UADER | FCyT
Universidad Autónoma de Entre Ríos
Facultad de Ciencia y Tecnología

Ejercicio 1: Configuración global de una aplicación

Problema: Imagina que estás desarrollando una aplicación y necesitas una clase que almacene la configuración global, como el idioma, el tema de la aplicación, etc. Quieres garantizar que solo haya una instancia de esta clase en toda la aplicación.

Ejercicio 2: Registro de eventos en una aplicación
Problema: En una aplicación, necesitas un registro de eventos para registrar errores y eventos importantes en diferentes partes del código. Quieres que todas las partes de la aplicación compartan el mismo registro de eventos.

Ejercicio 3: Administrador de bases de datos

Problema: Estás desarrollando un sistema que se conecta a una base de datos y necesitas garantizar que solo haya una instancia del administrador de la base de datos en toda la aplicación.

Ejercicio 1: Configuración global de una aplicación

Solución con Explicación: En este ejercicio, necesitamos garantizar que solo haya una instancia de la clase `AppConfig` que almacena la configuración global de la aplicación. Para lograrlo, utilizamos el patrón Singleton. En la clase `AppConfig`, hemos definido un atributo estático `_instance` que almacenará la única instancia de la clase.

El método `__new__` se utiliza para crear nuevas instancias de la clase, pero hemos personalizado este método para que verifique si ya existe una instancia (`_instance`). Si no existe, crea una nueva instancia; de lo contrario, devuelve la instancia existente. Esto asegura que siempre haya una única instancia de `AppConfig`.

Ejercicio 2: Registro de eventos en una aplicación

Solución con Explicación: En este ejercicio, necesitamos crear un registro de eventos para registrar eventos importantes en la aplicación. Queremos que todas las partes de la aplicación compartan el mismo registro de eventos, por lo que utilizamos el patrón Singleton.

La clase `EventLogger` tiene un atributo estático `_instance` que almacena la única instancia de la clase. Al personalizar el método `__new__`, verificamos si ya existe una instancia. Si no existe, creamos una nueva instancia y, si ya existe, devolvemos la instancia existente. De esta manera, todas las partes de la aplicación utilizan el mismo registro de eventos.

Ejercicio 3: Administrador de bases de datos

Solución con Explicación: En este ejercicio, necesitamos crear un administrador de bases de datos que se conecte a una base de datos y asegurar que solo haya una instancia de este administrador en toda la aplicación. Utilizamos el patrón Singleton para lograr esto.

La clase `DatabaseManager` tiene un atributo estático `_instance` que almacena la única instancia de la clase. Al personalizar el método `__new__`, verificamos si ya existe una instancia. Si no existe, creamos una nueva instancia y, si ya existe, devolvemos la instancia existente. De esta manera, garantizamos que todas las partes de la aplicación compartan la misma conexión de base de datos a través del administrador de bases de datos único.

En cada solución, el patrón Singleton asegura que haya una única instancia de la clase correspondiente en toda la aplicación, lo que es útil para administrar recursos globales y evitar la creación de instancias innecesarias.

Ejercicio 1: Configuración global de una aplicación

Solución UML:

plaintext Copy code

```
+---------------------------------------+
|            AppConfig                  |
+---------------------------------------+
| - instance: AppConfig                 |
| - language: String                    |
| - theme: String                       |
+---------------------------------------+
| + getInstance(): AppConfig            |
| + initConfig(): void                  |
| + getLanguage(): String               |
| + setLanguage(language: String): void |
| + getTheme(): String                  |
| + setTheme(theme: String): void       |
+---------------------------------------+
```

Explicación:

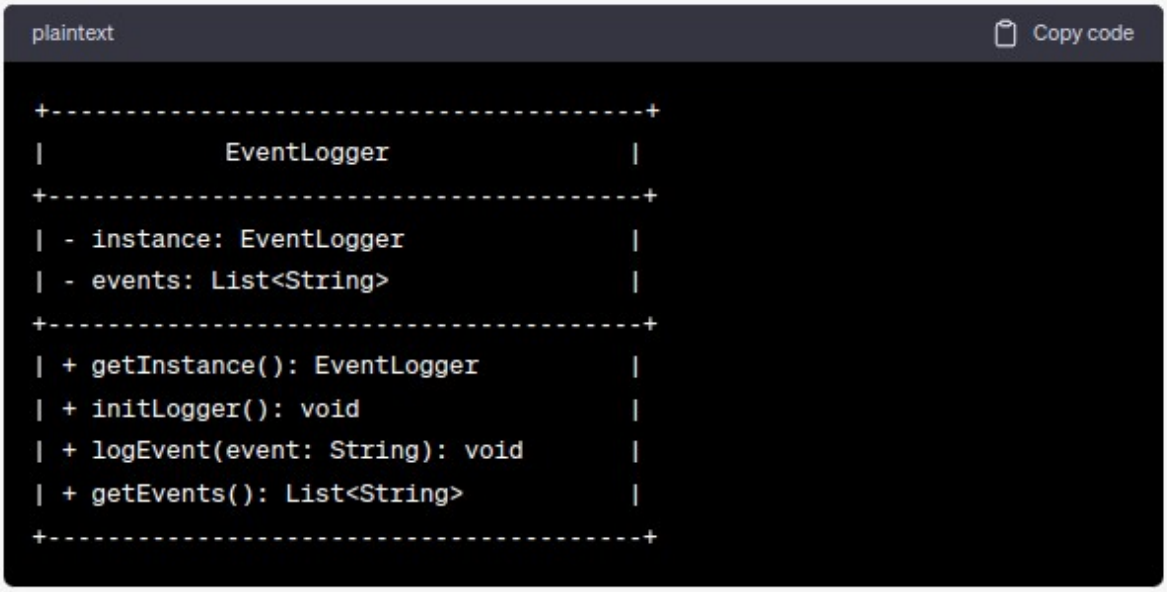
- `AppConfig` es la clase Singleton que almacena la configuración global.
- `instance` es un atributo privado estático que almacena la única instancia de `AppConfig`.
- `language` y `theme` son propiedades de configuración.
- `getInstance()` es un método estático que devuelve la instancia única de `AppConfig`.
- `initConfig()` inicializa la configuración.
- `getLanguage()` y `setLanguage()` permiten obtener y establecer el idioma.
- `getTheme()` y `setTheme()` permiten obtener y establecer el tema.

Regenerate

Ejercicio 2: Registro de eventos en una aplicación

Solución UML:

plaintext Copy code



- **Explicación:**
  - `EventLogger` es la clase Singleton que registra eventos.
  - `instance` es un atributo privado estático que almacena la única instancia de `EventLogger`.
  - `events` es una lista que almacena los eventos registrados.
  - `getInstance()` es un método estático que devuelve la instancia única de `EventLogger`.
  - `initLogger()` inicializa el registro de eventos.
  - `logEvent(event)` registra un evento en la lista.
  - `getEvents()` devuelve la lista de eventos registrados.

Ejercicio 3: Administrador de bases de datos

Solución UML:

plaintext Copy code

```
+---------------------------------------+
|        DatabaseManager                |
+---------------------------------------+
| - instance: DatabaseManager           |
| - connection: String                  |
+---------------------------------------+
| + getInstance(): DatabaseManager      |
| + initDatabase(): void                |
| + connect(databaseUrl: String): void  |
| + getConnection(): String             |
+---------------------------------------+
```

- **Explicación:**
  - `DatabaseManager` es la clase Singleton que administra la conexión de la base de datos.
  - `instance` es un atributo privado estático que almacena la única instancia de `DatabaseManager`.
  - `connection` almacena la cadena de conexión a la base de datos.
  - `getInstance()` es un método estático que devuelve la instancia única de `DatabaseManager`.
  - `initDatabase()` inicializa la base de datos.
  - `connect(databaseUrl)` establece la conexión de la base de datos.
  - `getConnection()` devuelve la cadena de conexión actual.