1.**Intent**

The intent of the code is to demonstrate the **Factory Design Pattern** for dynamically creating instances of different database connection types. The main goal is to decouple the logic of creating a database connection from the main logic of interacting with the database, thereby allowing flexibility for future extensions to support additional database types.

The code allows users to specify a database type (e.g., MySQL, PostgreSQL, or SQLite) and dynamically connect and disconnect from the respective database. The Factory Design Pattern helps centralize the logic of creating database connections while maintaining loose coupling between the database connection classes and the main program.

**2. Role of Participants**

* **Main Class**: This class is the user interface that prompts the user for a database name (e.g., "mysql", "postgresql", "sqlite"). Based on the user input, it calls the FactoryDatabase class to get an appropriate DatabaseConnection instance, and then calls connect() and disconnect() methods on it
* **FactoryDatabase Class**: The FactoryDatabase class creates and returns instances of the correct database connection based on the database name provided by the user. It ensures that only one type of database connection is instantiated at a time.
* **MySQL, PostgreSQL, SQLite Classes**: These classes provide the specific implementation for connecting to and interacting with the respective databases. They implement the DatabaseConnection interface and handle queries and connections related to their respective database systems.

**3. Mapping of Participants with Actual Classes**

|  |  |
| --- | --- |
| **Participant** | **Class/Interface** |
| **Main User Interface** | Main |
| **Database Factory** | FactoryDatabase |
| **MySQL Database** | mysql |
| **PostgreSQL Database** | postgresql |
| **SQLite Database** | sqlite |

|  |  |
| --- | --- |
|  |  |
|  | 4. ****Consequences****Positive Consequences:  1. **Flexibility**: The design is easily extendable to support additional database types. New database classes (e.g., Oracle, SQL Server) can be added by implementing the DatabaseConnection interface and updating the FactoryDatabase class to support these new types. 2. **Loose Coupling**: The main class only interacts with the factory and the DatabaseConnection interface, making it unaware of the actual database classes. This reduces the dependencies between the main program and the database-specific code. 3. **Separation of Concerns**: The factory handles the creation of database connection objects, while the database classes manage their own connection and disconnection logic. This separation improves code readability and maintainability.  Negative Consequences:  1. **Factory Growth**: As more databases are added, the FactoryDatabase class may grow in size, making it harder to maintain. One possible improvement is to use configuration files or a more dynamic mapping mechanism. 2. **Redundancy**: Each database class contains its own implementation for connecting and disconnecting, meaning there might be repeated code for managing connections, error handling, and queries. Centralizing some common functionality could reduce redundancy.   5. **Structure Diagram:** |
|  |  |
|  |  |
|  |  |

6. Class Diagram:

