Exercise 11: Implementing Dependency Injection

Objective

To develop a Customer Management Application using Dependency Injection where the CustomerService depends on CustomerRepository, and dependencies are injected using constructor injection.

Java Code – DependencyInjectionExample.java

// Step 2: Repository Interface interface CustomerRepository {

String findCustomerById(String customerId);

}

// Step 3: Concrete Repository Implementation

class CustomerRepositoryImpl implements CustomerRepository {

@Override

public String findCustomerById(String customerId) {

// Simulate database lookup

return "Customer[ID: " + customerId + ", Name: Deepika]";

}

}

// Step 4: Service Class depending on Repository class CustomerService {

private CustomerRepository repository;

// Step 5: Constructor Injection

public CustomerService(CustomerRepository repository) { this.repository = repository;

}

public void displayCustomer(String customerId) {

String customerInfo = repository.findCustomerById(customerId); System.out.println("Customer Info: " + customerInfo);

}

}

// Step 6: Test Class

public class DependencyInjectionExample { public static void main(String[] args) {

// Inject dependency manually

CustomerRepository repo = new CustomerRepositoryImpl(); CustomerService service = new CustomerService(repo);

// Use service service.displayCustomer("C102");

}

}

Sample Output

Customer Info: Customer[ID: C102, Name: Deepika]

Key Benefits of Dependency Injection

Feature Description

Loose Coupling Objects do not create their dependencies directly Easier Testing Dependencies can be easily mocked or stubbed Better Maintainability Easy to update or switch implementations

Promotes Reusability Services can be reused with different repository classes