**Exercise 11: Implementing Dependency Injection**

**Overview**

This implementation demonstrates the Dependency Injection (DI) pattern in a customer management system. It showcases how DI can be used to decouple the high-level modules from the low-level modules, promoting loose coupling and improving flexibility and testability.

Implementation

Link: [LINK](https://github.com/Akashmondal55/Akash_5016855/tree/main/Week-1/Design%20patern%20and%20princple/Exercise-11)

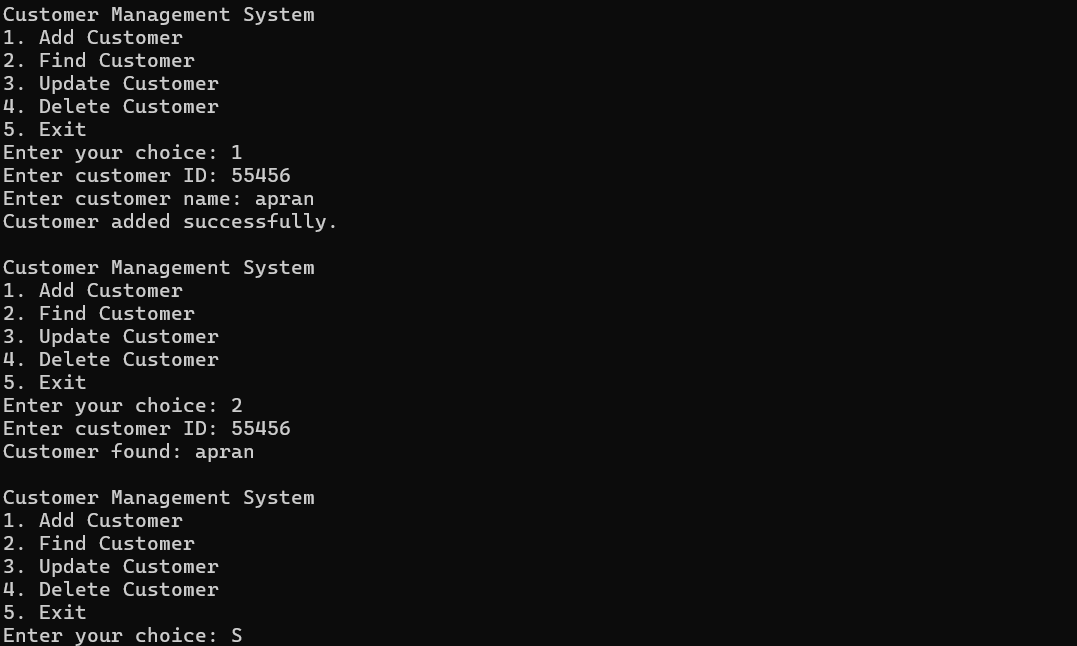
**How It Works**

1. The CustomerRepository interface defines the contract for customer data operations.
2. CustomerRepositoryImpl provides a concrete implementation of this interface.
3. CustomerService depends on the CustomerRepository interface, not on a concrete implementation.
4. In DependencyInjectionTest, we create an instance of CustomerRepositoryImpl and inject it into CustomerService.
5. The main method provides a menu-driven interface for the user to interact with the customer management system.

**Benefits of This Implementation**

1. **Loose Coupling**: CustomerService is not tightly coupled to a specific implementation of CustomerRepository.
2. **Flexibility**: We can easily switch to a different implementation of CustomerRepository (e.g., database-backed) without changing CustomerService.
3. **Testability**: We can easily mock CustomerRepository for unit testing CustomerService.
4. **Separation of Concerns**: Each class has a clear, single responsibility.
5. **Potential Improvements**
6. Use a Dependency Injection container (e.g., Spring) for more complex applications.
7. Implement error handling and input validation.
8. Add a persistent storage mechanism instead of in-memory storage.
9. Implement logging for better debugging and monitoring.

Output



**Conclusion**

This implementation demonstrates the power of Dependency Injection in creating loosely coupled, flexible, and testable code. By depending on abstractions (interfaces) rather than concrete implementations, we've created a system that's easy to modify and extend. The user-defined aspect allows for interactive testing of the system, showcasing how DI works in a practical scenario.