**Exercise 1: Implementing the Singleton Pattern**

**Scenario:**

You need to ensure that a logging/audit utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **SingletonDemo**.
2. **Define a Singleton Class:**
   * Create a class named AuditTrail that has a private static instance of itself.
   * Ensure the constructor of Logger is private.
   * Provide a **public static method** to return the single instance.
3. **Implement the Singleton Pattern:**
   * The AuditTrail class uses lazy initialization to ensure only **one object is created**.
4. **Test the Singleton Implementation:**
   * In the SingletonDemo class, use the getInstance() method to obtain and use the same instance of AuditTrail.

**Source Code :**

package week1.designpatterns;  
  
class AuditTrail  
{  
 // Singleton instance variable  
 private static AuditTrail instance;  
  
 // Private constructor to restrict instantiation  
 private AuditTrail()  
 {  
 System.out.println("AuditTrail instance created.");  
 }  
  
 // Method to get the singleton instance  
 public static AuditTrail getInstance()  
 {  
 if (instance == null)  
 {  
 instance = new AuditTrail();  
 }  
 return instance;  
 }  
  
 public void record(String event)  
 {  
 System.out.println("Audit Record: " + event);  
 }  
}  
  
public class SingletonDemo  
{  
  
 public static void main(String[] args)  
 {  
 AuditTrail audit1 = AuditTrail.getInstance();  
 AuditTrail audit2 = AuditTrail.getInstance();  
  
 audit1.record("User logged in.");  
 audit2.record("User performed an action.");  
  
 if (audit1 == audit2)  
 {  
 System.out.println("Both audit instances are the same.");  
 }  
 else  
 {  
 System.out.println("Different audit instances exist.");  
 }  
 }  
}

Output :

Logger instance created.

Log: This is the first message.

Log: This is the second message.

logger instances are the same