Implementing the Singleton Pattern

Definition : The Singleton Pattern is a design pattern that ensures a class has only one instance throughout the application and provides a global point of access to that instance.

Understanding Singleton :

- I understood the **Singleton Design Pattern** more clearly through a **cricket scorecard example**.
- In a live match, the **same scorecard** must be visible across the **commentary box**, **statistics panel**, and the **main homepage**.
- If each section had its own instance, inconsistent scores could appear.
- Hence, only **one shared instance** should exist — this is exactly what Singleton ensures.

My Implementation :

In my Logger class:

- I used a **private constructor** to restrict direct object creation.
- A **static method getInstance()** controls access and ensures only one object is ever created.
- Additional calls reuse the same object and ignore new data.

This confirms that the Logger behaves like a **singleton**: one consistent source used everywhere.

What I Learned :

- Singleton is useful when only **one consistent object** should manage data or behavior (e.g., logging, scoreboards, settings).
- I now understand how to **control object creation** and ensure consistency across different parts of an application.

There are multiple ways to implement the Singleton pattern in Java, such as:

- Eager Initialization
- Lazy Initialization
- Thread-safe Singleton
- Enum-based Singleton

I have prepared a handwritten explanation of the Singleton pattern covering different implementations and insights.

To view the notebook:

- **Click here to view it on GitHub**
- Or refer to the attached file: **SingletonPattern_Handwritten.pdf (Page Number : 2-5)**