Nathalia Dos Santos Bruggemann

Parte I: Ejercicios sobre los pilares de POO

Práctica presente en los archivos .ts de la presente carpeta.

Parte II: Identificando patrones

```
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); // Output: true
db1.query("SELECT * FROM users");
```

Patrón creacional: Singleton

Se presenta una aplicación de Singleton donde una clase tiene un acceso global y siempre se regresa el mismo objeto y sus instancias al ser llamada en cualquier parte del código.

La clase Database tiene un constructor y un método para aplicar una consulta a la base de datos. En las variables db1 y db2 se llama a Database como constructor y luego se comparan pues ambas son del mismo tipo y tienen los mismos atributos.

```
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");
```

Patrón creacional: Singleton

Una clase debe tener una sola instancia que proporcione un acceso global a la misma. Tiene como características: a) un constructor privado que impide a otro objetos de usar el operador new, b) el método de creación estático que llama al constructor privado y lo guarda en una variable estática. Esto permite que siempre sea el mismo objeto el que es retornado ante el llamado de la función *singleton* en cualquier parte del código.

En este código la clase singleton es la de Logger que tiene un constructor privado y un método estático que crea un nuevo objeto Logger. Las variables acceden únicamente a getInstance para crear un Logger nuevo.

```
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
```

Patrón creacional: Factory

Permite la creación de nuevas clases con base a una ya existente permitiendo que las subclases puedan alterar el tipo de objeto que se cree. Este patrón sugiere que se reemplace la construcción directa a partir de un método nombrado *factory*.

En este código se puede visualizar a la función *factory* y como es llamada para la creación de nuevos usuarios en las variables *user1* y *user2*.