**Singleton Pattern: Database Connection**

**Description:**

The **Singleton Pattern** is a creational design pattern that ensures a class has only one instance and provides a global point of access to that instance. This pattern is useful when exactly one object is needed to coordinate actions across the system, such as managing a database connection.

**Components:**

1. **Singleton Class (DatabaseConnection):**  
   Contains a private static instance of the class and a private constructor to prevent direct instantiation. Provides a public static method to access the unique instance.
2. **Client Code (SingletonPatternDemo):**  
   Uses the singleton class to access the single instance and interact with it, ensuring that all operations are performed on the same object.

**How It Works:**

1. **Private Constructor:** The constructor of the singleton class is private, preventing other classes from instantiating new objects directly.
2. **Static Instance:** A private static variable holds the single instance of the class. It is initialized when first accessed.
3. **Public Access Method:** A public static method (getInstance) provides access to the singleton instance. If the instance does not exist, it is created. Subsequent calls return the existing instance.
4. **Global Access:** Clients access the singleton instance via the static method, ensuring that all operations use the same object.

**Real-Life Example: Database Connection**

In a database connection scenario:

* **DatabaseConnection (Singleton Class):** Manages a single connection to the database. It ensures that only one instance of the connection is created and used throughout the application.
* **Client Code (SingletonPatternDemo):** Requests access to the database connection using the getInstance method. Any database operations are performed on the same connection instance.

**Scenario:**

1. **Initial Request:** The client calls DatabaseConnection.getInstance() to obtain a connection. Since no instance exists, it is created, and the connection to the database is established.
2. **Subsequent Requests:** Any further calls to getInstance() return the existing instance, which is the same as the one created initially.
3. **Query Execution:** The client performs operations (queries) using the connection. Both db1 and db2 refer to the same connection instance, ensuring consistent access to the database.

**Benefits:**

1. **Controlled Access:** Ensures that only one instance of the class is created, providing a controlled point of access.
2. **Resource Management:** Prevents multiple instances that could lead to resource waste, such as multiple database connections.
3. **Global Access:** Provides a global point of access to the instance, simplifying the management of shared resources.