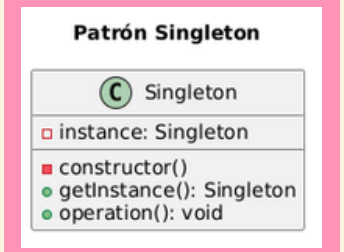
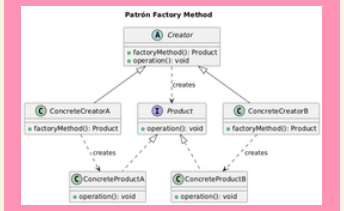
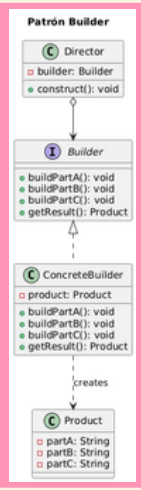
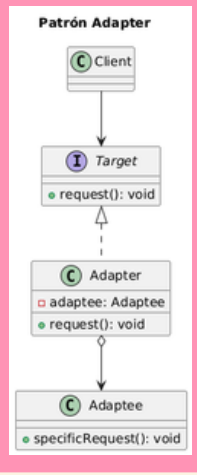
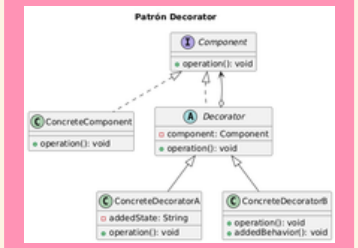
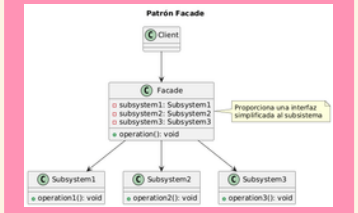
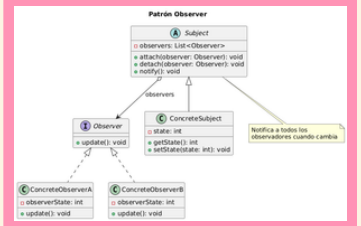
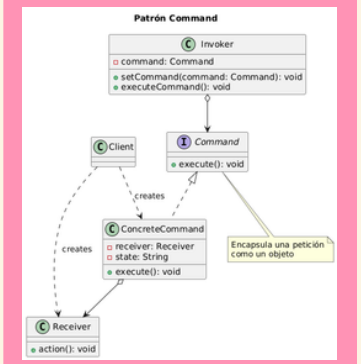
PATRONES DE DISEÑO ARQUITECTONICO

NOMBRE DEL PATRON	DEFINICIÓN	CLASIFICACIÓN	EJEMPLO EN UML	IMPLEMENTACIÓN DE JAVASCRIPT
SINGLETON	Garantiza que una clase tenga una única instancia y proporciona un punto de acceso global a ella.	Creacional		class Singleton { static instance = null; constructor() { if (Singleton.instance) { return Singleton.instance; } Singleton.instance = this; this.data = []; } addData(item) { this.data.push(item); }}
FACTORY METHOD	Una interfaz para crear objetos, pero permite a las subclases decidir qué clase instanciar.	Creacional		class VehicleFactory { createVehicle(type) { switch(type) { case 'car': return new Car(); case 'motorcycle': return new Motorcycle(); default: throw new Error('Tipo no válido'); } } } class Car { drive() { return 'Conduciendo carro'; } } class Motorcycle { drive() { return 'Conduciendo moto'; } } // Uso const factory = new VehicleFactory(); const car = factory.createVehicle('car');

NOMBRE DEL PATRON	DEFINICIÓN	CLASIFICACIÓN	EJEMPLO EN UML	IMPLEMENTACIÓN DE JAVASCRIPT
BUILDER	Separa la construcción de un objeto complejo de su representación, permitiendo crear diferentes representaciones.	Creacional		class ComputerBuilder { constructor() { this.computer = {}; } setCPU(cpu) { this.computer.cpu = cpu; return this; } setRAM(ram) { this.computer.ram = ram; return this; } setStorage(storage) { this.computer.storage = storage; return this; } build() { return this.computer; } } // Uso const computer = new ComputerBuilder() .setCPU('Intel i7') .setRAM('16GB') .setStorage('512GB SSD') .build();
ADAPTER	Permite que interfaces incompatibles trabajen juntas, actuando como un puente entre dos interfaces.	Estructural		class OldPrinter { printOld(text) { return `Old: ${text}`; } } class PrinterAdapter { constructor(oldPrinter) { this.oldPrinter = oldPrinter; } print(text) { return this.oldPrinter.printOld(text); } } // Uso const oldPrinter = new OldPrinter(); const adapter = new PrinterAdapter(oldPrinter); console.log(adapter.print('Documento'));

NOMBRE DEL PATRON	DEFINICIÓN	CLASIFICACIÓN	EJEMPLO EN UML	IMPLEMENTACIÓN DE JAVASCRIPT
DECORATOR	Añade funcionalidades adicionales a objetos de forma dinámica sin alterar su estructura.	Estructural		class Coffee { cost() { return 5; } description() { return 'Café simple'; } } class MilkDecorator { constructor(coffee) { this.coffee = coffee; } cost() { return this.coffee.cost() + 2; } description() { return this.coffee.description() + ' + Leche'; } } // Uso let coffee = new Coffee(); coffee = new MilkDecorator(coffee); console.log(coffee.description()); // Café simple + Leche console.log(coffee.cost()); // 7
FACADE	Proporciona una interfaz simplificada para un subsistema complejo.	Estructural		class CPU { start() { console.log('CPU iniciado'); } } class Memory { load() { console.log('Memoria cargada'); } } class HardDrive { read() { console.log('Disco leído'); } } class ComputerFacade { constructor() { this.cpu = new CPU(); this.memory = new Memory(); this.hardDrive = new HardDrive(); } startComputer() { this.cpu.start(); this.memory.load(); this.hardDrive.read(); } } // Uso const computer = new ComputerFacade(); computer.startComputer();

NOMBRE DEL PATRON	DEFINICIÓN	CLASIFICACIÓN	EJEMPLO EN UML	IMPLEMENTACIÓN DE JAVASCRIPT
OBSERVER	Una dependencia uno a muchos entre objetos, de forma que cuando un objeto cambia su estado, todos sus dependientes son notificados.	Comportamiento		class Subject { constructor() { this.observers = []; } subscribe(observer) { this.observers.push(observer); } unsubscribe(observer) { this.observers = this.observers.filter(obs => obs !== observer); } notify(data) { this.observers.forEach(observer => observer.update(data)); } } class Observer { update(data) { console.log(`Recibido: ${data}`); } } // Uso const subject = new Subject(); const observer1 = new Observer(); subject.subscribe(observer1); subject.notify('Nuevo evento');
STRATEGY	Define una familia de algoritmos, encapsula cada uno y los hace intercambiables.	Comportamiento		class PaymentContext { constructor(strategy) { this.strategy = strategy; } setStrategy(strategy) { this.strategy = strategy; } pay(amount) { return this.strategy.pay(amount); } } class CreditCard { pay(amount) { return `Pagando $${amount} con tarjeta`; } } class PayPal { pay(amount) { return `Pagando $${amount} con PayPal`; } } // Uso const payment = new PaymentContext(new CreditCard()); console.log(payment.pay(100)); payment.setStrategy(new PayPal()); console.log(payment.pay(200));

NOMBRE DEL PATRON	DEFINICIÓN	CLASIFICACIÓN	EJEMPLO EN UML	IMPLEMENTACIÓN DE JAVASCRIPT
COMMAND	Encapsula una petición como un objeto, permitiendo parametrizar clientes con diferentes peticiones.	Comportamiento		class Light { on() { console.log('Luz encendida'); } off() { console.log('Luz apagada'); } } class TurnOnCommand { constructor(light) { this.light = light; } execute() { this.light.on(); } } class TurnOffCommand { constructor(light) { this.light = light; } execute() { this.light.off(); } } class RemoteControl { submit(command) { command.execute(); } } // Uso const light = new Light(); const remote = new RemoteControl(); remote.submit(new TurnOnCommand(light));