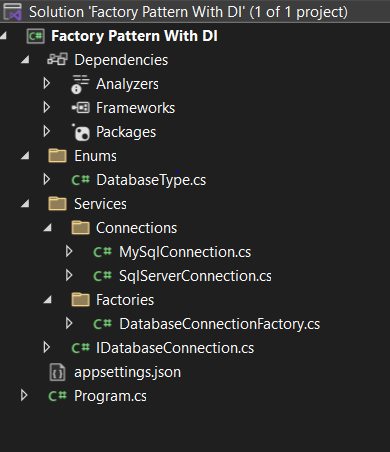
Factory Pattern with Dependency Injection (DI) for Dynamic Connection Strings in C#

**Introduction**

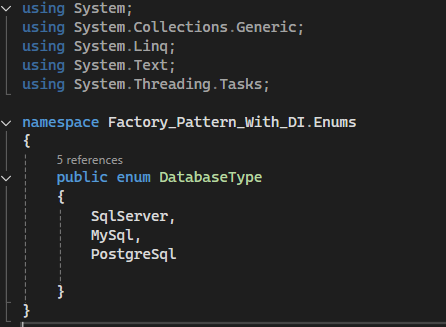
**Factory Pattern** is a design pattern of the creational kind that offers an interface for the creation of objects in a superclass but permits the subclasses to extend the type of objects to be created.  
When used in conjunction with **Dependency Injection (DI)**, we can inject dependencies like database connection strings dynamically from a configuration file **(appsettings.json)** rather than having to hardcode them.  
This paper describes the usage of a **Factory Pattern** with **DI** to dynamically select database type and connection string at runtime for dynamic database connections in C#.

**Project Structure**

**Factory Pattern With DI**

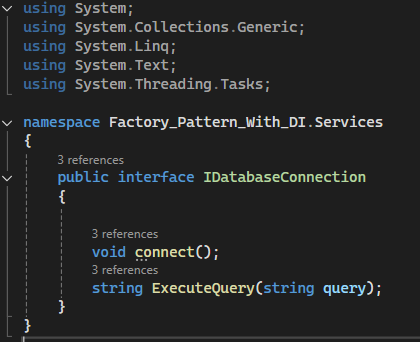


**Step 1: Define Database Type Enum**

The DatabaseType enum specifies various database types that are supported in the factory. This makes it easy to add support for additional databases in the future.

**Step 2: Define the Interface for Database Connections**

The IDatabaseConnection interface gives us a contract that all database connection classes need to implement. This provides a uniform structure for various database implementations.

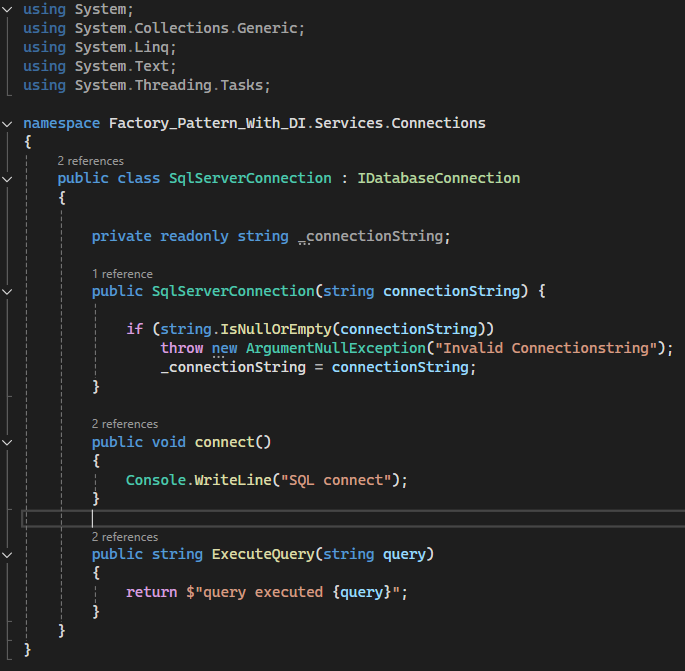


**Step 3: Implement Database Connection Classes**

Each class implements IDatabaseConnection for a specific database type.

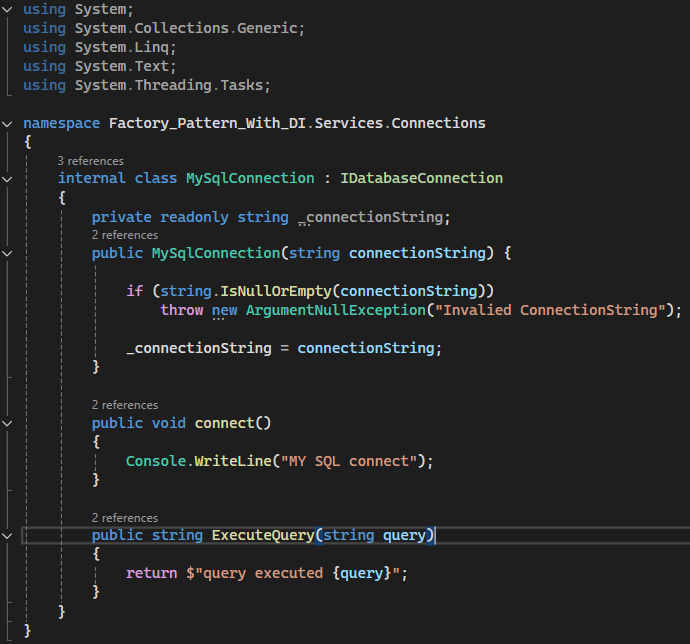
**SQL Server Connection Implementation**

This class handles database connection logic specific to SQL Server. It receives a connection string through dependency injection.



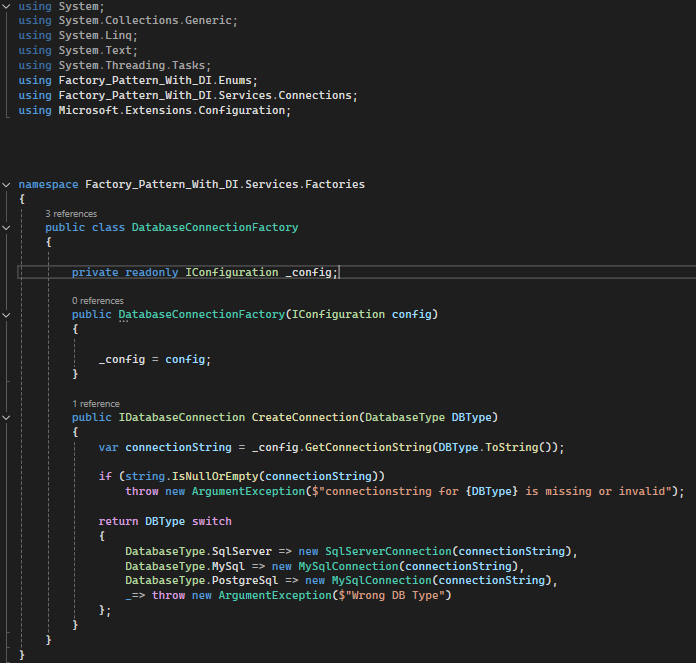
**MySQL Connection Implementation**

Similar to the SQL Server implementation, but for MySQL.



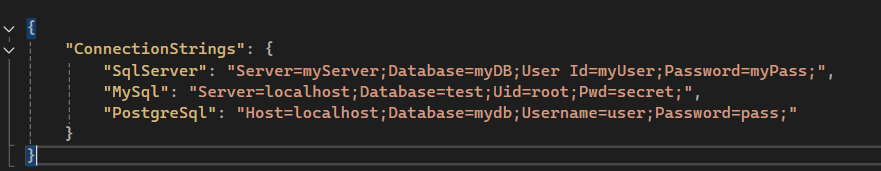
**Step 4: Implement the Factory Class**

The DatabaseConnectionFactory class dynamically creates database connection instances based on the DatabaseType enum. This eliminates the need for hardcoding database-specific logic elsewhere in the codebase.



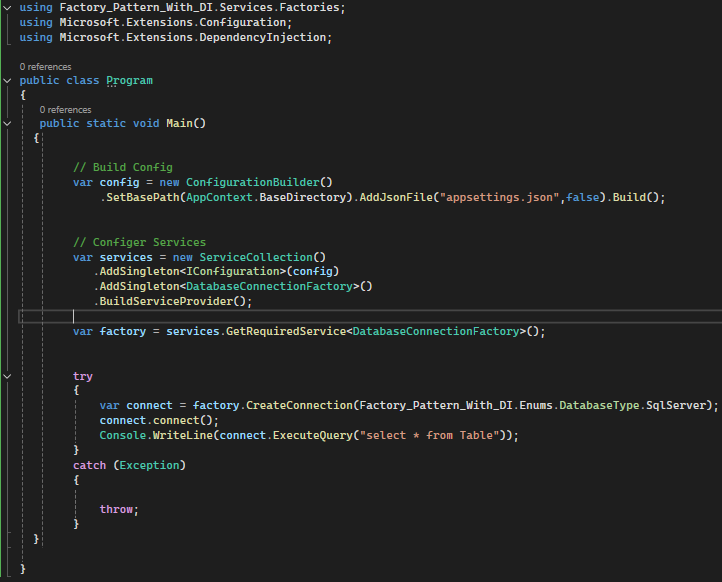
**Step 5: Configuration File (appsettings.json)**

This file stores connection strings for different databases, allowing them to be updated without modifying the code.



**Step 6: Implement the Main Program**

The Program.cs file sets up dependency injection, retrieves the required database connection, and executes a query.



**Conclusion**

With Factory Pattern and Dependency Injection, we get:

1. Loose coupling: Factory dynamically generates objects without the need for hardcoding dependencies.
2. Scalability: Add new database types without changing existing logic.
3. Dynamic Configuration: Connection strings are handled externally in appsettings.json.
4. Dependency Injection: Implements DI to handle configurations in a clean manner.

This method makes it possible to have a maintainable and adaptable solution for working with multiple database connections in C# applications.