FPT Education
FPT UNIVERSITY

Java

Introduction to Spring Framework

Objectives

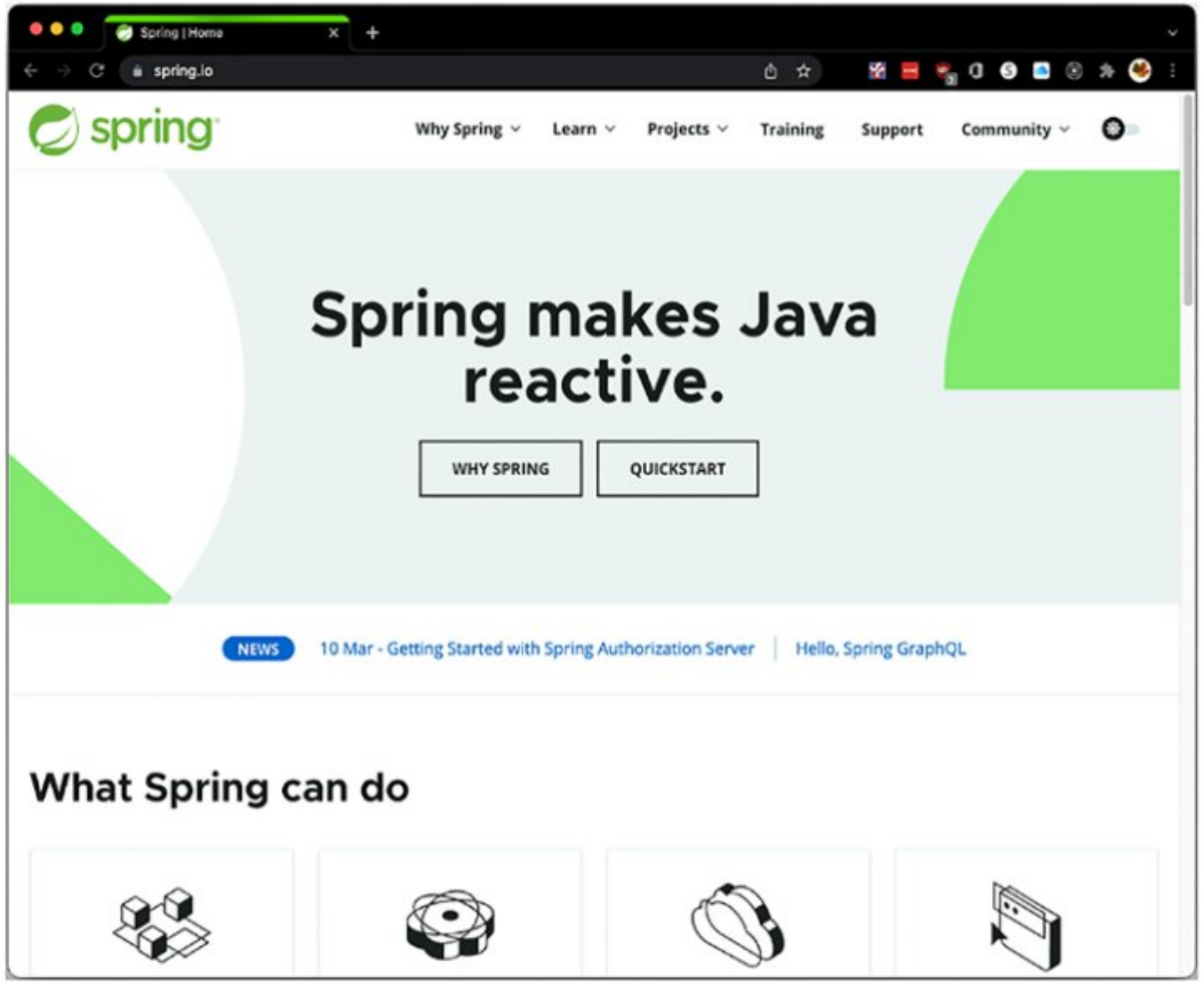
- What is Spring Framework?
- Advantages of using Spring Framework
- Key features of Spring Framework
  - Dependency Injection and Inversion of Control
  - Aspect Oriented Programming

Spring Framework Overview

What is Spring Framework?

- Spring is the most popular application development framework for enterprise Java.
- Open source Java platform since 2003.
- Spring supports all major application servers and JEE standards.
- Spring handles the infrastructure so you can focus on your application.
- Spring is the most popular application development framework for enterprise Java.
- Millions of developers around the world use Spring Framework to create high performing, easily testable, reusable code.

What is Spring Framework?



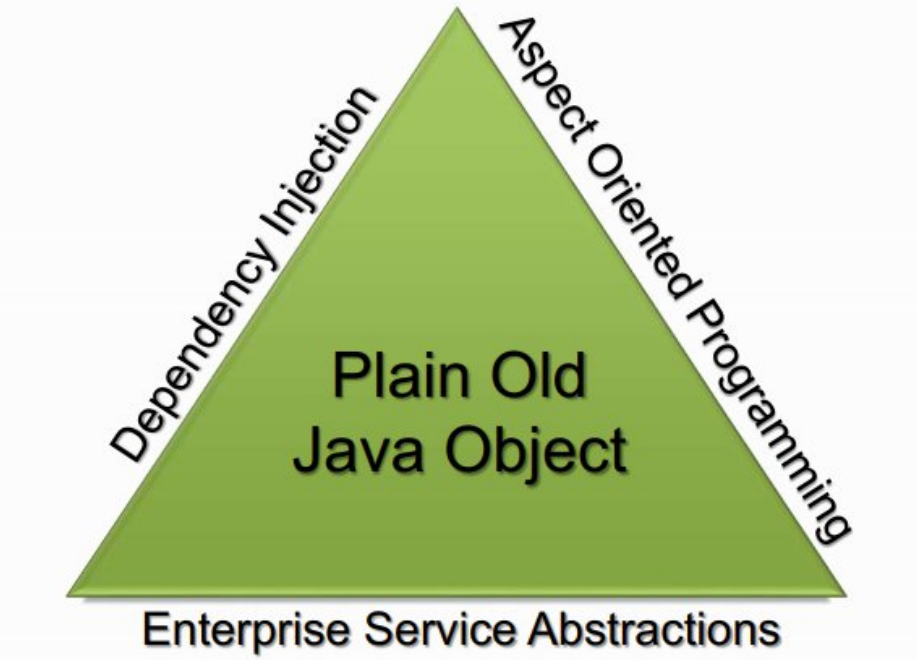
Spring Features

- **Dependency Injection:** Spring uses dependency injection to create all the dependencies and collaboration between classes.
- **Configuration Model:** The book mentions Spring Framework as having the best programming and configuration model for modern Java-based enterprise applications. Spring supports different ways to configure its container.
- **Web Applications:** Can be used to create web applications and expose RESTful APIs.
- **Messaging:** Used for sending messages via JMS, AMQP, RabbitMQ, and MQTT.
- **Dynamic Languages:** Supports dynamic languages like Groovy, Ruby, and Bean Shell for Spring.

Maven

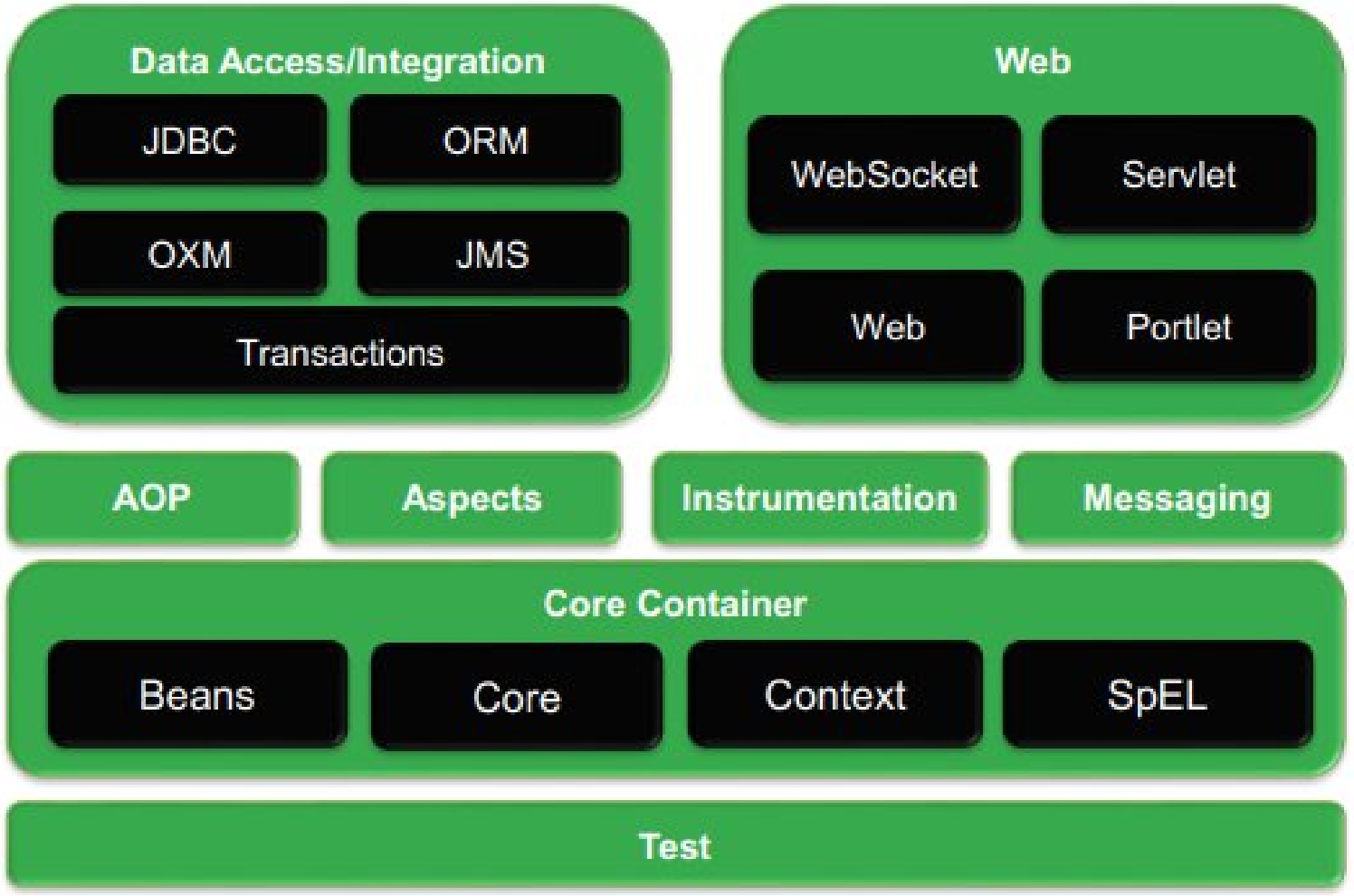
- Using Maven for Spring Framework development streamlines the dependency management, build process, project structure, and overall development workflow.
- It enhances collaboration, reduces configuration overhead, and ensures a standardized approach to building Spring applications.
- Reasons to use Maven for Spring applications
  - Dependency Management
  - Build Automation
  - Consistent Project Structure
  - Dependency Scope Management
  - Plugin Ecosystem
  - Transitive Dependency Resolution
  - Easy Project Configuration

The Spring Triangle



- *"Spring's main aim is to make J2EE easier to use and promote good programming practice. It does this by enabling a POJO-based programming model that is applicable in a wide range of environments." – Rod Johnson*

Architectural overview of Spring Framework



Core Container

The Core Container consists of the Core, Beans, Context, and Expression Language modules the details:

- The **Core** module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
- The **Bean** module provides BeanFactory, which is a sophisticated implementation of the factory pattern.
- The **Context** module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured. The ApplicationContext interface is the focal point of the Context module.
- The **SpEL** module provides a powerful expression language for querying and manipulating an object graph at runtime.

Data Access/Integration

The Data Access/Integration layer consists of the modules:

- The **JDBC** module provides a JDBC-abstraction layer that removes the need for tedious JDBC related coding.
- The **ORM** module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.
- The **OXM** module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
- The Java Messaging Service **JMS** module contains features for producing and consuming messages.
- The **Transaction** module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

Web

The Web layer consists of the Web, Web-MVC, Web-Socket, and Web-Portlet modules:

- The **Web** module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
- The **Web-MVC** module contains Spring's Model-View-Controller (MVC) implementation for web applications.
- The **Web-Socket** module provides support for WebSocket-based, two-way communication between the client and the server in web applications.
- The **Web-Portlet** module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

Spring Framework 6

- **Java 17 Support** - Spring Framework 6 is compatible with Java 17, providing developers with access to the newest Java functionalities and improvements.
- **Reactive Programming** - Enhanced support for reactive programming is a significant highlight of Spring Framework 6, enabling developers to build responsive, scalable applications efficiently.
- **Modularization** - The framework is modularized, allowing developers to choose and include only the necessary components, reducing the overhead of unused features.
- **Microservices Architecture** - Spring Framework 6 caters to the needs of building microservices-based applications, promoting flexibility and scalability in the development process.

Spring Framework 6

- Kotlin Support - Kotlin, a programming language that runs on the Java Virtual Machine and is fully interoperable with Java, is well-supported in Spring Framework 6 for developers who prefer it as an alternative language for their projects.
- Containerization and Cloud-native Development - With enhanced containerization support, Spring Framework 6 facilitates cloud-native development, aligning well with modern deployment practices such as container orchestration.
- Simplified Configuration - Spring Framework 6 simplifies configuration options, making it easier for developers to set up and manage their applications with less complexity.

Spring Framework 6

- **Developer Productivity** - The framework aims to enhance developer productivity by offering streamlined tools, integrations, and documentation to expedite the development process.
- **Spring Boot 3.0** is also the first Spring Boot GA release to support Spring Framework 6.0. As a developer, we need to be aware of these updates in order to work smoothly with Spring Framework. Undoubtedly, one of the biggest turns in the Spring Framework 6 release was the dropping of support for older versions of Java.
- The Spring Framework 6 is migrated towards Jakarta EE from Java EE. Hence, It will use the 'jakarta' packages namespace in place of 'javax' namespace.

Spring Version History

Version	Year	Note
	2002	First version was written by Rob Johnson
1.0	March 2004	
2.0	October 2006	
2.5	November 2017	
3.0	December 2009	
4.0	December 2013	Support for Java SE (Standard Edition) 8, Groovy 2, some aspects of Java EE 7, and Web Socket
4.3	June 2016	
5.0	March 2017	5.2.2 (December 2019)
6.0	November 2022	6.1.5 (March 2024), JDK 17

Spring, Java and Java EE compatibility

Spring	Java Version	Java EE Version
1.x	Full Support for JDK 1.3, 1.4	Java EE 1.3, 1.4 are fully supported.
2.0.x	Full Support for JDK 1.3, 1.4 and 1.5	Java EE 1.3, 1.4 are fully supported.
2.5	Full support for Java SE 1.4.2 or later, Early Support for Java SE 6.	Java EE 1.3, 1.4 are fully supported. Early support for Java EE 5.
3.x	Java SE 5 and 6 are fully supported	Java EE 1.4, 5 are fully supported. Early support for Java EE 6.
4.x	Java SE 6, 7 are fully supported. Early support for Java SE 8.	Java EE 6 is fully supported Early support for Java EE 7
5.x	Java SE 6, 7, 8 are fully supported Early Support for Java SE 9	Java EE 6, 7 are fully supported
6.x	JDK 17	Jakarta EE

Dependency Injection & Inversion of Control

Understanding Dependency Injection

- SOLID is a set of five object-oriented design principles that help in creating more maintainable, flexible, and scalable software.
  - Single Responsibility Principle (SRP)
  - Open/Closed Principle (OCP)
  - Liskov Substitution Principle (LSP)
  - Interface Segregation Principle (ISP)
  - Dependency Inversion Principle (DIP)
- Dependency Injection (DI) is a design pattern used to facilitate Inversion of Control (IoC) and improve the modularity and maintainability of software systems.

Benefits of Dependency Injection

- **Improved Testability** - By injecting dependencies into classes, it becomes easier to isolate components for unit testing without the need for complex setup or mocking frameworks. This promotes a more reliable and efficient testing process.
- **Reduced Coupling** - Dependency Injection helps reduce tight coupling between classes by allowing dependencies to be provided from external sources.
- **Enhanced Modularity** - Implementing Dependency Injection promotes modular design by clearly defining and managing the dependencies between various components.
- **Flexibility and Scalability** - Dependency Injection enables a more flexible and scalable architecture, as components can be easily replaced or extended without modifying the existing codebase.

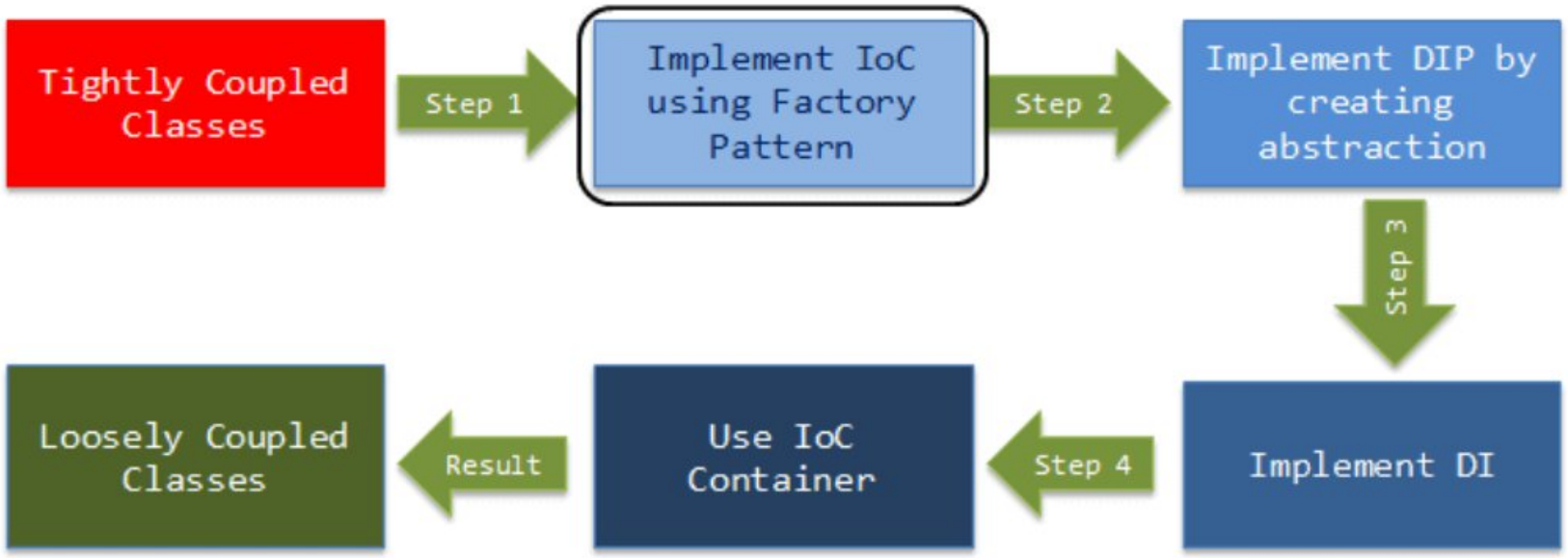
Benefits of Dependency Injection

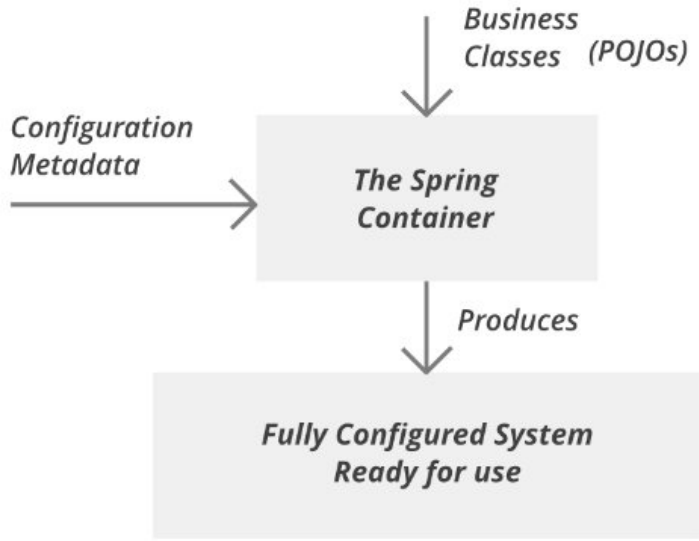
- **Promotes Design Patterns** - Dependency Injection encourages the adoption of design patterns such as Inversion of Control (IoC) and Dependency Inversion Principle (DIP). These patterns contribute to better code organization, improved readability, and overall software design quality.
- **Encourages Separation of Concerns** - Dependency Injection supports the separation of concerns principle by clearly defining the roles and responsibilities of different components.
- **Integration with DI Containers** - Dependency Injection can be utilized in conjunction with Dependency Injection Containers (such as Spring Framework or Google Guice), which help manage the injection of dependencies and further streamline the development process.

Inversion of Control

- Inversion of Control (IoC) is a design principle where the control over object creation and management is taken away from the objects themselves and inverted (controlled externally).
- In traditional programming, objects are responsible for creating and managing their dependencies, leading to tightly coupled and less maintainable code. With Inversion of Control, the responsibility of creating and managing objects and their dependencies is shifted to an external entity.

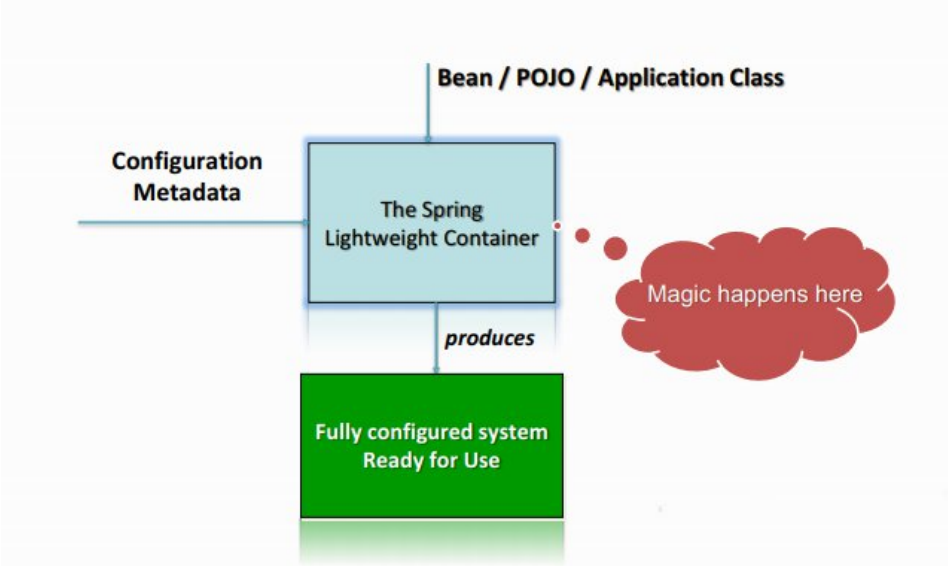
Inversion of Control





Spring IoC Container

- The Spring IoC creates the objects, wire them together, configure them, and manage their complete lifecycle from creation till destruction.
- The Spring container uses dependency injection (DI) to manage the components that make up an application.



Types of Dependency Injection

- *Dependency injection (DI)* comes with two flavors
  - Constructor-based Dependency Injection
    - Accomplished by the container invoking a constructor with a number of arguments, each representing a dependency.
    - Calling a static factory method with specific arguments to construct the bean is nearly equivalent, and this discussion treats arguments to a constructor and to a static factory method similarly.
  - Setter-based Dependency Injection
    - Accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

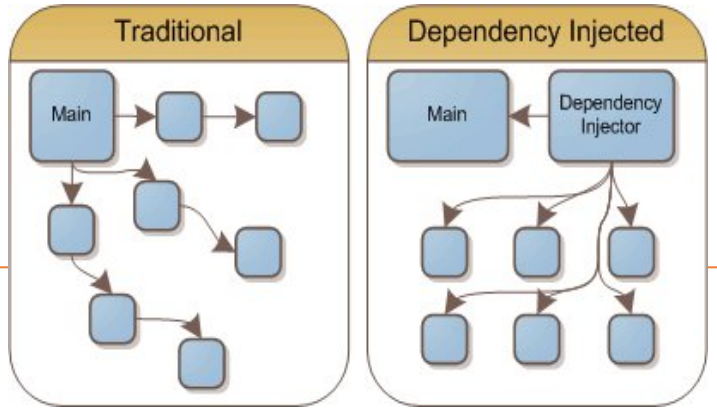
Beans

- The objects that form the backbone of your application and that are managed by the Spring IoC container are called beans.
- A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.
- These beans are created with the *configuration metadata* that you supply to the container, for example, in the form of XML <bean/> definitions or @Bean

Bean Scopes

Scope	Description
singleton	This scopes the bean definition to a single instance per Spring IoC container (default).
prototype	This scopes a single bean definition to have any number of object instances.
request	This scopes a bean definition to an HTTP request.
session	This scopes a bean definition to an HTTP session.
application	This scopes a bean definition to a ServletContext
websocket	This scopes a bean definition to a WebSocket session.

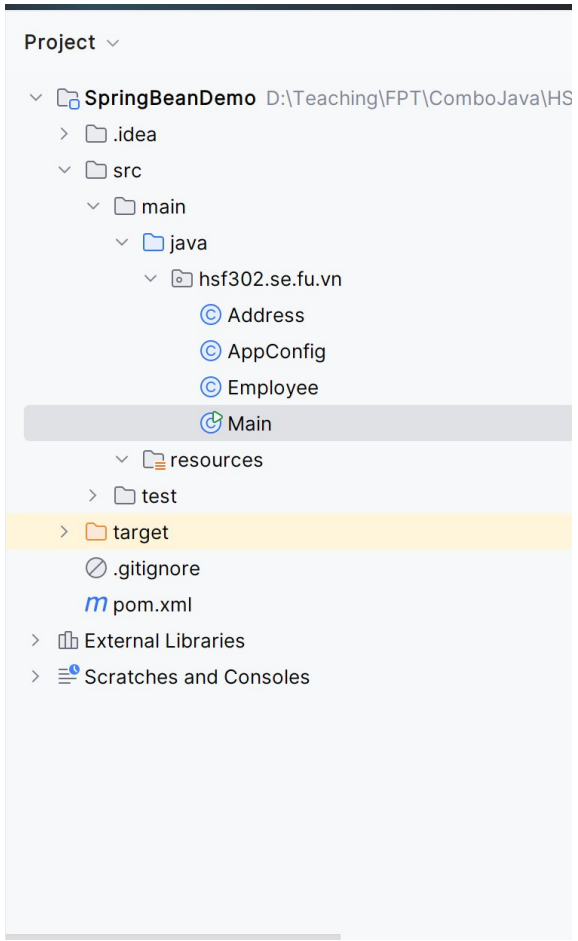
Dependency Injection Demo



- Step 01. Create Project Java
- Step 02. Configure Project
- Step 03. Configure the dependencies of Spring in pom.xml file
- Step 04. Create class named Address
- Step 05. Create class named Employee. Make the dependency between Employee and Address, inject Address to Employee class using Constructor/Setter.
- Step 06. Create the AppConfig.java
- Step 07. Test and run program

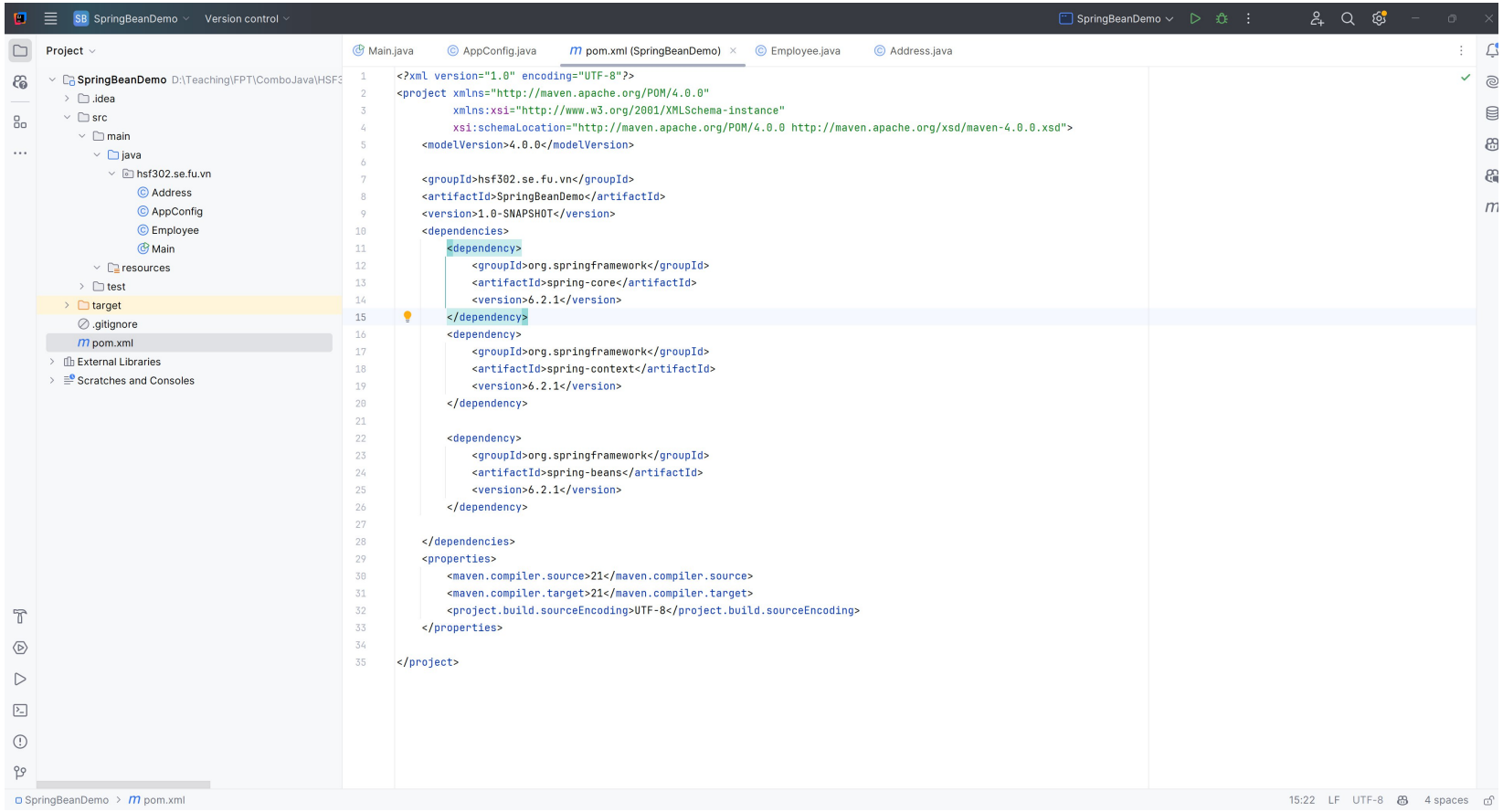
Dependency Injection Demo

- Step 01. Create Project Java
- Step 02. Configure Project



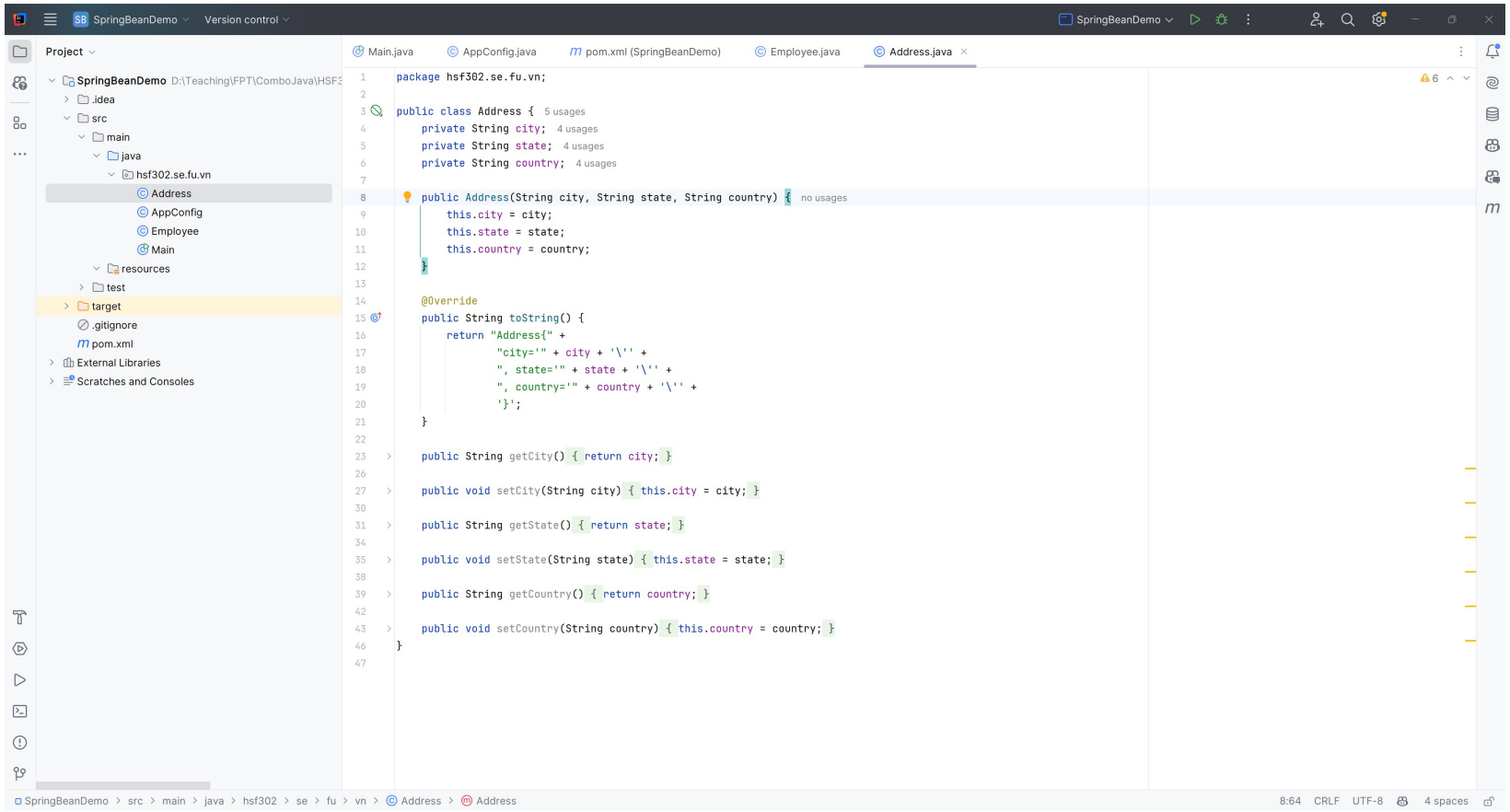
Dependency Injection Demo

- Step 03. Configure the dependencies of Spring in pom.xml file



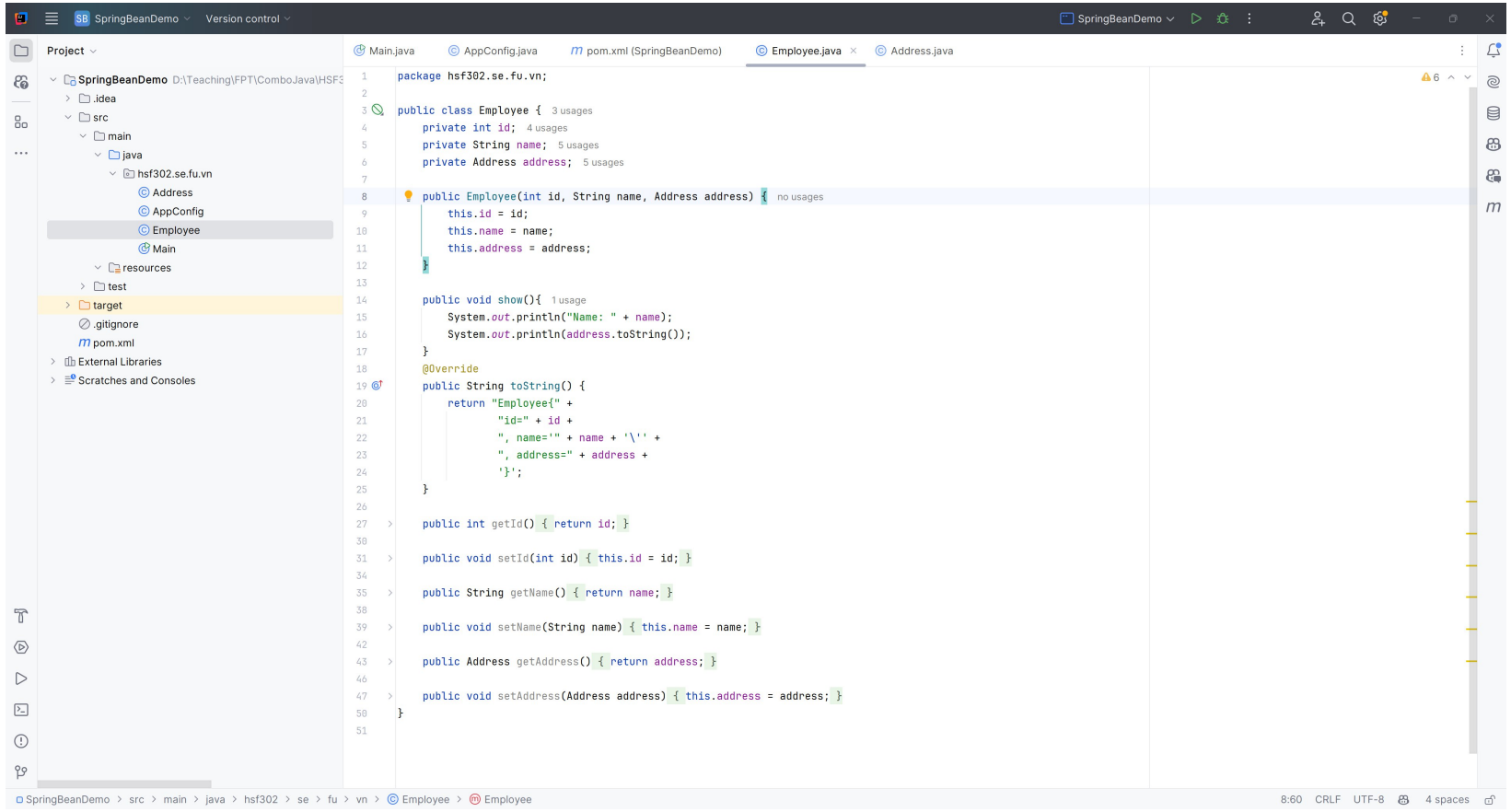
Dependency Injection Demo

- Step 04. Create class named Address



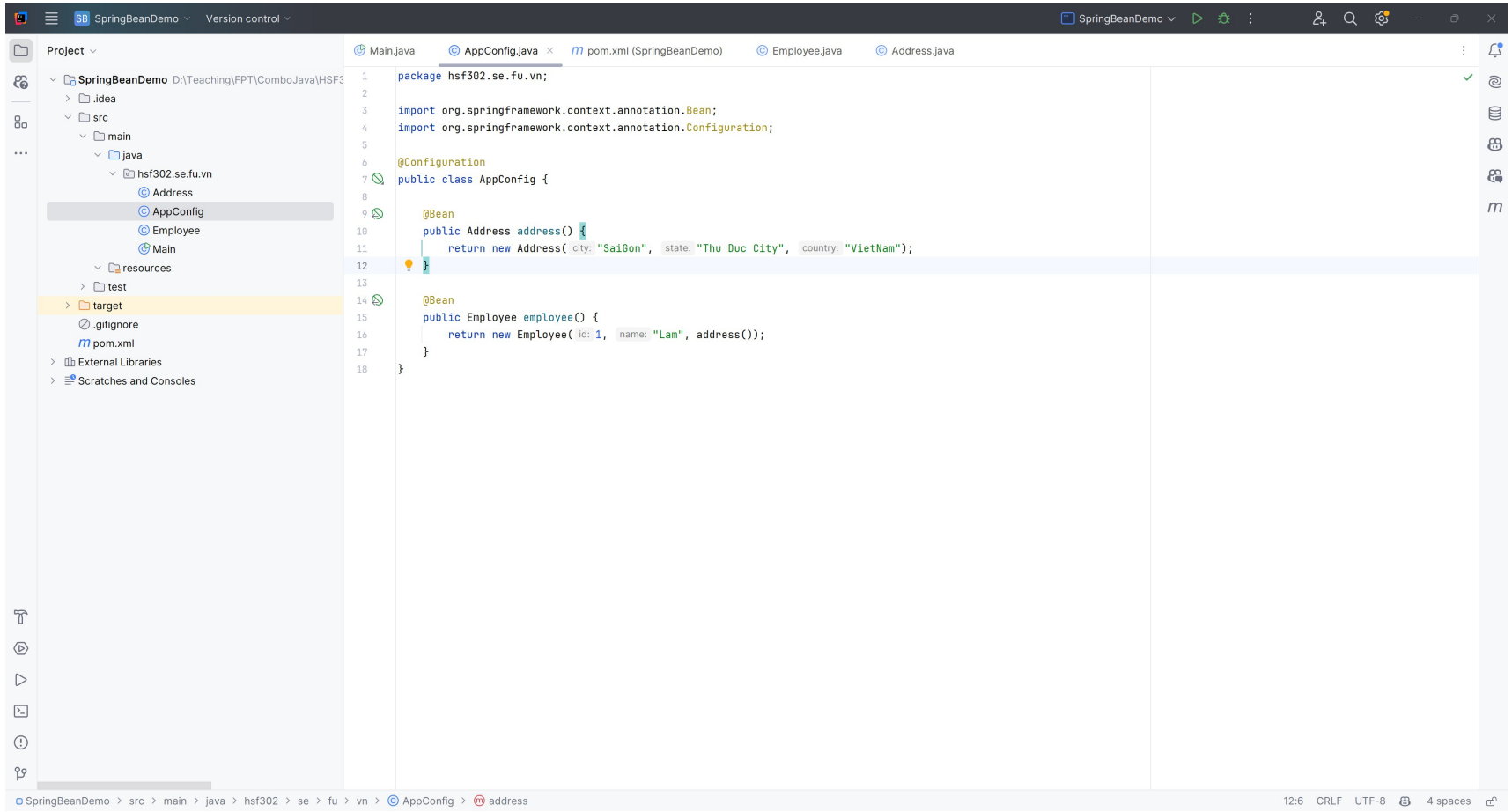
Dependency Injection Demo

- Step 05. Create class named Employee. Make the dependency between Employee and Address, inject Address to Employee class using Constructor/Setter.



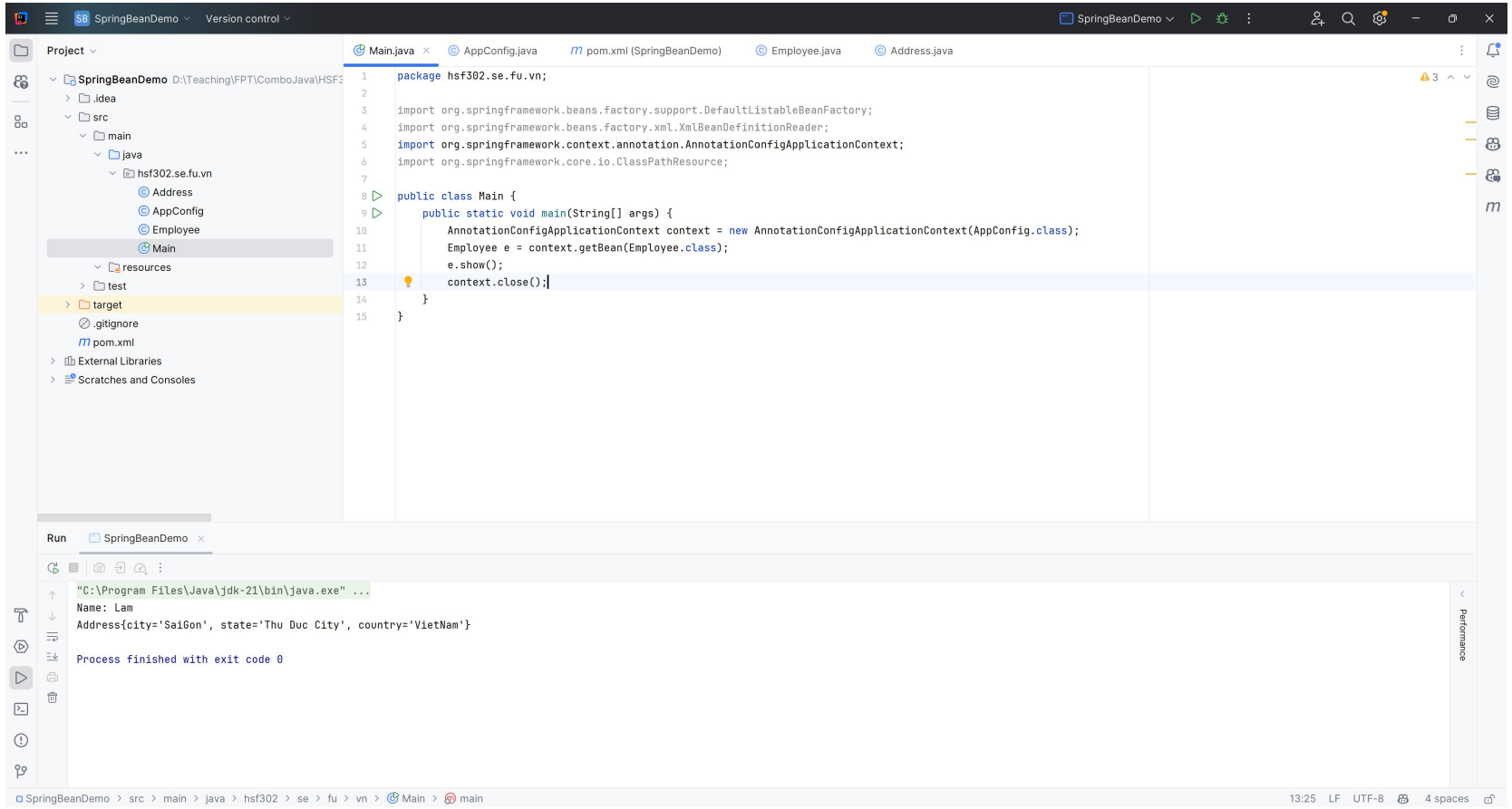
Dependency Injection Demo

- Step 06. Create the file : AppConfig.java



Dependency Injection Demo

- Step 08. Test and run program



Naming Spring Beans

- Every bean has one or more identifiers.
- Identifiers must be unique within the container that hosts the bean.
- A bean usually has only one identifier, but if it requires more than one, the extra ones can be considered *aliases*.
- The id attribute allows you to specify exactly one id.
- If you want to introduce other aliases to the bean, you can specify them in the name attribute, separated by a comma (,), semicolon (;), or white space.
- You are not required to supply a name or id for a bean. If no name or id is supplied explicitly, the container generates a unique name for that bean. However, if you want to refer to that bean by name you must provide a name.
- Motivations for not supplying a name are related to using inner beans and *autowiring* collaborators.

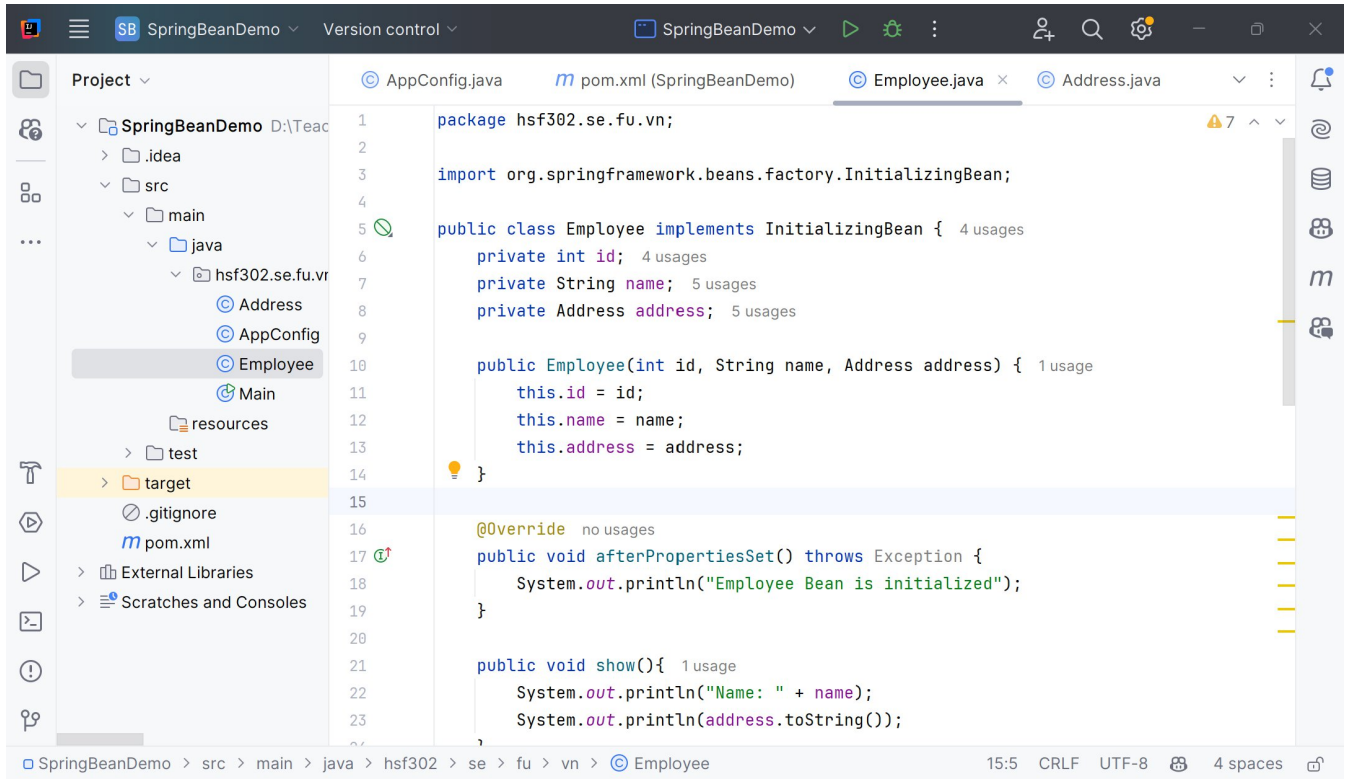
Beans - Lifecycle

- The life cycle of a Spring bean is easy to understand.
- When a bean is instantiated, it may be required to perform some initialization to get it into a usable state.

- When the bean is no longer required and is removed from the container, some cleanup may be required.

Beans – Lifecycle – Initialization

- The org.springframework.beans.factory.InitializingBean interface specifies a single method: void afterPropertiesSet() throws Exception;



Beans – Lifecycle – Initialization

- Annotate the method with @PostConstruct

```
@PostConstruct
public void init() {
    ...
}
```

- A PostConstruct interceptor method must not throw application exceptions, but it may be declared to throw checked exceptions including the java.lang.Exception if the same interceptor method interposes on business or timeout methods in addition to lifecycle events.
- If a PostConstruct interceptor method returns a value, it is ignored by the container.

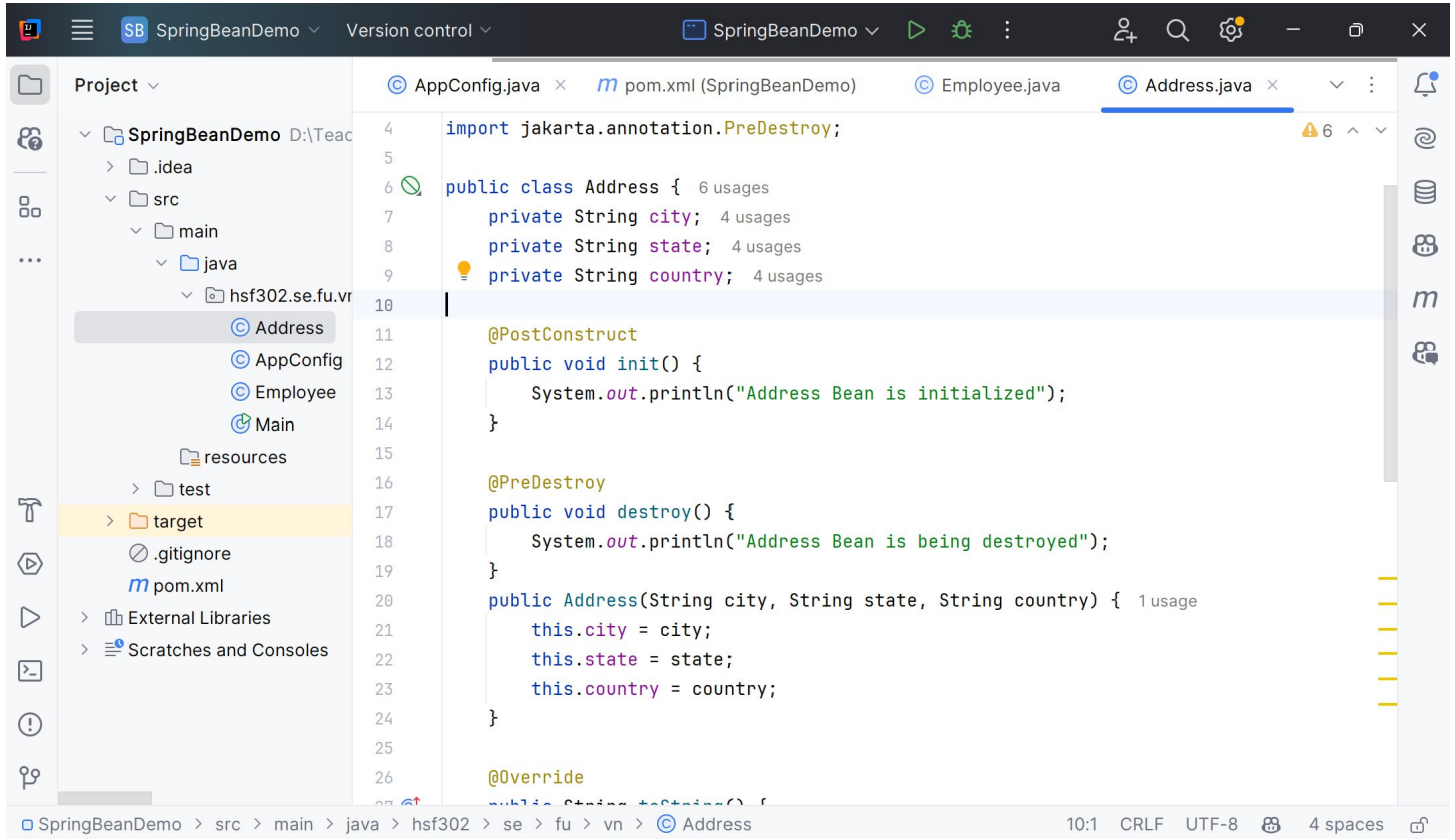
Beans – Lifecycle – Destruction

- Annotate the method with @PreDestroy

```
@PreDestroy
public void destroy() {
    ...
}
```

Beans – Lifecycle – Initialization

- Annotate the method with @PostConstruct and @PreDestroy



Configuration & Advanced Features

The @Autowired annotation in the Spring

- Dependency Injection: The @Autowired annotation is used to automatically inject dependent beans into the associated fields, constructors, or methods within a Spring component.
- Field Injection: In the case of field injection, you can directly annotate the field to be injected using @Autowired.

```
@Component
public class MyComponent {
    @Autowired
    private MyDependency dependency;
    // ...
}
```

The @Autowired annotation in the Spring

- Constructor Injection: Alternatively, you can use @Autowired on a constructor to automatically inject the dependencies.

```
@Service
public class MyService {
    private final MyRepository repository;
    @Autowired
    public MyService(MyRepository repository) {
        this.repository = repository;
    }
    // ...
}
```

The @Autowired annotation in the Spring

- Method Injection: The @Autowired annotation can also be used on methods, allowing for method-level dependency injection.

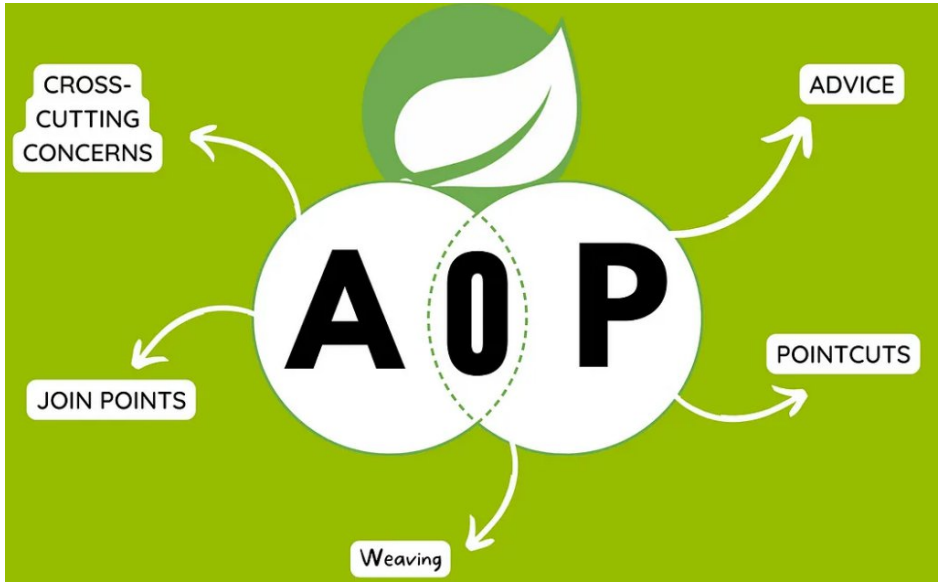
```
@Controller
public class MyController {
    private MyService service;
    @Autowired
    public void setService(MyService service) {
        this.service = service;
    }
    // ...
}
```

The @Autowired annotation in the Spring

- Qualifiers: In situations where there are multiple beans of the same type, the @Qualifier annotation can be combined with @Autowired to specify the exact bean to be injected.
- Optional Injection: The @Autowired annotation supports optional injections, meaning that if the specified bean is not found, the injection will be gracefully handled.

Aspect-Oriented Programming (AOP)

- AOP is a programming paradigm that enables modularization of cross-cutting concerns in software systems.
- In Spring, AOP complements OOP by providing a way to dynamically add behavior to the existing code without modifying it directly.



AOP Core Concepts

- **Aspect:** A module that encapsulates behaviors affecting multiple classes. In Spring AOP, aspects are implemented using regular classes annotated with @Aspect.
- **Advice:** Defines the additional behavior to be applied at a particular join point. Types of advice include "before", "after", "around", etc.
- **Join Point:** A point during the execution of a program where an aspect can be plugged in.
- **Pointcut:** A set of join points where advice should be executed. It defines the expressions that target specific methods.

Benefits of AOP

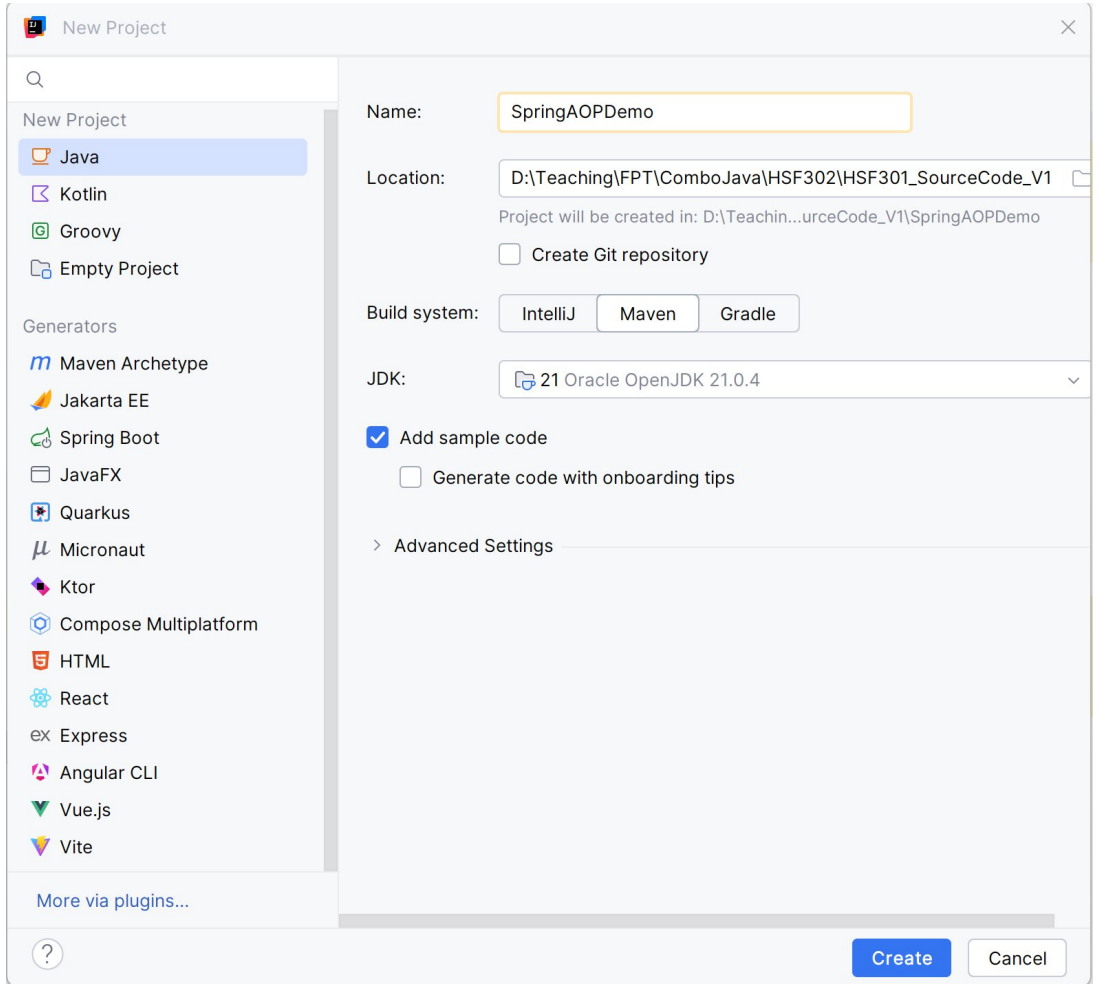
- Modularity: Separation of cross-cutting concerns into aspects promotes cleaner and more maintainable code.
- Reusability: Aspects can be applied to multiple classes or components.
- Cleaner Code: Business logic remains clean and focused, undisturbed by cross-cutting concerns like logging, security, etc.
- Dynamic Application: Aspects can be added or removed without modifying the core application code.

FPT Education
FPT UNIVERSITY

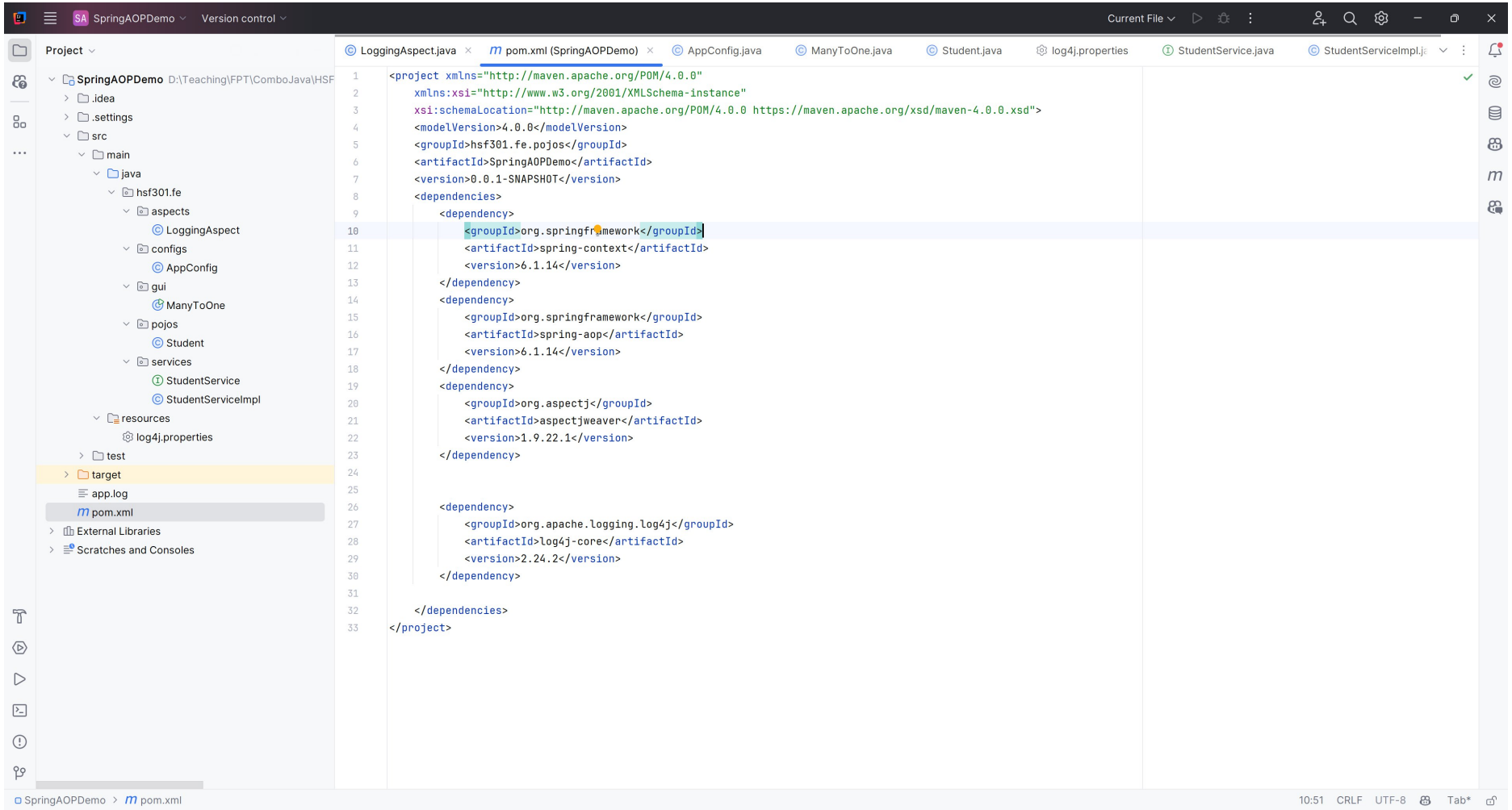
Java

Aspect-Oriented Programming Demo

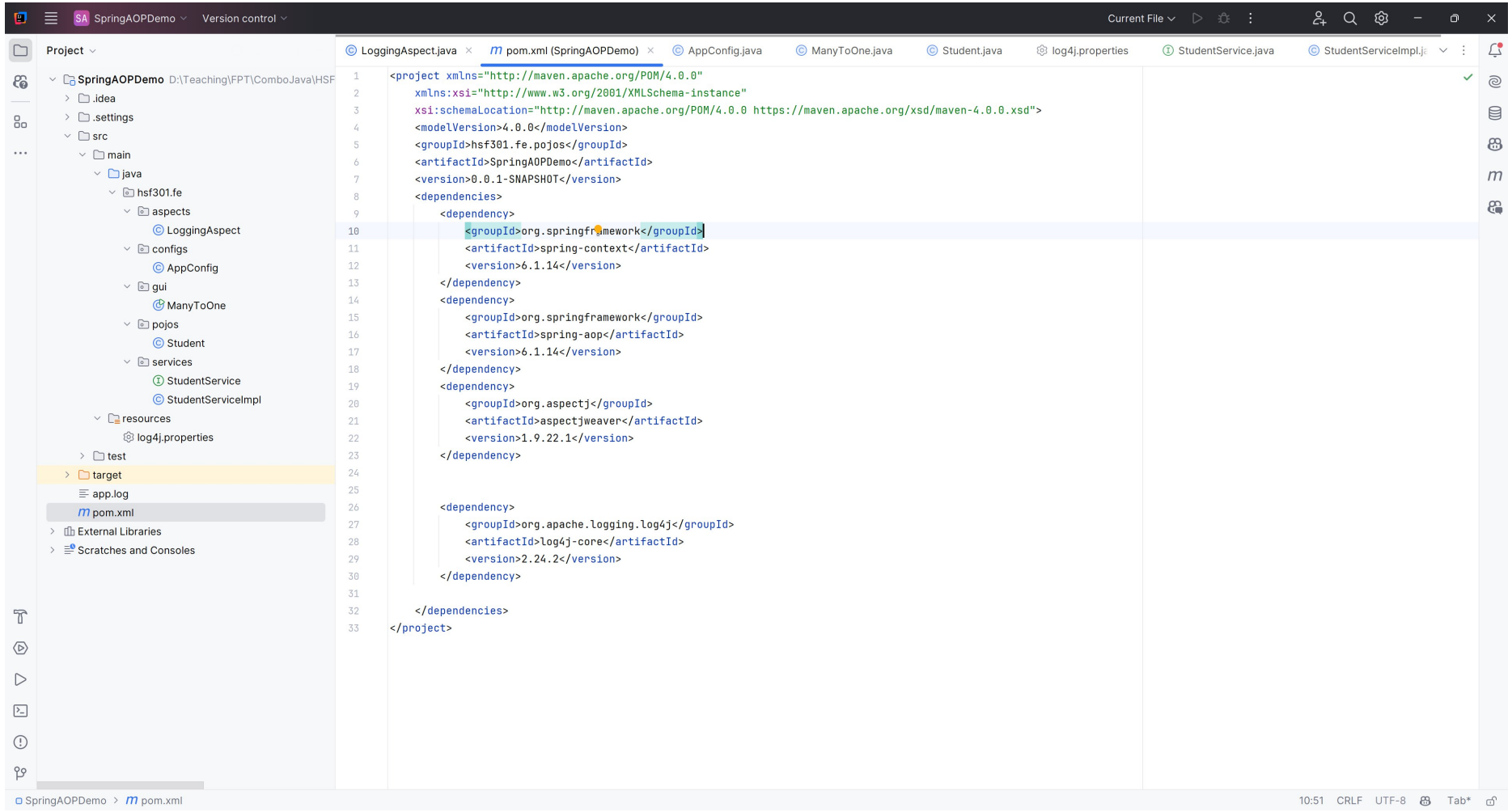
Open IntelliJ, File | New | Maven Project



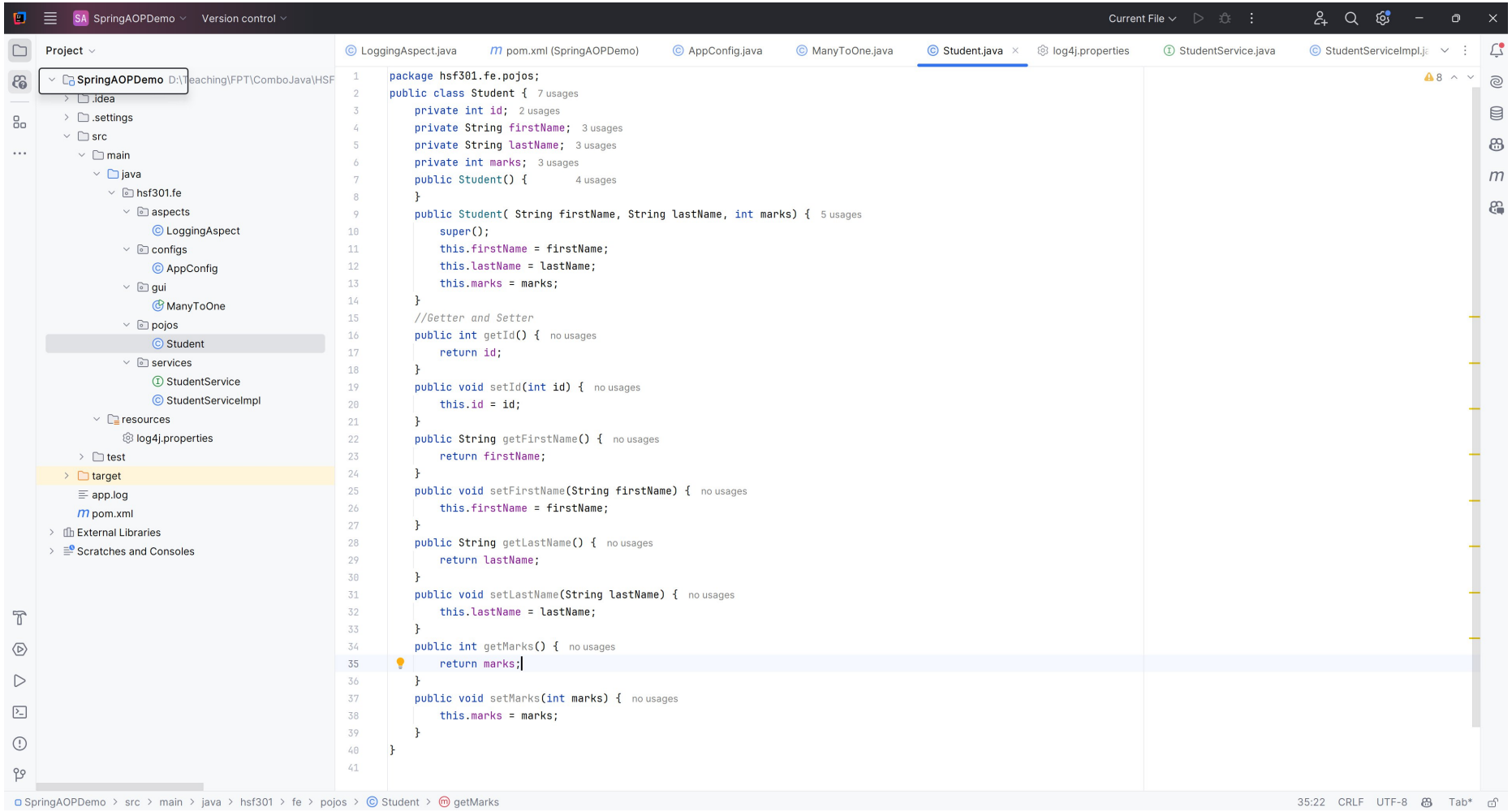
Create the Structure Project



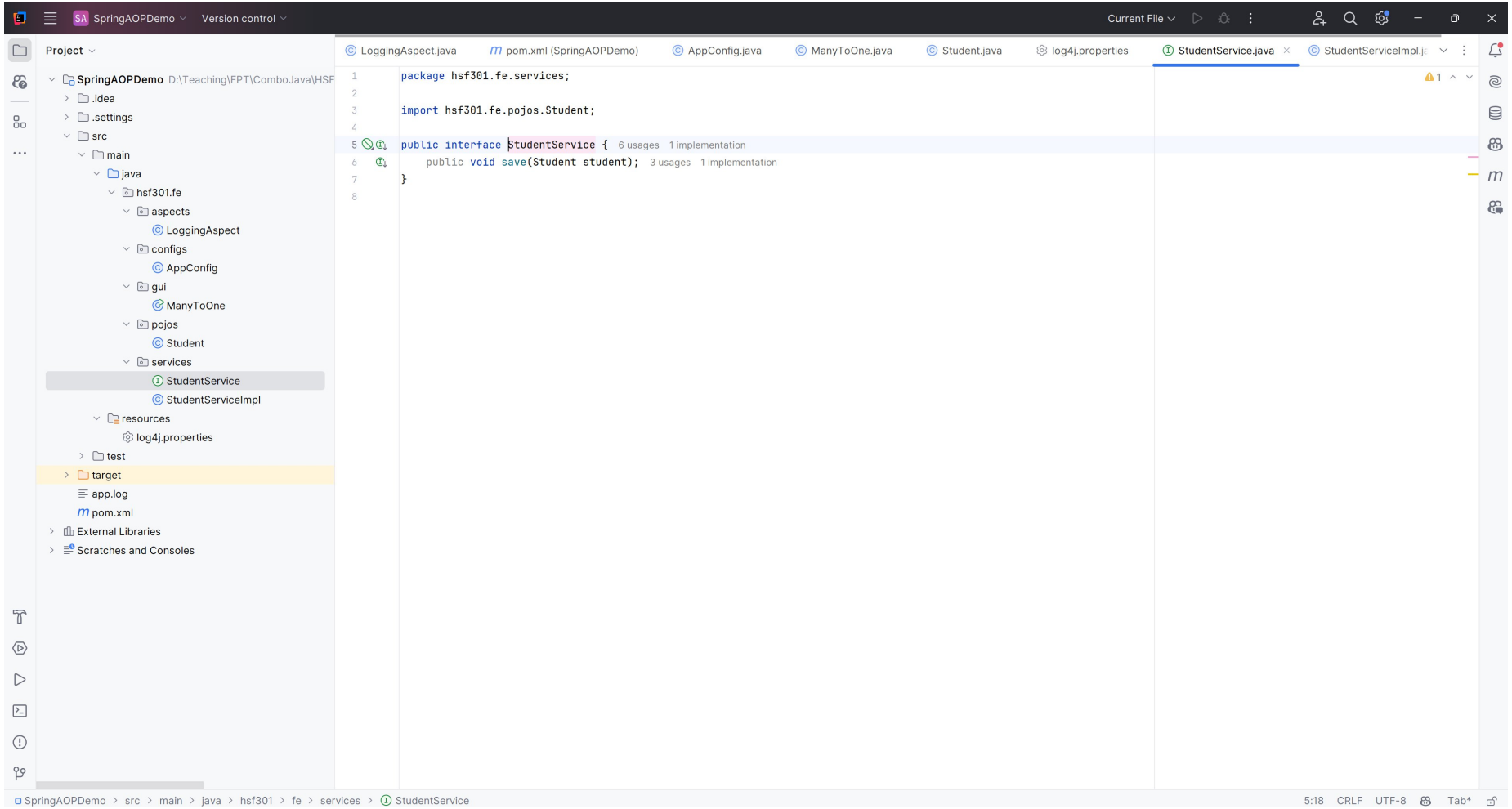
Edit pom.xml


SpringAOPDemo
Version control
Current File
Project
SpringAOPDemo D:\Teaching\FPT\ComboJava\HSF
.idea
.settings
src
main
java
hsf301.fe
aspects
LoggingAspect
configs
AppConfig
gui
ManyToOne
pojos
Student
services
StudentService
StudentServiceImpl
resources
log4j.properties
test
target
app.log
pom.xml
External Libraries
Scratches and Consoles
LoggingAspect.java
pom.xml (SpringAOPDemo)
AppConfig.java
ManyToOne.java
Student.java
log4j.properties
StudentService.java
StudentServiceImpl.j
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>hsf301.fe.pojos</groupId>
<artifactId>SpringAOPDemo</artifactId>
<version>0.0.1-SNAPSHOT</version>
<dependencies>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>6.1.14</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-aop</artifactId>
<version>6.1.14</version>
</dependency>
<dependency>
<groupId>org.aspectj</groupId>
<artifactId>aspectjweaver</artifactId>
<version>1.9.22.1</version>
</dependency>
<dependency>
<groupId>org.apache.logging.log4j</groupId>
<artifactId>log4j-core</artifactId>
<version>2.24.2</version>
</dependency>
</dependencies>
</project>
SpringAOPDemo > pom.xml
10:51 CRLF UTF-8 Tab*

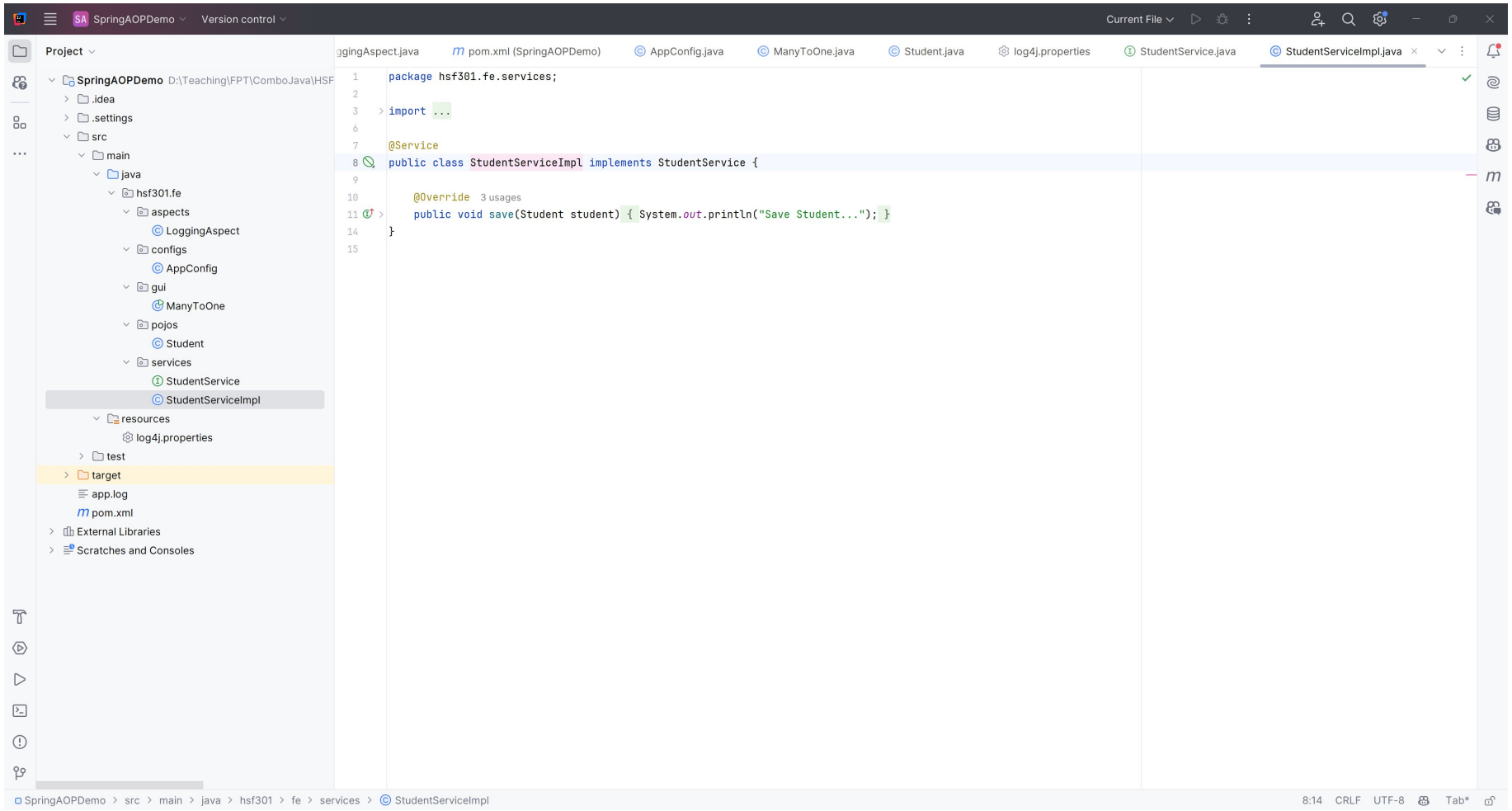
4. Create the Student.java in pojo



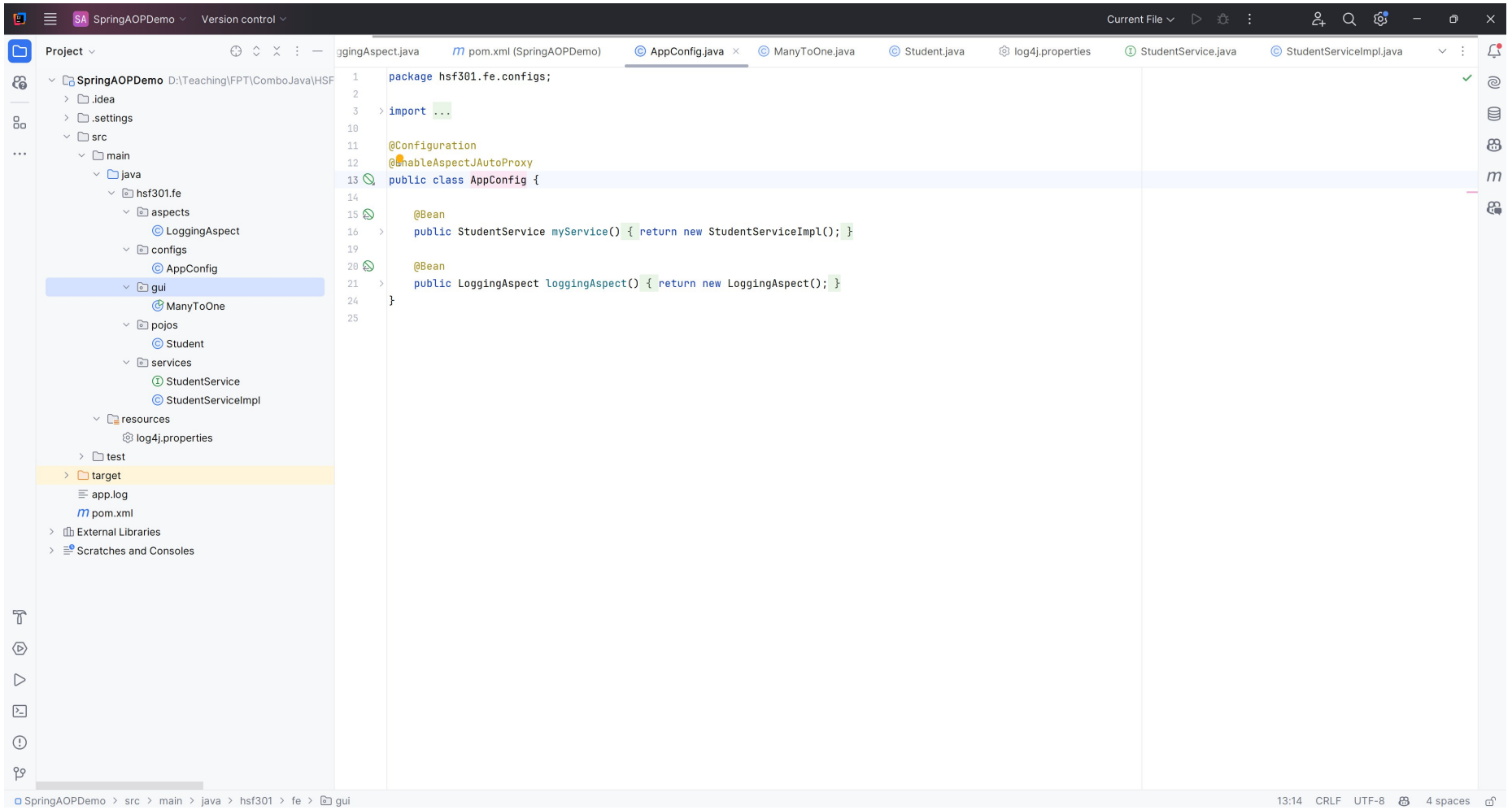
Create the StudentService.java in services package



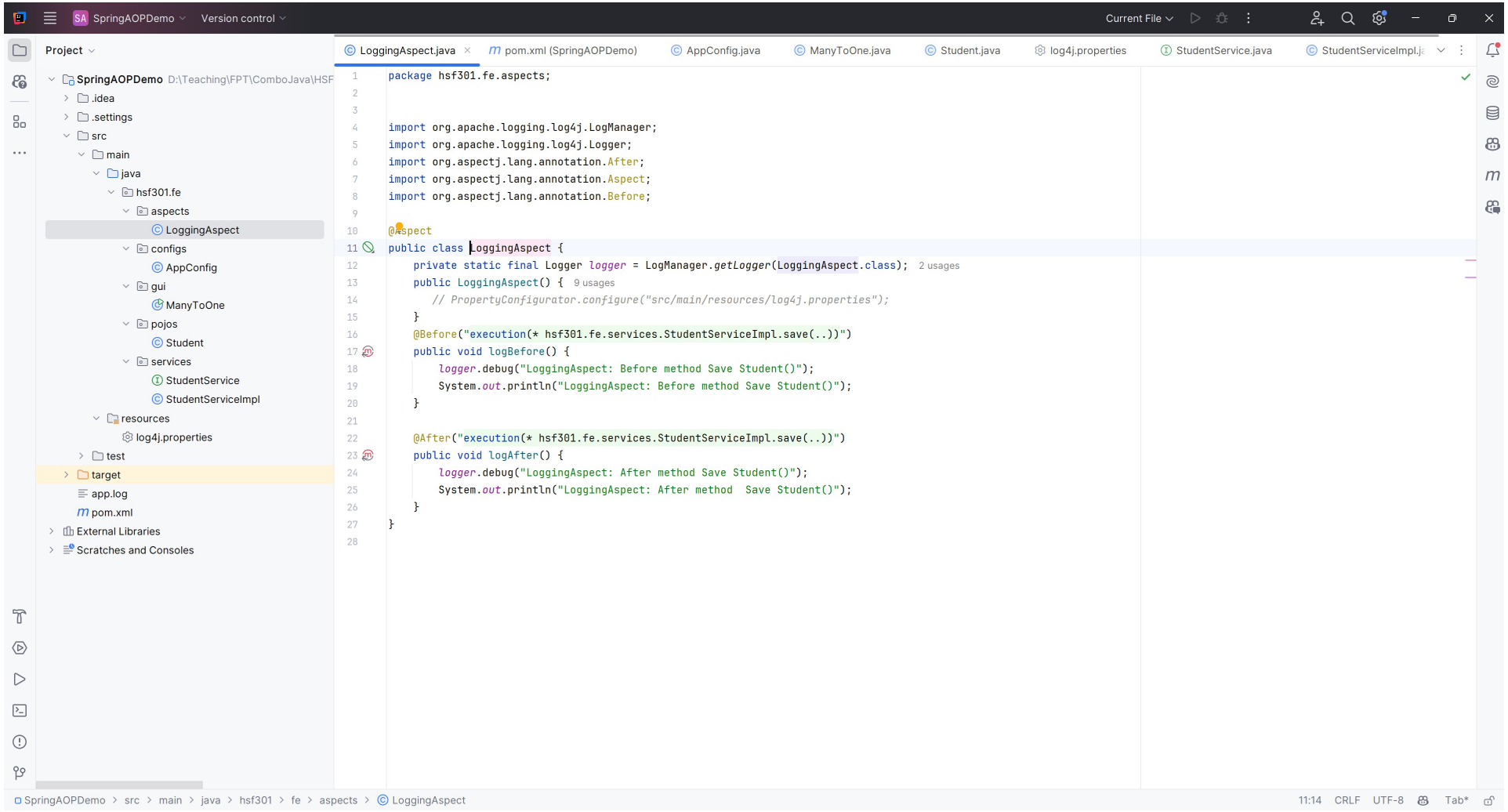
Create the StudentServiceImpl.java in services package



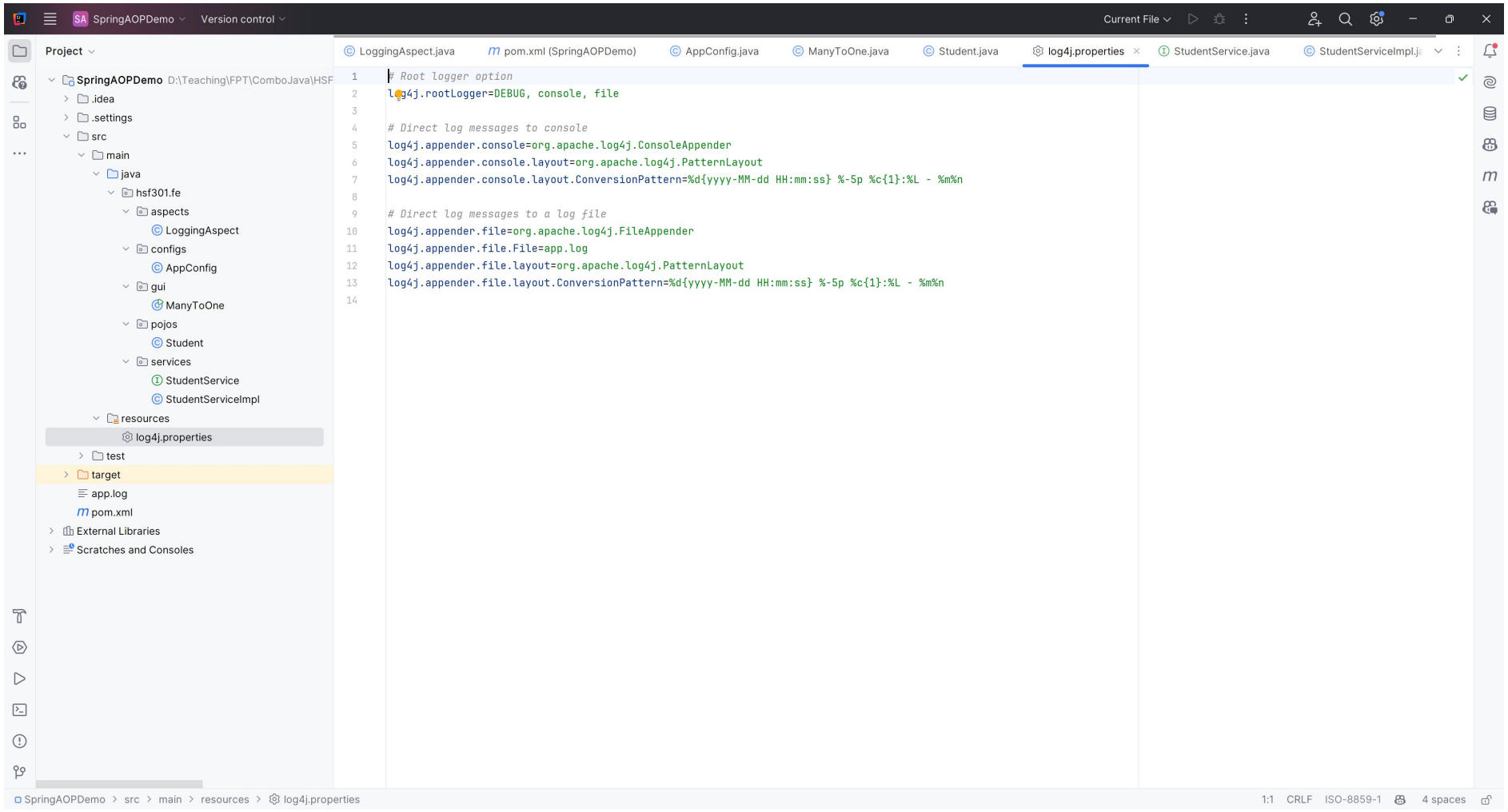
Create the AppConfig.java in configs package



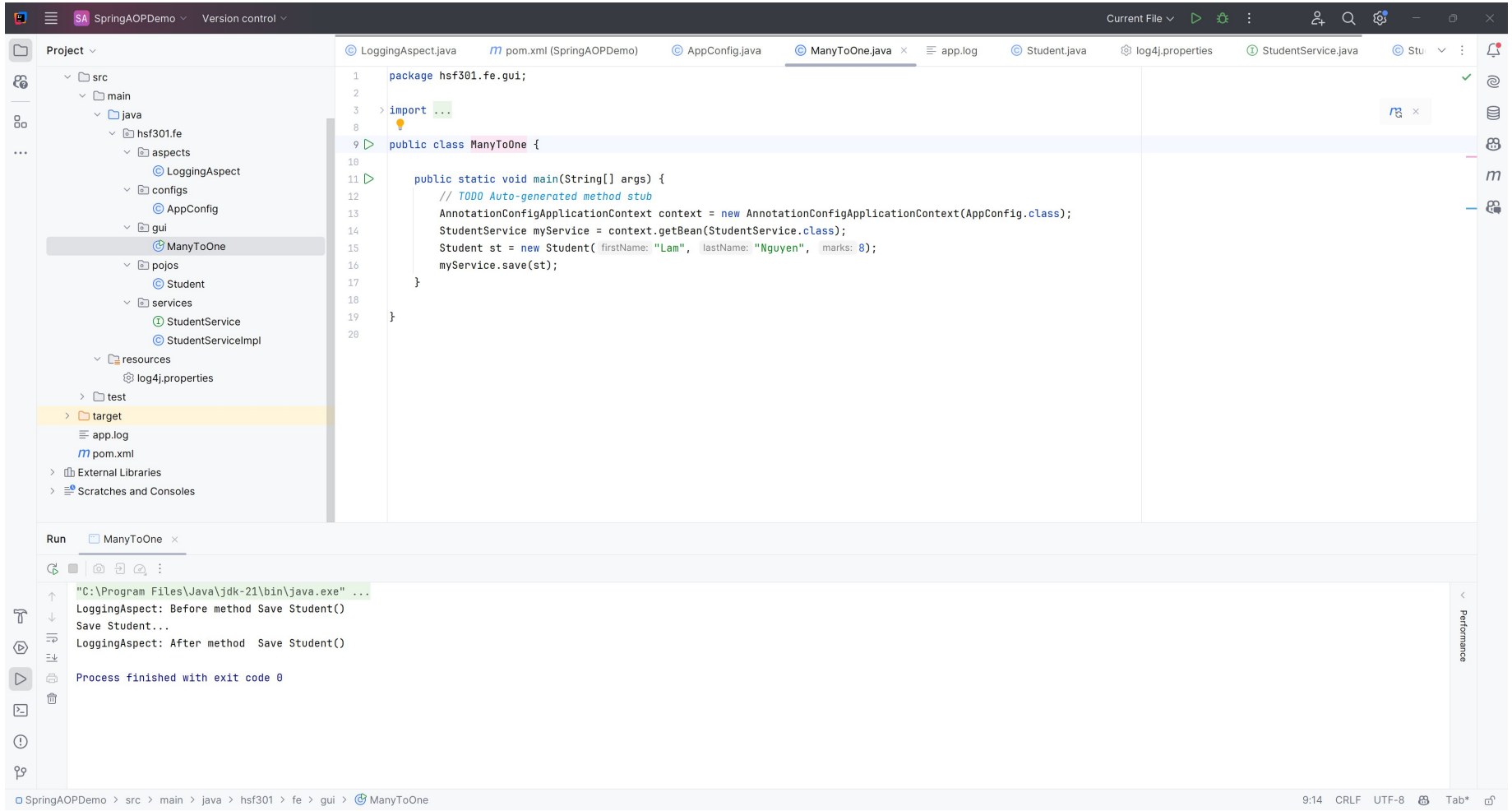
Create the LogginAspect.java in aspects package



Create the log4j.properties in resources



11. Run Program



Summary

Concepts were introduced:

- Spring Framework
- Advantages of using Spring Framework
- Key features of Spring Framework
  - Dependency Injection and Inversion of Control
  - Aspect Oriented Programming