EXERCISES QUE TRABAJAREMOS EN EL CUE:

- EXERCISE 1: CREAR LA CLASE USER, UN CONSTRUCTOR INICIAL, E INSTANCIACIÓN.
- EXERCISE 2: CREANDO MÉTODOS ACCESADORES (GETTERS) Y MUTADORES (SETTERS).
- EXERCISE 3: MÉTODOS PERSONALIZADOS.
- EXERCISE 4: ACCEDIENDO A LA CLASE Y SUS MÉTODOS EN OTRO SCRIPT.

El objetivo de este ejercicio es plantear una guía paso a paso para implementar una Clase en JavaScript, especificando la definición de una clase, el método constructor, métodos accesadores y mutadores, métodos personalizados, e instanciación de una clase.

EXERCISE 1: CREAR LA CLASE USER, UN CONSTRUCTOR INICIAL, E INSTANCIACIÓN.

Para comenzar, necesitaremos configurar el proyecto. Abre Visual Studio Code, navegando a un directorio a elección en tu máquina, o abriéndolo en la terminal.

```
mkdir class_example

cd class_example

npm init -y
```

Procedemos a crear un archivo llamado userClass.js, con el siguiente código:

```
// Declaración de una clase
class User {
    // Creamos el constructor
    constructor(email) {
        this._email = email

    }
}
// Instanciación de la clase
let user = new User("jose@test.com")
```

```
11
12 // Accediendo a algún atributo de la clase
13 console.log(user._email)
14
15 // Muestra el tamaño de argumentos
16 console.log(User.length)
17
18 // Nombre de la clase
19 console.log(User.name)
```

En ECMAScript 2015 (ES6), las clases se declaran de manera similar a otros lenguajes: usando la palabra "class", seguida del nombre de la clase que estamos creando.

Generalmente, los nombres de las clases, así como las variables, son sensibles a mayúsculas y minúsculas. Podemos nombrarla como deseemos, pero por regla y convención, siempre se usa la primera letra en mayúscula.

Seguidamente de la palabra class, se colocan unas llaves { }, dentro de ellas va el código de la clase. Por lo general, en los lenguajes como C++ y Java, serían típicamente los atributos y los métodos, pero en Javascript los atributos de la instancia de la clase se crean en el constructor, es decir, dentro de él.

Para instanciar objetos a partir de una clase, se utiliza la palabra "new" para crear nuevos objetos de una clase, acompañado del nombre de la clase, y todos los parámetros que invocan al constructor.

```
1 let user = new User("jose@test.com")
```

Este proceso de instanciación crea el nuevo objeto, y su inicialización es realizada por el constructor. En este caso particular, se ha creado una propiedad para la instancia user que asigna el valor de email.

Los objetos tienen propiedades que llamamos atributos, éstos se definieron en el momento de la creación, o en otro momento que podemos modificar según la vida del objeto. En nuestro caso, tenemos el atributo email del objeto user.

Para acceder a los atributos de los objetos, lo hacemos a través del operador punto ("."). Utilizamos el nombre del objeto creado, el operador punto, y el nombre del atributo al que queramos acceder.

```
// Accediendo a algún atributo email del objeto user
console.log(user._email)
```

EXERCISE 2: CREANDO MÉTODOS ACCESADORES (GETTERS) Y MUTADORES (SETTERS)

Ahora vamos a profundizar un poco en los getters y setters de Javascript, implementando un nuevo caso de uso. Además, veremos cómo funciona set, para asignar un valor "computado".

Adecuaremos nuestro script **userClass.js** de la siguiente manera:

```
class User {
    // Creamos el constructor
    constructor(id, email, firstname, lastname, age) {
        this._id = id;
        this._email = email;
        this._firstname = firstname;
        this._lastname = lastname;
        this._age = age;
    }
}
```

En JavaScript, los métodos getters se utilizan para acceder a las propiedades de un objeto. Por ejemplo, agregamos los métodos setters:

```
class User {
    // Creamos el constructor
    constructor(id, email, firstname, lastname, age) {
        this._id = id;
        this._email = email;
        this._firstname = firstname;
        this._lastname = lastname;
        this._age = age;
    }
    // Definición de los métodos Getters
    get email() {
        return this._email
    }
    get firstname() {
        return this._firstname
    }
    get lastname() {
```

```
        return this._lastname
    }
}
// Inicializamos la clase
let user = new User(1, "usuario@test.com", "José", "Perez", 25)

// Accesando a la propiedad del objeto, atributo email
console.log(user._email)

// Obteniendo el atributo de email por medio del método getter
console.log(user.email)

// Accediendo como un método
// TypeError: user.email is not a function
console.log(user.email())
```

En el programa anterior, se crea un método get email(), get firstname(), y get lastname() para acceder a la propiedad de un objeto.

```
// Definición de los métodos Getters
    get email() {
        return this._email
    }
    get firstname() {
        return this._firstname
    }
    get lastname() {
        return this._lastname
    }
```

Para crear un método getter, se utiliza la palabra reservada **get.**

Y accedemos al valor del objeto. Para esto, lo hacemos como una propiedad.

```
// Obteniendo el atributo de email por medio del método getter
console.log(user.email)
```

Cuando se intenta acceder al valor como método, se produce un error.

```
// Accediendo como un método
// TypeError: user.email is not a function
console.log(user.email())
```

En JavaScript, los métodos setter se utilizan para cambiar los valores de un objeto. Por ejemplo, cambiando el valor de **lastname** del objeto **user**:

```
class User {
    // Creamos el constructor
    constructor(id, email, firstname, lastname, age) {
        this._id = id;
        this._email = email;
        this._firstname = firstname;
        this._lastname = lastname;
        this._age = age;
    }
    // Definición de los métodos Getters
    get email() {
        return this._email
    }
    get firstname() {
        return this._firstname
    }
    get lastname() {
        return this._lastname
    }

    // Definición de Métodos Setters
    set lastname(newLastname) {
        newLastname = newLastname.trim();
        if (newLastname === '') {
            console.log("El lastname no puede ser vacío");
        }
        this._lastname = newLastname;

    }
}
// Inicializamos la clase
let user = new User(1, "usuario@test.com", "José", "Perez", 25)

// Obteniendo el atributo de lastname por medio del método getter
console.log(user.lastname)

// Cambiando el lastname por setter
user.lastname = "Sánchez"

// Verificando el cambio por el getter
console.log(user.lastname)
```

En el ejemplo anterior, el método setter se usa para cambiar el valor de un objeto, específicamente el lastname.

```
set lastname(newLastname) {
        newLastname = newLastname.trim();
        if (newLastname === '') {
            console.log("El lastname no puede ser vacío");
        }
        this._lastname = newLastname;

    }
```

Para crear un método setter, se utiliza la palabra clave **set**.

Como se muestra en el programa anterior, el valor de lastname es Pérez. Seguidamente, lo cambiamos a Sánchez.

```
// Cambiando el lastname por setter
user.lastname = "Sánchez"
```

Setter debe tener exactamente un parámetro formal.

EXERCISE 3: MÉTODOS PERSONALIZADOS

Éstos, en las clases en ES6, pueden declarar de una manera resumida, pues nos ahorramos la palabra "**function**" en la declaración de la función.

Un método es una función asociada a un objeto. En nuestro ejemplo, vamos a un método que nos devuelva todos los atributos del objeto user en un Aray[], entonces los métodos son funciones que se ejecutan solamente en un objeto que haya sido creado. Podemos tener otro método, por ejemplo: agregar saldo a una cuenta.

```
class User {
    // Creamos el constructor
    constructor(id, email, firstname, lastname, age) {
        this._id = id;
        this._email = email;
        this._firstname = firstname;
        this._lastname = lastname;
        this._age = age;
    }
    // Definición de los métodos Getters
    get email() {
        return this._email
```

```
    }
    get firstname() {
        return this._firstname
    }
    get lastname() {
        return this._lastname
    }

                    // Método personalizado
    allUserData() {
        let data = [];
        data.push(this._id);
        data.push(this._email);
        data.push(this._firstname);
        data.push(this._lastname);
        data.push(this._age);
        return data;

    }

    // Definición de Métodos Setters
    set lastname(newLastname) {
        newLastname = newLastname.trim();
        if (newLastname === '') {
            console.log("El lastname no puede ser vacío");
        }
        this._lastname = newLastname;

    }
}
// Inicializamos la clase
let user = new User(1, "usuario@test.com", "José", "Perez", 25)
// Obteniendo el atributo de email
console.log(user.email)

// Obteniendo el atributo de firstname
console.log(user.firstname)

// Obteniendo el atributo de lastname
console.log(user.lastname)

// Obteniendo toda la información por el método
let data = user.allUserData();
console.log(data)

// Cambiando el lastname del user con setters
user.lastname = "Sánchez"

```

```
// Verificando los cambios
console.log(user.lastname)
console.log(user.allUserData())
```

Al ejecutar el código anterior, tenemos como respuesta:

```
$ node userClass.js
usuario@test.com
José
Perez
[ 1, 'usuario@test.com', 'José', 'Perez', 25 ]
Sánchez
[ 1, 'usuario@test.com', 'José', 'Sánchez', 25 ]
```

Recuerda que los métodos se invocan sobre el nombre del objeto, con el operador punto, y el nombre del método, seguido por sus parámetros.

EXERCISE 4: ACCEDIENDO A LA CLASE Y SUS MÉTODOS EN OTRO SCRIPT

Para esto, debemos exportar la clase como un módulo. En este caso, el script userClass.js nos queda de la siguiente forma:

```
module.exports = class User {
    // Creamos el constructor
    constructor(id, email, firstname, lastname, age) {
        this._id = id;
        this._email = email;
        this._firstname = firstname;
        this._lastname = lastname;
        this._age = age;
    }
    // Definición de los metodos Getters
    get email() {
        return this._email
    }
    get firstname() {
        return this._firstname
    }
    get lastname() {
        return this._lastname
    }
    allUserData() {
        let data = [];
        data.push(this._id);
        data.push(this._email);
```

```
        data.push(this._firstname);
        data.push(this._lastname);
        data.push(this._age);
        //console.log("informacion del usuario: " + this._id + " " +
this._email + " " + this._firstname + " " + this._lastname + " " +
this._age)
        return data;

    }

    // Definición de Métodos Setters
    set lastname(newLastname) {
        newLastname = newLastname.trim();
        if (newLastname === '') {
            console.log("El lastname no puede ser vacio");
        }
        this._lastname = newLastname;

    }
}
```

Y el script app.js, de la siguiente manera:

```
const User = require('./userClass.js');

let user = new User(1, "usuario@test.com", "José", "Perez", 25)
// Obteniendo el atributo de email
console.log(user.email)

// Obteniendo el atributo de firstname
console.log(user.firstname)

// Obteniendo el atributo de lastname
console.log(user.lastname)

// Obteniendo toda la información por el metodo
let data = user.allUserData();
console.log(data)

// Cambiando el lastname del user con setters
user.lastname = "Sánchez"

// Verificando los cambios
console.log(user.lastname)
console.log(user.allUserData())
```