Practica de Patrones de Diseño

Bonilla Nandayapa Bryan Gersain – 307086

Universidad Autónoma de Querétaro

Facultad de Informática

Taller de Patrones de Diseño

Note:

Licenciatura, Maestro: Enrique Aguilar, Universidad Autónoma de Querétaro, Facultad de Informática

SINGLETON

```
1  class Database {
2    constructor() {
3      if (!Database.instance) {
4        Database.instance = this;
5      }
6      return Database.instance;
7    }
8
9    query(sql) {
10     console.log("Ejecutando consulta:", sql);
11   }
12 }
13
14 const db1 = new Database();
15 const db2 = new Database();
16
17 console.log(db1 === db2); // Output: true
18 db1.query("SELECT * FROM users");
```

En este código, la clase Database utiliza el patrón Singleton para asegurarse de que solo se cree una instancia de la DB, independientemente de cuántas veces se intente crear una nueva instancia. La instancia única se almacena en la propiedad instance de la clase Database, y se devuelve esa instancia en lugar de crear una nueva cuando se llama al constructor.

De esta manera, cuando se crea una nueva instancia de Database (como en las líneas 14 y 15), en realidad se devuelve la instancia única existente, lo que se verifica con la línea 17 console.log (db1 === db2); // Output: true.

SINGLETON

```
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); // Output: true
logger1.log("Error: No se puede conectar al servidor");
```

En este código, la clase Logger utiliza el patrón Singleton para garantizar que solo se cree una instancia de la clase Logger y que esa instancia sea accesible a través del método getInstance(). De esta manera, cuando se llama a Logger.getInstance() por primera vez, se crea una instancia de la clase Logger y se asigna a la variable Logger.instance. Luego, cuando se llama a Logger.getInstance() nuevamente, se devuelve la instancia existente en lugar de crear una nueva.

Esto es útil cuando se necesita una instancia única de una clase que deba ser accesible desde cualquier parte del programa, como en el caso de un logger que debe registrar mensajes en una sola instancia.

FACTORY METHOD

```
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(); // Output: Hola, soy Juan
user2.greet(); // Output: Hola, soy Maria
```

El código en si es muy corto pero la clase User define un objeto que puede saludar.

UserFactory es una clase que actúa como fábrica para crear objetos de usuario. Tiene un método createUser que toma un nombre como argumento y devuelve un nuevo objeto User con ese nombre.

Al utilizar UserFactory para crear objetos de User, el código puede crear objetos de la clase de Usuario sin tener que preocuparse por los detalles de su creación. Esto hace que el código sea más modular y fácil de mantener.