DI & IoC

CYBERSOFT.EDU.VN

Nội dung

❑ Apache Maven Tool

- ➢ Apache Maven là gì?
- ➢ Cách cài đặt và sử dụng.
- ➢ Tạo dự án bằng Maven.

❑ Design Pattern

- ➢ Design pattern là gì?
- ➢ Tại sao cần Design Pattern.
- ➢ Phân nhóm Design Pattern.
- ➢ Tìm hiểu về Dependency Injection.
- ➢ Tìm hiểu về IoC.
- ➢ Bài tập DI và IoC.

❑ Spring Core

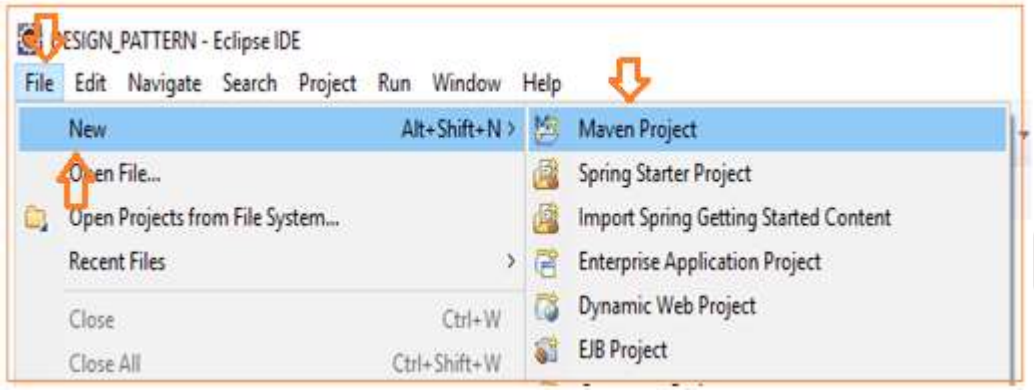
- ➢ Bean là gì?
- ➢ Cấu hình bean.
- ➢ Bài tập trên lớp.
- ➢ Bài tập về nhà.

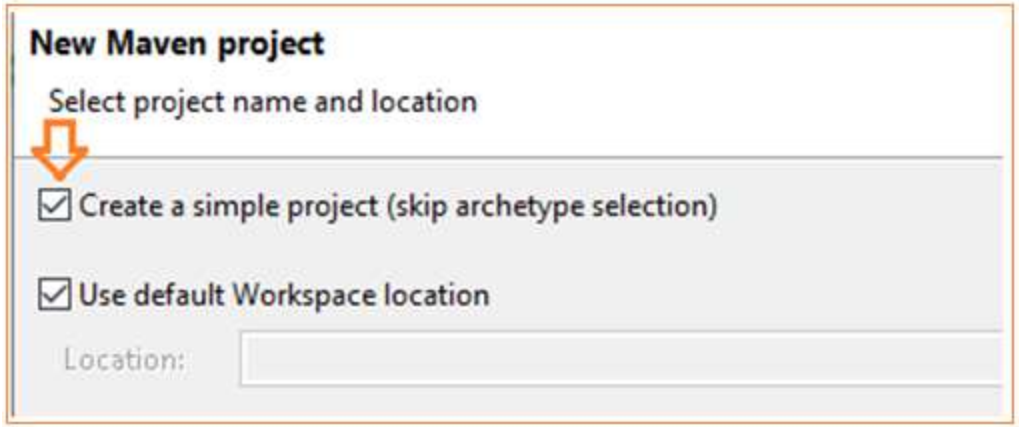
Maven Tool

- **Apache Maven** là một **opensource được phát triển bởi Apache**, ra đời nhằm mục đích quản lý các project hiệu quả. Giúp các lập trình viên giảm thiểu thời gian quản lý và build các project trong công việc.
- **Maven** hỗ trợ việc tự động hóa các quá trình tạo dự án ban đầu, download thư viện, thực hiện biên dịch, kiểm thử, đóng gói và triển khai sản phẩm.
- **Developer** chỉ cần tập trung vào việc code, còn Maven đảm nhận việc build và triển khai.
- **Apache maven** hiện nay đã được tích hợp sẵn vào trong eclipse (trừ những bản rất cũ).
- Link tìm kiếm và down load các thư viện: https://mvnrepository.com

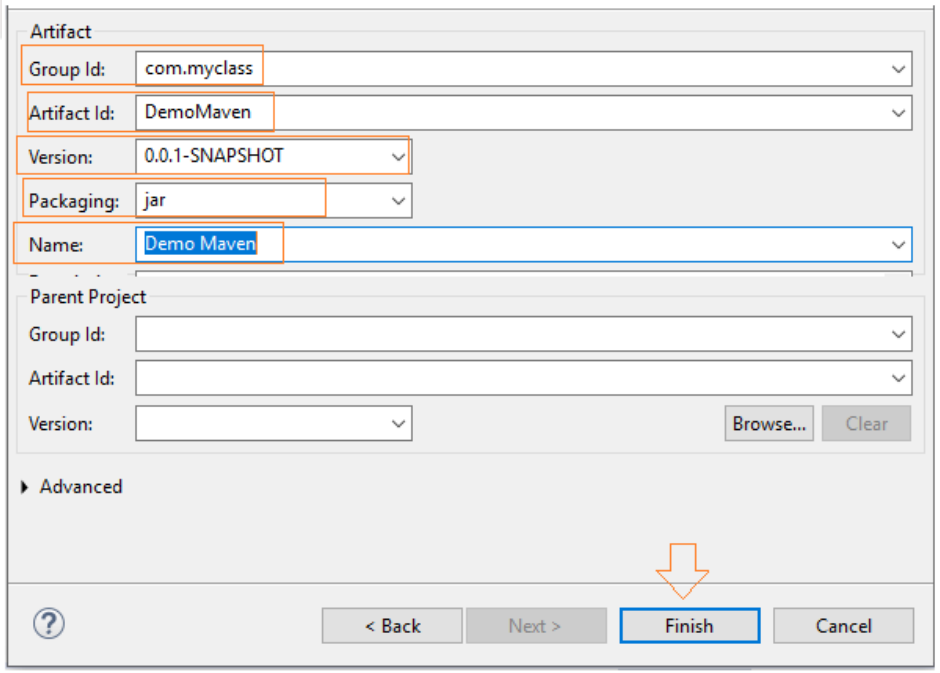
Maven Tool

❑ Tạo dự án bằng Maven





- **Group Id** : Tên tổ chức / công ty / tên website.
- **Artifact Id** : Tên của packge, dự án.
- **Version** : version của project.
- **Package** : jar có nghĩa là thư viện or java application, war là web application.
- **Name** : Tên project (trong eclipse).



Maven Tool

❑ File porm.xml

- File **pom.xml** là nơi khai báo tất cả những gì liên quan đến dự án được cấu hình qua maven, như khai báo các dependency, version của dự án, tên dự án.
- Cặp thẻ **<dependencies>** là cặp thẻ cha, chúng ta sẽ khai báo các thư viện con bên trong cặp thẻ này.
- Khai báo các thư viện bên trong cặp thẻ **<dependency>** với các thông tin bao gồm tên thư viện và version của thư viện.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.myclass</groupId>
    <artifactId>DemoMaven</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Demo Maven</name>

    <dependencies>          Khai báo thư viện spring-context
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.1.5.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>
    </dependencies>          Khai báo thư viện JSTL
</project>
```

Design Pattern là gì?

- **Design pattern** là một **kỹ thuật trong lập trình hướng đối tượng**, được các nhà nghiên cứu đúc kết và tạo ra các mẫu thiết kế chuẩn.
- **Design pattern** là **thiết kế dựa trên code**, **nó nằm ở một tầm vực cao hơn code**, do đó bất kì ngôn ngữ nào (C#, Java, Python) cũng có thể áp dụng vào được.
- Nói một cách đơn giản, **design pattern** là các **mẫu thiết kế có sẵn dùng để giải quyết một vấn đề**. Áp dụng mẫu thiết kế này sẽ làm code dễ bảo trì và mở rộng hơn.

Design Pattern

❑ Tại sao nên sử dụng?

- ✓ **Design pattern** giúp cho dự án **dễ bảo trì**, **nâng cấp** và **mở rộng**.
- ✓ **Design pattern** đã được các nhà nghiên cứu đúc kết ra nên khi sử dụng chúng ta sẽ **hạn chế được các lỗi tiềm ẩn**.
- ✓ Và cuối cùng là khi sử dụng **design pattern** thì sẽ giúp code của chúng ta sẽ **dễ đọc**, **có lợi khi làm việc nhóm**.

❑ Điều kiện để học

- ✓ Nắm chắc được kiến thức OOP: **thừa kế**, **đa hình, trừu tượng**, **bao đóng**.
- ✓ Có kiến thức về **interface** và **abstract class**, **static**.
- ✓ Bỏ tư duy theo lối cấu trúc, nâng tư duy hoàn toàn OOP.

Design Pattern

❑ **Có 3 nhóm chính:**

❖ **Creational Pattern** (nhóm khởi tạo): **Abstract Factory**, **Factory Method**, **Singleton**, **Builder**, **Prototype**,...

✓ *Creational Pattern sử dụng một số thủ thuật để khởi tạo đối tượng mà không cần sử dụng từ khóa new.*

❖ **Structural Pattern** (nhóm cấu trúc): **Adapter**, **Bridge**, **Composite**, **Decorator**, **Facade**, **Proxy**, **Flyweight**,...

✓ *Nhóm này sẽ giúp chúng ta thiết lập, định nghĩa quan hệ giữa các đối tượng.*

❖ **Behavioral Pattern** (nhóm ứng xử): **Interpreter**, **Template Method**, **Chain of Responsibility**, **Command**, **Iterator**, **Mediator**, **Memento**, **Observer**, **State**, **Strategy** và **Visitor**.

✓ *Nhóm này sẽ tập trung thực hiện các hành vi của đối tượng.*

Dendency

- **Dependency** nghĩa là sự phụ thuộc, ở đây có thể hiểu là **sự phụ thuộc giữa các class**.
- **Dependency** khiến cho **việc bảo trì**, **nâng cấp dự án gặp nhiều khó khăn**.

❖ **Ví dụ:**

- ✓ Tạo 2 class **EmailService** và **FacebookService** đều có chung phương thức **sendMessage()** tuy nhiên nội dung in ra sẽ khác nhau.
- ✓ Tạo 2 class **HomeController** và **ContactController** sẽ tạo mới một trong hai service để in ra message.

➡ Lúc này sẽ ra sự phụ thuộc giữa các class (dependency).

Ví dụ

❖ **Class EmailService**

```
public class EmailService {
    public void sendMessage() {
        System.out.println("Email sending...");
    }
}
```

❖ **Class FacebookService**

```
public class FacebookService {
    public void sendMessage() {
        System.out.println("Facebook sending...");
    }
}
```

Ví dụ

❖ Controller

```
public class ContactController {
    EmailService service = null;
    public ContactController(){
        service = new EmailService();
    }

    public void send(){
        service.sendMessage();
    }
}
```

```
public class HomeController {
    EmailService service = null;
    public HomeController(){
        service = new EmailService();
    }

    public void send(){
        service.sendMessage();
    }
}
```

Ví dụ

❖ Hàm Main

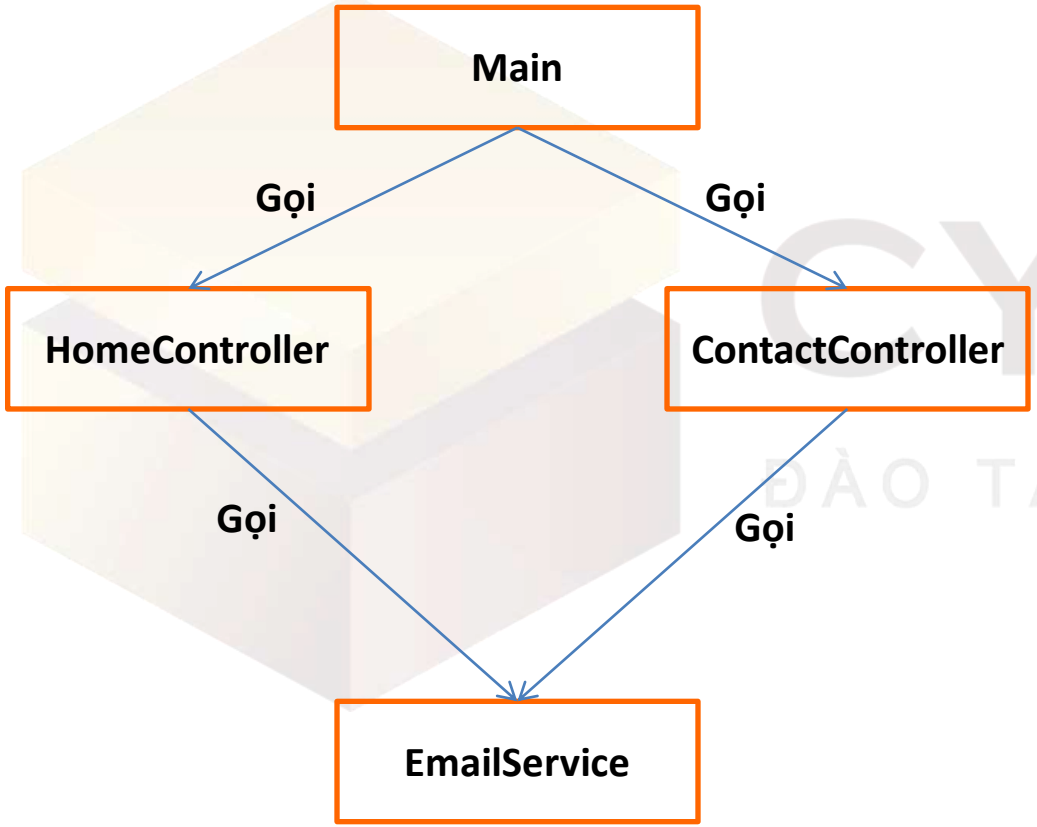
```
public class Main {
    public static void main(String[] args) {

        HomeController homeController = new HomeController();
        homeController.send();

        ContactController contactController = new ContactController();
        contactController.send();
    }
}
```

Dependency

❑ Sự phụ thuộc

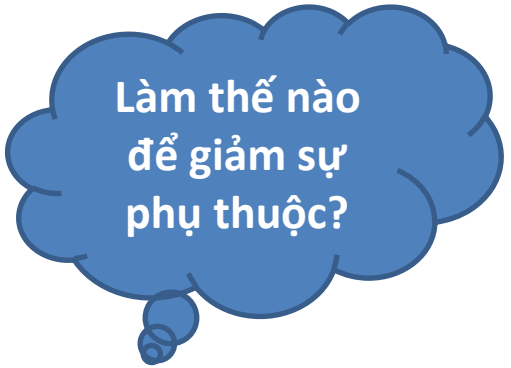


➢ Chương trình khởi chạy

1. Hàm Main gọi tới HomeController và ContactController .
2. Controller gọi tới EmailService để thực thi phương thức sendMessage().

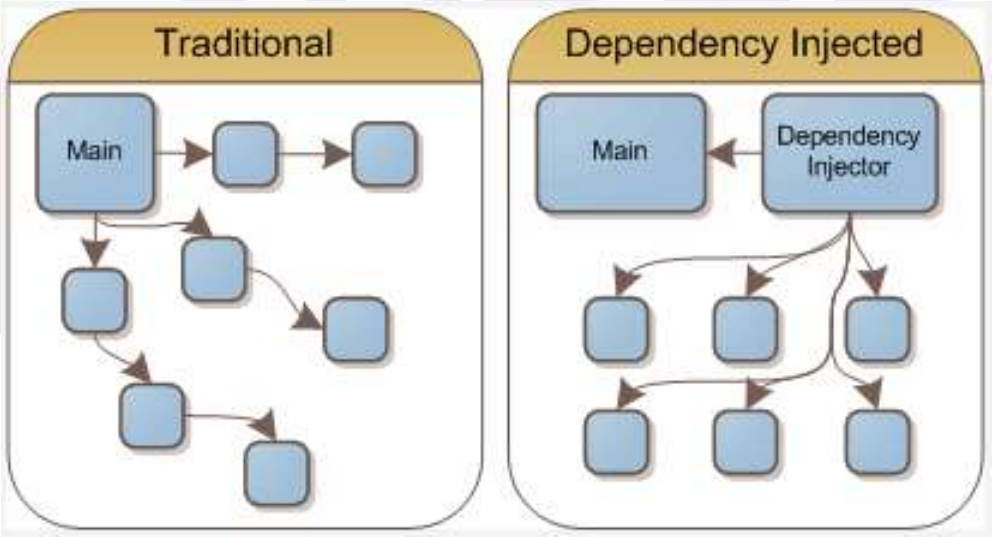
➡ Sinh ra **sự phụ thuộc** (**Dependency**).

➡ Câu hỏi đặt ra:



Dependency Injection

- **DI** (Dependency Injection) là một dạng **Design Pattern** được thiết kế với **mục đích ngăn chặn sự phụ thuộc lần nhau giữa các class**.
- Dễ dàng hơn trong việc thay đổi module, bảo trì code và testing.



Nguyên lý hoạt động

- Các module **không giao tiếp trực tiếp với nhau, mà thông qua interface**.
- Các **module cấp thấp sẽ implement interface** và **module cấp cao sẽ gọi module cấp thấp thông qua interface**.
- Việc **khởi tạo các module cấp thấp sẽ do IoC Container thực hiện**.
- Việc **module nào gắn với interface nào sẽ được config trong code** hoặc **config trong file XML** (IoC Container).

Nguyên lý thứ nhất

- **Thứ nhất**

> Các **module không giao tiếp trực tiếp** với nhau, mà **thông qua interface**.

- **Thứ hai**

> Các **module cấp thấp sẽ implement interface** và **module cấp cao sẽ gọi module cấp thấp thông qua interface**.

Giải pháp

- ✓ **Bước 1**: Tạo interface **MessageService**
- ✓ **Bước 2**: Class **EmailService** và **FacebookService** imlement từ **MessageService**.
- ✓ **Bước 3**: Trong controller khai báo thuộc tính **messageService** có kiểu **MessageService** và sử dụng hàm khởi tạo để gán giá trị cho thuộc tính.
- ✓ **Bước 4**: Tại hàm **Main** khởi tạo loại **Service** muốn sử dụng và truyền vào cho các Controller.

➧ **Nên nhớ**:

- ✓ Các module không giao tiếp với nhau mà thông qua Interface.
- ✓ Module cấp thấp sẽ implement Interface và module cấp cao sẽ gọi module cấp thấp thông qua Interface.

Ví dụ

❑ Interface MessageService

```
public interface MessageService {
    void sendMessage();
}
```

❑ Class EmailService

```
public class EmailService implements MessageService{
    public void sendMessage() {
        System.out.println("Email sending...");
    }
}
```

❑ Class FacebookService

```
public class FacebookService implements MessageService {
    public void sendMessage() {
        System.out.println("Facebook sending...");
    }
}
```

Ví dụ

❑ ContactController

```
public class ContactController {
    MessageService messageService;
    public ContactController(
        MessageService messageService
        ) {
        this.messageService = messageService;
    }
    public void send() {
        messageService.sendMessage();
    }
}
```

❑ HomeController

```
public class HomeController {
    MessageService messageService;
    public HomeController(
        MessageService messageService
        ) {
        this.messageService = messageService;
    }
    public void send() {
        messageService.sendMessage();
    }
}
```

Ví dụ

❑ Main

```
public class Main {

    public static void main(String[] args) {

        MessageService messageService = new  EmailService();

        HomeController homeController = new HomeController(messageService);
        homeController.send();

        ContactController contactController = new ContactController(messageService);
        contactController.send();

    }
}
```

Dependency Injection

❑ Các dạng DI

- ❖ **Contructor Injection**: Các dependency sẽ được container truyền vào (inject vào) một class thông qua constructor của class đó. Đây là cách thông dụng nhất.
- ❖ **Setter Injetcion**: Các dependency sẽ được truyền vào 1 class thông qua các hàm Setter.
- ❖ **Fields/ properties Injection**: Các dependency sẽ được truyền vào 1 class một cách trực tiếp vào các field.
- ❖ **Interface Injection**: Cách này ít dùng nhất.

Dependency Injection

```
public class ContactController {
    MessageService messageService;

    public void setMessageService(MessageService messageService){
        this.messageService = messageService;
    }

    public void send() {
        messageService.sendMessage();
    }
}
```



```
public class UserController {

    MessageService messageService;

    public UserController(MessageService messageService) {
        this.messageService = messageService;
    }

    public void send() {
        messageService.sendMessage();
    }
}
```

Nguyên lý thứ hai

❑ Thứ nhất

Việc khởi tạo các module cấp thấp sẽ do IoC Container thực hiện.

❑ Thứ hai

Việc module nào gắn với interface nào sẽ được config trong code hoặc trong file XML (IoC Container).

Inversion of Control

- **Inversion of Control** (**IoC**) nghĩa là đảo ngược điều khiển. Ý của nó là làm **thay đổi luồng điều khiển của ứng dụng**, giúp tăng tính mở rộng của một hệ thống.
- Theo **cách truyền thống một đối tượng được tạo ra từ một class**, các trường (field) của nó sẽ được gán giá trị từ chính bên trong class đó. **IoC làm ngược lại với cách truyền thống**, các đối tượng được tạo ra bởi IoC Container và được tiêm vào bằng cách sử dụng **Dependency Injection**.

IoC Container

- **IoC Container** là **cốt lõi của Spring Framework**.
- **Ioc Container** **tạo ra các đối tượng, nối chúng lại với nhau, cấu hình chúng và quản lý vòng đời của chúng** từ lúc tạo ra đến khi bị hủy. Các đối tượng đó được gọi là **Spring Bean**.
- **IoC Container** sử dụng **Dependency Injection** để quản lý các thành phần tạo nên một ứng dụng.
- **IoC Container** được **cung cấp thông tin từ tập tin cấu hình xml** hoặc **Java code** (Spring 5).

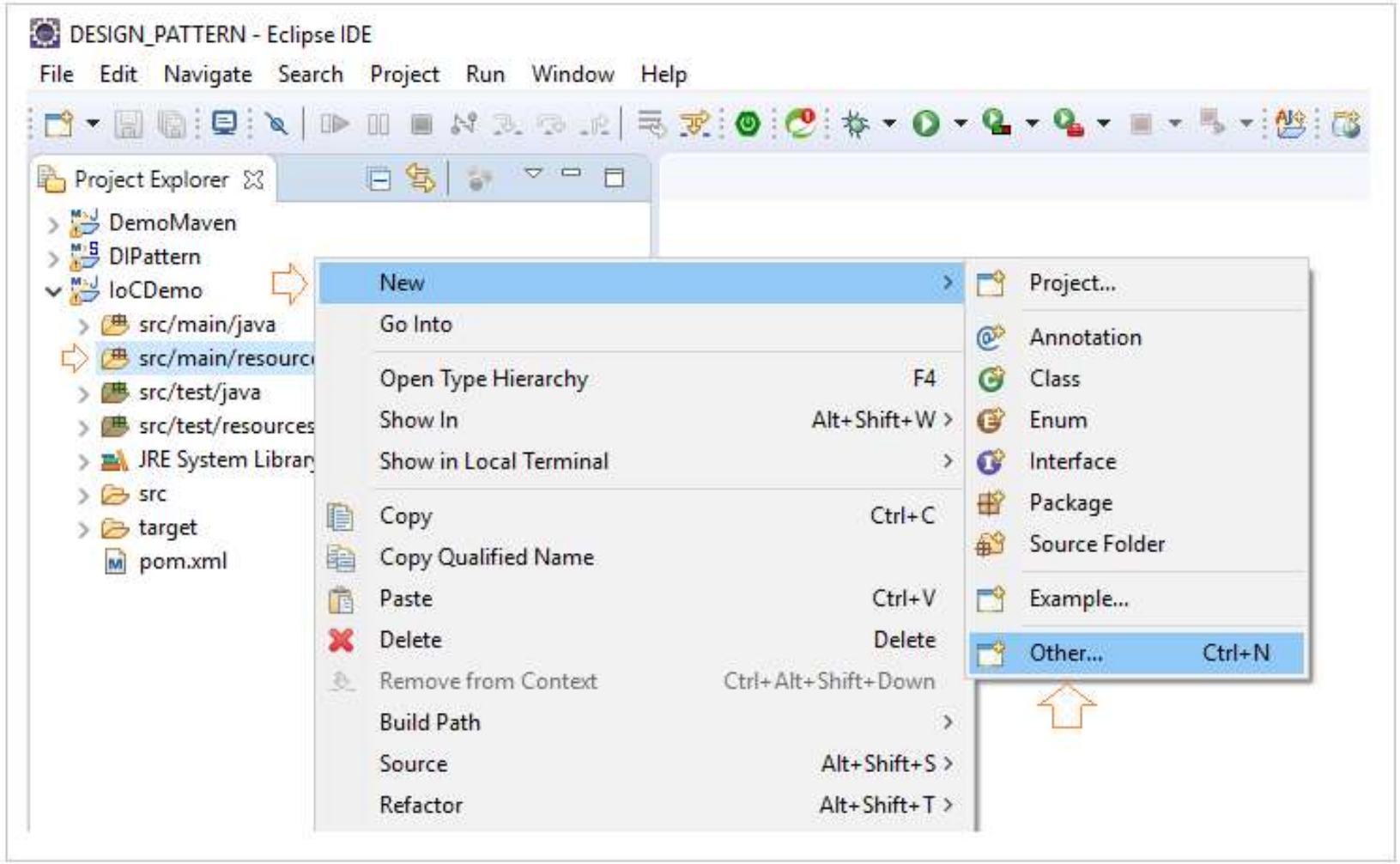
IoC Container

- **BeanFactory**: Đây là **container đơn giản nhất** cung cấp hỗ trợ cơ bản cho Dependency Injection.
- **ApplicationContext**: được xây dựng bằng cách **kế thừa BeanFactory** nhưng nó có thêm một số chức năng mở rộng như tích hợp với Spring AOP, xử lý Message, Context cho web application.

❖ ***Note**: Chúng ta sẽ chủ yếu sử dụng Applicaton Context.*

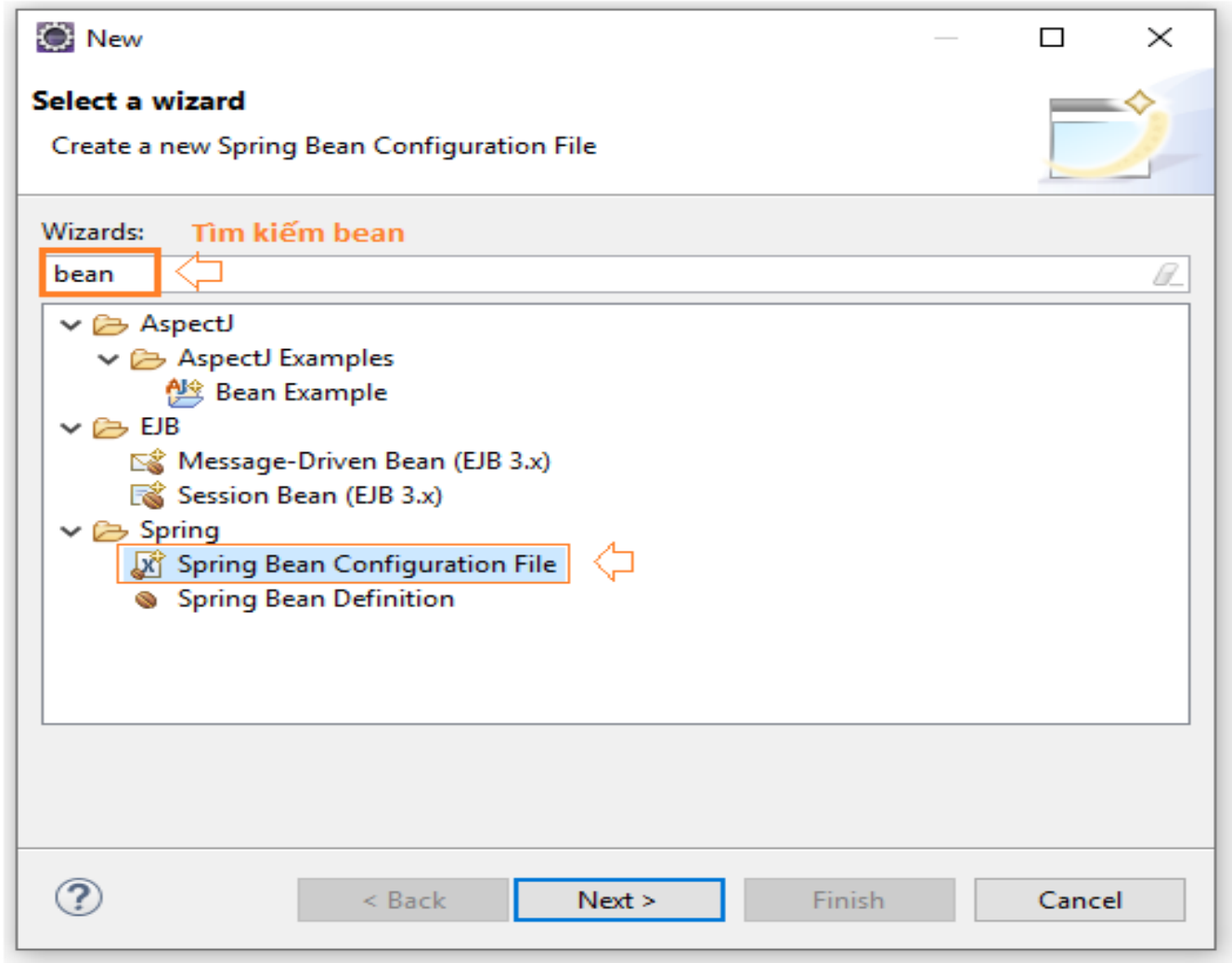
Tạo file cấu hình Container

❑ **Bước 1:** Chuột phải vào **src/main/resource → New → Other**



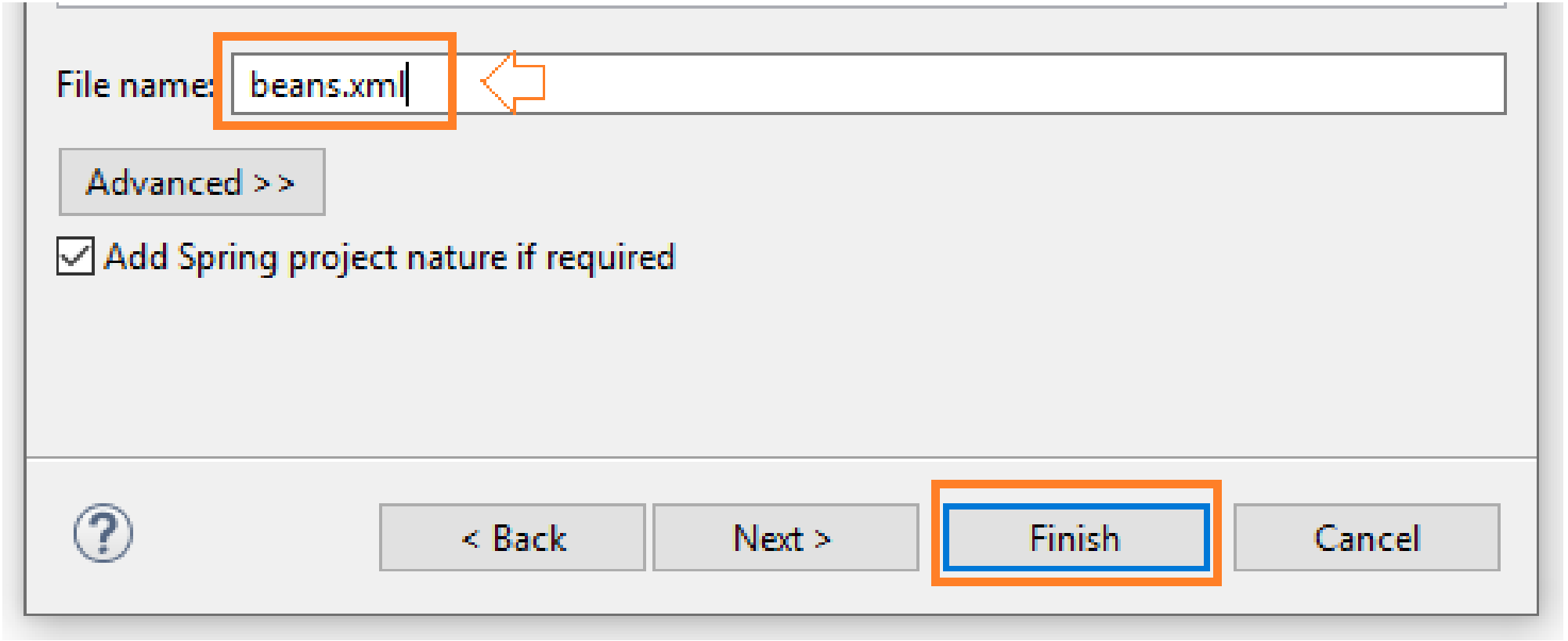
Tạo file cấu hình Container

❑ **Bước 2:** Chọn **Spring Bean Configuration File** → **Next**



Tạo file cấu hình Container

❑ **Bước 3:** Đặt tên file → Chọn **Finish** để hoàn thành.



Tạo file cấu hình Container

❑ Áp dụng Ioc Container vào ví dụ

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.1.5.RELEASE</version>
    </dependency>
</dependencies>
```

Thêm thư viện spring-context vào file porm.xml

Thêm mới bean vào file beans.xml (Ioc Container)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="emailService" class="com.myclass.service.EmailService"></bean>

    <bean id="homeController" class="com.myclass.controller.HomeController">
        <constructor-arg ref="emailService"/>
    </bean>

    <bean id="contactController" class="com.myclass.controller.ContactController">
        <constructor-arg ref="emailService"/>
    </bean>

</beans>
```

Ioc Container

```java
public class Main {

    private static ClassPathXmlApplicationContext context;

    public static void main(String[] args) {
        context = new ClassPathXmlApplicationContext("beans.xml");

        System.out.println("---=== HOME CONTROLLER ===---");
        HomeController home = (HomeController)context.getBean("homeController");
        home.send();

        System.out.println("---=== CONTACT CONTROLLER ===---");
        ContactController contact = (ContactController)context.getBean("contactController");
        contact.send();
    }
}
```

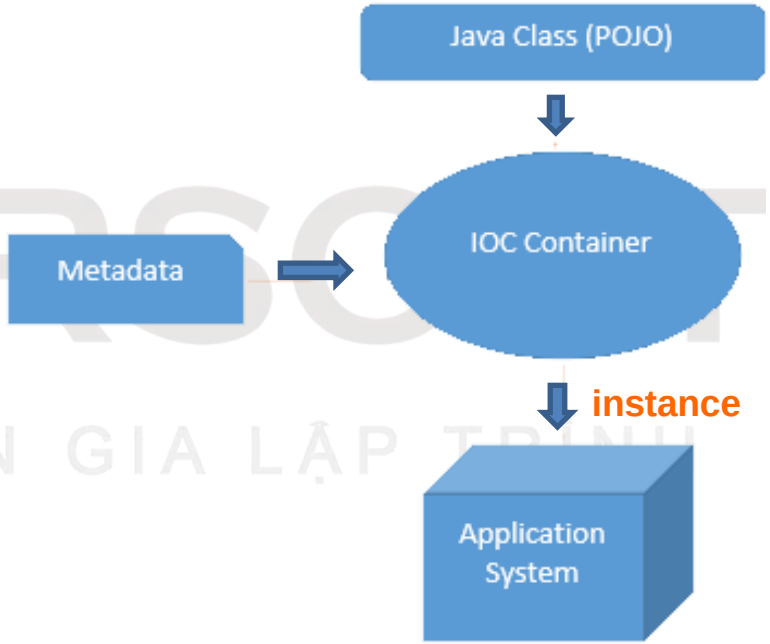
Hàm main

Kết quả

```
---=== HOME CONTROLLER ===---
Email sending...
---=== CONTACT CONTROLLER ===---
Email sending...
```

Spring Bean

- **Spring Bean** là **các object trong Spring Framework**, được khởi tạo thông qua Spring Container.
- Bất kỳ **class Java POJO thông thường nào cũng có thể là Spring Bean** nếu nó được cấu hình để khởi tạo thông qua Container bằng cách cung cấp thông tin (metadata) cấu hình (các file config .xml, .properties..).



- ✓ ***POJO (Plain Old Java Object)**: Đối tượng Java thuần túy.*

Spring Bean

Spring Bean

```xml
<bean id="account1" class="com.myclass.entity.Account">
    <property name="id" value="1" />
    <property name="ownerName" value="Nguyễn Văn A" />
    <property name="balance" value="20000000" />
</bean>
<bean id="account2" class="com.myclass.entity.Account">
    <property name="id" value="2" />
    <property name="ownerName" value="Nguyễn Văn B" />
    <property name="balance" value="10000000" />
</bean>
```

⇔

Account instance

```java
Account account1 = new Account();
account1.setId(1);
account1.setOwnerName("Nguyễn Văn A");
account1.setBalance(20000000);

Account account2 = new Account();
account1.setId(2);
account1.setOwnerName("Nguyễn Văn B");
account1.setBalance(10000000);
```

Spring Bean Scopes

❖ **Có 5 scope định nghĩa cho Spring Bean**:

- **Singleton** – Chỉ **một instance của bean được tạo trong mỗi container** --> đây là scope mặc định cho Spring Beans.
- **Prototype** – Một instance mới sẽ được **tạo mỗi lần bean được request**.
- **Request** – Khá giống prototype scope, tuy nhiên nó được sử dụng cho ứng dụng web. Một instance mới của bean sẽ được **tạo cho mỗi HTTP request**.
- **Session**: Mỗi thể hiện của bean sẽ được **tạo cho mỗi HTTP Session**.
- **Global-Session**: Mỗi thể hiện của bean sẽ được **tạo cho mỗi global HTTP Session**.

➢ *3 Scope cuối chỉ dùng trong ứng dụng web.*

Spring Bean Configuration

- Có 3 cách cấu hình Spring Bean:
- ✓ **XML Configuration**: Cấu hình bằng file XML.
- ✓ **Java Configuration**: Cấu hình bằng Java code.
- ✓ **Annotation Configuration**: Cấu hình bằng cách sử dụng Annotation.

POJO class

```java
public class Contact {
    private String fullname;
    private String email;
    private String phone;

    public String getFullname() {
    public void setFullname(String fullname) {
    public String getEmail() {
    public void setEmail(String email) {
    public String getPhone() {
    public void setPhone(String phone) {
}
```

Spring Bean

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans

    <bean id="contact" class="com.myclass.model.Contact">
        <property name="fullname" value="Trần Văn Hào"></property>
        <property name="email" value="tvhao@gmail.com"></property>
        <property name="phone" value="0366546789"></property>
    </bean>

</beans>
```

Spring Bean

❑ Java Configure

```java
public class Contact {
    private String fullname;
    private String email;
    private String phone;

    public String getFullname() {
    public void setFullname(String fullname) {
    public String getEmail() {
    public void setEmail(String email) {
    public String getPhone() {
    public void setPhone(String phone) {
}
```

POJO Class

```java
@Configuration
public class BeanConfiguration {

    @Bean
    public Contact contact() {
        Contact contact = new Contact();
        contact.setFullname("Trần Văn Hào");
        contact.setEmail("tvhao@gmail.com");
        contact.setPhone("0366546789");
        return contact;
    }
}
```

Bean

❑ Annotation Configure

```java
@Repository
public class MessageRepository {
    public void send() {
        System.out.println("Respository send...");
    }
}
```

```java
@Service
public class MessageService {
    public void send() {
        System.out.println("Respository send...");
    }
}
```

```java
@Controller
public class ContactController {
    public String index() {
        return "index";
    }
}
```

Bài tập trên lớp

❑ Viết chương trình chuyển tiền online với yêu cầu như sau:

❖ Thông tin user:

- Mã người dùng (kiểu số)
- Tên người dùng (kiểu chuỗi)
- Số tiền trong tài khoản (kiểu số)

❖ Chức năng:

- Thêm mới user.
- Cập nhật thông tin user.
- Tìm kiếm user theo id.

→ **Chức năng chính**: Chuyển tiền giữa hai tài khoản.

Mô tả bài tập

❑ **POJO** bean **Account** :

- **id** : (kiểu số)
- **ownerName**: Tên tài khoản (Kiểu chuỗi).
- **balance**: Số dư (kiểu số thực).

❑ Lớp **AccountDao** bao gồm list chứa danh sách Account và 3 phương thức:

- **insert** : Thêm mới account.
- **update**: Cập nhật account.
- **find**: Tìm account theo id.

❑ Lớp **AccountService** gồm 2 phương thức:

- **transferMoney**: Chuyển tiền.
- **getAccount**: Lấy thông tin account theo id.

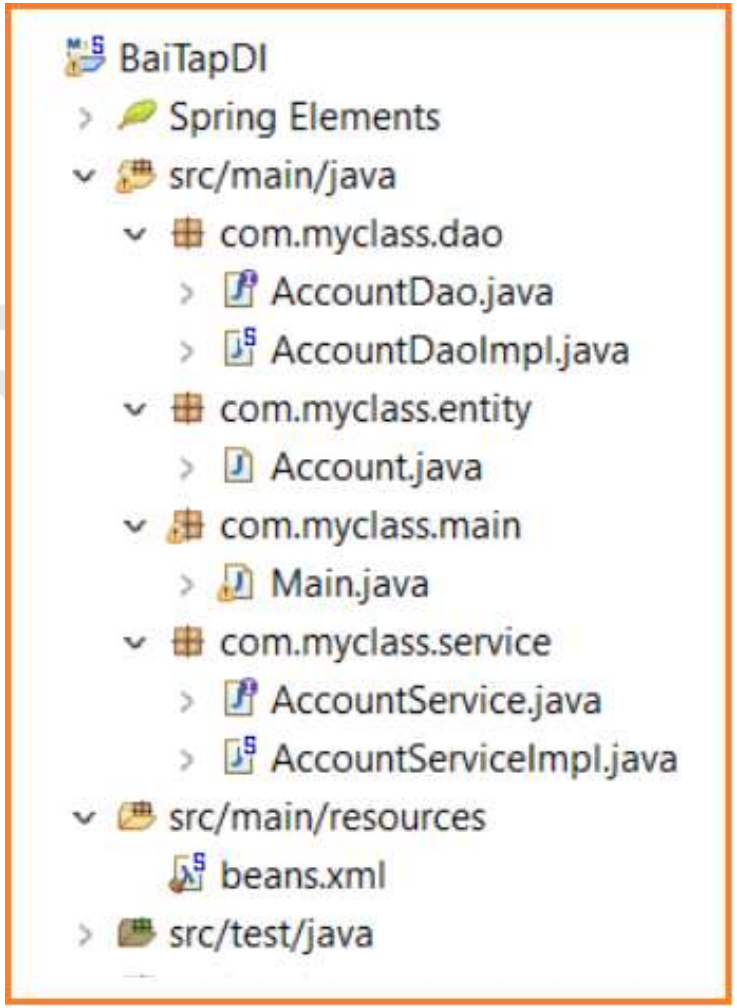
➔ Cấu hình Bean theo cả 2 cách XML Config và Java Cofig.

➔ Áp dụng Denpendency Injection và thực hiện test các chức năng.

Các bước thực hiện

❑ Các bước thực hiện

- ✓ **Bước 1**: Tạo POJO Bean.
- ✓ Bước 2: Tạo lớp AccountDao.
  - ✓Tạo danh sách Account
  - ✓Định nghĩa phương thức insert, update, find.
- ✓ **Bước 3**: Tạo lớp AccountService.
  - ✓Định nghĩa phương thức transferMoney, getAccount.
- ✓ **Bước 4**: Cấu hình Container.
- ✓ **Bước 5**: Tạo hàm Main để chạy chương trình.



AccountDao

```java
public class AccountDaoImpl implements AccountDao{

    private List<Account> accounts;
    public AccountDaoImpl() {
        accounts = new ArrayList<Account>();
    }
    public Account findById(int id) {
        for(Account account : accounts) {
            if(account.getId() == id)
                return account;
        }
        return null;
    }
    public void insert(Account account) {
        accounts.add(account);
    }
    public void update(Account account) {
        Account model = findById(account.getId());
        model.setOwnerName(account.getOwnerName());
        model.setBalance(account.getBalance());
    }
}
```

AccountService

```java
public class AccountServiceImpl implements AccountService{
    private AccountDao _accountDao;
    public AccountServiceImpl(AccountDao accountDao) {
        _accountDao = accountDao;
    }
    public void insert(Account account) {
        _accountDao.insert(account);
    }
    public void transferMoney(int fromId, int toId, long amount) {

        //Lấy ra thông tin tài khoản sẽ chuyển tiền
        Account source = _accountDao.findById(fromId);
        //Lấy ra thông tin tài khoản sẽ nhận tiền
        Account target = _accountDao.findById(toId);
        // Trừ tiền của tài khoản cần chuyển tiền
        source.setBalance(source.getBalance() - amount);
        // Cộng tiền cho tài khoản cần nhận tiền
        target.setBalance(target.getBalance() + amount);
        // Cập nhật thông tin vào danh sách trong tầng DAO
        _accountDao.update(source);
        _accountDao.update(target);
    }
    public Account findById(int id) {
        return _accountDao.findById(id);
    }
}
```

Beans.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.sp
        http://www.springframework.org/schema/context http://www.springframework.

    <beans>
        <bean id="accountDao" class="com.myclass.dao.AccountDaoImpl"></bean>
        <bean id="accountService" class="com.myclass.service.AccountServiceImpl">
            <constructor-arg ref="accountDao" />
        </bean>
    </beans>
</beans>
```

Hàm main

```java
public static void main(String[] args) {
    ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
    AccountService _accountService = (AccountServiceImpl) context.getBean("accountService");

    _accountService.insert(new Account(1, "Nguyễn Văn A", 20000000));
    _accountService.insert(new Account(2, "Nguyễn Văn B", 15000000));

    System.out.println("-------- TRƯỚC KHI CHUYỂN --------");
    System.out.println("-------- Thông Tin Người Chuyển --------");
    Account account1 = _accountService.findById(1);
    System.out.println("Họ tên: " + account1.getOwnerName());
    System.out.println("Tài khoản: " + account1.getBalance());

    System.out.println("-------- Thông Tin Người Nhận --------");
    Account account2 = _accountService.findById(2);
    System.out.println("Họ tên: " + account2.getOwnerName());
    System.out.println("Tài khoản: " + account2.getBalance());

    // Thực hiện chuyển tiền
    _accountService.transferMoney(1, 2, 10000000);

    System.out.println("-------- SAU KHI CHUYỂN --------");
    System.out.println("-------- Thông Tin Người Chuyển --------");
    Account account3 = _accountService.findById(1);
    System.out.println("Họ tên: " + account3.getOwnerName());
    System.out.println("Tài khoản: " + account3.getBalance());

    System.out.println("-------- Thông Tin Người Nhận --------");
    Account account4 = _accountService.findById(2);
    System.out.println("Họ tên: " + account4.getOwnerName());
    System.out.println("Tài khoản: " + account4.getBalance());
}
```

Sử dụng Java Config

❑ Trường hợp sử dụng **Java Config** bạn chỉ cần làm theo các bước sau:

✓ **Bước 1**: Tạo lớp cấu hình

✓ **Bước 2**: Thay đổi lớp thực thi đọc cấu hình tại hàm main:

```
//ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
AccountService _accountService = (AccountServiceImpl) context.getBean("accountService");

_accountService.insert(new Account(1, "Nguyễn Văn A", 20000000));
_accountService.insert(new Account(2, "Nguyễn Văn B", 15000000));
```

Class Java Config

Đánh dấu đây là file config

```java
@Configuration
public class AppConfig {

    @Bean
    public AccountDao accountDao() {
        return new AccountDaoImpl();
    }

    @Bean
    public AccountService accountService() {
        return new AccountServiceImpl(accountDao());
    }
}
```

Đánh dấu đây là bean

Bài tập về nhà

Bài 1:

- Tạo POJO bean Student gồm các thông tin: id, email, fullname, address, phone.
- Xây dựng lớp AccountRepository và lớp AccountService để quản lý danh sách các sinh viên.
- Thực hiện các chức năng: in danh sách sinh viên, thêm, sửa, xóa, tìm kiếm.
- ✓ Cấu trúc dự án theo mô hình phân lớp như đã học.
- ✓ Áp dụng Dependency vào dự án.

Tổng kết

❑ Nội dung cần nắm

- ➢ Sử dụng Meven Tool.
- ➢ Tổng quan về Design Pattern.
- ➢ Hiểu về Dependency và Ioc trong lập trình.
- ➢ Khái niệm và cách sử dụng Spring Bean.
- ➢ Cấu hình Spring Bean trong ứng dụng.