Universidad Autónoma de Querétaro
Facultad de Informática

Actividad: Análisis de código

Alumno: Duarte Fregoso Josafat Axel
Profesor: Enrique Aguilar

Fecha: 17/05/2024

Analiza el siguiente código en JS y determina que patrón de diseño se está utilizando y descríbelo

1. Singleton

```
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");
```

El patrón de diseño Singleton asegura que una clase tenga una única instancia y proporciona un punto de acceso global a dicha instancia. Este patrón es útil cuando se necesita exactamente una instancia de una clase para coordinar acciones en todo el sistema.

Implementación del Singleton

1. Propiedad estática instance:

   Database.instance es una propiedad estática que se utiliza para almacenar la única instancia de la clase Database.

2. Constructor:

   El constructor verifica si Database.instance ya ha sido definida.

   Si no está definida, asigna this a Database.instance, creando así la única instancia de la clase.

   Si ya está definida, el constructor retorna Database.instance, asegurando que siempre se retorna la misma instancia de la clase Database.

3. Método query:

   Este método es un ejemplo de una operación que puede ser realizada por la instancia de la clase Database.

4. Uso de la clase Database:

   Cuando se crean dos instancias (db1 y db2) de la clase Database, ambas referencias (db1 y db2) apuntan a la misma instancia.

   La línea console.log(db1 === db2); // Output: true demuestra que ambas variables (db1 y db2) son de hecho la misma instancia.

Beneficios del Patrón Singleton

- Control de acceso a la única instancia: Permite tener control preciso sobre cómo y cuándo se accede a la única instancia.
- Ahorro de recursos: Garantiza que solo se crea una instancia de la clase, lo que puede ser útil para ahorrar memoria en aplicaciones que requieren muchos recursos.
- Consistencia: Al tener una única instancia, se asegura que el estado sea consistente en toda la aplicación.

Consideraciones

- Problemas con pruebas unitarias: Los Singletons pueden ser difíciles de probar porque introducen una dependencia global en el estado de la aplicación.
- Problemas de concurrencia: En un entorno multihilo, asegurar que una clase tenga una única instancia puede ser complicado y podría requerir técnicas adicionales como sincronización.

2. 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");
```

El patrón de diseño Singleton asegura que una clase tenga una única instancia y proporciona un punto de acceso global a dicha instancia. Este patrón es útil cuando se necesita exactamente una instancia de una clase para coordinar acciones en todo el sistema.

Implementación del Singleton

1. Propiedad estática instance:

   Logger.instance es una propiedad estática que se utiliza para almacenar la única instancia de la clase Logger.

2. Constructor:

   El constructor inicializa la propiedad logs como un array vacío.

   No hay restricciones en el constructor para limitar la creación de instancias, ya que el control se realiza en el método estático getInstance.

3. Método log:

   Este método agrega un mensaje al array logs y lo registra en la consola.

4. Método estático getInstance:

Este método es responsable de controlar el acceso a la instancia única de la clase Logger.

Verifica si Logger.instance está definida. Si no es así, crea una nueva instancia de Logger y la asigna a Logger.instance.

Retorna la instancia única de Logger.

5. Uso de la clase Logger:

   Cuando se crean dos instancias (logger1 y logger2) utilizando Logger.getInstance(), ambas referencias (logger1 y logger2) apuntan a la misma instancia.

   La línea console.log(logger1 === logger2); // Output: true demuestra que ambas variables (logger1 y logger2) son de hecho la misma instancia.

Diferencia con el patrón anterior

La segunda implementación (Logger) es más típica y clara en el uso del patrón Singleton, ya que encapsula el control de la instancia dentro de un método estático dedicado (getInstance), lo que puede hacer que el código sea más fácil de entender y mantener.

3. 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 patrón Factory Method es un patrón de diseño creacional que proporciona una interfaz para crear objetos en una clase base, pero permite a las subclases alterar el tipo de objetos que se crearán. En este caso, la fábrica (UserFactory) es responsable de crear instancias de la clase User.

Implementación del Factory Method

1. Clase User:

La clase User tiene una propiedad name y un método greet que imprime un saludo utilizando el nombre del usuario.

2. Clase UserFactory:

La clase UserFactory tiene un método createUser que toma un nombre como parámetro y retorna una nueva instancia de la clase User con ese nombre.

3. Uso de la fábrica:

Se crea una instancia de UserFactory.

Se utilizan los métodos createUser de la instancia UserFactory para crear instancias de User.

Beneficios del Patrón Factory Method

- Desacoplamiento: Desacopla el código de creación del objeto del código que utiliza el objeto, lo que puede hacer que el código sea más flexible y fácil de mantener.
- Extensibilidad: Facilita la adición de nuevos tipos de objetos a la fábrica sin modificar el código existente.

Referencias

- Wikipedia. (n.d.). Singleton pattern. Retrieved May 17, 2024, from https://en.wikipedia.org/wiki/Singleton_pattern
- Refactoring Guru. (n.d.). Singleton pattern. Retrieved May 17, 2024, from https://refactoring.guru/design-patterns/singleton
- Wikipedia. (n.d.). Factory method pattern. Retrieved May 17, 2024, from https://en.wikipedia.org/wiki/Factory_method_pattern
- Refactoring Guru. (n.d.). Factory Method pattern. Retrieved May 17, 2024, from https://refactoring.guru/design-patterns/factory-method