

Guia P1 - POO

Ejercicio: Identificación de Patrones

Lee el siguiente código y responde:

```
class DatabaseConnection {  
    private static instance: DatabaseConnection;  
    private constructor() {}  
  
    static getInstance(): DatabaseConnection {  
        if (!this.instance) {  
            this.instance = new DatabaseConnection();  
        }  
        return this.instance;  
    }  
}
```

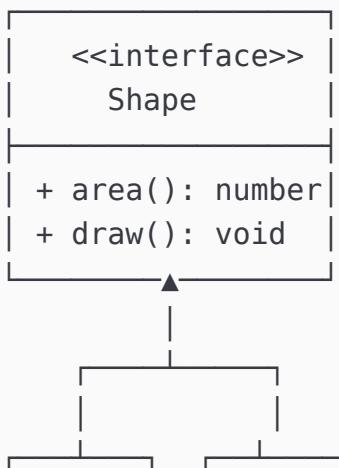
Preguntas:

1. ¿Qué patrón de diseño se está implementando?
2. ¿Cuál es el propósito de hacer el constructor privado?
3. ¿En qué situaciones del mundo real usarías este patrón?

Solución

Ejercicio: Implementación desde UML

Implementa el siguiente diagrama de clases en TypeScript:



Circle	Square
-radius	-side

Requisitos:

- Implementa la interfaz `Shape`
- Implementa las clases `Circle` y `Square`
- Agrega un método `describe()` que retorne una descripción de la forma
- Crea un array de formas mixtas y calcula el área total

Solución

Ejercicio: Refactorización con Decorator

El siguiente código calcula el precio de pizzas con ingredientes adicionales:

```
class Pizza {
    private base: number = 10;
    private queso: boolean = false;
    private pepperoni: boolean = false;
    private champinones: boolean = false;
    private extraQueso: boolean = false;

    agregarQueso() { this.queso = true; }
    agregarPepperoni() { this.pepperoni = true; }
    agregarChampinones() { this.champinones = true; }
    agregarExtraQueso() { this.extraQueso = true; }

    calcularPrecio(): number {
        let precio = this.base;
        if (this.queso) precio += 2;
        if (this.pepperoni) precio += 3;
        if (this.champinones) precio += 2.5;
        if (this.extraQueso) precio += 1.5;
        return precio;
    }

    getDescripcion(): string {
        let desc = "Pizza base";
        if (this.queso) desc += " + queso";
        if (this.pepperoni) desc += " + pepperoni";
        if (this.champinones) desc += " + champiñones";
        if (this.extraQueso) desc += " + extra queso";
        return desc;
    }
}
```

```
}
```

Refactoriza usando el patrón Decorator para:

- Eliminar los booleanos
- Hacer el código más extensible (agregar nuevos ingredientes sin modificar código existente)
- Permitir agregar el mismo ingrediente múltiples veces
- Mantener la funcionalidad de `calcularPrecio()` y `getDescripcion()`

Solución

Ejercicio: Sistema de Tarifas UrbanRide

Eres desarrollador en "*UrbanRide*", una aplicación de transporte urbano que necesita calcular tarifas dinámicas para sus viajes. El sistema debe ser lo suficientemente flexible para adaptarse a diferentes tipos de viajes y aplicar recargos según las circunstancias.

Requisitos del Sistema:

1. Cálculo Base de Tarifas:

- **Viajes Cortos** (< 8 km): Tarifa base de \$1.50 + \$0.80 por kilómetro
- **Viajes Medios** (≥ 8 km y < 12 km): Tarifa base de \$1.80 + \$0.70 por kilómetro
- **Viajes Largos** (≥ 12 km): Tarifa base de \$2.00 + \$0.60 por kilómetro

2. Recargos Aplicables:

- **Nocturno**: +15% sobre el total
- **Aeropuerto**: +\$3.00 fijo
- **Fin de Semana**: +25% sobre el total

3. Funcionalidad Requerida:

- Seleccionar automáticamente la estrategia de tarifa base según la distancia
- Permitir aplicar múltiples recargos de forma encadenada
- Mostrar desglose detallado del costo (base + cada recargo)
- Calcular el precio final total

Solución
