Practica 1

Daniel Rubi Alvarado

Patrones de Diseño

Código 1

```
class Database {
    constructor() {
        if (!Database.instance) {
            Database.instance = this;
        }
        return Database.instance;
    }
    query(sql) {
        console.log("Ejecutando consulta:", sql);
    }
}
const db1 = new Database();
const db2 = new Database();
console.log(db1 === db2);  // true
db1.query("SELECT * FROM users");
```

El patrón que se puede apreciar en el código es el patrón singleton. Este patrón asegura que una clase tenga una única instancia y proporciona un punto de acceso global a ella.

Singleton

Única Instancia: El patrón Singleton garantiza que una clase solo tenga una única instancia en todo el ciclo de vida de una aplicación. En el código, esto se logra mediante el uso de una propiedad estática **instance** en la clase **Database**.

Control de Acceso Global: El patrón proporciona un punto de acceso global a la instancia. Esto significa que cualquier código que necesite usar la instancia puede obtenerla sin crear una nueva instancia.

1. **Constructor:**

El constructor de la clase Database verifica si ya existe una instancia (Database.instance).

Si no existe (!Database.instance), se crea y se asigna this a Database.instance.

Si ya existe una instancia, el constructor devuelve esa instancia existente, asegurando que solo haya una instancia de Database.

2. **Método query:**

Este método es simplemente una simulación de una consulta de base de datos que imprime la consulta SQL proporcionada.

3. **Instanciación y Comparación:**

Se crean dos variables db1 y db2 que intentan instanciar la clase Database.

La comparación console.log (db1 === db2); imprime true, demostrando que ambas variables apuntan a la misma instancia.

Código 2

```
class Logger {
    constructor() {
        this.logs = [];
    }

    log(message) {
        this.logs.push(message);
        console.log("Log Registrado:", message);
    }

    static getInstance() {
        if (!Logger.instance) {
```

```
            Logger.instance = new Logger();
        }
        return Logger.instance;
    }
}
const logger1 = Logger.getInstance();
const logger2 = Logger.getInstance();
console.log(logger1 === logger2);  // true
logger1.log("Error: No se puede conectar al servidor");
```

En este código se aprecia una combinación de los patrones de diseño Singleton y Factory Method.

Singleton:

El patrón Singleton garantiza que una clase tenga una única instancia y proporciona un punto de acceso global a esa instancia.

1. **Instancia Única:**

La clase Logger tiene un método estático getInstance que controla la creación de la instancia única.

Dentro de getInstance, se verifica si Logger.instance existe. Si no existe, se crea una nueva instancia de Logger.

Esto asegura que solo haya una instancia de Logger durante la ejecución de la aplicación.

2. **Punto de Acceso Global:**

El método estático getInstance proporciona un punto de acceso global a la instancia única de Logger.

Factory Method:

Es un patrón de creación que define una interfaz para crear objetos, pero permite a las subclases alterar el tipo de objetos que se crearán.

En lugar de instanciar objetos directamente, el cliente llama a un método de fábrica, que se encargará de la creación de objetos.

1. **Método de Creación:**

El método **getInstance** actúa como un Factory Method al encapsular la lógica de creación de la instancia de **Logger**.

Aunque no hay subclases en este ejemplo, la idea básica es similar: **getInstance** controla cómo y cuándo se crea la instancia.

Código 3

```
class User {
    constructor(name) {
        this.name = name;
    }

    greet() {
        console.log("Hola, soy", this.name);
    }
}

class UserFactory {
    createUser(name) {
        return new User(name);
    }
}
const factory = new UserFactory();
const user1 = factory.createUser("Juan");
const user2 = factory.createUser("Maria");
user1.greet();  // Hola, soy Juan
user2.greet();  // Hola, soy Maria
```

En el último código se puede observar el patron de diseño Factory Method.

Factory Method:

El patrón Factory Method es un patrón de diseño creacional que define una interfaz para crear un objeto, pero permite a las subclases alterar el tipo de objetos que se crearán. Este patrón delega la responsabilidad de la creación de objetos a métodos especializados, conocidos como "fábricas".

1. **Clase User:**

Esta clase representa un usuario con un nombre y un método greet que imprime un saludo en la consola.

El constructor inicializa el nombre del usuario.

2. **Clase UserFactory:**

Esta clase actúa como una fábrica que encapsula la lógica de creación de objetos User.

El método createUser toma un nombre como argumento y retorna una nueva instancia de User.

3. **Uso de la Fábrica:**

Se crea una instancia de UserFactory.

Se crean dos usuarios (user1 y user2) utilizando el método createUser de UserFactory.

Ambos usuarios utilizan su método greet para saludar.