**Exercise 11: Implementing Dependency Injection**

**CustomerRepository.java**

**package** mypackage;

**public** **interface** CustomerRepository {

String findCustomerById(String id);

}

**Implement Concrete Repository**

**CustomerRepositoryImpl.java**

**package** mypackage;

**import** java.util.HashMap;

**import** java.util.Map;

**public** **class** CustomerRepositorylmpl **implements** CustomerRepository {

**private** Map<String, String> customerData;

**public** CustomerRepositorylmpl() {

customerData = **new** HashMap<>();

customerData.put("C001", "Alice");

customerData.put("C002", "Bob");

customerData.put("C003", "Charlie");

}

@Override

**public** String findCustomerById(String id) {

**return** customerData.getOrDefault(id, "Customer Not Found");

}

}

**Define Service Class**

**CustomerService.java**

**package** mypackage;

**public** **class** CustomerService {

**private** CustomerRepository customerRepository;

// Constructor Injection

**public** CustomerService(CustomerRepository customerRepository) {

**this**.customerRepository = customerRepository;

}

**public** **void** getCustomerDetails(String id) {

String customerName = customerRepository.findCustomerById(id);

System.***out***.println("Customer ID: " + id + ", Name: " + customerName);

}

}

**Test Dependency Injection Implementation**

**TestDependencyInjection.java**

**package** mypackage;

**import** java.util.Scanner;

**public** **class** TestDependencyInjection {

**public** **static** **void** main(String[] args) {

// Create repository implementation

CustomerRepository repository = **new** CustomerRepositorylmpl();

// Inject repository into service using constructor injection

CustomerService service = **new** CustomerService(repository);

// Get user input

Scanner sc = **new** Scanner(System.***in***);

System.***out***.print("Enter Customer ID to search: ");

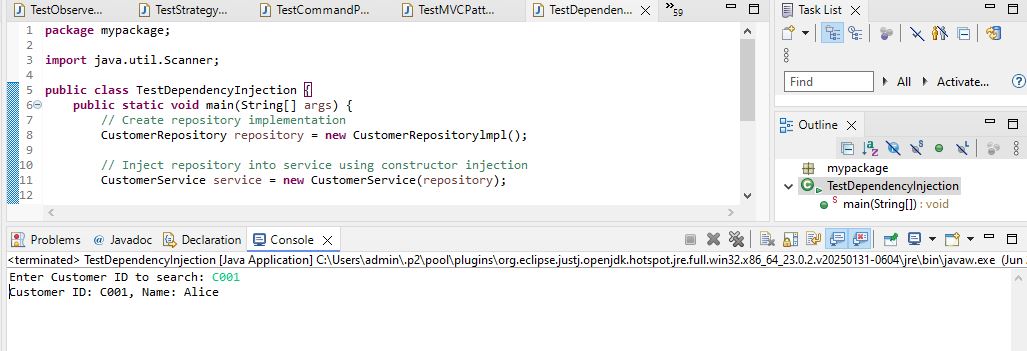
String id = sc.nextLine();

service.getCustomerDetails(id);

sc.close();

}

}

****

We are developing a customer management application where CustomerService depends on CustomerRepository to retrieve customer data. We use Dependency Injection to manage dependencies.

**1. What is Dependency Injection?**

Dependency Injection (DI) is a design pattern where the dependencies of a class are provided from the outside rather than created inside the class. This promotes loose coupling and improves flexibility and testability.

**2. Types of Dependency Injection**

* **Constructor Injection (used in this code):** Dependencies are injected via class constructor.
* **Setter Injection:** Dependencies are injected via setter methods.
* **Field Injection:** Dependencies are injected directly into fields (common in frameworks like Spring).

**3. Advantages of Dependency Injection**

* Reduces coupling between classes.
* Easier to test (mock dependencies during testing).
* Easier to maintain and extend.
* Improves code reusability.

**4. Time Complexity**

* Fetching customer data: O(1) (using HashMap)

**5. Real-Life Applications**

* Used extensively in frameworks like Spring, Angular.
* Web services, enterprise applications, microservices.