섹션 8. Spring Boot의 이모저모

49강. build.gradle 이해하기

이번 시간에는 아직까지 미지의 영역으로 남겨두었던 `build.gradle` 파일을 확인해 보자! 😊

'빌드 스크립트'라고도 불리는 `build.gradle` 파일은 gradle을 이용해 프로젝트를 빌드하고 의존성을 관리하기 위해 작성한 파일로, 현재 groovy라는 언어를 이용해 작성되어 있다.

gradle이 빌드툴이고, groovy가 이 빌드툴을 사용하기 위한 언어이다! `build.gradle` 을 작성하기 위해 groovy 대신 kotlin이라는 언어를 사용할 수도 있다.

자 그럼, 가장 먼저 `build.gradle`의 윗부분부터 살펴보자.

```
plugins {
  id 'org.springframework.boot' version '2.7.6'
  id 'io.spring.dependency-management' version '1.0.12.RELEASE'
  id 'java'
}
```

가장 먼저 `plugins`라는 단어와 중괄호가 눈에 띈다! `plugins`라는 블락(중괄호) 안에는 우리가 이 프로젝트에 적용하고 싶은 플러그인을 추가할 수 있다. 예를 들어, `id 'org.springframework.boot' version '2.7.6'`라고 적게 되면, `org.springframework.boot`라는 플러그인을 `2.7.6` 버전으로 사용한다는 의미이다.

`org.springframework.boot` 플러그인은 다음과 같은 역할을 갖고 있다.

1. 스프링을 빌드했을 때 실행 가능한 jar 파일이 나오게 도와주고
2. 스프링 애플리케이션을 실행할 수 있게 도와주며
3. 또 다른 플러그인들이 잘 적용될 수 있게 해준다.

다음으로, `io.spring.dependency-management` 플러그인은 외부 라이브러리나 프레임워크의 버전을 관리하는 데 도움을 주고 서로 얽혀 있는 의존성들을 처리하는 데 도움을 준다.

마지막으로 `java` 플러그인은 Java 프로젝트를 개발하는 데 필요한 기능들을 추가해 주고, 다른 JVM 언어 Gradle 플러그인들을 사용할 수 있는 기반을 마련해 준다.

```
group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'
```

다음 부분은 프로젝트에 관련된 부분이다.

- group : 프로젝트의 그룹을 의미한다. 빌드 결과물에 프로젝트 그룹에 대한 정보가 들어 있다.
- version : 프로젝트의 버전을 의미한다. 우리가 빌드한 jar 파일을 생각해 보면 `library-app-0.0.1-SNAPSHOT.jar`로 `0.0.1-SNAPSHOT`이 바로 version에서 나온 것이다.
- sourceCompatibility : 프로젝트가 사용하고 있는 JDK 버전을 의미한다. 우리는 Java11을 기반으로 프로젝트를 만들었으니 11으로 되어 있다.

```
repositories {
  mavenCentral()
}
```

다음으로는 `repositories`라는 블락이 눈에 들어온다. 이 블락은 외부 라이브러리나 프레임워크를 가져오는 장소를 설정한다. 예를 들어 `mavenCentral()`이라고 하면 “maven 중앙 저장소”를 가리키는 것이다.

```
dependencies {
  implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
  implementation 'org.springframework.boot:spring-boot-starter-web'
  runtimeOnly 'com.h2database:h2'
  runtimeOnly 'mysql:mysql-connector-java'

  testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

그다음 `dependencies` 블락은 우리가 사용하는 라이브러리/프레임워크를 표시하는 곳이다.

예를 들어, `implementation 'org.springframework.boot:spring-boot-starter-data-jpa'`라고 한다면, `org.spring.framework.boot` 그룹이 만든 `spring-boot-starter-data-jpa`를 가져오라는 의미이다.

하지만 아래 있는 h2 관련 의존성이나 mysql 관련 의존성은 `implementation` 대신 `runtimeOnly`를 사용하고 있는데, `implementation`은 해당 의존성을 가져와 항시 사용한다는 의미이고, `runtimeOnly`는 코드를 실행할 때에만 의존성을 사용하겠다는 의미이다.

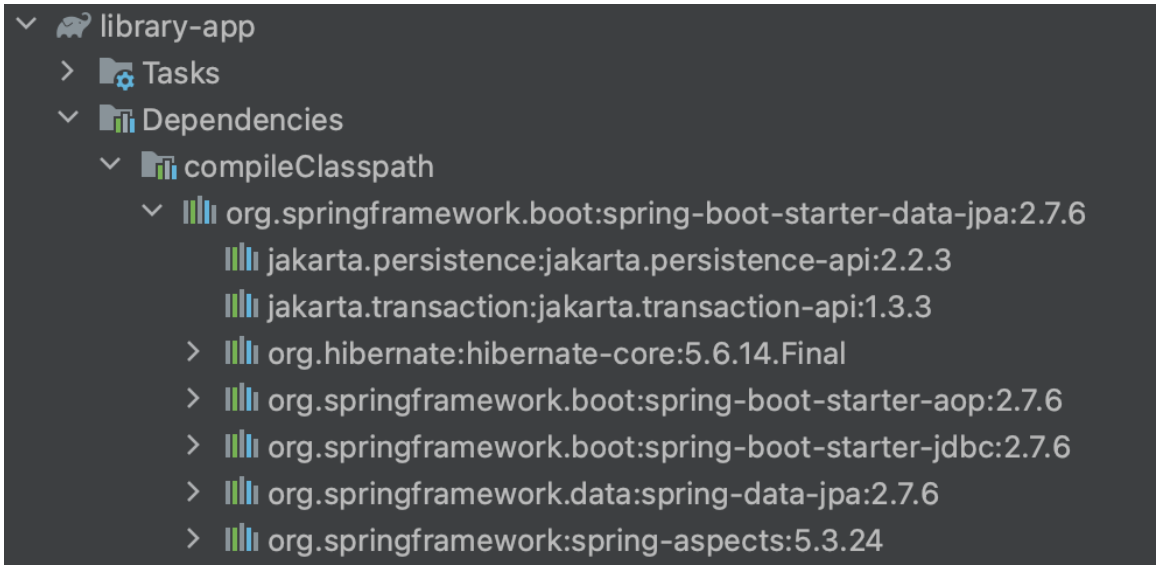
`testImplementation` 같은 경우는 테스트 코드를 컴파일하거나 실행시킬 때 항시 사용한다는 의미이다.

이렇게 앞 부분에 나오는 문구를 gradle의 **dependency configuration**이라고 부른다.

자 그럼 이제 우리의 프로젝트들이 어떤 의존성을 갖고 있는지 하나씩 살펴보자.

먼저 `org.springframework.boot:spring-boot-starter-data-jpa` 가 있다. 이때 특이한 단어가 하나 들어 있는데 바로 'starter'이다. 이 starter라는 의미는 <50강. Spring과 Spring Boot>에서 조금 더 자세히 설명되겠지만 Spring에서 JPA를 사용하기 위해 필요한 여러 라이브러리와 프레임워크가 모아져 있다는 의미이다.

실제로 IntelliJ 오른쪽 탭의 Gradle을 열어, `library-app` > `Dependencies` > `compileClasspath` > `org.springframework.boot:spring-boot-starter-data-jpa:2.7.6` 을 확인해보면



이 starter는 jakarta persistence, jarkarta transaction, hibernate, 또 다른 starter들, 다른 spring jpa 등 다양한 프레임워크/의존성을 가지고 있는 것을 확인할 수 있다.

그래서 우선은 JPA를 스프링과 함께 사용하기 위해 필요한 모든 것이 모여져 있는 의존성이라고 생각하며 된다.

다음으로 `org.springframework.boot:spring-boot-starter-web` 은 방금 다루었던 것과 유사하게, 스프링에서 web 관련 개발을 하기 위해 필요한 모든 것이 모여져 있는 의존성이다. 이 안에 tomcat이라거나 API를 만들 수 있게 해주는 어노테이션이라거나 하는 기능들이 들어 있다.

그다음은 h2와 mysql을 사용하기 위해 필요한 의존성이고, 마지막으로 `org.springframework.boot:spring-boot-starter-test` 는 스프링을 사용하며 테스트를 작성하기 위해 필요한 라이브러리나 프레임워크가 모여 있는 의존성이다.

테스트 코드에 대해 조금 더 알아보고 싶다면 무료로 열려 있는 **<실전! 코틀린과 스프링 부트로 도서 관리 애플리케이션 개발하기 - 2강. 테스트 코드란 무엇인가, 그리고 왜 필요한가?!>**를 참고해 보아도 좋다.

```
tasks.named('test') {
  useJUnitPlatform()
}
```

마지막으로 적혀 있는 부분은 테스트를 수행할 때에 Junit5를 사용하겠다는 의미이다. Junit5는 Java나 Kotlin 프로젝트를 할 때 자주 사용되는 테스트 프레임워크로, 테스트 코드라는 것을 작성할 때 사용한다.

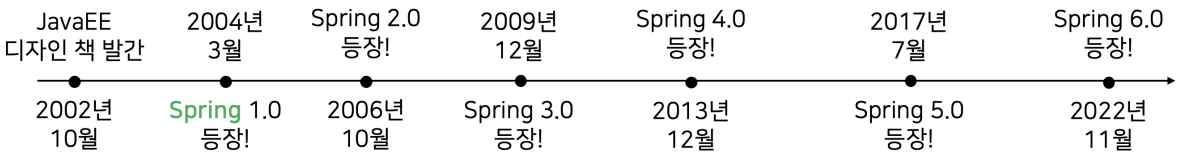
매우 좋다~! 😊 👍 이렇게 우리는 `build.gradle`의 모든 문구 하나하나를 파악해 보았다.

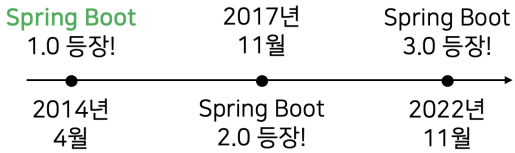
다음 시간에는 Spring과 Spring Boot의 차이점이 무엇인지 간단히 다뤄보도록 하자! 🔥

50강. Spring과 Spring Boot

> 💡 강의 영상과 PPT를 함께 보시면 좋습니다 😊

이번 시간에는 Spring과 Spring Boot의 차이점에 대해 알아보자. Spring과 Spring Boot는 비슷하지만 조금씩 다른 부분이 많다. 우선 둘의 역사를 간략히 살펴보자.

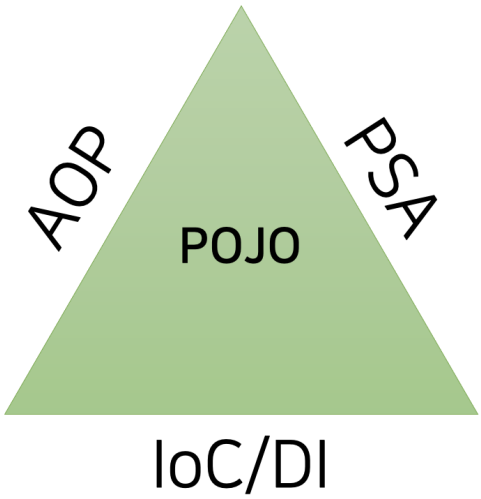




둘의 역사에서 확인할 수 있는 것처럼 Spring Boot는 Spring이 나온 뒤 10년 정도가 흘러서 나왔다는 것을 알 수 있다. 그러니까 Spring Boot가 Spring의 업그레이드 버전인 것 같은데, 정확히 어떤 부분이 업그레이드된 걸까?

[1] 간편한 설정

Spring은 강력한 기능을 제공한다. 우리가 집중적으로 살펴보았던, 의존성 주입이라는 개념 외에도 트랜잭션을 간편하게 사용할 수 있게 해준다거나, 다양한 어노테이션이 잘 동작할 수 있게 해준다거나 하는 기능들이 있다. 하지만 Spring Boot가 없던 시절에는 강력한 기능들이 동작하게 하기 위해 정말 정말 많은 설정을 해야 했다!!!



이때 xml이라는 문법을 따로 익혀 트랜잭션 관련 설정~ 의존성 주입 관련 설정~ 톰캣 관련 설정~~ 이런 모든 설정들을 하나씩 해주어야 했다. 그 덕분에 알아야 할 것도 많았고 설정도 정말 길었다. 당연히 이렇게 많은 설정을 하는 것도 힘들고, 설정이 너무 많아 놓치거나 빠지는 것도 생기기 때문에 불편했다.

그래서 `Spring Boot`에서는 이러한 모든 설정을 xml 대신 Java의 어노테이션 기반으로 가능하게 하고, 기본적인 설정들은 모두 자동으로 해주었다. 우리가 사용한 `JdbcTemplate` 역시 스프링 부트가 자동으로 설정을 해주었기 때문에 사용할 수 있었던 것이다.

[2] 간단한 의존성 관리

좋다! Spring Boot의 등장으로 스프링을 활용하기 위한 설정이 간단해졌다! 하지만 여전히 불편한 부분이 존재한다. 바로 의존성 관리이다! Spring을 사용할 때에는 서버 개발을 하는데 필요한 라이브러리나 프레임워크를 모두 하나하나 적어주어야 했다.

어떤 라이브러리를 사용할지, 어떤 프레임워크를 사용할지, 어떤 버전을 사용할지 등등을 모두 처리해 줬는데 이때 라이브러리끼리 충돌되거나 프레임워크끼리 충돌되는 경우도 잦았다. 또한 지금 우리의 dependencies는 몇 줄 밖에 되지 않지만, Spring을 사용할 때에는 다양한 의존성들을 한 땀 한 땀 모두 설정해 주어야 했기 때문에 알아야 할 것도 많았고 당연히 관리도 어려웠다.

그래서 `Spring Boot`에서는 이런 의존성을 필요한 것끼리 한데 묶어 `starter`로 관리하기 시작했다! 우리가 살펴보았던 JPA starter만 하더라도 안을 열어보면 수십 개의 의존성이 하나로 묶어져 있는 것을 확인할 수 있다.

starter 안에 담겨 있는 내용들이 많아 한 화면에 담기지도 않는다!! (중복된 것도 있다!)

이 라이브러리들을 모두 추적하고 버전 업데이트되면 바뀐 내용을 읽어보고, 코드에 한 땀 한 땀 버전을 적어줘야 한다고 생각해 보자. 정말 아찔하다…! 🥺

다행히~ `Spring Boot`의 starter 덕분에 의존성 관리가 매우 쉬워졌다.

[3] 강력한 확장성

이렇게 간편해진 설정과 starter가 등장하며 의존성 관리가 쉬워지자, `Spring Boot`에서는 기능의 확장성 또한 매우 강력해졌다. 이전에는 새로운 기술을 도입하려고 하면, 해당 기술을 공부하고 설정을 하나하나 처리해 주고, 필요한 의존성을 모두 찾아 등록해 주어야 했는데 이제는 `starter` 추가만으로 우리가 원하는 기술을 도입할 수 있게 된 것이다.

예를 들어 우리가 했던 것처럼 jpa를 사용하고 싶다면 jpa starter를, 보안 관련 모듈을 적용하고 싶다면 security starter를 추가하는 것만으로 그 기술을 바로 가져다 사용할 수 있게 된 것이다.

이런 확장성 덕분에 <45강. 빌드와 실행, 그리고 접속>에서 살펴보았던 것처럼 Spring Boot 내부에 톰캣을 내장하게 되었고, 배포 과정 역시 무척 간단해졌다.

너무 좋다~! 👍 이번 시간에는 Spring과 Spring Boot의 차이점인 1) 간편한 설정 2) 간단한 의존성 관리 3) 강력한 확장성에 대해 알아보았다.

이제 다음 시간에는 application.yml의 문법과 Java로 스프링을 사용할 때 활용할 수 있는 lombok에 대해 알아보자! 🔥

51강. application.yml과 application.properties, lombok

이번 시간에는 application.properties와 lombok에 대해 살펴보도록 하자! 👍

우리가 작성한 application.yml 파일은 YAML 문법을 사용한다. YAML은 2가지 의미를 가지고 있는데, `Yet Another Markup Language` 와 `YAML Ain't Markup Language` 두 가지이다.

YAML은 복잡한 마크업 언어인 XML 대신하려 등장했기 때문에 원래는 '또 다른 마크업 언어'라고 이름 붙여 나왔다. 그러다가, YAML의 핵심은 문서 마크업이 아닌 데이터 그 자체라는 것을 보여주기 위해 이름을 바꾸었다.

YAML의 확장자는 `.yaml` 또는 `.yml`로 현재 우리가 `application.yml`이라 작성한 것은 두 번째 확장자를 사용한 것이다. 그래서 사실 `.yaml`로 바꾸더라도 문제없다.

YAML의 문법을 간단히 살펴보자. YAML은 기본적으로 key : value 형식으로 데이터를 정의한다.

```
spring:
  config:
    activate:
      on-profile: dev
```

예를 들어 위의 데이터는 `spring.config.activate.on-profile`이 key이고 value가 dev인 것이다.

또한 각 계층은 들여 쓰기를 통해 구분하게 되며 이를 이용해 중복을 제거할 수 있다.

```
spring:
  datasource:
    url: "jdbc:mysql://localhost/library"
    username: "root"
    password: "Abcd1234!"
    driver-class-name: com.mysql.cj.jdbc.Driver
```

위의 예시에서 `spring.datasource.url`과 `spring.datasource.username`에 있는 `spring.datasource`를 들여 쓰기를 사용해 중복 제거한 것이다!

value에 들어갈 수 있는 타입에는 참 / 거짓 그리고 숫자와 문자열이 있다.

참 / 거짓은 true / false로 표현할 수 있고, 숫자의 경우 큰따옴표를 사용하지 않고 적으면 숫자로 인식된다. 문자열의 경우는 큰따옴표를 사용해도 되고 사용하지 않아도 된다.

또한 value에는 배열도 들어갈 수 있는데, 배열은 `-`을 사용하게 된다.

```
person:
  name: lannstark
  age: 99
  dogs:
    - 초코
    - 망고
```

YAML에서 주석을 표현하려면 `#`을 쓰게 된다.

```
# 주석
spring:
  config:
    activate:
      on-profile: dev
```

매우 좋다~!! 👍 이렇게 간단히 YAML의 문법에 대해 정리해 보았는데 사실 스프링의 설정을 하기 위해서 반드시 YAML 문법을 사용해야 하는 것은 아니다!

YAML 파일인 `application.yml` 대신 `application.properties`를 사용할 수도 있다!

`application.properties`를 사용할 경우, yaml처럼 들여 쓰기를 통해 중복되는 부분을 제거할 수는 없지만, key - value 형식으로 데이터를 기록하는 것은 똑같다.

```
spring.datasource.url="jdbc:mysql://localhost/library"
spring.datasource.username="root"
spring.datasource.password="Abcd1234!"
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

또한 profile을 설정할 때는, `application-프로파일이름.properties`라는 파일을 새로 만들어 처리하게 된다.

YAML 형식에 비해 중복이 많고 가독성이 좋지 않아 개인적으로는 YAML 방법을 선호하는 편이다!

다음으로는 `lombok`에 대해 알아보자! 우리가 도서관리 애플리케이션 프로젝트를 진행하며 getter나 생성자 등을 필드별로 만들어주었는데, 필드가 많아지고 프로젝트가 커질수록 이러한 보일러 플레이트 코드(boiler plate code)를 만드는 것은 번거롭다.

이런 번거로움을 피하기 위해 `lombok`이라는 라이브러리를 사용할 수 있는데, `lombok`을 사용하면 getter나 생성자, setter, equals, toString 등을 자동으로 만들어줄 수 있다!!

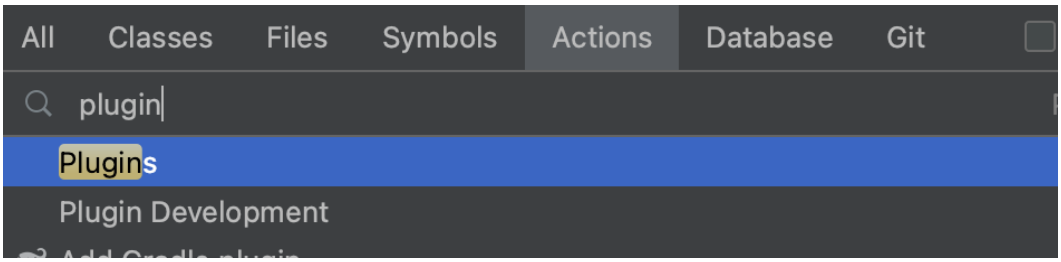
우리 프로젝트에 간단히 도입해 보자. `lombok`을 도입하기 위해서는 3가지 설정을 해주어야 한다.

1. lombok 의존성 추가
2. IntelliJ lombok 플러그인 추가
3. IntelliJ Annotation Processor 설정

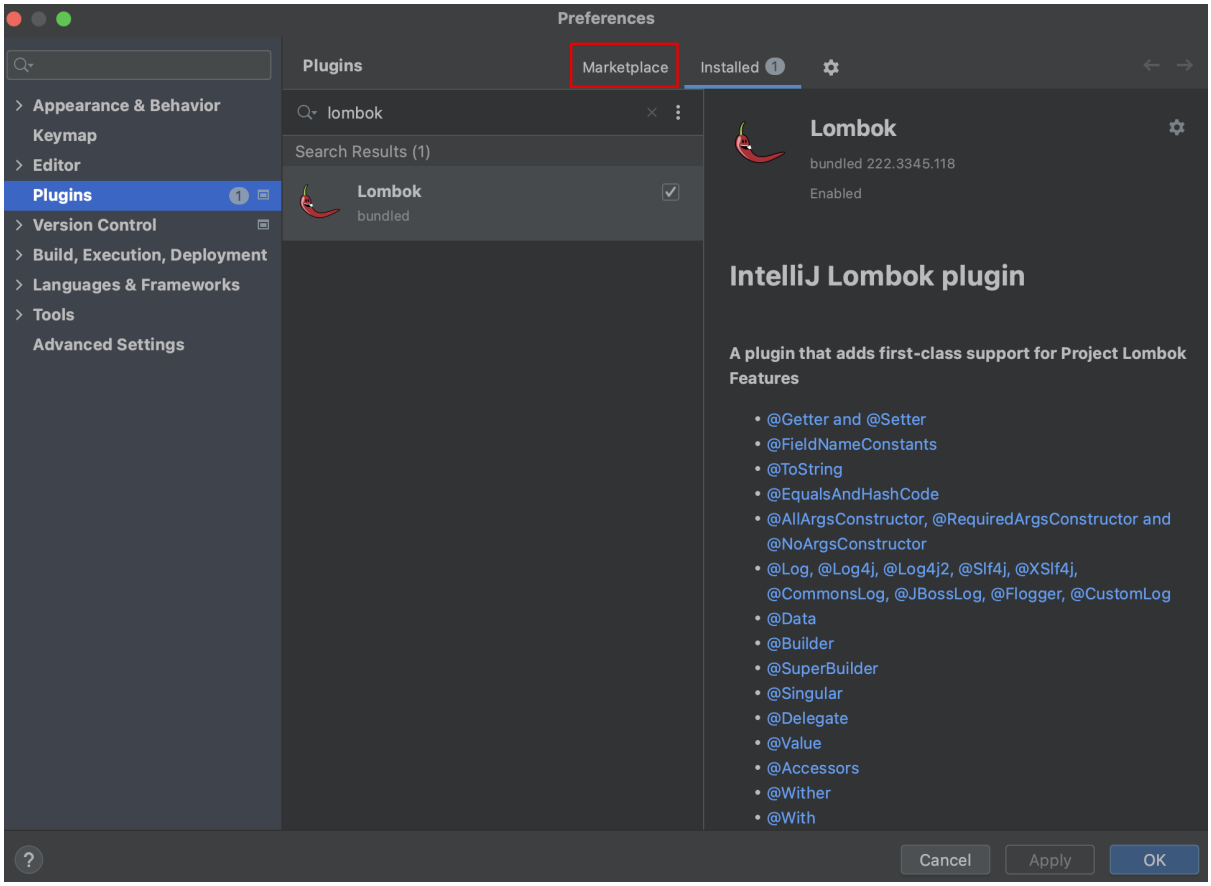
자 먼저 lombok 의존성부터 추가해 주자! `build.gradle`에 들어가 `dependencies` 블락에 아래 내용을 추가해 주자. 그런 다음 Gradle Refresh를 한 번 해주고,

```
dependencies {
  implementation 'org.projectlombok:lombok'
}
```

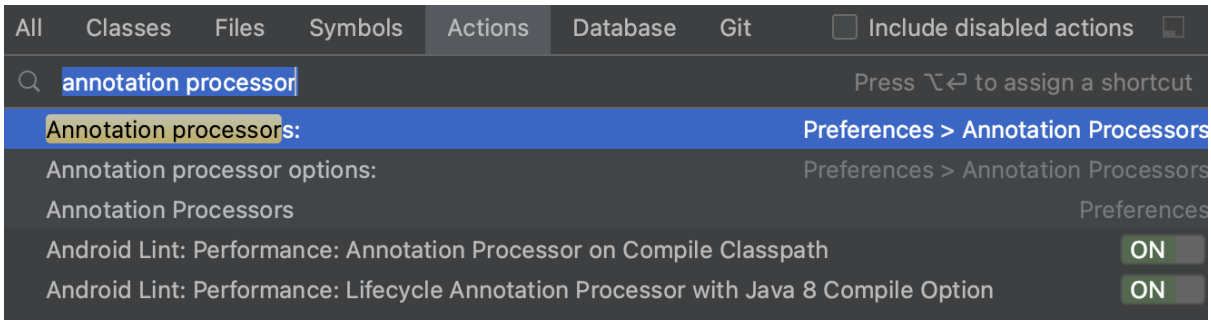
그다음 IntelliJ lombok 플러그인을 추가해 주자! 맥북 기준 `command + shift + A` (Windows는 `Ctrl + Shift + A`)를 눌러 Actions 창을 열어 주고 여기에 plugin을 검색해 plugins에 들어가자.



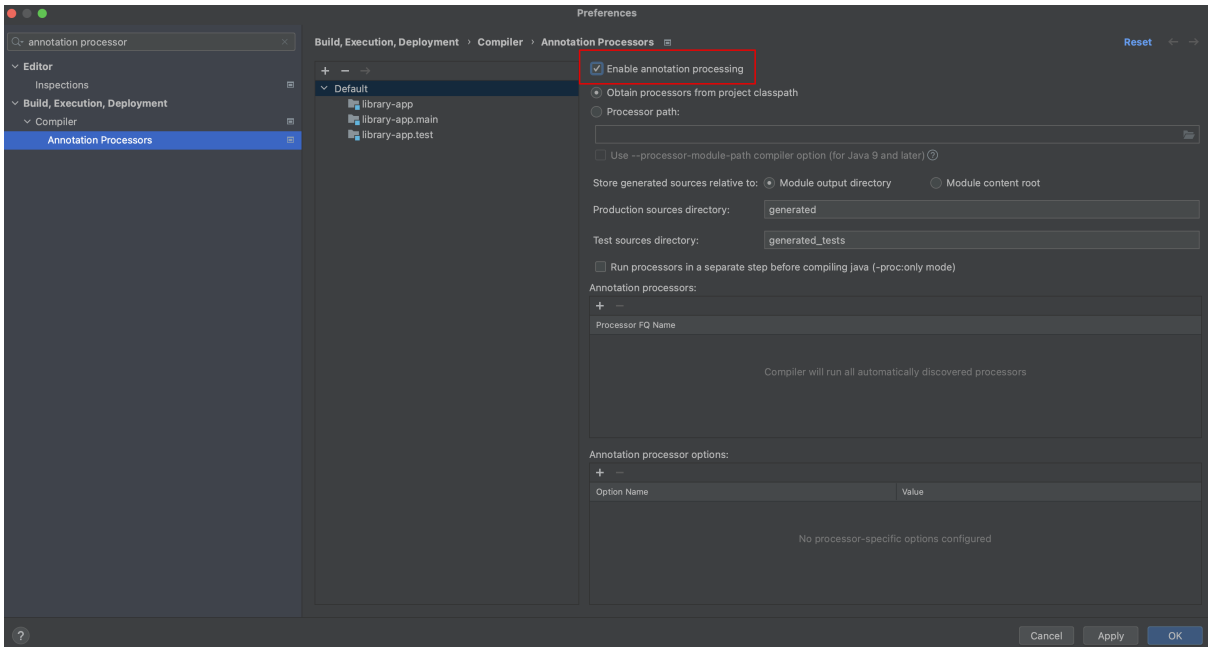
그런 다음 Marketplace에서 lombok을 검색해 설치하면 된다.



좋다~ 이제 마지막 단계이다. 아까와 마찬가지로 `command + shift + A` (Windows는 `Ctrl + Shift + A`)를 눌러 Actions 창을 열어 주고 여기에 annotation processor라고 검색해 들어가자.



그다음 나오는 화면에서 Enable annotation processing을 선택하면 끝이다!!



자 그럼 이제 lombok을 사용할 준비가 끝났으니 사용해 보도록 하자!

getter를 사용하고 있는 `UserCreateRequest` 객체에 가서 기존 getter를 모두 지워주고 class 위에 어노테이션으로 `@Getter`를 추가해 주자!

```
@Getter
public class UserCreateRequest {

  private String name;
  private Integer age;

}
```

이렇게 되면 이 클래스에 존재하는 필드들에 자동으로 getter가 생긴 것처럼 동작하게 된다. 실제 `UserServiceV2`의 `saveUser` 메소드를 보면, getter가 잘 동작하는 것을 확인할 수 있다.

한 가지 더 작업해 보자! `User` 클래스로 들어가 `protected User() {}`를 지우고 클래스 위에 `@NoArgsConstructor`라고 적어보자! 그러면 마치 기본 생성자가 있는 것처럼 JPA 에러가 나지 않는 것을 확인할 수 있다! 이 어노테이션이 기본 생성자를 자동으로 만들어주기 때문에 가능한 일이다.

```
@Entity
@NoArgsConstructor
public class User {
 // 생략...
}
```

만약 기본 생성자의 접근 권한을 protected로 변경하고 싶다면 `@NoArgsConstructor(access = AccessLevel.PROTECTED)`를 사용하면 된다.

마지막으로 `BookService`로 들어가 생성자를 지우고 클래스에 `@RequiredArgsConstructor`라고 적어보자!

```
@Service
@RequiredArgsConstructor
public class BookService {

  private final BookRepository bookRepository;
  private final UserLoanHistoryRepository userLoanHistoryRepository;
  private final UserRepository userRepository;

  // 생략...

}
```

이 어노테이션은 final이 붙어 있어 필수로 값이 필요한 변수들을 가지고 있는 생성자를 만들어준다! 그렇게 되면, 자동으로 만들어진 생성자를 통해 `BookRepository`, `UserLoanHistoryRepository`, `UserRepository`가 주입되고 우리는 생성자를 굳이 작성하지 않아도 코드가 동작하게 된다.

지금까지 활용한 `@Getter` `@NoArgsConstructor` `@RequiredArgsConstructor` 외에도 `@Setter` `@EqualsAndHashCode` `@ToString` 등이 보일러 플레이트 코드를 줄이기 위해 사용되곤 한다! 이 외에도 추가적인 어노테이션들이 있으니 필요할 때 검색해서 활용해 보면 좋을 것 같다.

매우 좋다~! 😊 이렇게 몇 가지 설정과 어노테이션을 사용해 Java에서 반복되는 코드를 줄여보았다.

만약 Java14+를 사용한다면, record class라는 것을 사용해 몇몇 경우에 lombok을 대체할 수 있고

`Kotlin`을 사용한다면 `Kotlin` 언어만을 특징을 이용해 간결한 코딩이 가능하다.

매우 좋다~!! 이제 마지막으로 다음 시간에는 우리가 사용한 스프링 2.7.6 버전에서 (2022년 12월 말 기준) 가장 최신 버전인 3.0.1로 업데이트해 보도록 하자!

52강. Spring Boot 2.7.x에서 3.0.x로 업데이트하기

> 💡 강의 영상과 PPT를 함께 보시면 좋습니다 😊

이번 시간에는 Spring Boot가 2.7.x에서 3.0.0으로 넘어오며 달라진 점 몇 가지를 간단히 소개하고, 우리의 프로젝트를 최신 버전인 3.0.1로 업데이트해 볼 것이다! 시간이 흘러 다음 버전이 지속적으로 출시되더라도, 비슷한 방법으로 업데이트할 수 있을 것이다.

2022년 11월 말에 출시된 Spring Boot 3.0은 Major 버전이 바뀐 만큼 많은 부분에 변화가 있었다. Spring Boot 2.7.x에서 3.0으로 넘어오며 어떤 변화가 있었는지 간단히 살펴보자!

1. Java 최소 버전이 17로 업그레이드되었다.
2. 많은 스프링 프로젝트, Thrid-Party Library 버전업이 이루어졌다.
    a. 대표적으로 Spring Framework가 6.0, Hibernate이 6.1로 업그레이드되었다.
3. AOT 기초 작업이 이루어졌다.
    a. AOT(Ahead Of Time)란 빌드를 할 때 스프링 애플리케이션을 분석하고 최적화하는 도구로, 애플리케이션 시작 시간과 메모리 사용량을 줄일 수 있게 해준다.
4. javax 대신 jakarta 패키지를 사용하게 되었다.
    a. 원래 javax란 패키지 이름은 썬 마이크로 시스템즈가 개발해 관리하고 있었고, 애플리케이션 개발에 있어 가장 기반이 되는 코드들의 모음이었다.

b. <50강. Spring과 Spring Boot>에서 살펴보았던 것처럼 이 javax 패키지 (java Enterprise Edition)는 스프링에서 사용되고 있었는데

c. 점점 사람들이 javax 패키지를 직접 사용하기보다 스프링을 사용하게 되어 javax 패키지의 인기가 없어지고 최신 기술이 반영되지 않았다

d. 이런 상황에서 오라클이라는 회사가 썬 마이크로시스템즈를 인수하고 이 프로젝트를 비영리 단체인 이클립스에 이관하였는데

e. java라는 이름의 상표권은 오라클이 가지고 있다 보니 javax라는 이름을 사용하지 못하게 되었다.

f. 때문에 이클립스는 javax라는 패키지 이름을 모두 jarkarta로 변경했다.

5. 모니터링 기능들이 강화되었다.

이 외에도 정말 많은 세부사항에 변경이 있었고, 자세한 내용은 release note를 확인해 보면 좋다!

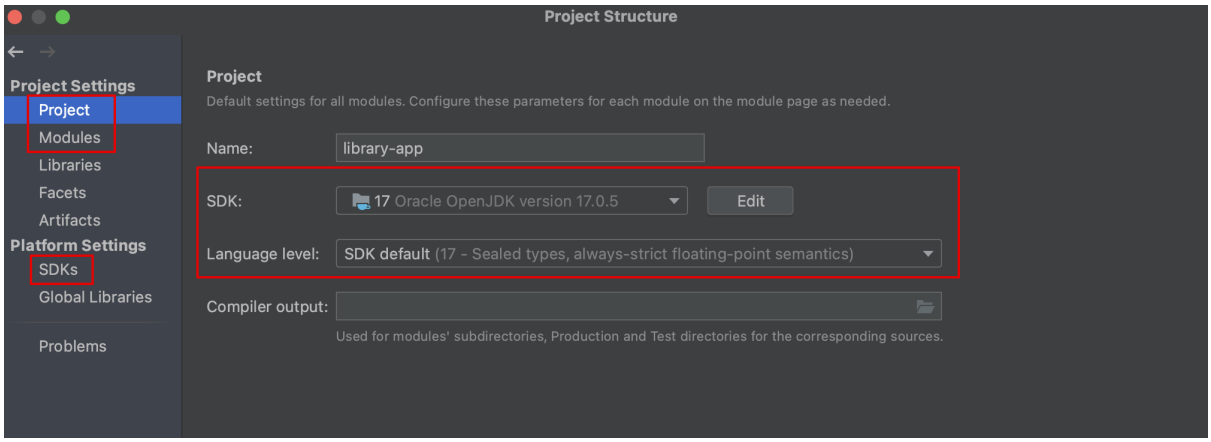
- https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-3.0-Release-Notes

자 그럼 이제 Spring Boot 3.0.1로 버전을 업그레이드해 주자! 가장 먼저 해야 할 일은, Spring Boot 3.0의 최소 Java 버전을 맞추기 위해, JDK 17을 설치하는 일이다. JDK 17 설치가 완료되었다면 우리가 사용하는 프로젝트와 IDE (IntelliJ)의 Java 버전도 모두 변경해 주어야 한다!

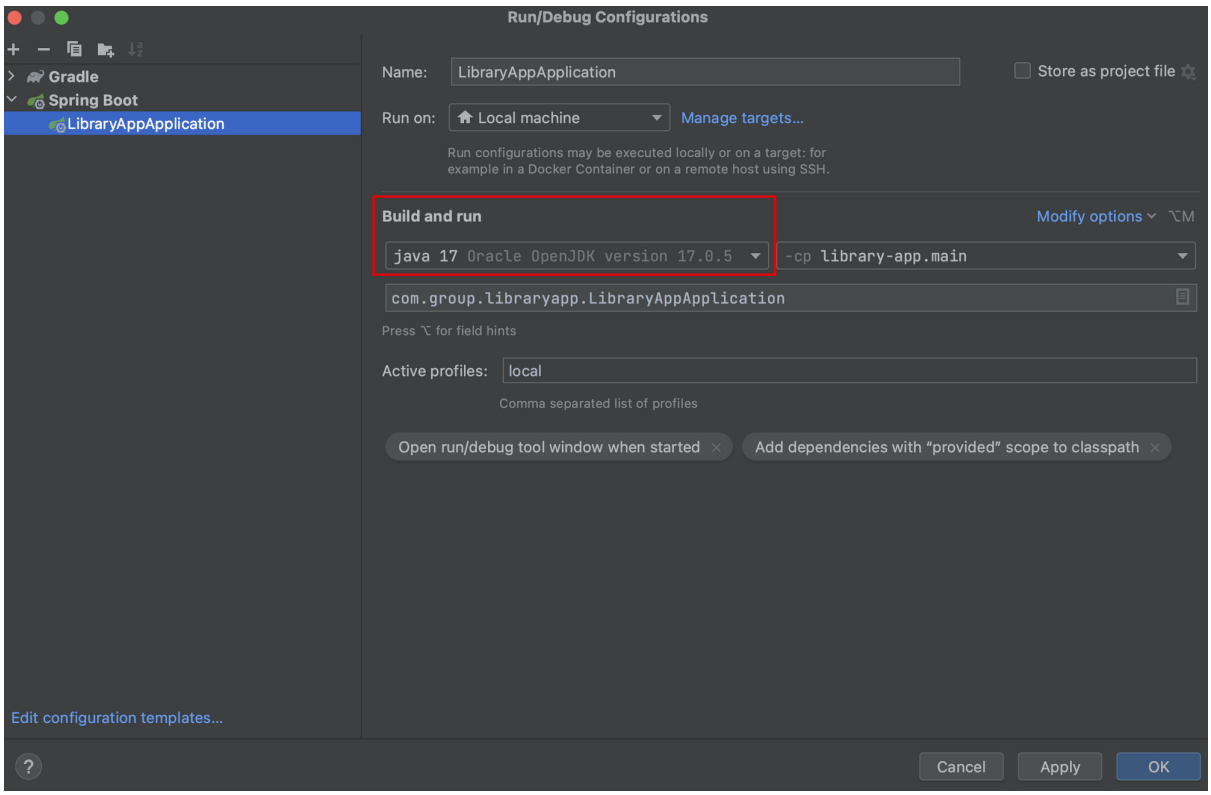
[1. JDK17로 변경하기]

JVM 버전 변경

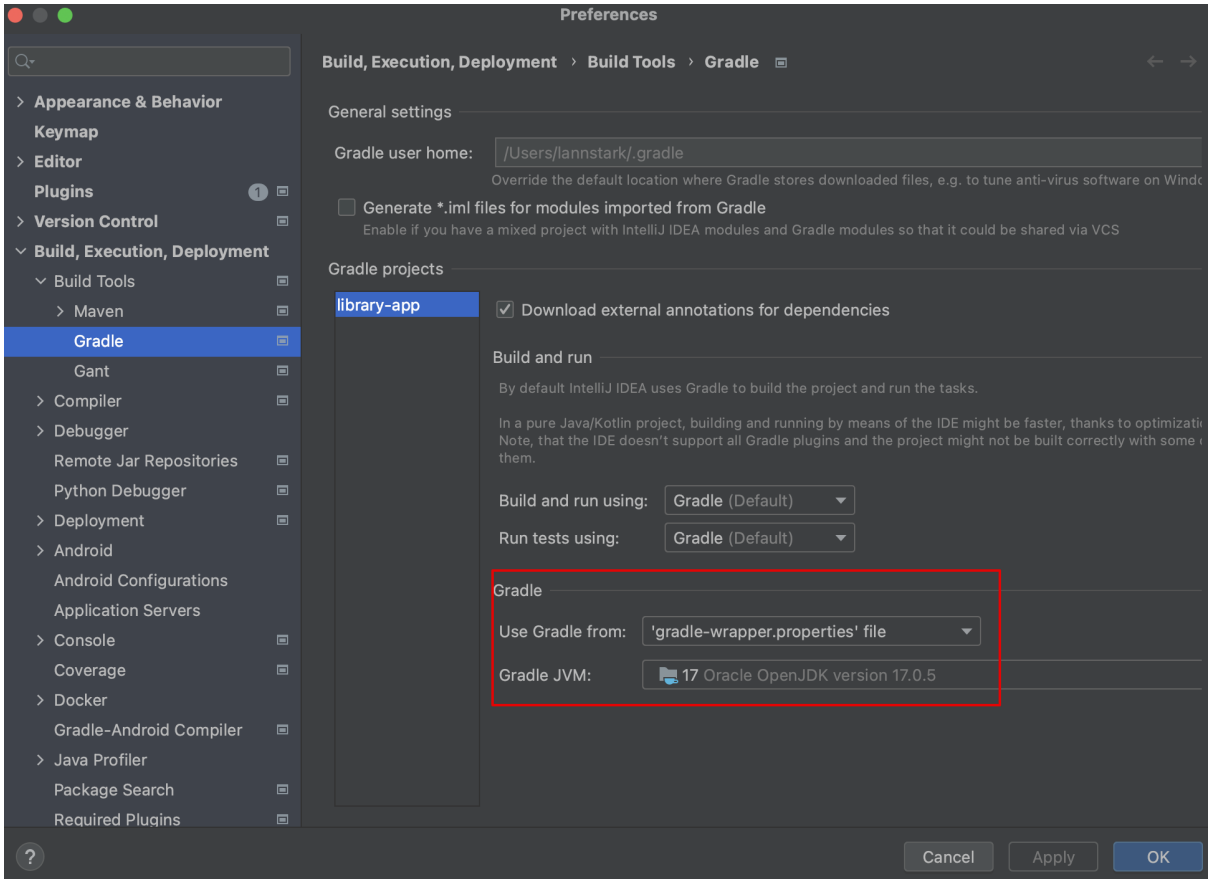
- [File] - [Project Structure]에 있는 모든 SDK를 JDK 17으로 변경하기!



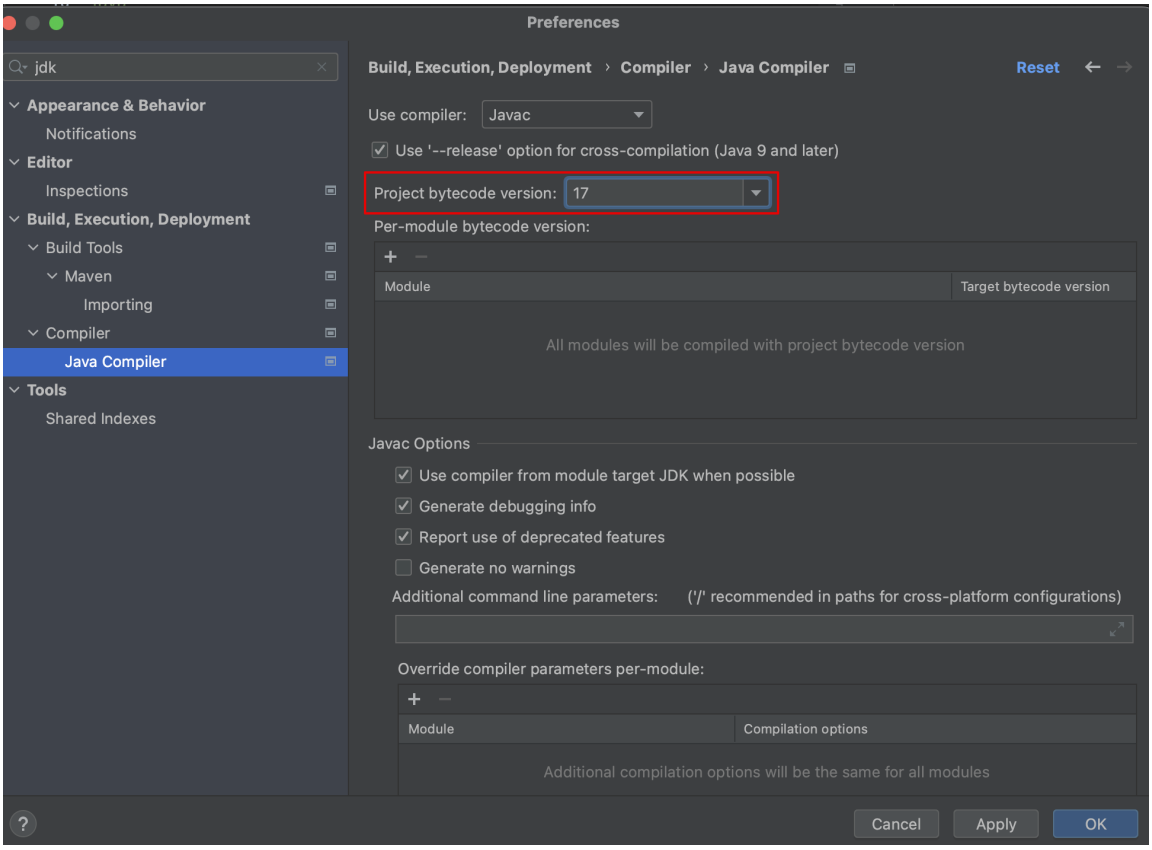
- [Edit Configurations] Build and run JDK 17로 변경하기!



- [Preferences] - [Build, Execution, Deployment] - [Build Tools] - [Gradle] - [Gradle JVM]에서 JDK 17로 변경하기



- [Preferences] - [Build, Execution, Deployment] - [Compiler] - [Java Compiler]에서 JDK 17로 변경하기



- `build.gradle` 의 Java 버전 변경하기

```
sourceCompatibility = '17'
```

[2. Spring Boot 버전을 3.0.1로 변경하기]

이제 Java 버전을 바꾸었으니, `build.gradle`에서 스프링 부트의 버전을 변경하자!

```
plugins {
  id 'org.springframework.boot' version '3.0.1' // 버전업!
  id 'java'
}
```

[3. 마이그레이션 툴 의존성 추가하기]

다음으로는 Spring Boot 3.0 마이그레이션 가이드에서 언급한 대로 마이그레이션 툴 의존성을 추가하자!

```
dependencies {
  runtimeOnly "org.springframework.boot:spring-boot-properties-migrator"
}
```

[4. `javax`로 되어 있는 패키지를 모두 `jakarta` 기반으로 변경하기]

우리가 살펴보았던 Spring Boot 3.0 변경점 중, 직접적으로 영향을 받는 것은 패키지 이름 변경이다! JPA 코드에서 사용하고 있던 `javax` 패키지를 모두 `jakarta`로 변경해 주자!

```
import jakarta.persistence.*;

@Entity
public class User {
 // ...
}
```

[5. 로그 확인 후 마이그레이션 툴 제거하기]

모든 `jakarta` 패키지를 적용했다면, 이제 서버를 동작시켜 확인해 보자!

서버 동작이 별다른 warning 없이 잘 되다면 아까 추가했던 마이그레이션 툴을 의존성에서 제거해 주자.

[6. 기능이 잘 동작하는지 확인하기]

마지막으로 서버를 재시작하여 모든 기능이 잘 동작하는지 웹 UI를 통해 확인해 보면 끝이다!!

이번 시간에 우리는 Spring Boot 2.7.x에서 3.0.x 버전으로 업그레이드해 보았다. 👍 SW 업계 특성상 매일같이 새로운 버전이 나오고 Spring Boot 역시 과거에 그래왔던 것처럼 계속해서 발전해 나갈 것이기 때문에 버전이 달라지면 어떤 부분이 달라졌는지, 프로젝트에 영향을 미칠 부분은 무엇인지 파악하고 대비하는 역량은 중요하다.

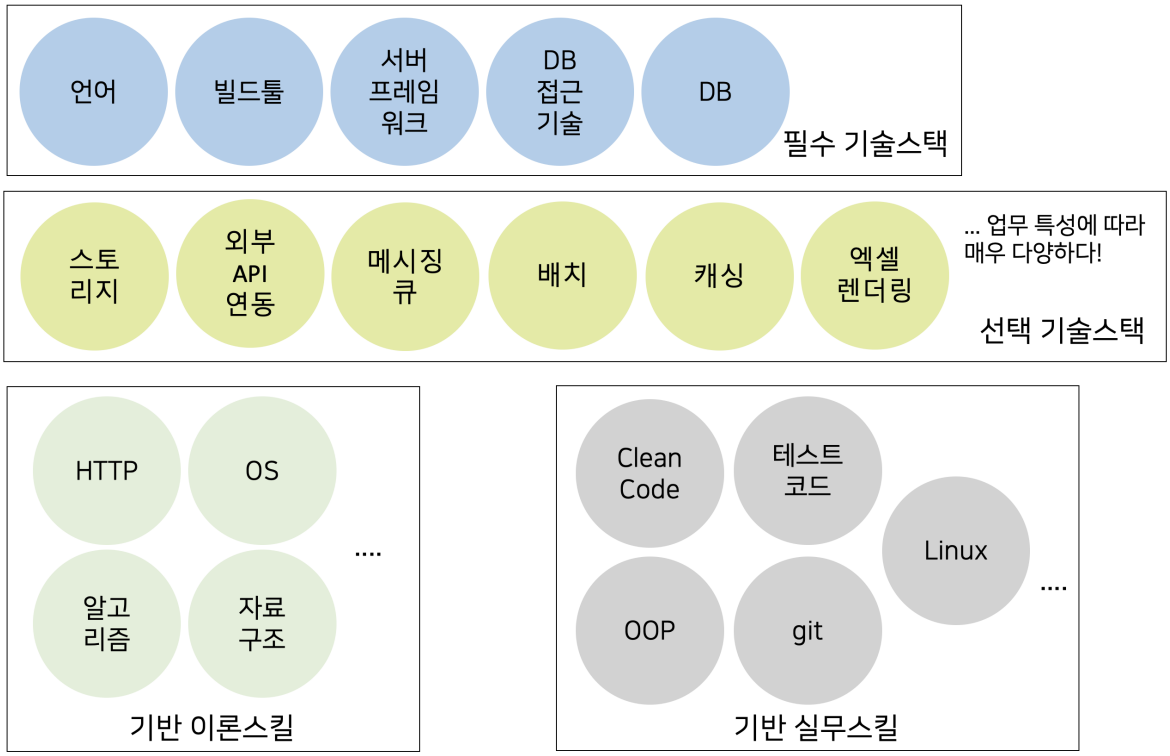
따라서 이번 강의에서 살펴본 것처럼 검색을 통해 자료를 수집하고 공식 자료를 읽어보며 버전을 업그레이드하는 경험은 큰 도움이 될 것이다! 🎉

마지막으로 더 공부하고 싶으신 분들께 드리는 말씀

> 💡 이렇게 Java와 Spring Boot를 활용해 서버 애플리케이션을 개발하고 배포하는 모든 여정은 마무리되었습니다. 정말 고생 많으셨습니다!!! 🙇

저희는 이번 강의를 통해 서버 개발의 전체적인 내용을 살펴보았습니다!

아래 표는 개인적으로 구분해 놓은 `서버 개발자가 알아야 하는 지식/기술` 모음입니다. 당연히 모든 기술이 적혀 있는 것은 아니지만, 저의 경험상 `필수 기술 스택`은 항상 필요하고, 진행하게 되는 프로젝트에 따라 선택 기술 스택, 기반 이론 스킬, 기반 실무 스킬은 갖춰야 하는 정도가 조금씩 달라지더라고요.



이를 기반으로 생각해보면, 저희

- 언어 : Java
- 빌드툴 : gradle
- 서버프레임워크 : Spring Boot
- DB 접근 기술 : JPA

- DB : MySQL (RDS)

이라는 `필수 기술 스택`과 서버 개발에 필요한 `기반 이론 스킬`(ex. 네트워크 / 트랜잭션), `기반 실무 스킬`들(ex. git / linux)을 갖추었습니다.

이제 추가적으로 서버 개발자가 되기 위해 더 긴 여정을 가려는 분들께 몇 가지 방향성을 말씀드립니다. 🙏 🏖️

서버 프레임워크 - Spring Boot

이번 강의를 통해 스프링 부트를 사용해 보았지만, 스프링 부트와 관련된 내용을 정말로 방대합니다!

스프링의 원리나 핵심가치, 스프링의 다양한 모듈들을 추가로 공부하실 수 있습니다.

다만, 스프링의 원리와 핵심가치를 온전히 이해하기 위해서는 OOP나 디자인 패턴 같은 실무 스킬에 어느 정도 익숙해져야 하고 스프링의 다양한 모듈들은 네트워크나 보안 등의 이론 스킬이 어느 정도 필요하다 보니 난이도는 조금 있는 편이라고 생각합니다.

기반 실무 스킬 - 클린 코드, 테스트 코드, OOP

기반 실무 스킬은 언어나 프레임워크가 변경되더라도 변하지 않는 스킬입니다. 예를 들어, 코드를 깔끔하게 작성하는 능력이나 객체나 함수 간의 관계를 적절히 활용하는 능력은 Java를 사용하건, Python을 사용하건, Kotlin을 사용하건 계속 유지될 수 있죠! 무협으로 비유하면 내공, 판타지로 비유하면 마나 서클 같은 기본기라고 할 수 있습니다.

이런 실무 스킬 역량을 기르는 몇 가지 방법을 소개 드립니다!

1. 관련 도서나 강의를 듣는다.
    a. 클린 코드, 리팩토링, 테스트, 객체지향을 키워드로 해서 검색해 보시면 다양한 자료가 나옵니다 🙂
2. 국내 IT 회사들의 기술 블로그 글들 중, <신입 개발자> 또는 <파일럿 프로젝트> 관련 글들을 읽는다.
3. 실습형 강의를 수강하며 다른 사람은 코드를 어떻게 작성했고, 왜 그렇게 작성했는지 이해하고 적용해 본다.
4. 프로젝트를 직접 참여하여 코딩하며 경험을 쌓아 나간다.
    a. 리뷰를 받을 수 있다면 더 좋습니다!

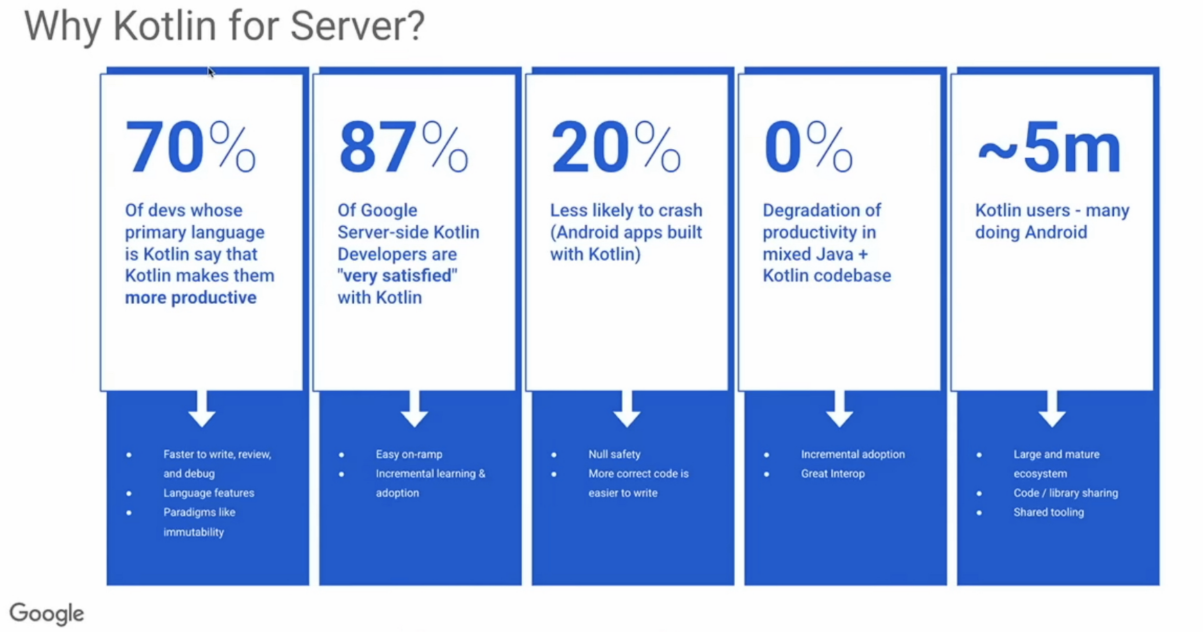
언어 - Kotlin

서버 개발을 할 때 Java라는 언어가 많이 사용되고 Java로 이루어진 프로젝트가 많이 있는 것은 맞지만, 최근에는 같은 JVM 계열의 언어인 Kotlin 역시 많이 사용되고 있습니다. 국내 IT 대기업들의 기술 블로그나 발표 자료에서 Kotlin을 찾아볼 수 있는 것은 물론, 해외에서도 사용 사례가 지속적으로 증가하고 있습니다.

예를 들어 2022년 10월 구글은<Google's Journey from Java to Kotlin for Server Side Programming>이라는 키노트를 통해 다음과 같이 이야기한 바 있습니다.

> Kotlin is now a recommended programming language for server-side JVM usage at Google, while still providing access to a large existing Java ecosystem.

> 코틀린은 존재하는 거대한 Java ecosystem을 활용할 수 있으며, 이제 구글에서 server-side JVM 프로그래밍 언어로 추천된다.



새로운 언어를 배워보며 프로그래밍 언어에 대한 통찰력을 얻고 싶은 분들께는 Kotlin을 추천드립니다.

인프라 - 배포 자동화

저희는 이번 강의에서 배포를 수동으로 진행했습니다! 하지만 배포를 자동으로 진행할 수 있는 여러 방법이 있고 각 방법마다 특징이나 장단점이 다릅니다. 배포 자동화, CI/CD를 검색 키워드로 하여 다양한 정보를 찾아보실 수 있습니다.

만약 코드를 작성해 비즈니스를 구현하는 것보다 배포를 자동화시키고 관리하는 것이 더 재밌으시다면, 인프라 관련 업무를 전문적으로 수행하는 DevOps 또는 SRE라는 직군도 있습니다. 반대로, 코드를 작성하고 비즈니스를 구현하는 것이 더 재미있으시다면, 인프라와 관련된 공부보다는 사용하시는 언어, 프레임워크, 그리고 기반 실무 스킬 (클린 코드, 테스트 등)에 집중하시는 것을 추천드립니다.

프로젝트 - 요구사항을 정하고 스스로 만들어보기

프로젝트를 통해 요구사항을 구현해 보는 것은 다양한 역량을 균형 있게 업그레이드할 수 있는 좋은 방법입니다!

요구사항을 처리하기 위해 다양한 해결책을 검색하고 구현하는 과정에서 다양한 스킬을 활용하기 때문이죠.

여러 사람과 진행하는 프로젝트도 좋고, 혼자만의 요구사항을 정해 만들어보는 것도 좋습니다. 😊

여러분들을 응원합니다! 감사합니다!! 🙇