Patrones de Diseño Arquitectonico

Aprendiz:

Juan David Buitrago Suarez

Instructor:

Ing. Néstor Montaño

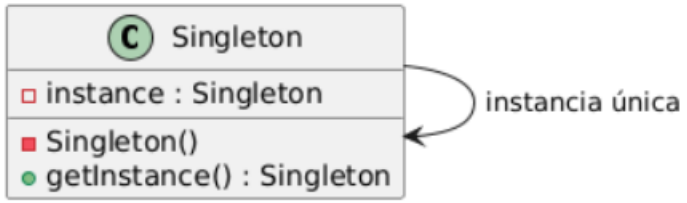
Tecnólogo Análisis y Desarrollo de Software

Ficha: 3064241

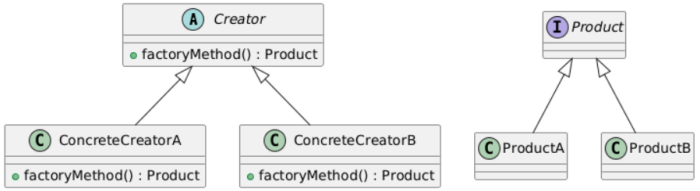
SENA Centro de Diseño y Metrología

Patrones Creacionales

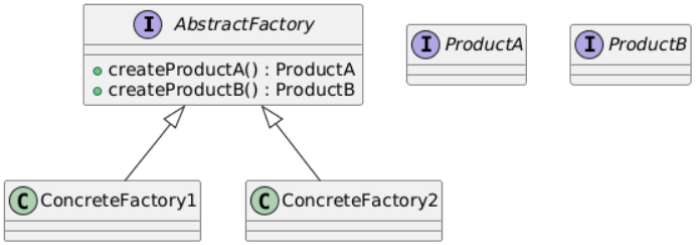
Nombre	Definición	Ejemplo en UML	Implementación en Node.js

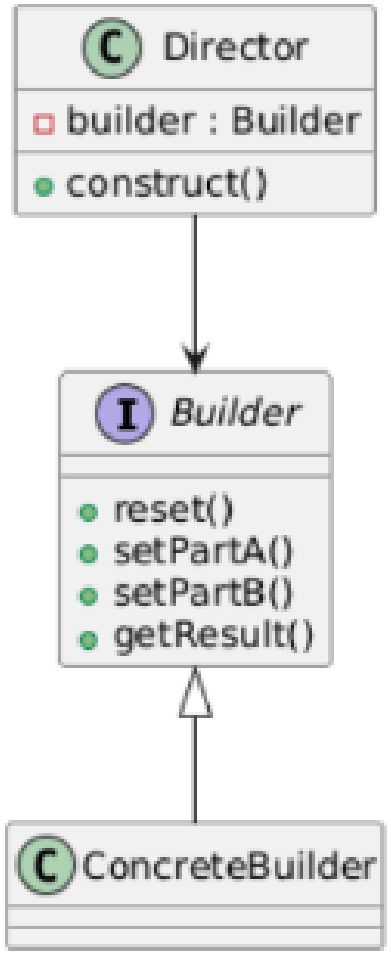
Singleton	Asegura que solo exista un objeto de ese tipo y que todos usen el mismo.		- class Singleton {
			constructor() {
			if (Singleton.instance) return Singleton.instance;
			Singleton.instance = this;
			}
			}
			const a = new Singleton();
			const b = new Singleton();
			console.log(a === b); // true

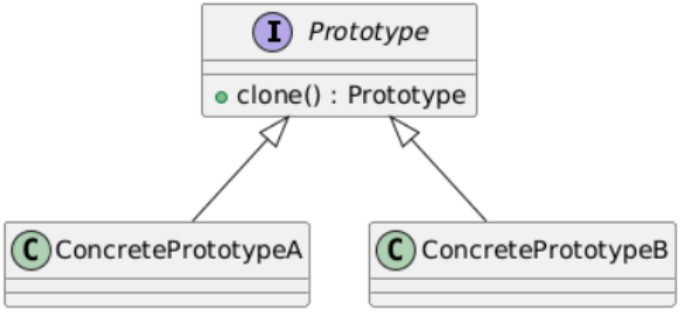
Nombre	Definición	Ejemplo en UML	Implementación en Node.js

Factory Method	Permite crear diferentes objetos sin decir exactamente cuál se va a crear.		-class ProductoA {
			crear() { return "Producto A creado";}}
			class ProductoB {
			crear() { return "Producto B creado";}}
			class Creador {
			factoryMethod(tipo) {
			return tipo === "A" ? new ProductoA() : new ProductoB();
			}
			}
			const creador = new Creador();
			console.log(creador.factoryMethod("A").crear());

Nombre	Definición	Ejemplo en UML	Implementación en Node.js

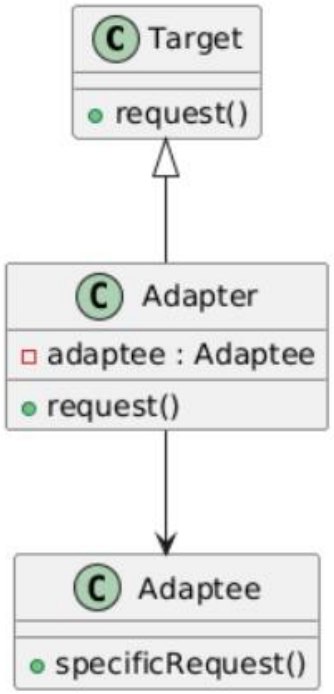
Abstract Factory	Crea varios tipos de objetos que combinan bien entre sí sin mostrar cómo se hacen.		- class ButtonWindows {
			draw() { return "Botón Windows"; }
			}
			class MenuWindows {
			show() { return "Menú Windows"; }
			}
			class UIFactoryWindows {
			createButton() { return new ButtonWindows(); }
			createMenu() { return new MenuWindows(); }
			}

Builder	Ayuda a armar objetos paso a paso para no confundir la construcción con el resultado final.		- class Builder {
			constructor() {
			this.data = {};
			}
			setA(v) { this.data.A = v; return this; }
			setB(v) { this.data.B = v; return this; }
			build() { return this.data; }
			}

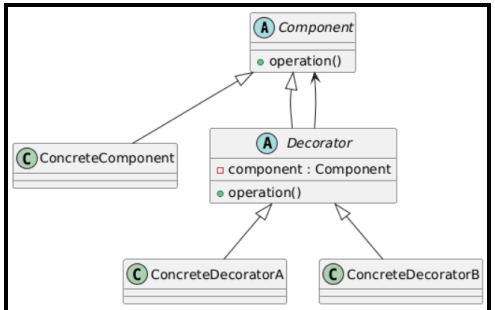
Nombre	Definición	Ejemplo en UML	Implementación en Node.js
Prototype	Permite copiar un objeto ya hecho para crear otro igual sin hacerlo desde cero.		- const proto = { a: 1 }; const obj = Object.create(proto);

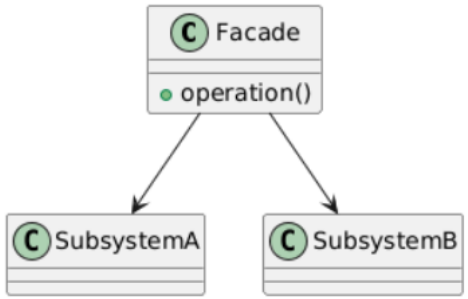
Patrones Estructurales

Nombre del Patrón	**Definición**	**Ejemplo en UML**	**Implementación en Node.js**

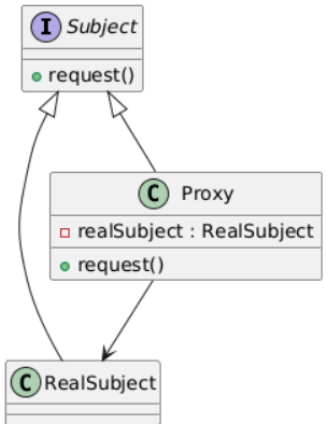
Adapter	Hace que dos cosas que no son compatibles puedan funcionar juntas.		- class Old {
			specific() { return "Función antigua"; }
			}
			
			class Adapter {
			request() { return new Old().specific(); }
			}

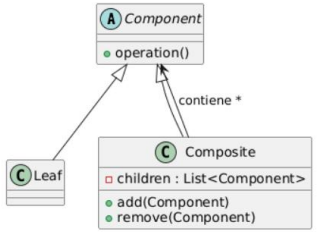
Nombre del Patrón	Definición	Ejemplo en UML	Implementación en Node.js

Decorator	Permite agregar nuevas funciones a un objeto sin cambiarlo por dentro.		- function deco(fn) {
			return () => fn() + " decorado";
			}

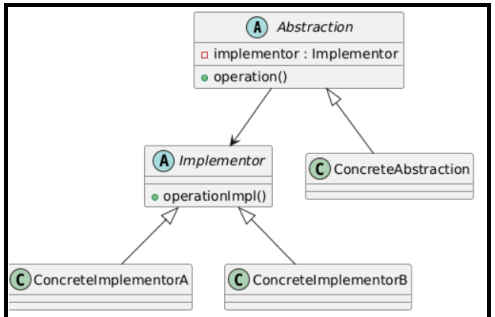
Facade	Da una forma fácil de usar un sistema complicado.		- class A { a() { console.log("A"); } }
			class B { b() { console.log("B"); } }
			class Facade {
			start() {
			new A().a();
			new B().b();
			}
			}

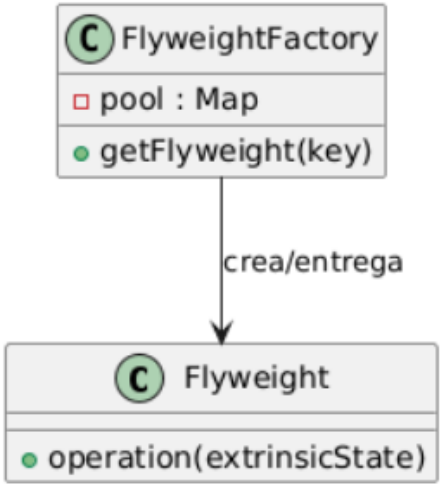
Nombre del Patrón	Definición	Ejemplo en UML	Implementación en Node.js

Proxy	Es un intermediario que controla el acceso a otro objeto.		- const obj = { name: "Real" };
			const proxy = new Proxy(obj, {
			get(target, prop) {
			console.log("Acceso controlado");
			return target[prop];
			}
			});

Composite	Permite manejar varios objetos como si fueran uno solo.		- class Component {
			add() {}
			operation() {}
			}

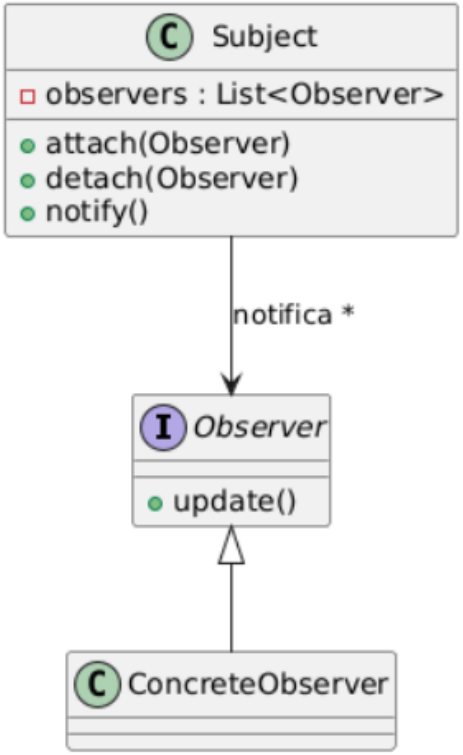
Nombre del Patrón	Definición	Ejemplo en UML	Implementación en Node.js

Bridge	Separa lo que hace una clase de cómo está hecha para que sean más flexibles.		- class Abstraction {
			constructor(imp) { this.imp = imp; }
			run() { this.imp.run(); }
			}

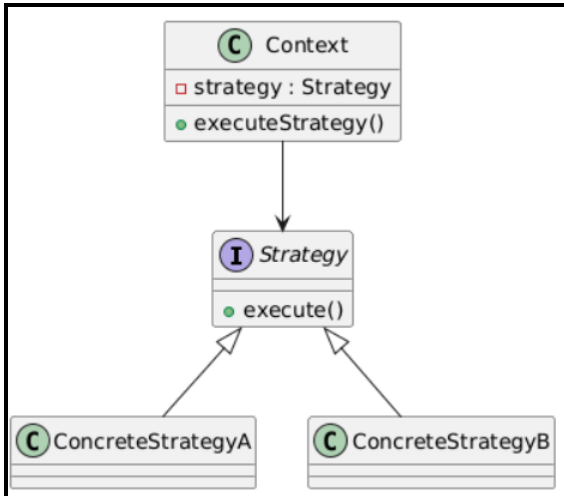
Flyweight	Reduce el uso de memoria compartiendo partes repetidas entre objetos.		- class Fly {
			constructor(s) { this.s = s; }
			}

Patrones de Comportamiento

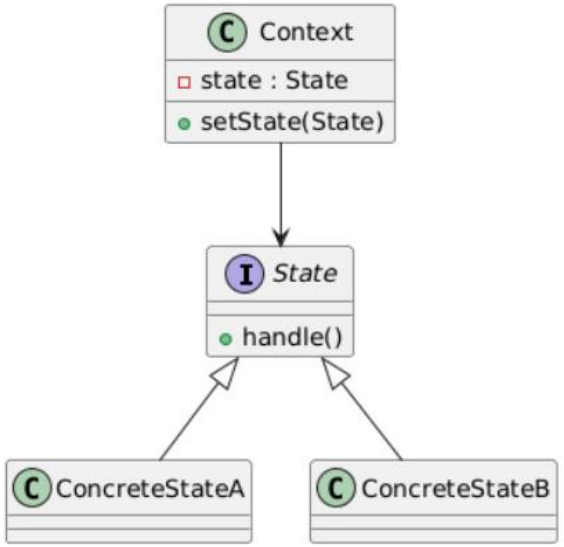
Nombre del Patrón	**Definición**	**Ejemplo en UML**	**Implementación en Node.js**

Observer	Varios objetos se enteran automáticamente cuando otro cambia.		- class Sub {
			constructor() { this.obs = []; }
			add(o) { this.obs.push(o); }
			notify(d) { this.obs.forEach(o => o.update(d)); }
			}

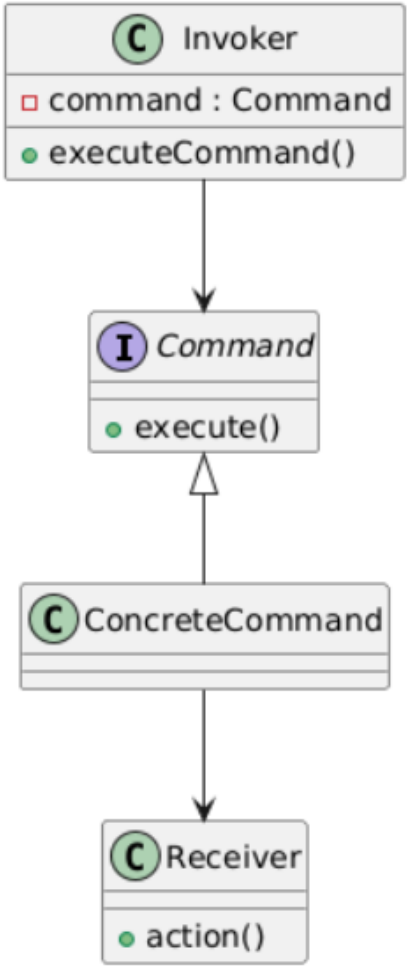
Nombre del Patrón	**Definición**	**Ejemplo en UML**	**Implementación en Node.js**

Strategy	Permite cambiar la forma de hacer algo sin modificar el objeto principal.		- class Ctx {
			set(s) { this.s = s; }
			run() { return this.s.exec(); }
			}

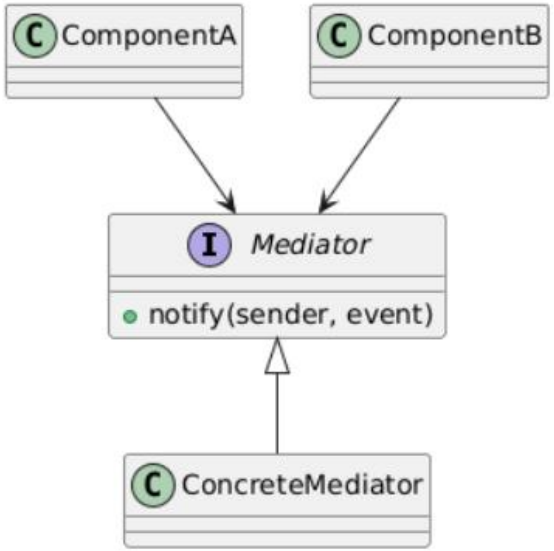
Nombre del Patrón	**Definición**	**Ejemplo en UML**	**Implementación en Node.js**

State	Cambia el comportamiento de un objeto según la situación en la que se encuentre.		- class StateA {
			handle() { return "A"; }
			}

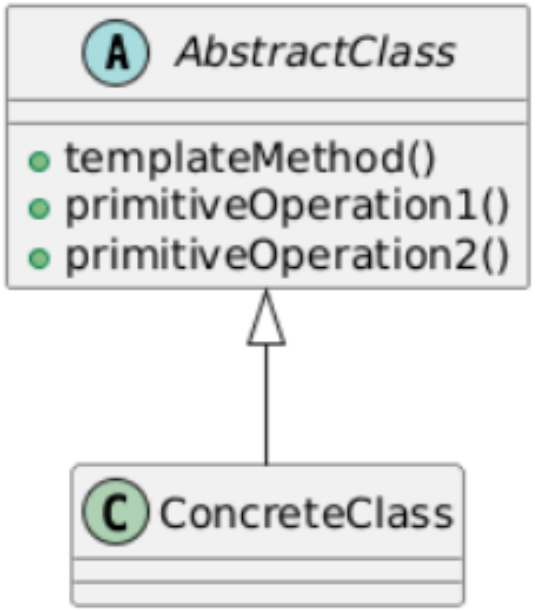
Nombre del Patrón	**Definición**	**Ejemplo en UML**	**Implementación en Node.js**

Command	Convierte acciones en objetos para poder guardarlas, deshacerlas o enviarlas.		- class Cmd {
			exec() {}
			}

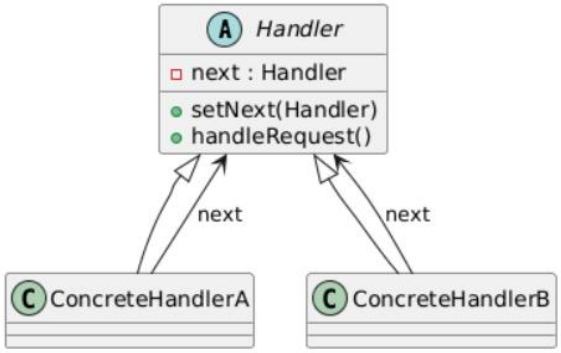
Nombre del Patrón	Definición	Ejemplo en UML	Implementación en Node.js

Mediator	Hace que varios objetos se comuniquen sin hablar entre ellos directamente.		- class Mediator {
			send(msg, to) { to.receive(msg); }
			}

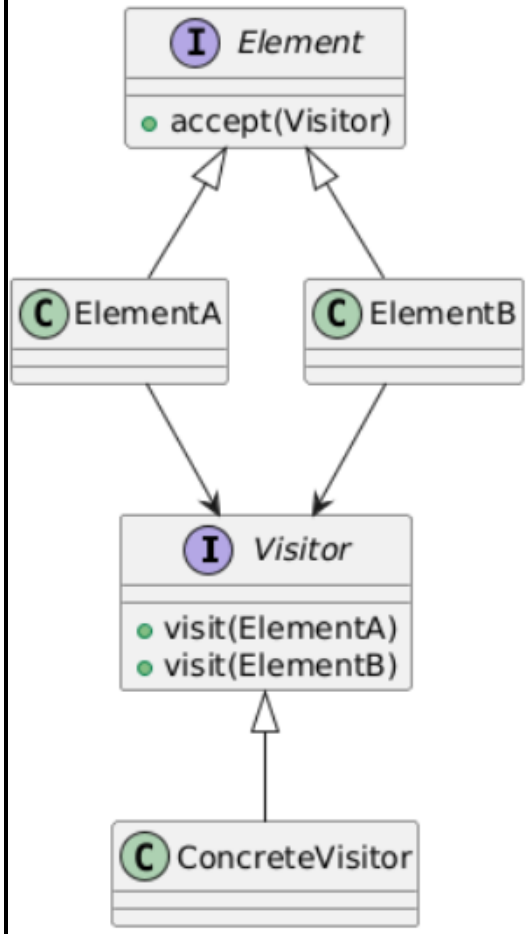
Nombre del Patrón	**Definición**	**Ejemplo en UML**	**Implementación en Node.js**

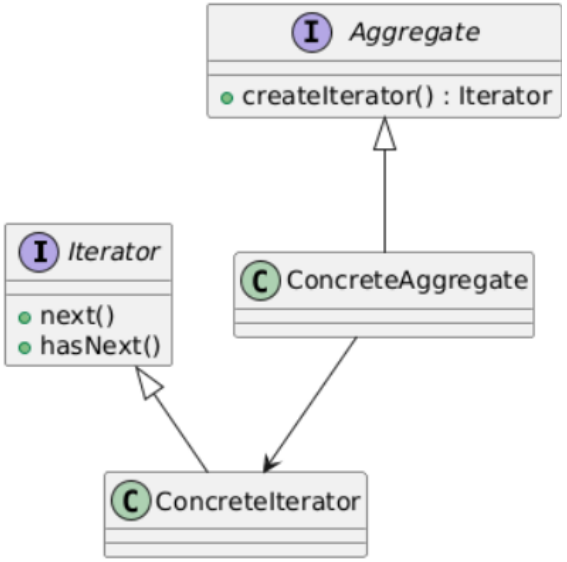
Template Method	Marca un “paso a paso” general y deja que cada clase cambie los detalles.		- class T {
			run() { this.a(); this.b(); }
			}

Nombre del Patrón	**Definición**	**Ejemplo en UML**	**Implementación en Node.js**

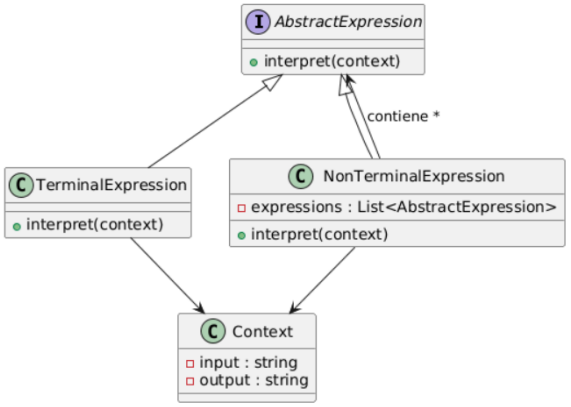
Chain of Responsibility	Pasa una petición por varios objetos hasta que uno pueda atenderla.		- class H {
			setNext(n) { this.n = n; }
			handle(r) { if (this.n) this.n.handle(r); }
			}

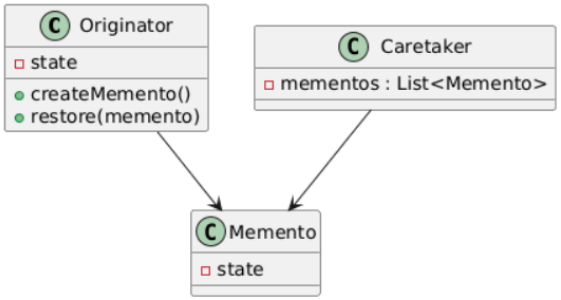
Nombre del Patrón	Definición	Ejemplo en UML	Implementación en Node.js

Visitor	Permite agregar funciones nuevas sin modificar las clases originales.		- class Visitor {
			visit(obj) { obj.accept(this); }
			}

Nombre del Patrón	**Definición**	**Ejemplo en UML**	**Implementación en Node.js**
Iterator	Da una forma sencilla de recorrer elementos uno por uno.		- for (const i of [1,2,3]) console.log(i);

Nombre del Patrón	Definición	Ejemplo en UML	Implementación en Node.js

Interpreter	Permite crear reglas para entender y ejecutar un pequeño lenguaje.		- class Expr {
			interp() {}
			}

Memento	Guarda y recupera estados anteriores de un objeto.		- class Origin {
			save() { return { ...this }; }
			}