Introducción a los Getters y Setters

En el paradigma orientado a objetos, el principio de encapsulamiento es fundamental. Implica que los datos de un objeto están protegidos y que el acceso a ellos está controlado. Los «getters» y «setters» permiten manipular propiedades de un objeto de manera segura, proporcionando una interfaz para acceder y modificar los datos sin exponer directamente las propiedades internas del objeto.

Getters

Un «getter» es un método que permite acceder al valor de una propiedad de un objeto. En lugar de acceder directamente a una propiedad, se usa el getter para obtener su valor. Esto es útil cuando se necesita lógica adicional para calcular el valor o para validar el acceso.

En JavaScript, se pueden definir getters utilizando la palabra clave **get**. Aquí hay un ejemplo de cómo se puede usar un getter para acceder a una propiedad:

```
class Persona {
  constructor(nombre, apellido) {
    this._nombre = nombre;
    this._apellido = apellido;
  }


  get nombreCompleto() {
    return `${this._nombre} ${this._apellido}`;
  }
}


const persona = new Persona("Juan", "Pérez");
console.log(persona.nombreCompleto); // "Juan Pérez"
```

view rawPersona4.js hosted with by GitHub

En este ejemplo, el getter **nombreCompleto** devuelve el nombre completo de la persona, concatenando el nombre y el apellido. Los getters pueden ser útiles cuando se necesita un acceso calculado a las propiedades o se desea realizar alguna acción antes de devolver el valor.

Setters

Un «setter» es un método que permite modificar el valor de una propiedad. Similar a los getters, se usa para encapsular la lógica de modificación y para validar el valor antes de asignarlo. Los setters son útiles para controlar cómo se modifican las propiedades y para implementar lógica adicional, como validaciones.

Para definir un setter, se usa la palabra clave **set**. Aquí hay un ejemplo de un setter que valida la entrada antes de asignar el valor a una propiedad:

```
class Producto {
  constructor(nombre, precio) {
    this._nombre = nombre;
    this._precio = precio;
  }


  get precio() {
    return this._precio;
  }


  set precio(nuevoPrecio) {
    if (nuevoPrecio < 0) {
      throw new Error("El valor del precio debe ser positivo");
    }
    this._precio = nuevoPrecio;
  }
}


const producto = new Producto("Laptop", 1000);
console.log(producto.precio); // 1000


producto.precio = 1200; // Cambia el precio
console.log(producto.precio); // 1200


try {
  producto.precio = -100; // Lanza un error
} catch (e) {
```

```
  console.error(e.message); // "precio tiene que ser positivo"
}
```

view rawProducto2.js hosted with by GitHub

En este ejemplo, el setter **precio** incluye una validación para asegurarse de que el nuevo precio no sea negativo. Si se intenta asignar un valor negativo, se lanza un error, evitando que el objeto quede en un estado no válido.

Ventajas de los Getters y Setters

Los getters y setters ofrecen varias ventajas:

1. **Encapsulamiento**: Permiten controlar el acceso y la modificación de propiedades, manteniendo la encapsulación.
2. **Validación**: Se puede añadir lógica de validación para asegurar que los valores sean correctos.
3. **Flexibilidad**: Permiten cambiar la implementación interna sin afectar a la interfaz externa.
4. **Acceso controlado**: Permiten realizar acciones adicionales cuando se accede o modifica una propiedad.

Desventajas de los Getters y Setters

Aunque los getters y setters son útiles, también tienen algunas desventajas:

1. **Complejidad añadida**: Introducen más código y lógica, lo que puede aumentar la complejidad.
2. **Rendimiento**: El uso excesivo de getters y setters puede tener un impacto en el rendimiento.
3. **Menor visibilidad**: Puede no ser obvio para los desarrolladores que están trabajando con métodos especiales en lugar de propiedades directas.