Instituto Politécnico Formosa

Farías Silvana Abigail
Taller de Lenguajes de Programación IV
12/08/2024

Trabajo de Investigación:
Programación Orientada a Objetos en JavaScript

POO es un paradigma de la programación que organiza el código en términos de objetos que contienen propiedades y métodos. En JavaScript, POO se basa en 4 pilares fundamentales:

1. Encapsulamiento: consiste en agrupar datos (propiedades) y métodos que actúan sobre esos datos dentro del objeto, ocultando la implementación interna y exponiendo solo la interfaz pública, protegiendo así los datos de implementaciones no intencionadas.

```
// Encapsulamiento

// Declaración de clase:
// Una clase es una plantilla para crear objetos, que tienen propiedades y métodos

// En JS lo definimos con la palabra reservada class

// Encapsulamos las propiedades nombre y edad y el método saludar dentro de la clase Persona
class Persona {
    #nombre
    constructor(nombre, edad) { // metodo especial que se llama automaticamente al crear una instancia de la clase
        this.#nombre = nombre, // encapsulamos el nombre, propiedad de dato privado
                               // son accesibles en el constructor de clases en la propia declaración de la clase
        this.edad = edad;
    }

    saludar() {
        console.log(`Hola, mi nombre es ${this.#nombre} y tengo ${this.edad} años`)
    }
}

// se llama al constructor y se inicilizan las propiedades
const persona1 = new Persona("Silvana", 19);
persona1.saludar();
```

2. Polimorfismo: Permite que objetos de diferentes clases respondan de manera diferente al mismo mensaje (petición de un objeto a otro para solicitar la ejecución de alguno de sus métodos o para obtener el valor de un atributo público). Es decir, métodos con el mismo nombre pueden tener diferentes implementaciones en diferentes clases.

```
// Polimorfismo
class Animal {
    hacerSonido() {
        console.log("El animal hace un sonido");
    }
}

// Herendan de la clase Animal
class Perro extends Animal { // Perro tiene todas las propiedades y métodos de Animal
    hacerSonido() { // sobreescribe el método hacerSonido de Animal y se comporta de manera diferente
        console.log("ladrido");
    }
}

class Gato extends Animal {
    hacerSonido() {
        console.log("maullido");
    }
}

// creacmos instancias
const perro = new Perro();
perro.hacerSonido();

const gato = new Gato();
gato.hacerSonido();
```

3. Herencia: permite crear nuevas clases (subclases) a partir de clases ya existentes (superclases) heredando sus propiedades y métodos. Es decir, es una característica donde la clase hija obtiene los datos y funciones de la clase padre porque se ha establecido una relación entre ambas, dicha relación se establece con la palabra reservada extends

```
// Herencia

// Clase padre
class Forma {
    constructor() {
        console.log("Soy una forma geométrica");
    }
}

// Clase hija o subclase
class Cuadrado extends Forma {
    constructor() {
        super(); // es una función especial que llama al constructor de la clase padre
        // se ejecuta el constructor de Forma
        console.log("Soy un cuadrado");
    }
}

const cuadrado = new Cuadrado();
// Soy una forma geométrica
// Soy un cuadrado
```

4. Abstracción: Se enfoca en las características esenciales de un objeto ocultando los detalles de la implementación. Las clases abstractas definen una interfaz común que las subclases deben implementar.

```
// Abstracción
class FiguraGeo {
    constructor(nombre) {
        this.nombre = nombre;
    }

    // método abstracto que debe ser implementado en las subclases
    calcularArea() {
        throw new Error ("Debes implementar el método calcularArea en las subclases");
        // obliga a las subclases a proporcionar su propia implementación
        // de cómo se calcula el área de la figura específica.
    }

    // método concreto que puede ser utilizado por las subclases
    mostrarNombre() {
        console.log(`Esta es una figura de tipo: ${this.nombre}`);
        // imprime el nombre de la figura
    }
}

// subclase de la clase FiguraGeo
class Circulo extends FiguraGeo {
    constructor(radio) {
        super("Circulo"); // inicializa el nombre de la figura como "Circulo"
        this.radio = radio;
    }

    // implementación del método abstracto definicion en la superclase o clase base
    calcularArea() {
        return Math.PI * Math.pow(this.radio, 2);
    }
}

// se crea una instancia de la clase Circulo con un radio de 5
const circulo = new Circulo(5)

// se calcula el tamaño de la figura llamando al método calcularArea
area = circulo.calcularArea();

// llama al método mostrarNombre de la clase FiguraGeo
circulo.mostrarNombre(); // Esta es una figura de tipo: Circulo
console.log(`Su area es: ${area}`); // 78.5398...
```

Ventajas de POO en JavaScript:

- Facilita la identificación y corrección de errores, ya que el código está organizado en objetos con funciones específicas.
- La herencia permite reutilizar el código existente, evitando la duplicación y mejorando la eficiencia del desarrollo.

- Facilita la gestión y escalabilidad del código a medida que el proyecto crece en complejidad y tamaño.

- Proporciona una estructura clara y organizada, lo que facilita la comprensión del código y colaboración en equipos de desarrollo.

Limitaciones:

- La creación de muchos objetos puede afectar el rendimiento en algunos casos
- Puede hacer que el código sea más complejo, especialmente en aplicaciones grandes.