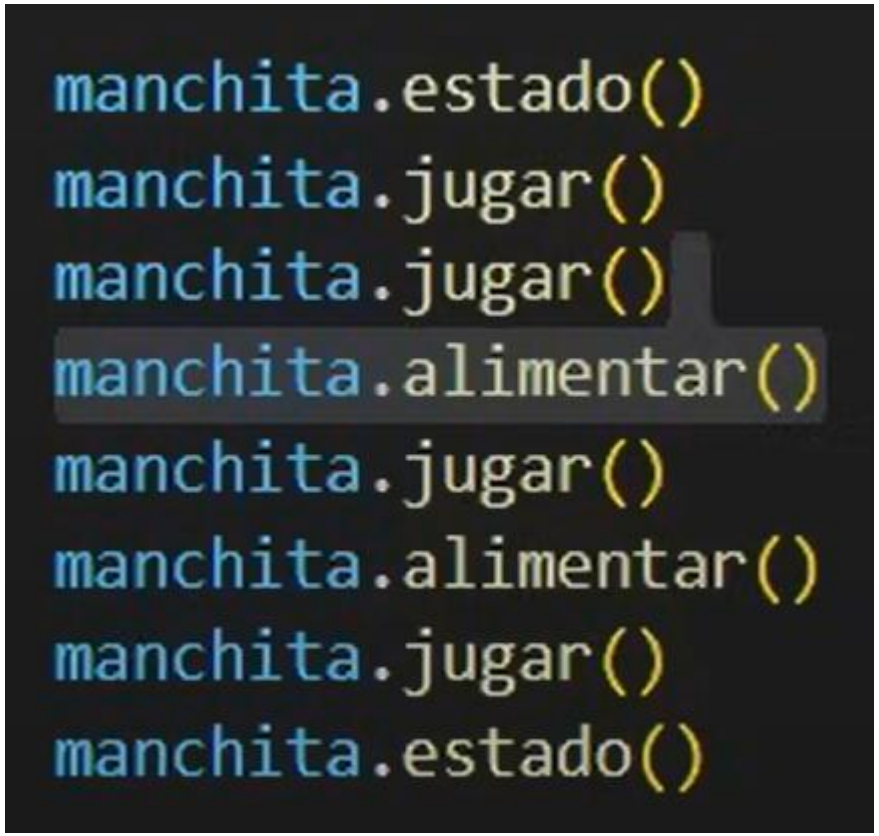
Uso de métodos (encapsular)

```
estado(){
    console.log(`${this.obtenerNombre()} es un ${this.#especie} tiene ${this.#energia}
    de energía y ${this.#hambre} de hambre.`)
}

}
```

```
const manchita = new Mascota("Manchita", "Perro")

manchita.estado()
manchita.jugar()
manchita.jugar()
manchita.alimentar()
manchita.jugar()
manchita.jugar()
manchita.estado()
```

Uso de métodos (encapsular)

- Volver a jugar



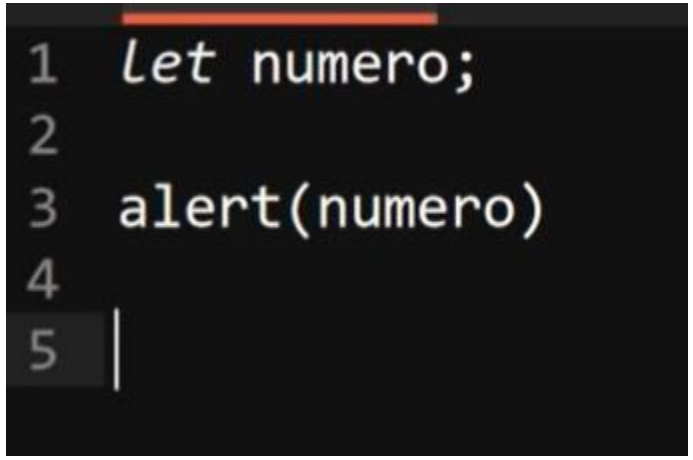
Números decimales en Javascript

- El separador de decimales es el punto.
- Se puede usar la función **toFixed** para indicar el número de decimales a utilizar.

```
let num=15.777777;
alert(num);
alert(num.toFixed(1));
alert(num.toFixed(2));
alert(num.toFixed(3));
alert(num.toFixed(4));
```

Otros tipos de datos: Undefined

- **Valor undefinido**: indica que la variable está declarada pero no ha sido inicializada.



- Vamos a trabajar declarando con let, porque de esta manera la variable tiene un alcance mắs restringuido.

Otros tipos de datos: Null

- **Valor null**: indica que la variable está declarada y con contenido vacío y por tanto se le asigna null. Es una asignación intencional.

```
let numero = null;
```

Otros tipos de datos: Not a Number (Nan)

- **Valor No es un número**: es un valor especial en JavaScript que representa un resultado de operación matemática indefinido, como la división por cero o la raíz cuadrada de un número negativo.
- Indica que una operación ha producido un resultado que no es un número válido.

```
let numero = 5;
let numero2 = "pedro";


alert(numero * numero2)
```

Otros tipos de datos: Not a Number (Nan)

- **Propagación de NaN:** Cuando NaN se utiliza en una operación, el resultado también será NaN.
- **Ejemplos de cuándo aparece NaN:**
  - **División por cero:** 1 / 0 devuelve Infinity, pero si se divide un número por cero, el resultado puede ser NaN, especialmente si se utiliza con valores negativos.
  - **Raíz cuadrada de un número negativo**: Math.sqrt(-1) devuelve NaN.
  - **Parseo de cadenas no numéricas:** parseInt("abc"), parseFloat("abc") devuelven NaN.

Otros tipos de datos: Not a Number (Nan)

- **Comprobación con isNaN():** se utiliza la función isNaN().

```
console.log(isNaN(NaN)); // true
console.log(isNaN(0)); // false
console.log(isNaN("abc")); // true
console.log(isNaN(10)); // false
```

Debugger

- Todos lo exploradores modernos y la mayoría de los otros ambientes soportan el “debugging” – una herramienta especial de UI para desarrolladores que nos permite encontrar y reparar errores más fácilmente.

Depuración

- Todos lo exploradores modernos y la mayoría de los otros ambientes soportan el “debugging” – una herramienta especial de UI para desarrolladores que nos permite encontrar y reparar errores más fácilmente.

cedia | TMO Training Management Office

GRACIAS

cedia.edu.ec
info@cedia.org.ec
(+593) 7-407-9300