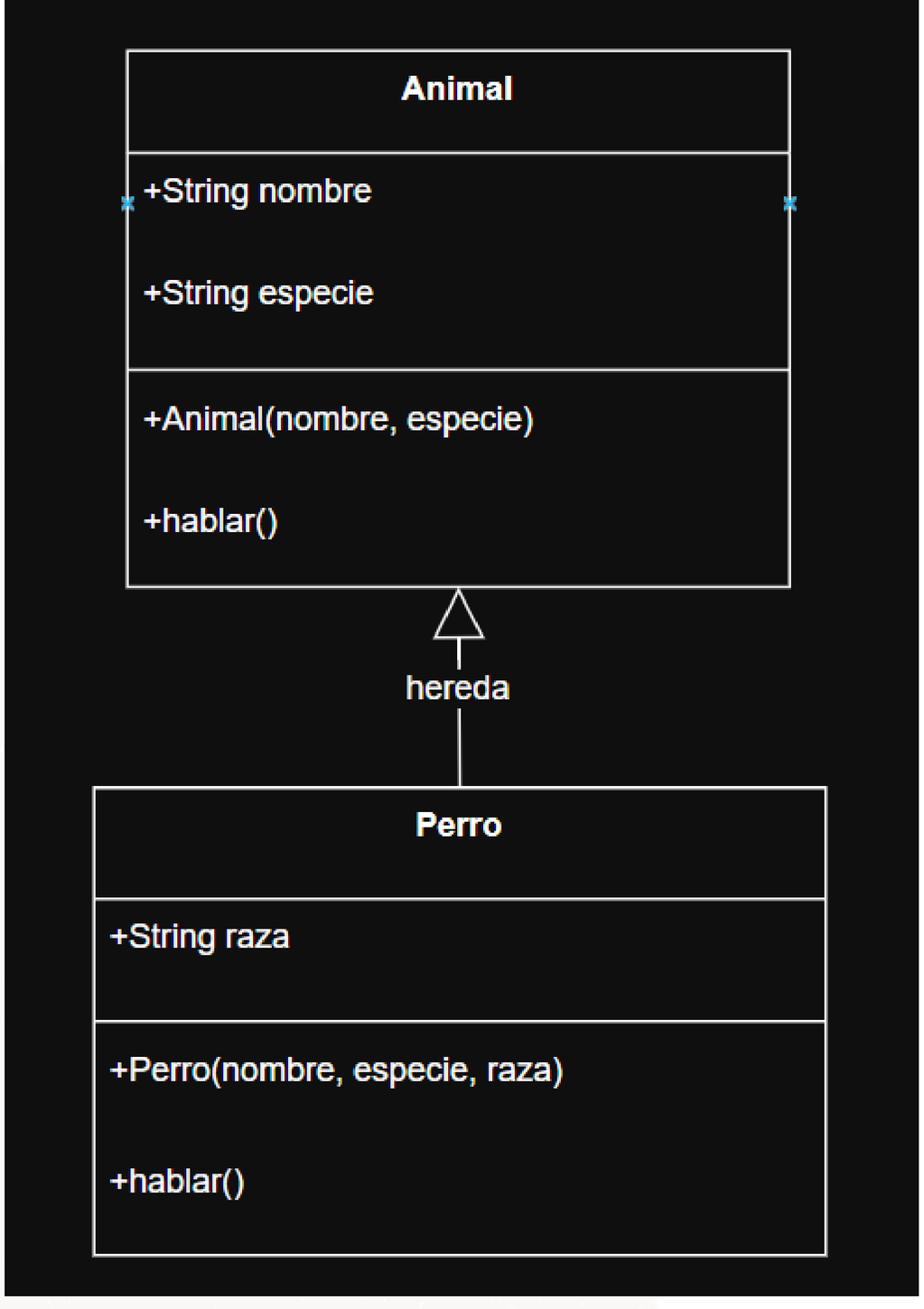
RELACIONES Y CONEXIONES EN LOS

DIAGRAMAS DE CLASES

Mauricio Ramirez Fulguera
Mayli Cortez Cano
Christian Marcelo Quiroga Thaine
Leandro Dennis Montaño



HERENCIA

Este diagrama muestra una jerarquía de clases "Animal" y "Perro".

La clase "Animal" tiene los siguientes atributos y métodos:
- String nombre
- String especie
- Animal(nombre, especie)
- hablar()

La clase "Perro" hereda de la clase "Animal" y tiene los siguientes atributos y métodos adicionales:
- String raza
- Perro(nombre, especie, raza)
- hablar()

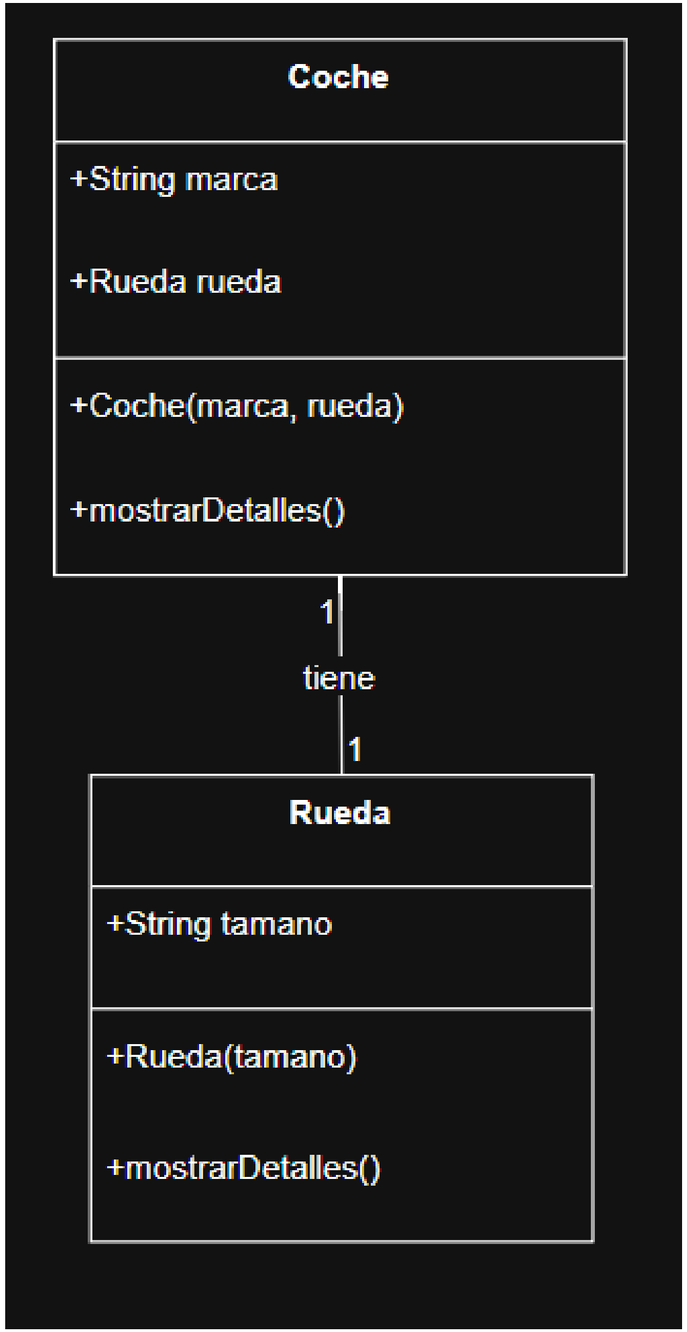
Esta estructura de clases permite modelar diferentes tipos de animales, donde el "Perro" hereda las propiedades y comportamientos generales de la clase "Animal" y agrega características específicas de los perros, como la raza.

El diagrama muestra cómo se relacionan estas clases a través de la herencia, lo cual es un concepto fundamental de la programación orientada a objetos.

```javascript
// Clase Superclase
class Animal {
    constructor(nombre, especie) {
        this.nombre = nombre;
        this.especie = especie;
    }

    // Método común para todas las subclases
    hablar() {
        console.log(`${this.nombre} hace un sonido.`);
    }
}
```

```javascript
// Clase Subclase (hereda de Animal)
class Perro extends Animal {
    constructor(nombre, especie, raza) {
        super(nombre, especie);  // Llamada al constructor de la superclase
        this.raza = raza;
    }

    // Sobreescribir el método hablar
    hablar() {
        console.log(`${this.nombre} dice: ¡Guau!`);
    }
}
```

```javascript
// Crear un objeto de la subclase
const miPerro = new Perro('Rex', 'Canino', 'Labrador');

// Mostrar detalles del perro
console.log(`${miPerro.nombre} es un ${miPerro.especie} de raza ${miPerro.raza}.`);

// Llamar al método hablar, que es sobrescrito en la subclase
miPerro.hablar();  // Llamará al método hablar de la clase Perro
```



ASOCIACIÓN

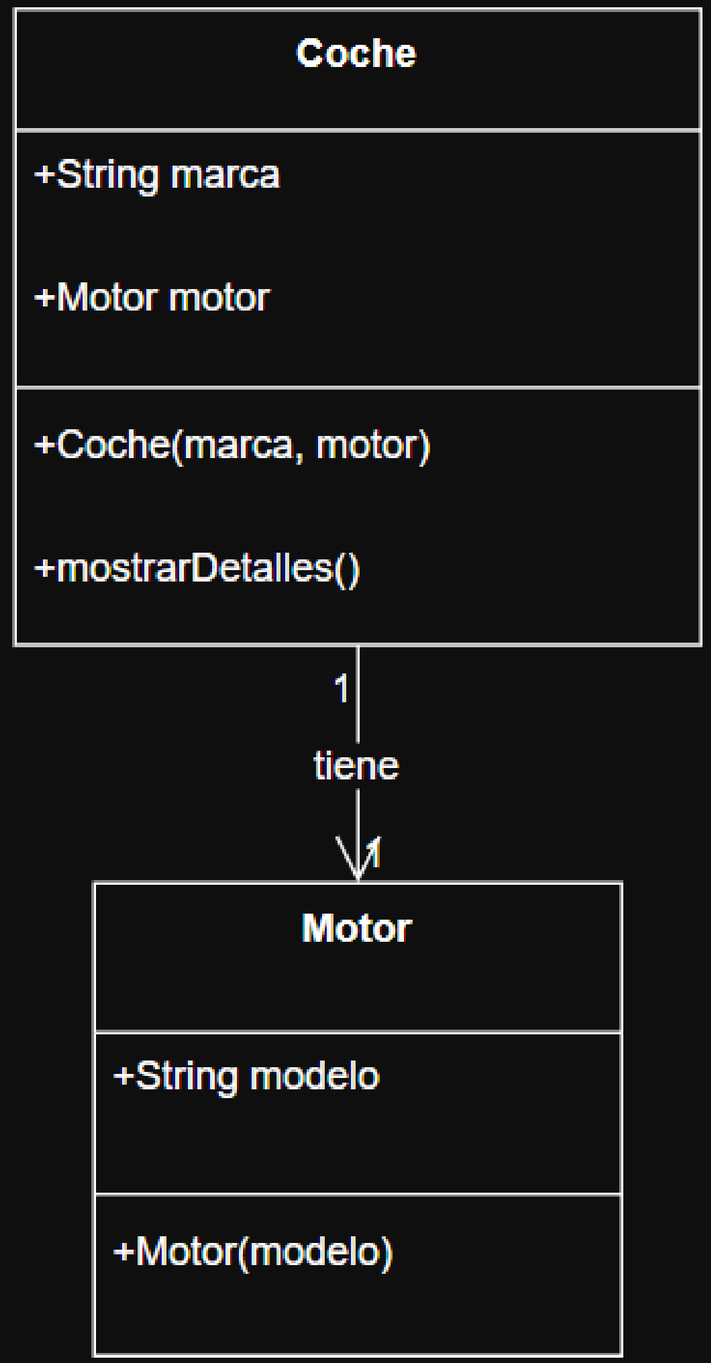
El diagrama representa la relación entre dos clases:

1. Clases:
   - Clase Rueda:
     - Tiene un atributo: tamano, que es un string (ej. "16 pulgadas").
     - Tiene un método: mostrarDetalles(), que imprime el tamaño de la rueda.
   - Clase Coche:
     - Tiene un atributo: marca, que es un string (ej. "Toyota").
     - Tiene un atributo: rueda, que es una instancia de la clase Rueda (un coche tiene una rueda).
     - Tiene un método: mostrarDetalles(), que imprime la marca del coche y luego llama a mostrarDetalles() de la Rueda asociada.
2. Relación (Asociación):
   - La relación entre Coche y Rueda se indica con "1" -- "1", lo que significa que un coche tiene una rueda. La rueda es un atributo de la clase Coche.
   - Esta relación es una asociación porque el Coche y la Rueda están conectados, pero ambos pueden existir por separado. Es decir, puedes tener un coche sin rueda y una rueda sin coche, aunque no sería muy lógico.

```javascript
// Clase Rueda
class Rueda {
    constructor(tamano) {
        this.tamano = tamano;
    }

    mostrarDetalles() {
        console.log(`Rueda de tamaño: ${this.tamano}`);
    }
}
```

```javascript
// Clase Coche
class Coche {
    constructor(marca, rueda) {
        this.marca = marca;
        this.rueda = rueda; // Asociación: El coche tiene una rueda
    }

    mostrarDetalles() {
        console.log(`Coche Marca: ${this.marca}`);
        this.rueda.mostrarDetalles();
    }
}
```

```javascript
// Crear una rueda
const rueda1 = new Rueda('16 pulgadas');

// Crear un coche y asociarle una rueda
const coche1 = new Coche('Toyota', rueda1);

// Mostrar detalles del coche y su rueda
coche1.mostrarDetalles();
```



AGREGACIÓN

Este diagrama de clases permite realizar las siguientes funcionalidades:

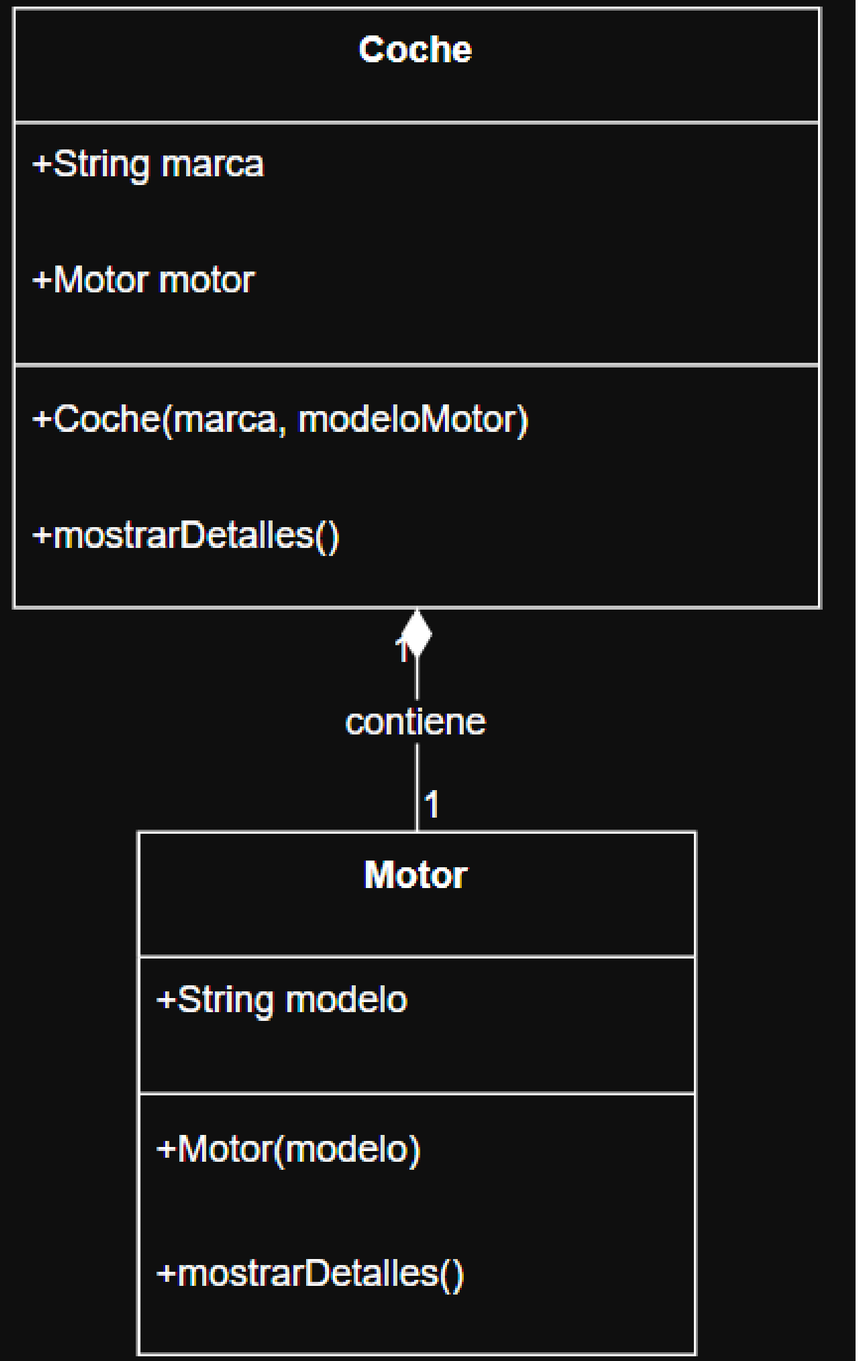
1. Crear un objeto Coche, especificando su marca y el Motor asociado.
2. Obtener información del Coche, como la marca y el Motor, a través del método mostrarDetalles().
3. Crear un objeto Motor, especificando su modelo.
4. Asociar el Motor al Coche a través de la composición, donde el Coche "tiene" un Motor.

Esto permite modelar la relación entre un Coche y su Motor, de forma que se pueda acceder a la información de ambos de manera integrada. Por ejemplo, se podría obtener la marca del Coche y el modelo del Motor a través de una sola llamada al método mostrarDetalles().

```javascript
// Clase Parte
class Motor {
    constructor(modelo) {
        this.modelo = modelo;
    }
}
```

```javascript
// Clase Todo
class Coche {
    constructor(marca, motor) {
        this.marca = marca;
        this.motor = motor; // Agregación: El coche tiene un motor, pero el motor puede existir por sí solo
    }

    mostrarDetalles() {
        console.log(`Coche Marca: ${this.marca}, Motor Modelo: ${this.motor.modelo}`);
    }
}
```

```javascript
// Crear un motor
const motor1 = new Motor('V8');

// Crear un coche y agregarle el motor
const coche1 = new Coche('Ford', motor1);

// Mostrar detalles del coche
coche1.mostrarDetalles();
```



COMPOSICIÓN

Este diagrama muestra la estructura de una clase llamada "Coche" :

Coche
String marca: Atributo que almacena la marca del coche.
Motor motor: Atributo que almacena el motor del coche.
Coche(marca, modeloMotor): Constructor de la clase Coche que toma la marca y el modelo del motor como parámetros.
mostrarDetalles(): Método que muestra los detalles del coche.

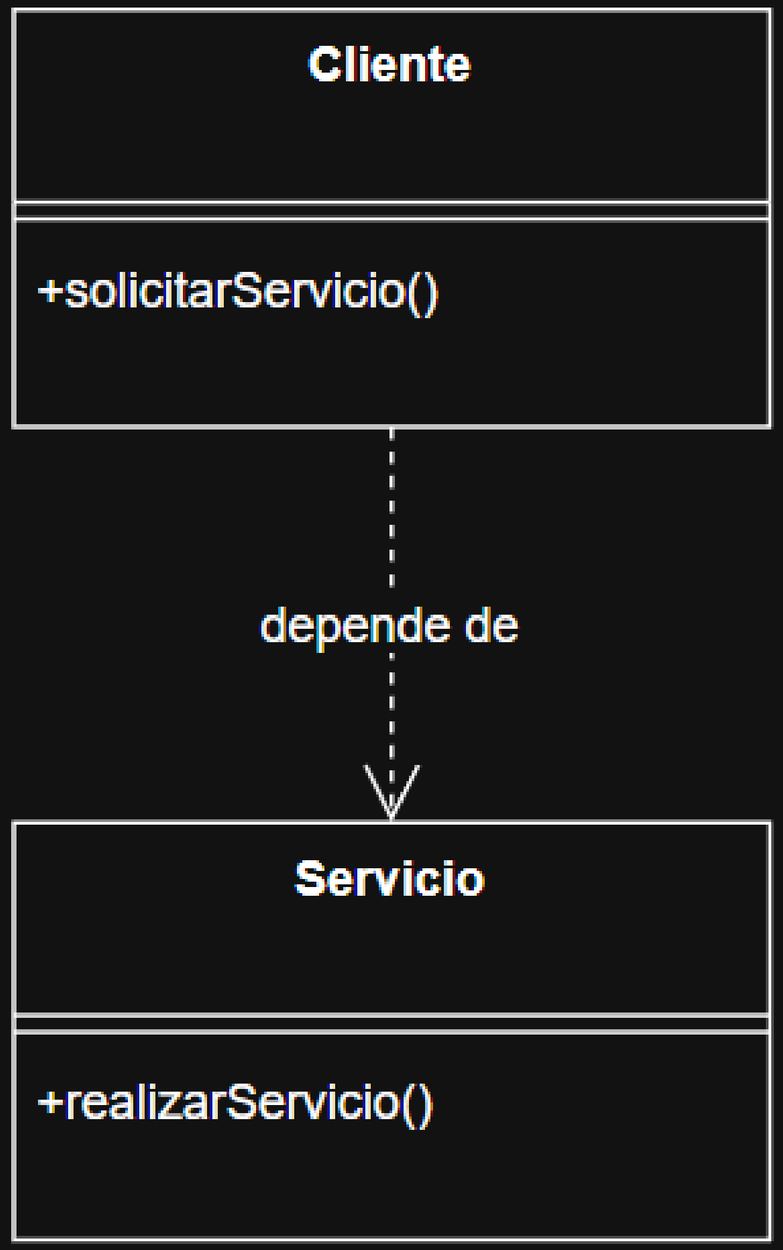
Motor
String modelo: Atributo que almacena el modelo del motor.
Motor(modelo): Constructor de la clase Motor que toma el modelo como parámetro.
mostrarDetalles(): Método que muestra los detalles del motor.

Este diagrama representa la estructura de una clase Coche que tiene un motor como parte de su composición. Permite crear objetos Coche y acceder a los detalles tanto del coche como del motor.

```javascript
// Clase Motor (no puede existir sin Coche)
class Motor {
    constructor(modelo) {
        this.modelo = modelo;
    }

    mostrarDetalles() {
        console.log(`Motor Modelo: ${this.modelo}`);
    }
}
```

```javascript
// Clase Coche (contiene Motor)
class Coche {
    constructor(marca, modeloMotor) {
        this.marca = marca;
        this.motor = new Motor(modeloMotor); // El coche crea y tiene un motor
    }

    mostrarDetalles() {
        console.log(`Coche Marca: ${this.marca}`);
        this.motor.mostrarDetalles();
    }
}
```

```javascript
// Crear un coche con un motor
const coche1 = new Coche('Toyota', 'V8');

// Mostrar detalles del coche y su motor
coche1.mostrarDetalles();
```



DEPENDENCIA

Este diagrama muestra la relación entre un Cliente y un Servicio en un sistema de software. Específicamente:

- El Cliente tiene la responsabilidad de solicitar un Servicio a través del método +solicitarServicio().
- El Servicio es responsable de realizar la funcionalidad solicitada a través del método +realizarServicio().
- La flecha que apunta de Cliente a Servicio indica que el Cliente "depende de" el Servicio para que se realice la funcionalidad requerida.

En resumen, este diagrama representa una interacción básica entre un Cliente que solicita un Servicio, y el Servicio que es responsable de ejecutar la funcionalidad solicitada. Es una representación simplificada de una arquitectura de software orientada a servicios.

```
// Clase Servicio
class Servicio {
    realizarServicio() {
        console.log("Servicio realizado.");
    }
}
```

```
// Clase Cliente
class Cliente {
    solicitarServicio() {
        console.log("Cliente solicitando servicio...");
        const servicio = new Servicio(); // Dependencia: Cliente depende
        de Servicio
        servicio.realizarServicio();    // Usamos el servicio en esta
        clase
    }
}
```

```
// Crear un cliente
const cliente1 = new Cliente();
cliente1.solicitarServicio();  // El cliente solicita el servicio, lo
cual depende de la clase Servicio
```

Muchas

GRACIAS