1. I have chosen the **Singleton Design Pattern**. This pattern ensures that a class has only **one instance** and provides a **global access point** to it. It is commonly used in scenarios where creating multiple instances can lead to inefficiencies or inconsistencies, such as **database connections, logging systems, and configuration settings**

2. A Database Connection Manager is a perfect example where the Singleton pattern is beneficial. If multiple instances of a database connection are created, it may cause unnecessary resource consumption and performance issues. Using Singleton ensures that only one connection instance exists throughout the application.

```
class DatabaseConnection {
    private static DatabaseConnection instance;

    private DatabaseConnection() {
        System.out.println("Database Connection Established");
    }

    public static DatabaseConnection getInstance() {
        if (instance == null) {
            instance = new DatabaseConnection();
        }
        return instance;
    }

    public void connect() {
        System.out.println("Using Database Connection...");
    }
}


public class SingletonExample {
    public static void main(String[] args) {
        DatabaseConnection db1 = DatabaseConnection.getInstance();
        DatabaseConnection db2 = DatabaseConnection.getInstance();

        db1.connect();

        // Checking if both references point to the same instance
        System.out.println(db1 == db2);
    }
}
```