

I. Tìm hiểu dependency injection

1. Tight coupling

- Phương pháp code mà các thành phần phụ thuộc trực tiếp vào nhau

=> Khó mở rộng

2. Loose coupling

- Phương pháp code mà các thành phần phụ thuộc vào interface hoặc các cơ chế (constructor, setter,...)

=> Dễ mở rộng bảo trì

3. Khái niệm DI

Dependency Injection (DI) là một kỹ thuật quan trọng trong Spring Framework, bao gồm cả Spring Boot. DI giúp quản lý và cung cấp các phụ thuộc giữa các thành phần của ứng dụng, giúp làm giảm sự phụ thuộc chặt chẽ (tight coupling) và tăng tính mở rộng (scalability) và dễ bảo trì.

- Các Kiểu Dependency Injection trong Spring Boot
 - + Constructor Injection
 - + Setter Injection
 - + Field Injection
- Spring Boot cung cấp hỗ trợ DI thông qua các annotation như `@Autowired`, `@Component`, và `@Bean`, giúp quản lý các phụ thuộc và cấu hình ứng dụng một cách dễ dàng.

4. Đóng gói dự án bằng maven

- Đóng gói đuôi .jar ,.war
- Pom(project object model):

```
<dependency>

    <groupId>org.springframework.boot</groupId>

    <artifactId>spring-boot-starter-web</artifactId>

</dependency>
```

- **Dependency:** Định danh thư viện
- **GroupId:** tên tổ chức, dự án phát triển
- **ArtifactId:** Định danh thư viện cụ thể

5. IOC

- Inversion of Control (IoC) là một nguyên tắc thiết kế phần mềm quan trọng trong Spring Framework và Spring Boot, giúp quản lý sự phụ thuộc giữa các thành phần trong ứng dụng. IoC cho phép chuyển giao quyền kiểm soát việc khởi tạo và quản lý các đối tượng từ mã nguồn ứng dụng sang một framework quản lý, thường là một container hoặc context.
- IoC Container trong Spring Boot

Spring IoC Container là trung tâm của Spring Framework. Nó chịu trách nhiệm tạo ra, cấu hình và quản lý vòng đời của các beans (đối tượng) trong ứng dụng. Trong Spring Boot, IoC Container được khởi tạo và cấu hình tự động khi bạn sử dụng annotation `@SpringBootApplication` hoặc `@ComponentScan`.

Các thành phần chính của IoC Container:

Bean Definition: Xác định cách các bean sẽ được tạo và cấu hình.

Bean Factory: Cung cấp các phương thức để truy xuất bean.

ApplicationContext: Một loại BeanFactory mở rộng với nhiều chức năng bổ sung.

- Các Loại IoC Containers

BeanFactory: Cung cấp cơ chế cơ bản để tạo và quản lý beans.

ApplicationContext: Mở rộng BeanFactory và cung cấp các dịch vụ bổ sung như hỗ trợ cho AOP, sự kiện, và các tính năng khác của Spring.

- Cấu Hình IoC trong Spring Boot

Sử dụng Annotation

@Component: Đánh dấu lớp như một bean để Spring quản lý. Có thể thay thế bằng các annotation khác như @Service, @Repository, hoặc @Controller.

@Configuration: Đánh dấu lớp chứa các phương thức @Bean để cấu hình các bean trong ứng dụng.

@Bean: Được sử dụng trong các lớp cấu hình để khai báo và cấu hình các bean.

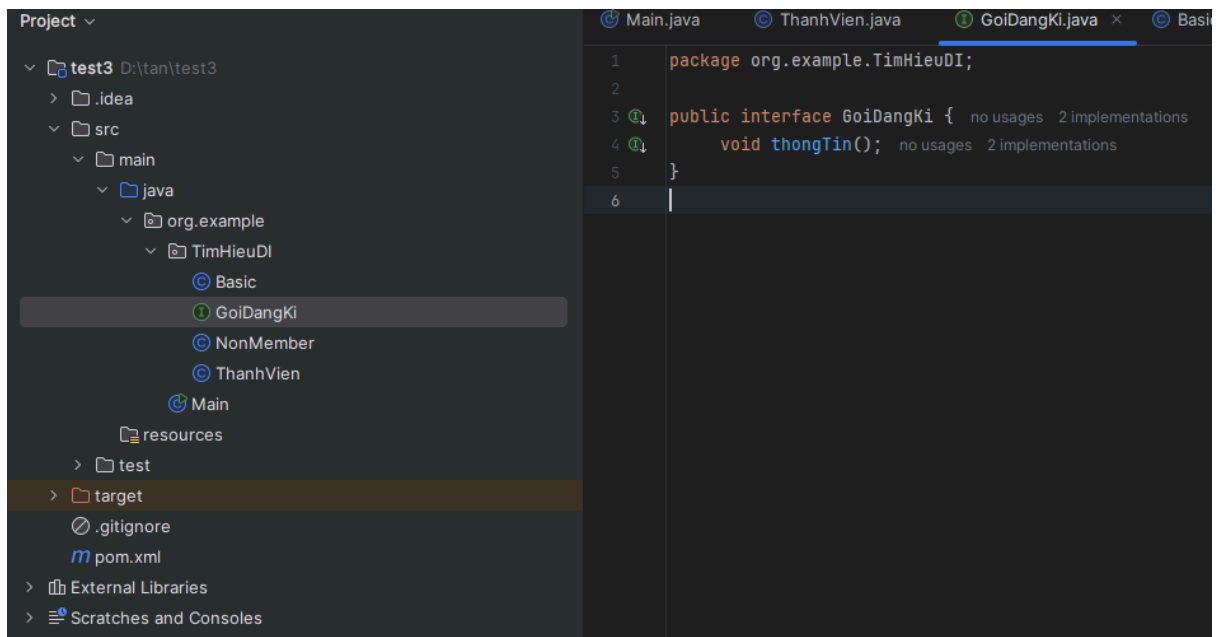
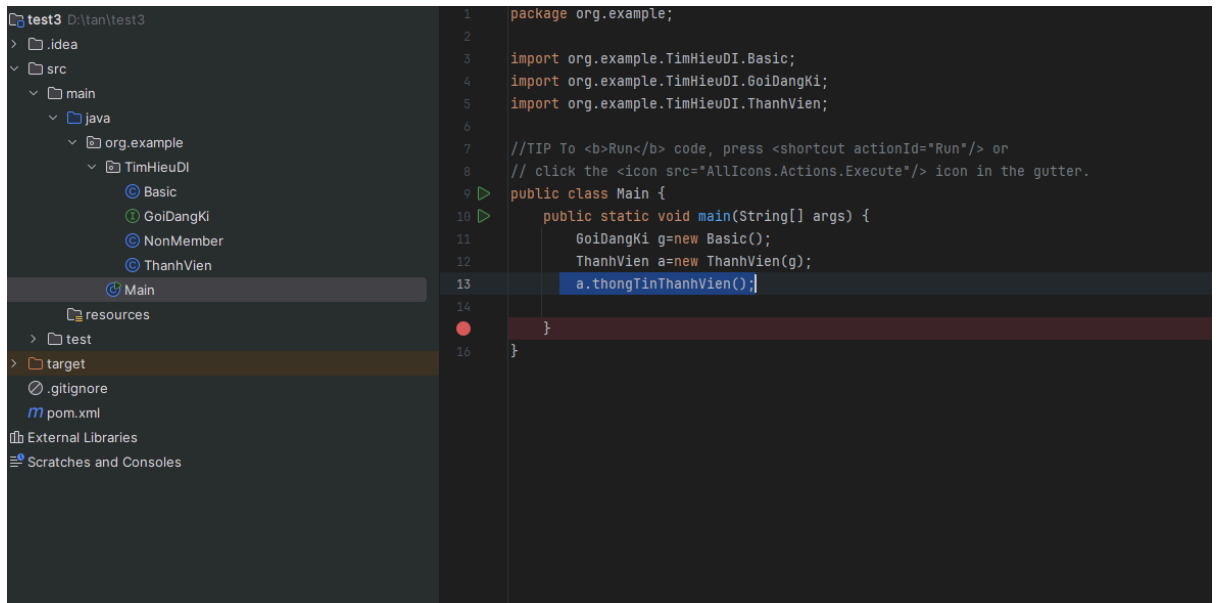
@Autowired: Được sử dụng để tự động tiêm các phụ thuộc vào các bean.

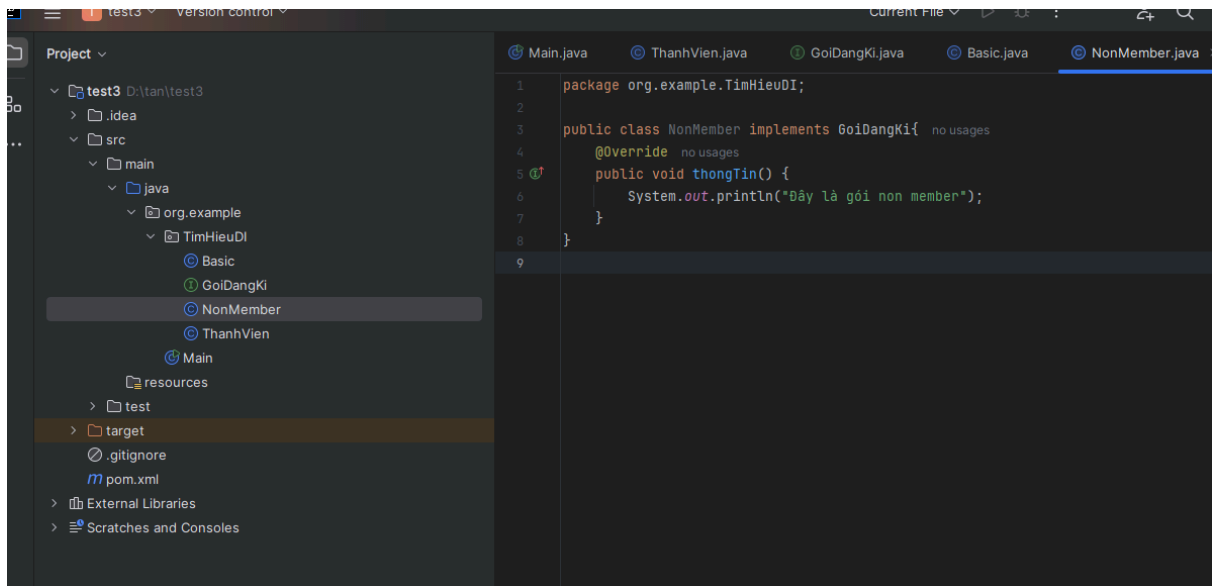
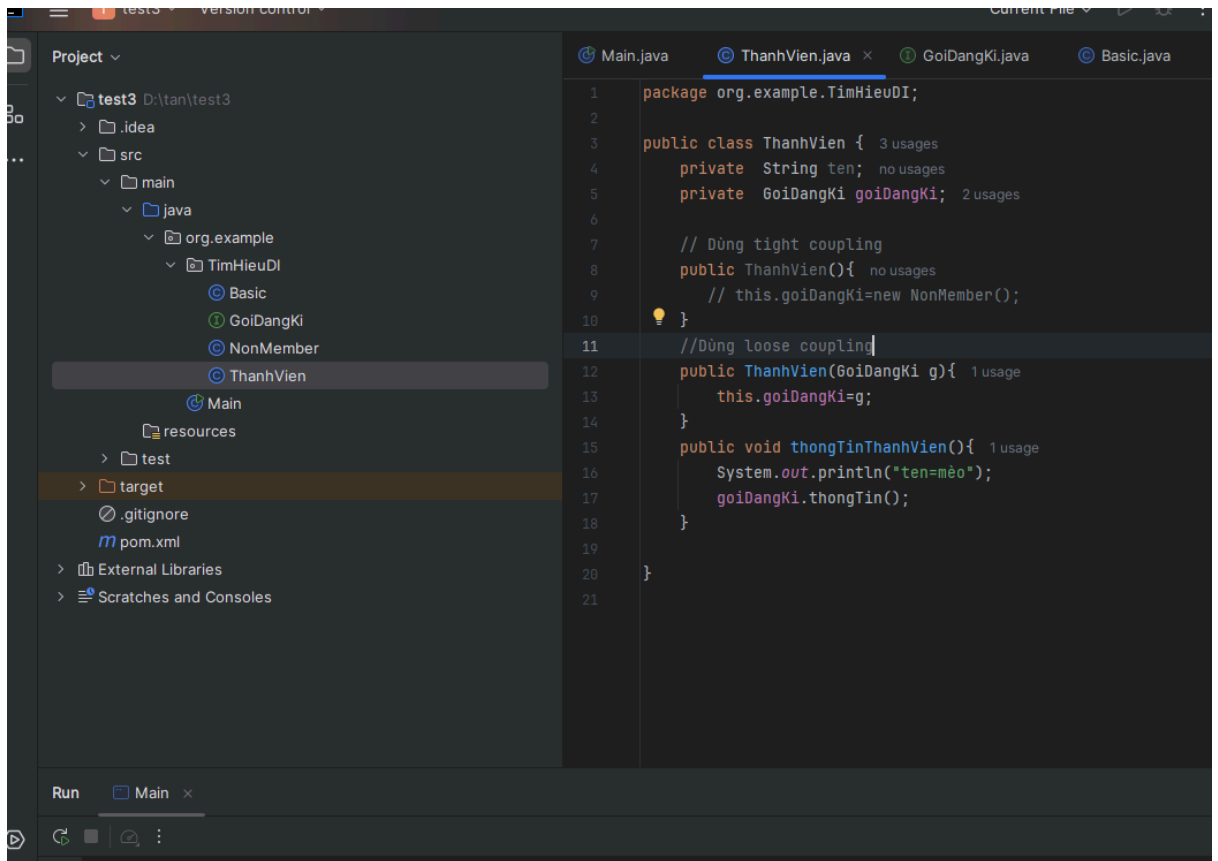
- **@Component**: Đánh dấu một lớp như là một bean Spring để Spring quản lý. @Component và các chuyên biệt hóa của nó , các subclass gồm (@Service, @Repository, @Controller) giúp Spring phát hiện và cấu hình các lớp này trong ứng dụng.
- **@Autowired**: Được sử dụng để tự động tiêm phụ thuộc vào các bean. Nó có thể được sử dụng trên constructor, setter methods, hoặc các trường (fields) để cung cấp các phụ thuộc cần thiết cho lớp.
- Sử dụng @Component và @Autowired giúp đơn giản hóa việc cấu hình và quản lý các thành phần của ứng dụng trong Spring Boot, làm giảm sự phụ thuộc chặt chẽ và tăng tính mở rộng của ứng dụng.

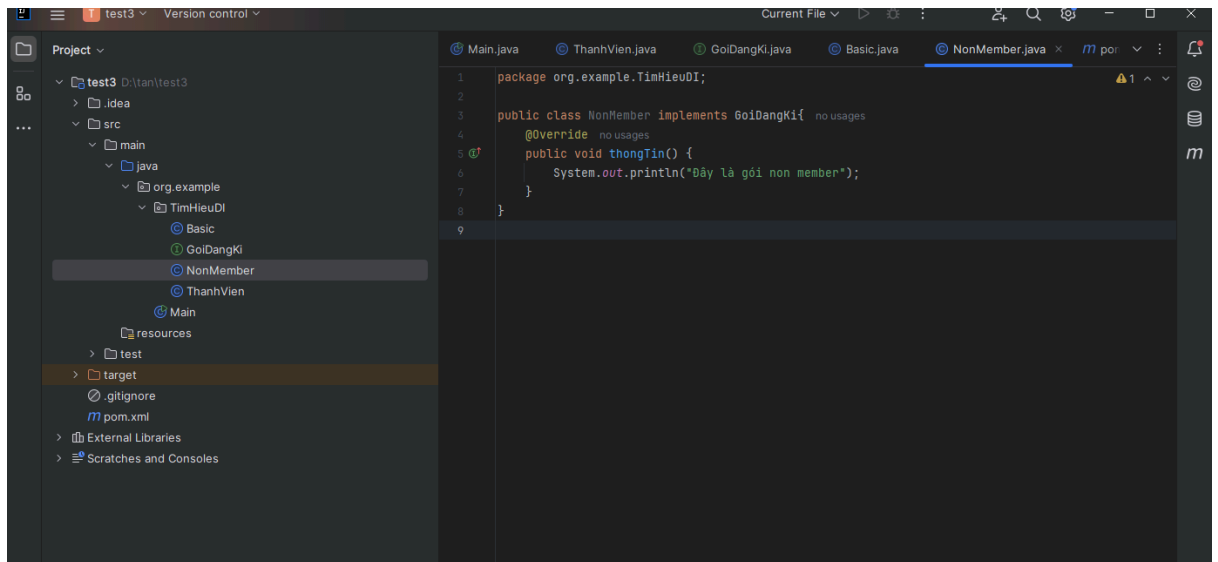
6. Tác dụng của dùng IOC

- Tách Rời và Linh Hoạt:

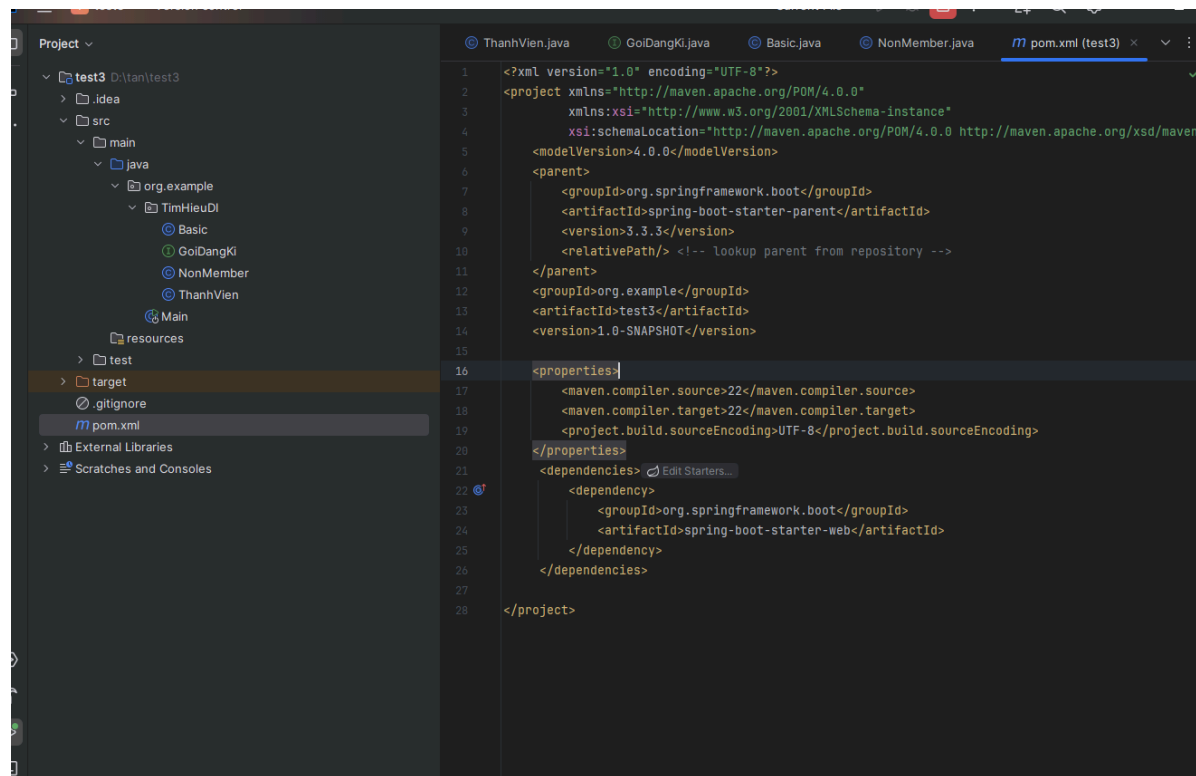
- **Tách Rời Các Thành Phần:** IoC container giúp tách rời các thành phần của ứng dụng bằng cách quản lý các phụ thuộc giữa chúng. Các thành phần không còn phải tự tạo hoặc quản lý các phụ thuộc của chúng, điều này giúp giảm sự phụ thuộc lẫn nhau và làm cho hệ thống linh hoạt hơn.
- **Dễ Thay Đổi:** Do các thành phần được tách rời, bạn có thể dễ dàng thay đổi các triển khai hoặc cấu hình mà không ảnh hưởng đến các phần khác của ứng dụng. Điều này làm cho hệ thống dễ bảo trì và mở rộng hơn.
- **Tiêm Phụ Thuộc Tự Động:**
 - **Tiêm Qua Constructor:** IoC container có thể tự động tiêm các phụ thuộc vào lớp qua constructor, giảm thiểu nhu cầu quản lý phụ thuộc một cách thủ công.
 - **Tiêm Qua Setter:** Các phụ thuộc có thể được tiêm qua các phương thức setter, cho phép các phụ thuộc là tùy chọn hoặc có thể thay đổi sau khi đối tượng được khởi tạo.
 - **Tiêm Vào Trường:** Các phụ thuộc có thể được tiêm trực tiếp vào các trường, mặc dù cách này ít được khuyến khích hơn so với tiêm qua constructor hoặc setter vì vấn đề về đóng gói và tính bất biến.
- **Quản Lý Cấu Hình:**
 - **Cấu Hình Tập Trung:** IoC container quản lý cấu hình của các thành phần ứng dụng. Cấu hình có thể được thực hiện thông qua các annotation, XML, hoặc các lớp cấu hình Java, giúp tập trung và tiêu chuẩn hóa việc quản lý cấu hình.
 - **Cấu Hình Theo Profile:** Hỗ trợ cấu hình khác nhau dựa trên các profile của ứng dụng (ví dụ: phát triển, kiểm thử, sản xuất), cho phép áp dụng các chiến lược cấu hình linh hoạt hơn.
- **Quản Lý Vòng Đời Đối Tượng:**
 - **Quản Lý Vòng Đời:** IoC container quản lý vòng đời của các đối tượng trong ứng dụng, bao gồm việc khởi tạo, sử dụng, và hủy các đối tượng. Điều này giúp giảm thiểu mã nguồn phải viết cho việc quản lý đối tượng và giúp đảm bảo rằng các đối tượng được quản lý một cách nhất quán.
- **Tăng Cường Tính Testable:**
 - **Dễ Kiểm Thử:** Do các thành phần được tách rời và phụ thuộc được tiêm tự động, việc kiểm thử các thành phần đơn lẻ trở nên dễ dàng hơn. Bạn có thể dễ dàng mock hoặc stub các phụ thuộc trong các bài kiểm thử.

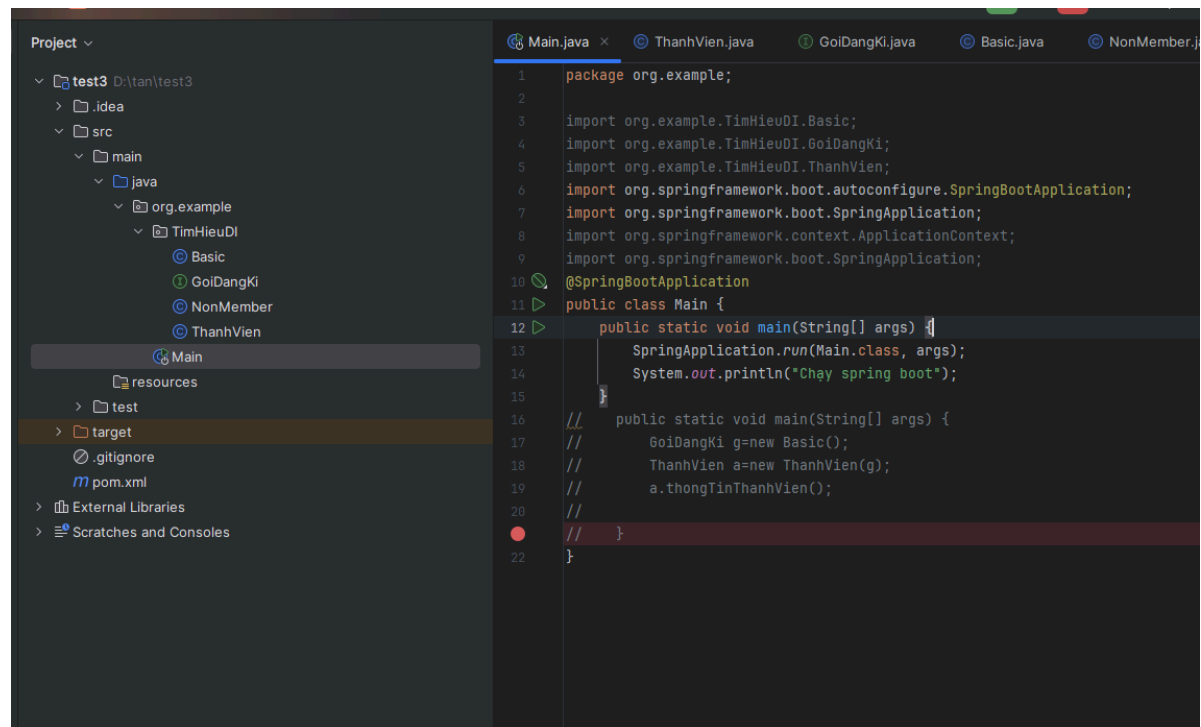




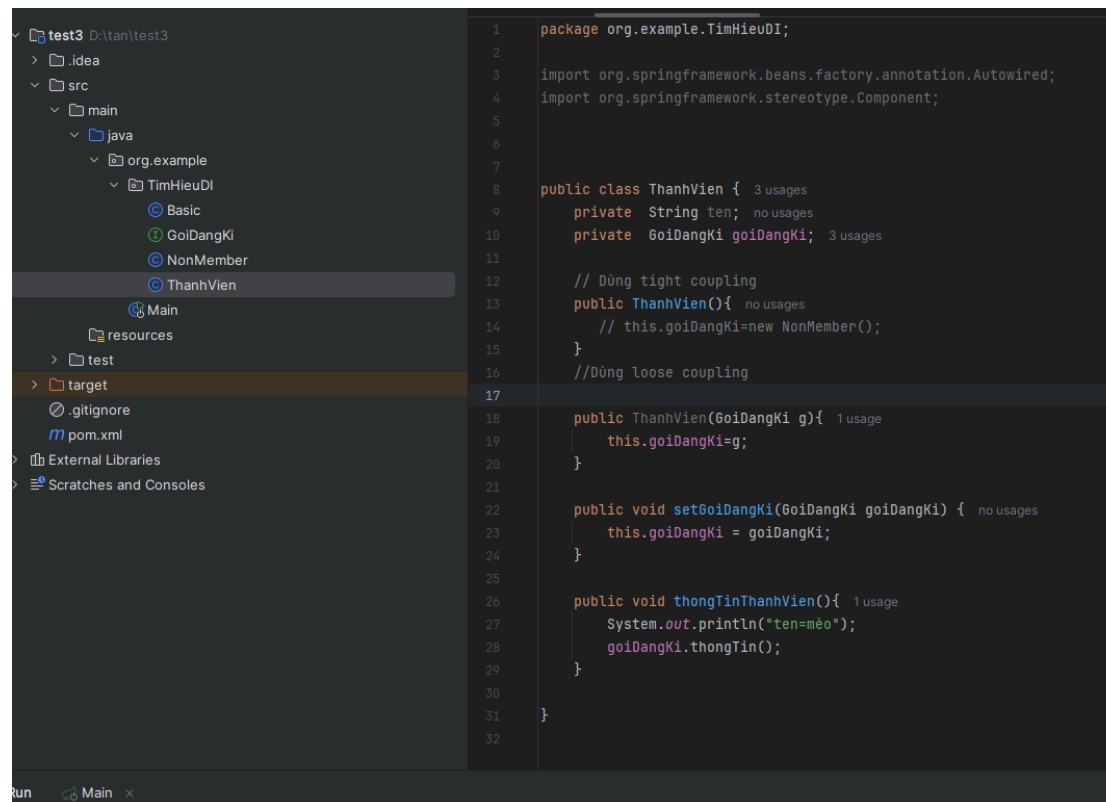


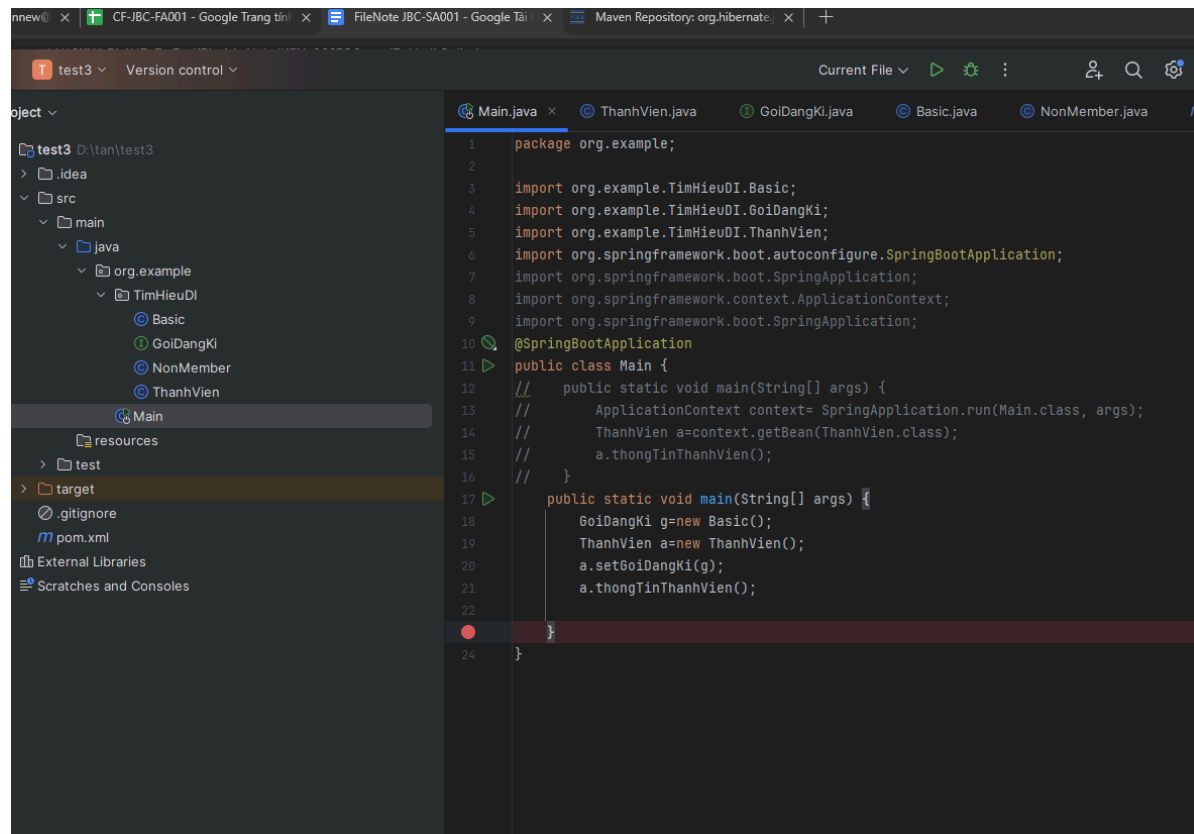
Tạo spring boot



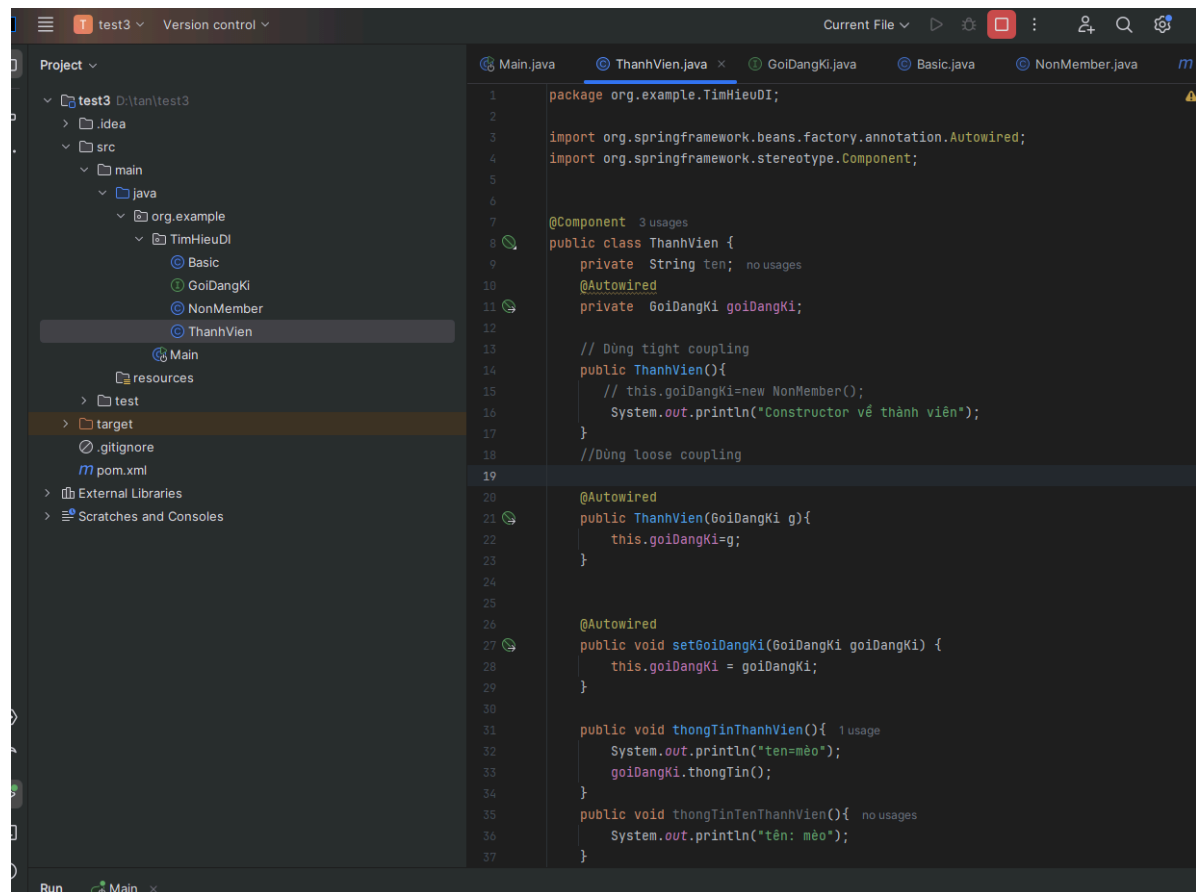
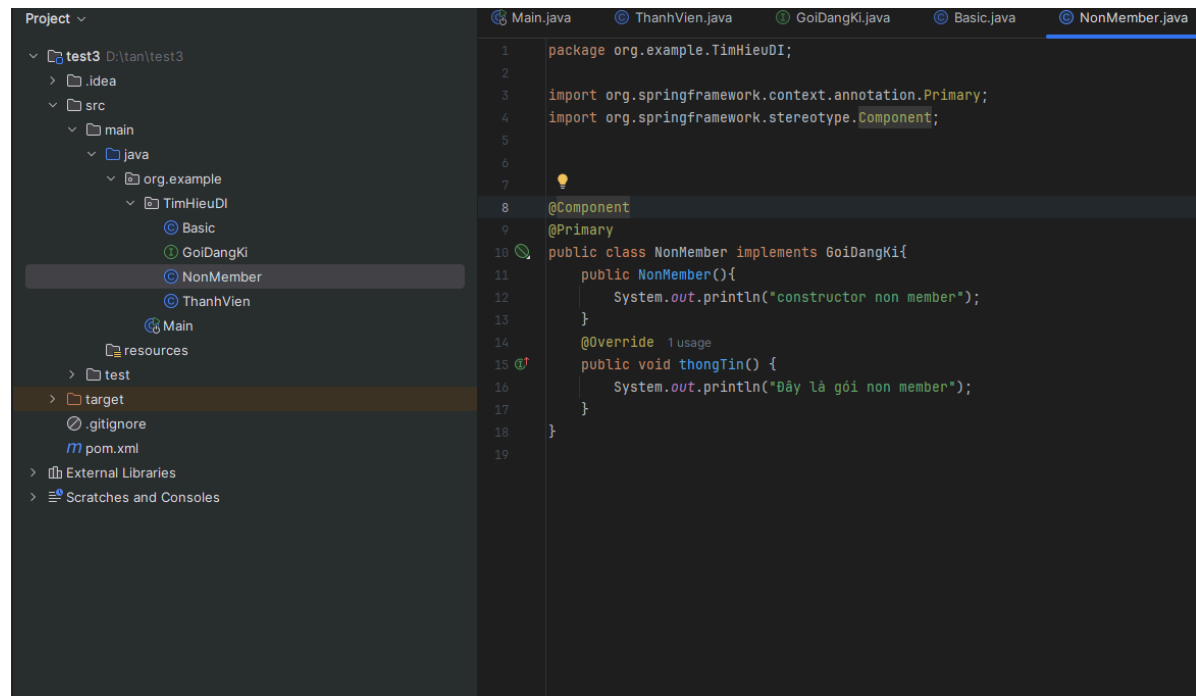


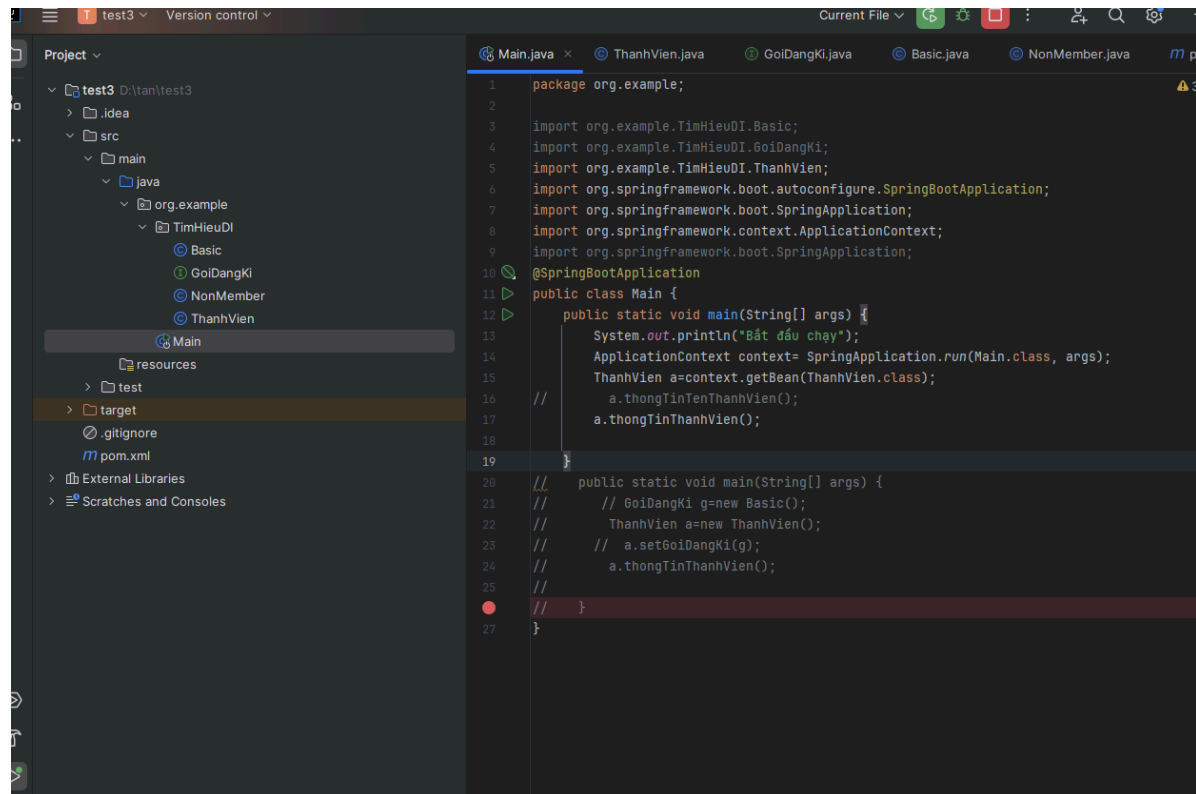
- Tiêm bằng setter



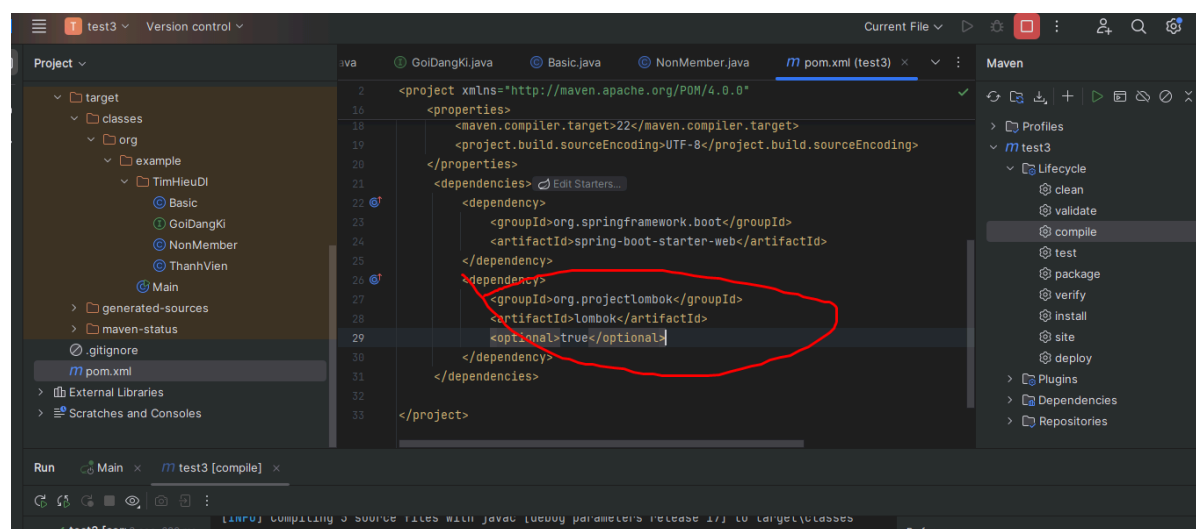


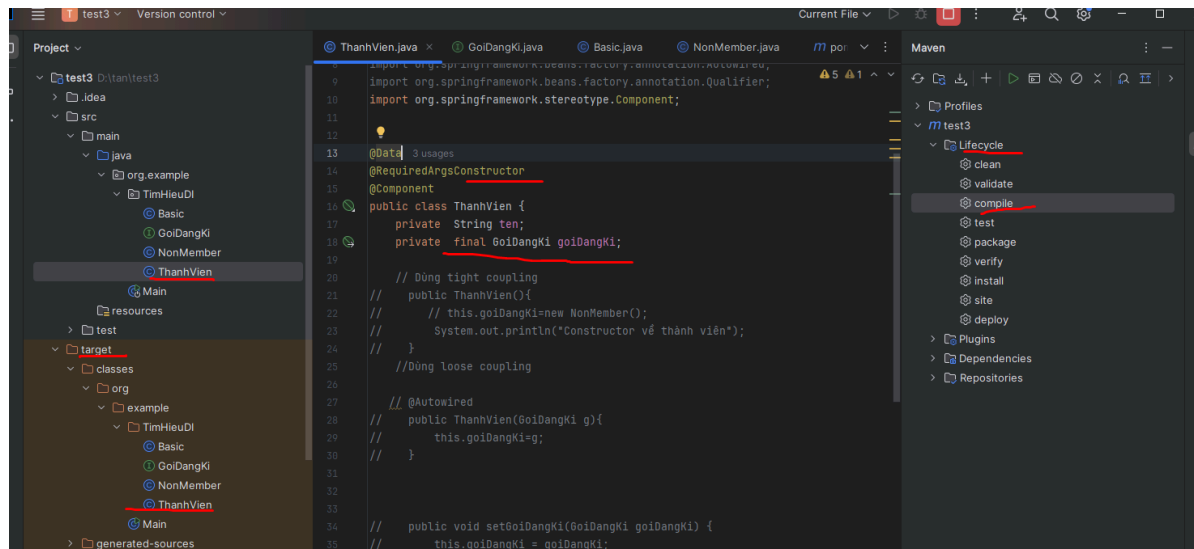
Cách tiêm tự động trong cơ chế IOC container



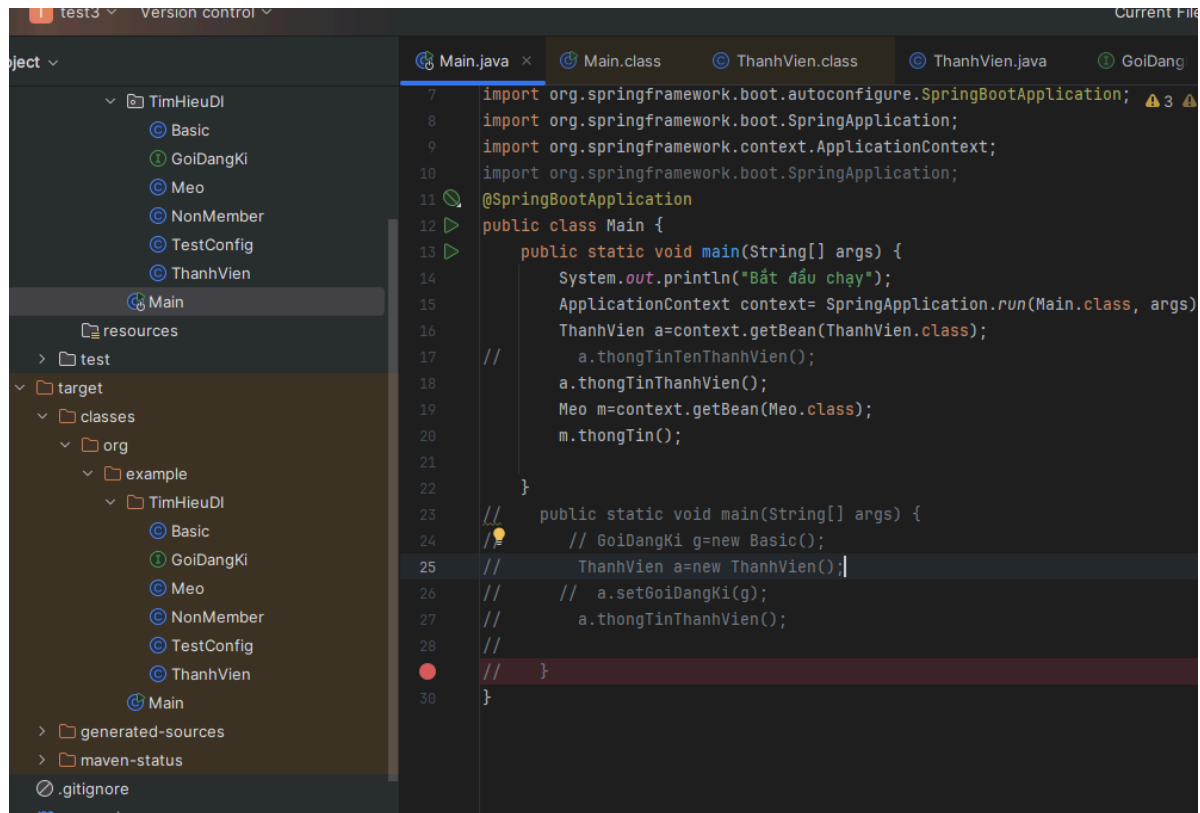


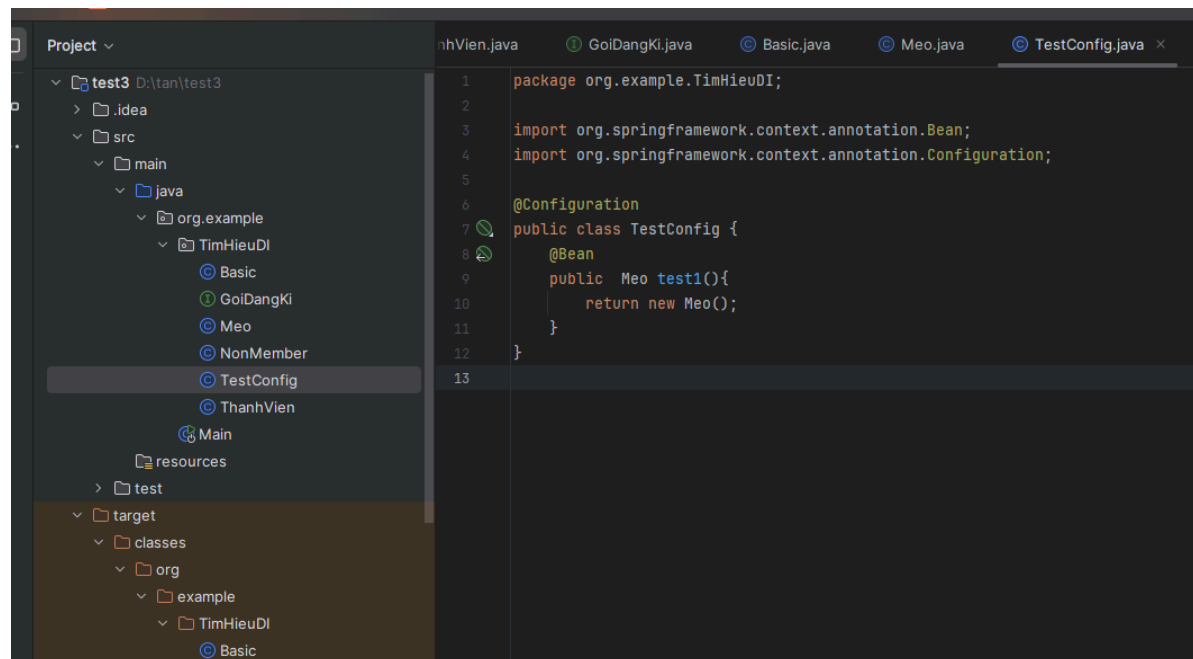
Tạo lombok





Dùng bean





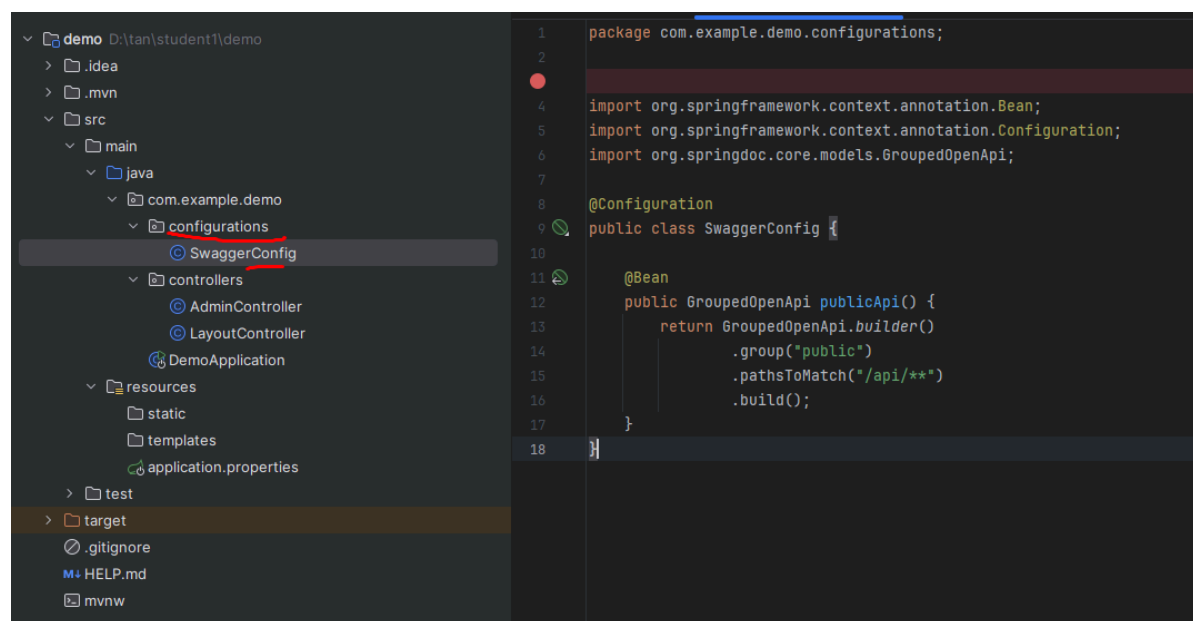
Tìm hiểu swagger

- springdoc-openapi-starter-webmvc-ui để cấu hình Swagger cho ứng dụng Spring Boot, quá trình này khá đơn giản và trực quan.
springdoc-openapi-starter-webmvc-ui là một starter library của Springdoc OpenAPI dành cho Spring MVC, giúp tích hợp Swagger UI dễ dàng vào ứng dụng Spring Boot của bạn.

<https://meet.google.com/ntc-ysie-jhb>

Swagger UI

công thức cấu hình:



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes a 'demo' directory with subdirectories like '.idea', '.mvn', 'src', 'main', 'java', 'com.example.demo', 'configurations', 'controllers', 'resources', 'static', 'templates', 'application.properties', 'test', and 'target'. The 'SwaggerConfig' class is highlighted in the 'configurations' directory. The code editor shows the following code:

```
1 package com.example.demo.configurations;
2
3
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.context.annotation.Configuration;
6 import org.springdoc.core.models.GroupedOpenApi;
7
8 @Configuration
9 public class SwaggerConfig {
10
11     @Bean
12     public GroupedOpenApi publicApi() {
13         return GroupedOpenApi.builder()
14             .group("public")
15             .pathsToMatch("/api/**")
16             .build();
17     }
18 }
```

```
src
├── main
│   └── java
│       ├── com.example.demo
│       │   ├── configurations
│       │   │   └── SwaggerConfig
│       │   ├── controllers
│       │   │   ├── AdminController
│       │   │   ├── LayoutController
│       │   │   └── DemoApplication
│       │   └── resources
│       │       ├── static
│       │       ├── templates
│       │       └── application.properties
│   └── test
├── target
├── .gitignore
├── HELP.md
├── mvnw
├── mvnw.cmd
├── pom.xml
├── External Libraries
└── Snippets and Consoles
```

```
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75

</dependency>
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.6.0</version>
</dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <excludes>
          <exclude>
```

tannguyen

Home Workspaces API Network

Search Postman

Overview GET http://loc PUT http://loc [CONFLICT] GET get all PUT New Req DEL New Req GET list New Env test1 GET index2 GET index1 meo No environment

meo

Filter variables

Variable	Type	Initial value	Current value
<input checked="" type="checkbox"/> baseurl	default	http://localhost:8080	http://localhost:8080
Add new variable			

spring initializr

Project

Language

☐ Gradle - Groovy

☐ Gradle - Kotlin

☒ Java

☐ Kotlin

☐ Groovy

☒ Maven

Spring Boot

☐ 3.4.0 (SNAPSHOT)

☐ 3.4.0 (M2)

☐ 3.3.4 (SNAPSHOT)

☒ 3.3.3

☐ 3.2.10 (SNAPSHOT)

☐ 3.2.9

Project Metadata

Group

com.example

Artifact

demo

Name

demo

Description

Demo project for Spring Boot

Package name

com.example.demo

Packaging

☒ Jar☐ War

Java

☐ 22☐ 21☒ 17

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web

WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Lombok

DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

MySQL Driver

SQL

MySQL JDBC driver.

Spring Data JPA

SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Validation

UIO

Bean Validation with Hibernate validator.

Project

demo

D:\tan\student1\demo

.idea

.mvn

src

main

java

com.example.demo

configurations

SwaggerConfig

controllers

AdminController

LayoutController

DemoApplication

resources

static

templates

application.properties

test

target

.gitignore

HELP.md

mvnw

mvnw.cmd

pom.xml

External Libraries

Scratches and Consoles

pom.xml (demo)

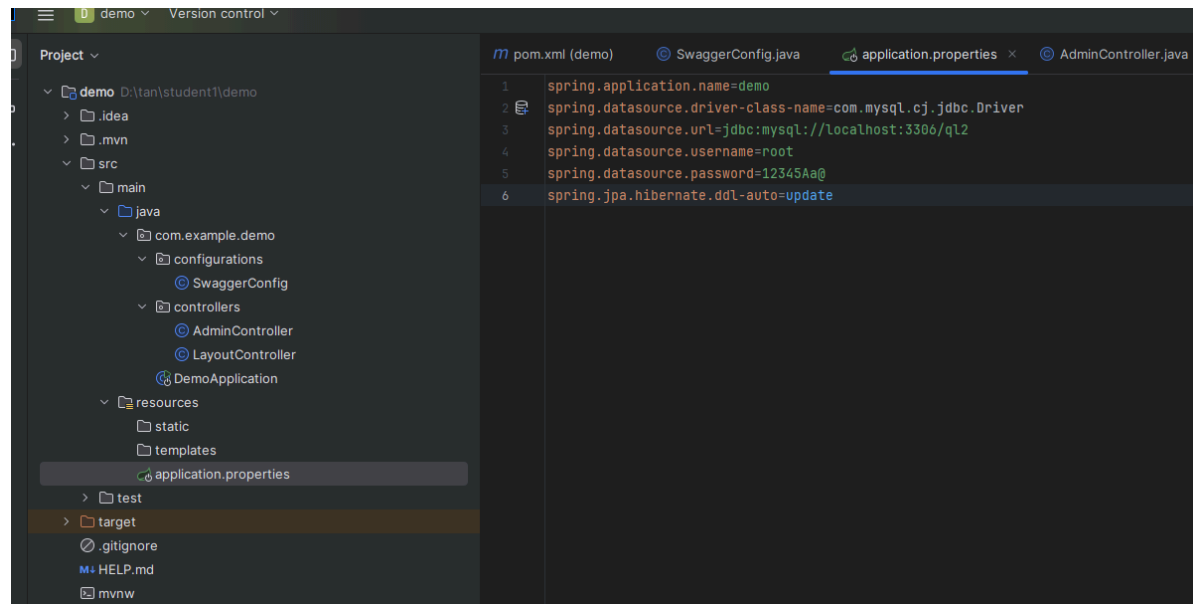
SwaggerConfig.java

application.properties

AdminController.java

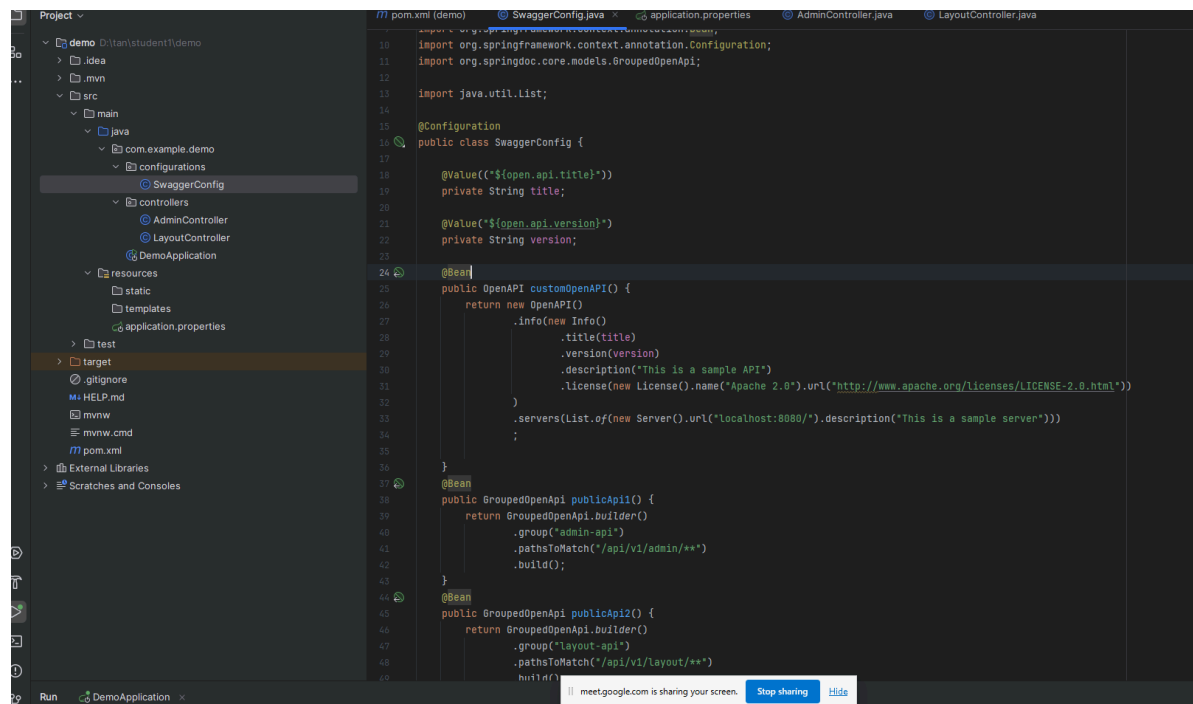
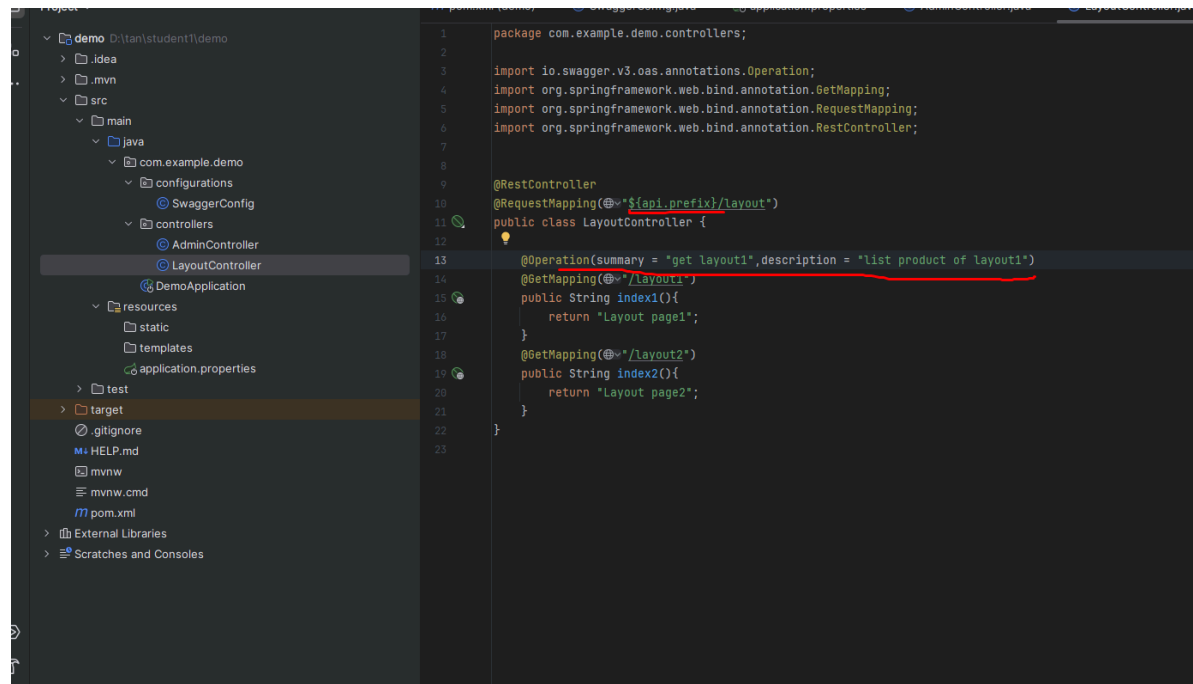
LayoutController.java

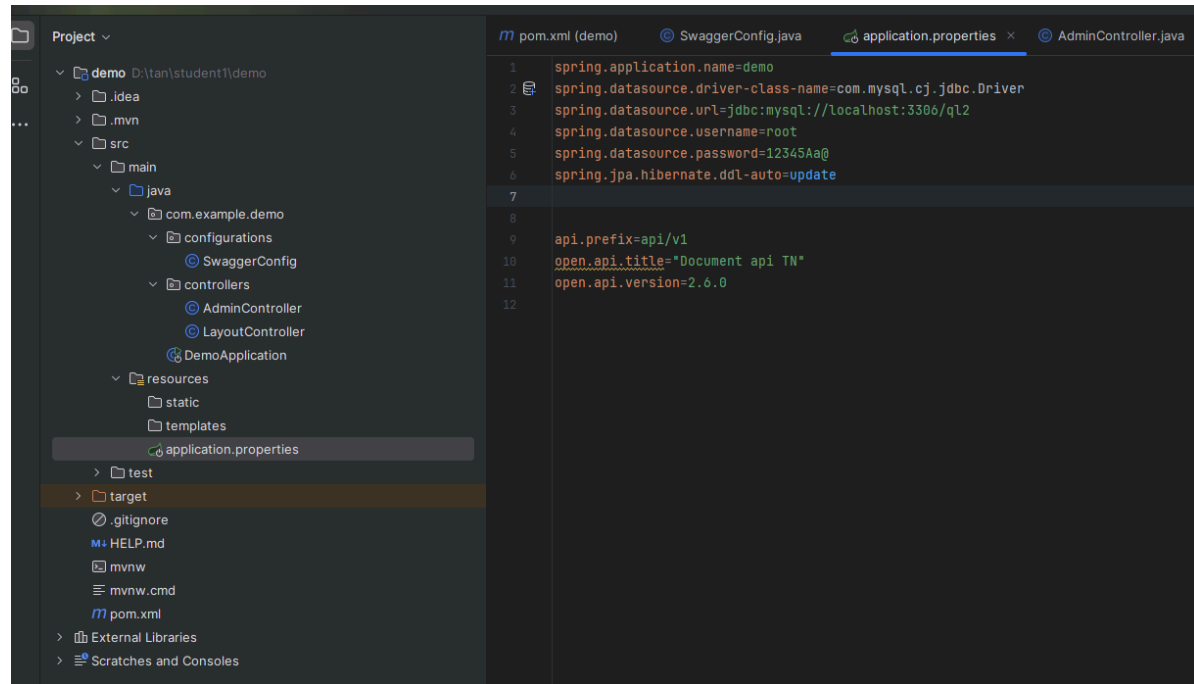
```
1 package com.example.demo.controllers;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.bind.annotation.RestController;
7
8 @RestController
9 @RequestMapping(value = "/api/admin")
10 public class AdminController {
11     @GetMapping(value = "/admin1")
12     public String index1(){
13         return "Admin page1";
14     }
15     @GetMapping(value = "/admin2")
16     public String index2(){
17         return "Admin page2";
18     }
19 }
20
```

- Bài tập: Crud student (tên, thành phố, ngày sinh, xếp loại)
- + Trong đó ngày sinh là ngày tháng năm
- + Xếp loại:Giỏi, khá, trung bình, yếu
- + Bắt lỗi validation cần thiết

<https://meet.google.com/div-nntm-kna>





Hướng dẫn crud

