Customer Management with Dependency Injection

public class DependencyInjectionExample {

public static void main(String[] args) {

// Create repository

CustomerRepository repository = new CustomerRepositoryImpl();

CustomerService customerService = new CustomerService(repository);

Customer customer = customerService.findCustomerById(123);

System.out.println("Found customer: " + customer);

// ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);

// CustomerService service = context.getBean(CustomerService.class);

}

// Model

static class Customer {

private int id;

private String name;

private String email;

public Customer(int id, String name, String email) {

this.id = id;

this.name = name;

this.email = email;

}

@Override

public String toString() {

return String.format("Customer[id=%d, name=%s, email=%s]", id, name, email);

}

}

// Repository Interface

interface CustomerRepository {

Customer findCustomerById(int id);

void saveCustomer(Customer customer);

}

// Concrete Repository Implementation

static class CustomerRepositoryImpl implements CustomerRepository {

private Map<Integer, Customer> customers = new HashMap<>();

public CustomerRepositoryImpl() {

customers.put(123, new Customer(123, "John Doe", "john@example.com"));

customers.put(456, new Customer(456, "Jane Smith", "jane@example.com"));

}

@Override

public Customer findCustomerById(int id) {

System.out.println("Finding customer with ID: " + id);

return customers.get(id);

}

@Override

public void saveCustomer(Customer customer) {

System.out.println("Saving customer: " + customer);

customers.put(customer.id, customer);

}

}

static class CustomerService {

private final CustomerRepository customerRepository;

public CustomerService(CustomerRepository customerRepository) {

this.customerRepository = customerRepository;

}

public Customer findCustomerById(int id) {

return customerRepository.findCustomerById(id);

}

public void saveCustomer(Customer customer) {

customerRepository.saveCustomer(customer);

}

}

static class AppConfig {

@Bean

public CustomerRepository customerRepository() {

return new CustomerRepositoryImpl();

}

@Bean

public CustomerService customerService() {

return new CustomerService(customerRepository());

}

}

}

Key Features:

1. **Constructor Injection**: Preferred method for dependency injection
2. **Interface-based Design**: Depend on abstractions (interfaces) not implementations
3. **Loose Coupling**: CustomerService doesn't know about CustomerRepositoryImpl
4. **Testability**: Easy to mock dependencies for unit testing
5. **Flexibility**: Can switch repository implementations without changing service code

Dependency Injection Benefits:

1. **Decoupling**: Components are loosely coupled
2. **Testability**: Easier to test with mock dependencies
3. **Reusability**: Components can be reused in different contexts
4. **Maintainability**: Easier to modify and extend
5. **Configurability**: Dependencies can be configured externally