Patrón Singleton en un Servicio de Registro (Logger)

Frida Cano Falcón

Academia Java MTY
Agosto 2024

Viernes 16 de agosto de 2024

Introducción al Patrón Singleton

El patrón Singleton es un patrón de diseño que garantiza que una clase tenga solo una instancia y proporciona un punto de acceso global a esa instancia. Es útil cuando se necesita una única instancia de una clase para coordinar acciones a lo largo del sistema. El servicio de registro (Logger) es un buen ejemplo para explicar el patrón Singleton debido a su naturaleza.

¿Por qué el Logger es un buen ejemplo para el Singleton?

Instancia Única: Un servicio de registro (Logger) necesita asegurar que solo exista una instancia en toda la aplicación. Tener múltiples instancias de un Logger puede llevar a inconsistencias en los mensajes de registro, duplicaciones o problemas en la gestión de archivos de registro.

Acceso Global: El patrón Singleton proporciona un punto de acceso global a la instancia del Logger. Esto significa que cualquier parte de la aplicación puede registrar mensajes sin tener que crear o gestionar múltiples instancias del Logger.

Gestión de Recursos: La creación de instancias múltiples de Logger podría resultar en la apertura de múltiples archivos de registro o flujos de salida, lo cual no es eficiente. El Singleton asegura que solo se gestione un recurso de salida para los mensajes de log.

Inicialización Controlada: El Singleton asegura que la instancia del Logger se inicializa solo una vez. Esto es importante si el Logger requiere configuración compleja, como establecer rutas de archivo o formatos de registro.

Seguridad en Entornos Multihilo: En aplicaciones con múltiples hilos, el patrón Singleton asegura que el Logger sea seguro para el acceso concurrente. Esto evita problemas como condiciones de carrera y entradas de log inconsistentes.

Implementación en Java

Clase - Logger

src > com > curso > tarea > Logger.java > Logger > log(String)

```java
package com.curso.tarea;

public class Logger {

    private static Logger instance;

    // Constructor privado previene la instanciación
    private Logger() {
    }

    // Método para obtener la instancia única
    public static Logger getInstance() {
        if (instance == null) {
            instance = new Logger();
        }
        return instance;
    }

    // Método para registrar un mensaje
    public void log(String message) {
        System.out.println("LOG: " + message);
    }
}

```

Codificación de la clase Logger

Private Constructor (private Logger()):

El constructor es privado para evitar que cualquier código fuera de la clase Logger pueda crear una nueva instancia de esta clase. Esto asegura que la única instancia del Logger sea controlada dentro de la misma clase. Al restringir la creación de instancias, el patrón Singleton garantiza que no se generen múltiples objetos Logger, manteniendo la unicidad de la instancia.

Static Instance Variable (private static Logger instance):

La variable instance es estática y almacena la única instancia del Logger que será creada. Al ser estática, esta variable pertenece a la clase en lugar de a un objeto en particular, permitiendo que la instancia del Logger sea compartida a lo largo de toda la aplicación.

Synchronized getInstance() Method:

El método getInstance() es sincronizado para asegurar que sólo una instancia del Logger sea creada, incluso si múltiples hilos intentan acceder a ella al mismo tiempo. La sincronización previene problemas de condición de carrera, asegurando que la instancia se inicialice de manera segura y correcta.

Global log() Method:

El método log() proporciona una interfaz simple para registrar mensajes desde cualquier parte de la aplicación. Esto demuestra cómo todas las partes de un sistema pueden acceder y utilizar la misma instancia del Logger para mantener un registro consistente y centralizado de los eventos.

Ejemplo de Uso en la Clase Principal

En este ejemplo, la clase Principal utiliza la instancia del Logger para registrar mensajes en diferentes etapas de la aplicación.

src > com > curso > tarea > Principal.java > Principal > main(String[])

```java
package com.curso.tarea;

public class Principal {
    Run | Debug
    public static void main(String[] args) {
        // Get the single instance of Logger
        Logger logger = Logger.getInstance();

        // Log some messages
        logger.log(message:"La aplicación ha iniciado.");
        logger.log(message:"Procesando datos...");
        logger.log(message:"La aplicación ha finalizado.");
    }
}
```

Codificación de la clase Principal

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  COMMENTS  DEBUG CONSOLE

PS C:\Users\HP\Documents\GitHub\EntregablesJavaAcademy2024\Semana 1\Singleton>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\HP\Documents\GitHub\EntregablesJavaAcademy2024\Semana 1\Singleton\bin' 'com.curso.tarea.Principal'
LOG: La aplicación ha iniciado.
LOG: Procesando datos...
LOG: La aplicación ha finalizado.
```

Resultados de la terminal

Ejemplo Verificación de la Instancia Única

```
src > com > curso > tarea2 > Principal.java > Principal > main(String[])
1   package com.curso.tarea2;
2
3   public class Principal {
        Run | Debug
4       public static void main(String[] args) {
5           // Obtener instancias del Logger
6           Logger logger1 = Logger.getInstance();
7           Logger logger2 = Logger.getInstance();
8
9           // Comprobar si las instancias son iguales
10          if (logger1 == logger2) {
11              System.out.println(x:"Ambas instancias son iguales, confirmando el patrón Singleton.");
12          } else {
13              System.out.println(x:"Las instancias son diferentes, lo cual indica un problema en el Singleton.");
14          }
15      }
16  }
17
```

Codificación de la clase Principal

```
PS C:\Users\HP\Documents\GitHub\EntregablesJavaAcademy2024\Semana 1\Singleton>  c:; cd 'c:\Users\HP\Documents\GitHub\EntregablesJavaAcademy2024\Semana 1\Singleton'; & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\HP\Documents\GitHub\EntregablesJavaAcademy2024\Semana 1\Singleton\bin' 'com.curso.tarea2.Principal'

Ambas instancias son iguales, confirmando el patrón Singleton.
```

Resultados de la terminal

Conclusión

El patrón Singleton es ideal para servicios como el Logger, donde se necesita una única instancia a lo largo de toda la aplicación. Al implementar el Singleton, garantizamos que el Logger maneje los mensajes de manera consistente y eficiente, evitando problemas asociados con múltiples instancias y facilitando la gestión de recursos.