Algoritmos: 9

Programación orientada a objetos

La **Programación Orientada a Objetos (POO)** es un paradigma de programación que organiza el diseño y desarrollo del software en torno a objetos. Un **objeto** es una representación de un ente del mundo real o conceptual, que combina datos (atributos o propiedades) y comportamientos (métodos o funciones) dentro de una estructura unificada.

Este enfoque se basa en varios principios clave:

Encapsulación

Abstracción

Herencia

Polimorfismo

Clase

Es un modelo o plantilla que define propiedades (atributos) y comportamientos (métodos) de un objeto.

Ejemplo: Una clase Persona puede tener atributos como nombre y edad, y métodos como hablar() o caminar()

Ejemplo: Clase Persona desde al cual se pueden crear diferentes objetos(instancias) Persona

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

Instancia

Es un objeto creado a partir de una clase. Representa un elemento concreto basado en el molde definido por la clase. Por ejemplo, una instancia de Persona sería una persona específica como Juan, con un nombre = "Juan" y edad = 25

Ejemplo: Una instancia u objeto de la clase Persona anterior

Usamos new para crear un nuevo objeto

const juan = new Persona("Juan", 25);

Propiedades y Métodos

Propiedades: Son las variables asociadas a un objeto o clase, que describen sus características.
Métodos: Son funciones asociadas a un objeto o clase, que definen su comportamiento.

Ejemplo: En la clase Persona, nombre y edad son las propiedades, saludar es un método(algo que hace una persona)

```
class Persona {
        constructor(nombre, edad) {

        this.nombre = nombre;
        this.edad = edad;
    }

    saludar() {

        console.log(`Hola, soy ${this.nombre}`);
    }
}
```

Propiedades y Métodos estáticos

Propiedades estáticas: Son variables asociadas a la clase, no a sus objetos.
Métodos estáticos: Son funciones asociadas a la clase, no a sus objetos. Ejemplo: En la clase Persona, **especie** es una propiedad de clase o estática, **describir** es un método de la clase o estático. Observar como ambos se acceden directamente sin crear instancias

```
class Persona {
        static especie = "Humano";
    static describir() {

        console.log("Los humanos son seres racionales.");

        }

    }

console.log(Persona.especie); // Acceso directo desde la clase
Persona.describir(); // Acceso directo desde la clase
```

Modificadores de acceso

Propiedad pública: Se puede acceder desde cualquier parte del código.
Propiedad privada: Solo accesible desde dentro de la clase, usa # para definirla.
Método público: Puede ser llamado desde cualquier parte del código.
Método privado: Solo puede ser llamado dentro de la misma clase, usa # para definirlo.

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

        saludar() {
                return `Hola, soy ${this.nombre} y tengo ${this.#obtenerEdad()} años.`;
               }
        #obtenerEdad() {
                return this.#edad;  }
 }
```

Encapsulación

Consiste en agrupar datos y métodos dentro de un objeto.

Los detalles internos del objeto se ocultan (ocultación de información) y se exponen solo las partes necesarias a través de interfaces públicas (por ejemplo, métodos **getter** y **setter**). Ejemplo:

Métodos públicos para acceder a la propiedad privada nombre

```
this.getNombre = function () {
        return _nombre;
};
this.setNombre = function (nuevoNombre) {
        _nombre = nuevoNombre;
        };
```

Herencia

Permite crear nuevas clases basadas en clases existentes, heredando sus atributos y métodos.

Facilita la reutilización del código y la creación de jerarquías de clases. Ejemplo:

Clase Perro que hereda de Animal

```
class Perro extends Animal {
        constructor(nombre, raza) {
                super(nombre); // Llama al constructor de la clase padre
                this.raza = raza; // Propiedad específica de Perro
}
```

Polimorfismo

Permite que diferentes objetos respondan de distintas formas al mismo mensaje o método.

Esto se logra mediante la sobrecarga de métodos (métodos con el mismo nombre, pero diferentes parámetros) o la sobrescritura (modificar el comportamiento de un método heredado). Ejemplo:

Clases Perro y Gato que heredan de Animal y sobreescriben el método hablar de la superclase

```
class Perro extends Animal {
        hablar() {
                console.log("El perro dice: ¡Guau!");
                }
}
class Gato extends Animal {
        hablar() {
                console.log("El gato dice: ¡Miau!");
        }
}
```

Algoritmos: 9

VankVersity