Event Bus or Messaging System

The Singleton Design Pattern is best used when you need exactly one instance of a class globally accessible and consistently shared across the application. It's ideal for managing shared resources or global state in a controlled way.

Best Use Cases

- Configuration Management

A class that loads and provides access to configuration values (e.g., from application.properties or YAML files).

- Logging

A single, centralized logger ensures consistency in how logs are written and prevents redundant output streams.

- Thread Pools / Executor Services

Managing a shared ExecutorService in a multi-threaded application ensures efficient resource usage.

- Database Connection Pooling (Manager)

A connection pool manager that all DAO classes share to ensure consistent connection reuse.

- Caching (In-Memory Cache Manager)

A singleton in-memory cache store (e.g., using Map) that's accessible throughout the application.

- Device or Hardware Access Layer

Coordinate access to a single device like a print spooler or file system manager in embedded systems.

- Service Registry / Lookup

A central registry to look up services or dependencies, often used in lightweight DI containers.

- Event Bus or Messaging System

A singleton event dispatcher ensures all components publish and subscribe to the same event pipeline.

When Not to Use Singleton

- When global state introduces tight coupling or breaks testability.
- When the class needs to be instantiated differently in unit tests (consider dependency injection instead).