**Exercise 11: Implementing Dependency Injection**

**Scenario:**

You are developing a customer management application where the service class depends on a repository class. Use Dependency Injection to manage these dependencies.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **DependencyInjectionExample**.
2. **Define Repository Interface:**
   * Create an interface **CustomerRepository** with methods like **findCustomerById()**.
3. **Implement Concrete Repository:**
   * Create a class **CustomerRepositoryImpl** that implements **CustomerRepository**.
4. **Define Service Class:**
   * Create a class **CustomerService** that depends on **CustomerRepository**.
5. **Implement Dependency Injection:**
   * Use constructor injection to inject **CustomerRepository** into **CustomerService**.
6. **Test the Dependency Injection Implementation:**
   * Create a main class to demonstrate creating a **CustomerService** with **CustomerRepositoryImpl** and using it to find a customer.

**Solution:**

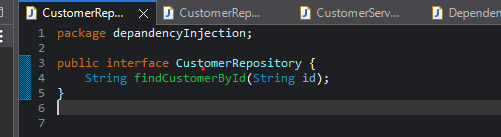
To develop a simple Customer Management application using Constructor-based Dependency Injection (DI) in Java.

| **Component** | **Role** | **Description** |
| --- | --- | --- |
| **CustomerRepository** | Repository Interface | Declares method findCustomerById() |
| **CustomerRepositoryImpl** | Concrete Repository Implementation | Implements data access logic |
| **CustomerService** | Service Class | Depends on the repository to perform operations |
| **Main** | Test Class | Demonstrates how dependencies are injected and used |

**Code:**

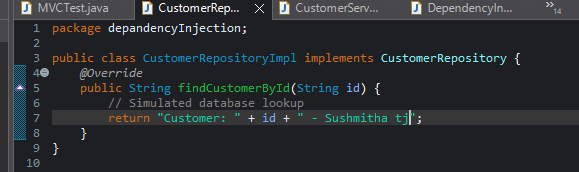
**🔹 1. Repository Interface: CustomerRepository.java**

Defines the contract for accessing customer data

****

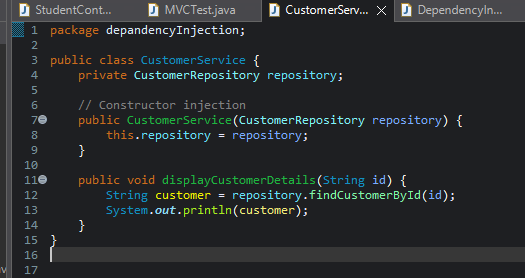
**🔹 2. Concrete Repository: CustomerRepositoryImpl.java**

* Simulates a database access method.
* Returns dummy data for demonstration



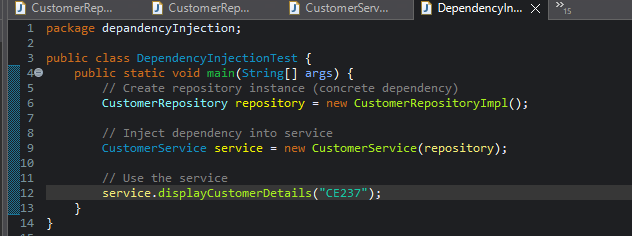
**🔹 3. Service Class: CustomerService.java**

* The repository is injected through the constructor.
* Service is decoupled from the specific implementation of the repository.

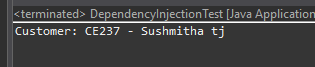
****

**🔹 4. Main/Test Class: DependencyInjectionTest.java**

* Demonstrates manual constructor-based DI.
* The service doesn't create the repository—it receives it from outside, promoting loose coupling.



**Output:**

****

**Benefits of Dependency Injection**

* ✔ Loose Coupling – Components are independent and easy to replace/test.
* ✔ Easier Testing – You can pass mock repositories for unit tests.
* ✔ Cleaner Code – Business logic is separate from object creation.
* ✔ Highly Scalable – Frameworks like Spring use this at scale.