Singleton Pattern Implementation for Logger in Java

Logger.java (Singleton Implementation)

public class Logger {

private static Logger instance;

private Logger() {

System.out.println("Logger instance created");

}

public static Logger getInstance() {

if (instance == null) {

synchronized (Logger.class) {

if (instance == null) {

instance = new Logger();

}

}

}

return instance;

}

public void logInfo(String message) {

System.out.println("[INFO] " + message);

}

public void logError(String message) {

System.out.println("[ERROR] " + message);

}

public void logWarning(String message) {

System.out.println("[WARNING] " + message);

}

}

LoggerTest.java (Test Class)

public class LoggerTest {

public static void main(String[] args) {

// Test the singleton pattern

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

System.out.println("Are logger1 and logger2 the same instance? " + (logger1 == logger2));

logger1.logInfo("Application started");

logger2.logWarning("Low memory detected");

logger1.logError("Failed to connect to database");

System.out.println("\nTesting with multiple threads:");

Runnable task = () -> {

Logger threadLogger = Logger.getInstance();

threadLogger.logInfo("Logging from thread " + Thread.currentThread().getId());

};

Thread thread1 = new Thread(task);

Thread thread2 = new Thread(task);

Thread thread3 = new Thread(task);

thread1.start();

thread2.start();

thread3.start();

try {

thread1.join();

thread2.join();

thread3.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

Key Features of This Implementation

* **Thread-safe Singleton**: Uses double-checked locking for thread safety
* **Lazy Initialization**: Instance is created only when first needed
* **Private Constructor**: Prevents external instantiation
* **Global Access Point**: Provides a single access point via getInstance().

