1. 스프링 부트 입문

Spring Framework vs Spring Boot

Spring Framework vs Spring Boot: 비교표

항목	Spring Framework	Spring Boot
출시 시기	2003년	2014년
설계 철학	유연함: 필요한 구성 요소만 수동으로 조립	간결함: 자동 설정과 최소 구성
설정 방식	XML 또는 Java Config를 이용한 수동 설정	자동 설정 + 최소한의 설정
시작 템플릿	없음 (처음부터 설정 필요)	`spring-boot-starter-*` 제공
내장 서버	없음 (Tomcat, Jetty 수동 배포 필요)	있음 (Tomcat, Jetty 내장)
프로젝트 구조	자유롭게 구성 가능 (유연하지만 복잡함)	표준 디렉터리 구조 권장
실행 방식	외부 WAS 배포 (WAR)	자체 실행 가능 (JAR 실행 가능)
배포 구조	WAR 파일 생성, WAS에 배포	실행 가능한 JAR 생성
의존성 관리	개발자가 수동 관리	Starters로 자동 의존성 관리
사용자 경험	유연하나 진입장벽 높음	빠르고 쉬운 개발, 배포
Spring Initializr	사용하지 않음	공식 제공 (https://start.spring.io)

핵심 차이 정리

1. 설정 방식의 차이

- **Spring Framework**는 XML 기반 또는 Java 기반 설정을 일일이 작성해야 함
  예: `applicationContext.xml`, `@Configuration`, `@Bean`
- **Spring Boot**는 `@SpringBootApplication` 하나로 대부분의 설정 자동 처리
  추가 설정은 `application.yml`이나 `application.properties`로 쉽게 관리

---

2. 내장 톰캣 (Embedded Tomcat)

- Spring Framework는 WAR를 만들어 **외부 Tomcat, JBoss**에 배포해야 실행됨
- Spring Boot는 내장 서버로 실행되기 때문에 단순히 `java -jar`로 실행 가능
  → 컨테이너 없이도 애플리케이션 바로 구동 가능

```
java -jar myapp-0.0.1-SNAPSHOT.jar
```

---

3. 의존성 관리

- Spring Framework는 필요한 의존성을 개발자가 직접 찾고 `pom.xml`에 설정해야 함
- Spring Boot는 `spring-boot-starter-*`만 입력하면 관련 의존성 일괄 포함됨

예:

```
<!-- Spring Framework -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
</dependency>

<!-- Spring Boot -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

---

4. 실행 및 배포 방식

- Spring Framework는 WAR 생성 → WAS에 배포 → 구동 필요
- Spring Boot는 단일 JAR로 자체 실행 가능 → DevOps, 클라우드 배포에 유리

---

5. 학습 곡선

- Spring Framework는 높은 자유도 → 설정 난이도와 학습 부담이 있음
- Spring Boot는 개발자 친화적 구조 → 빠른 개발 가능 → 기업에서 선호

정리 문장

> **Spring Framework는 고전적인 수동 조립형 프레임워크**이고,
> **Spring Boot는 그 조립을 자동으로 도와주는 반자동 프레임워크**라고 볼 수 있어.
> 즉, Spring Boot는 Spring Framework의 쉬운 진입점이자 생산성 향상을 위한 도구야.

Spring Boot의 철학 및 장점

1. Spring Framework

1.1 정의

Spring Framework는 자바 플랫폼을 위한 범용 애플리케이션 프레임워크이다. **IoC (제어의 역전)**, **DI (의존성 주입)**, **AOP**, **트랜잭션**, **웹 애플리케이션 개발(MVC)** 등을 지원하는 핵심 모듈들을 제공한다.

1.2 특징

- POJO 기반 개발
- 모듈화된 구성 (IOC, AOP, Web, JDBC, JMS 등)
- 유연한 설정 방식 (XML, JavaConfig, Annotation)
- DI 컨테이너 기반의 객체 관리
- View/Controller/Service 계층 분리 지원

1.3 사용 목적

- 전통적인 자바 엔터프라이즈 애플리케이션 개발
- XML 또는 JavaConfig 기반의 세밀한 구조 설계
- 프레임워크 구성 요소를 **직접 선택, 결합**하여 설계

2. Spring Boot

2.1 정의

Spring Boot는 Spring Framework 기반 애플리케이션을 **빠르고 간단하게** 개발할 수 있도록 돕는 프레임워크이다. 개발자가 직접 XML 설정을 하지 않아도 되며, **자동 설정(AutoConfiguration)**, **의존성 스타터(Starter)**, **내장 웹서버(Tomcat 등)** 등을 제공한다.

2.2 특징

- 의존성 자동 관리 (`spring-boot-starter-*`)
- 설정 최소화, 관례 기반 구성
- 내장 Tomcat, Jetty, Undertow 지원
- 실행 가능한 `jar` 패키지로 독립 실행 가능
- RESTful API, Web 애플리케이션, CLI 등 빠른 개발에 적합
- `application.yml` 또는 `application.properties`를 통한 외부 설정 바인딩

2.3 사용 목적

- 설정 없이 즉시 실행 가능한 Spring 앱 개발
- 개발과 테스트에 빠르게 진입하고 싶은 경우
- 표준 구조를 따르는 **단일 프로젝트 또는 모놀리식 구조**

3. Spring Cloud

3.1 정의

Spring Cloud는 마이크로서비스 환경을 구축하기 위한 **Spring Boot 기반의 분산 시스템 도구 모음**이다. 서비스 디스커버리, 구성 서버, 게이트웨이, 부하 분산, 분산 추적, 장애 복구 등을 추상화하여 제공한다.

3.2 특징

- 마이크로서비스 아키텍처 지원
- Eureka: 서비스 등록/탐색
- Config Server: 중앙 설정 관리
- Spring Cloud Gateway: API 게이트웨이
- Circuit Breaker: Resilience4j 기반 장애 대응
- Zipkin + Sleuth: 분산 트레이싱
- Spring Cloud Stream: Kafka, RabbitMQ 이벤트 스트리밍

3.3 사용 목적

- **분산 환경** 또는 **마이크로서비스 기반 시스템 구축**
- 클라우드 네이티브 애플리케이션
- 다양한 인프라 서비스(GCP, AWS, K8s)와 통합

4. 핵심 비교 표

항목	Spring Framework	Spring Boot	Spring Cloud
주 목적	자바 기반 엔터프라이즈 애플리케이션	빠른 애플리케이션 개발	마이크로서비스 및 분산 시스템 지원
추상화 수준	낮음 (직접 구성 필요)	중간 (자동 설정, 스타터 제공)	높음 (클라우드 패턴 내장)
설정 방식	수동 설정(XML, JavaConfig)	자동 설정 (관례 기반)	Spring Boot + 추가 의존성으로 구성 관리
학습 난이도	높음	낮음	중간~높음 (전체 아키텍처 이해 필요)
대표 모듈	IoC, AOP, JDBC, MVC	Starter, AutoConfiguration	Config, Discovery, Gateway, Stream 등
배포 형태	외부 웹 서버 필요	내장 서버로 단일 jar 실행 가능	컨테이너 기반, 클러스터 운영을 가정
대상 시스템	모놀리식 또는 계층형 시스템	모놀리식, API 서버 등	마이크로서비스, 클라우드 기반 시스템

5. 실제 사용 흐름과 계층 구조

Spring Framework → Spring Boot → Spring Cloud

1. **Spring Framework**는 핵심적인 기능을 제공
2. **Spring Boot**는 Spring을 **간편하게 사용할 수 있게 포장**
3. **Spring Cloud**는 Spring Boot 애플리케이션을 **마이크로서비스 환경에서 배포하고 운영할 수 있도록 확장**

6. 예시 적용 시나리오

- **Spring Framework 단독**
  - 레거시 시스템 유지보수
  - 기존 Java EE 기반 시스템 리팩토링
  - 대기업 내부 시스템 설계 시, 세밀한 설정 제어 필요할 때
- **Spring Boot**
  - 신규 프로젝트, 빠른 MVP 구축
  - 내부 API 서버
  - Restful 서비스 개발
- **Spring Cloud**
  - 마이크로서비스 기반 전자상거래 시스템
  - 금융/물류/IoT에서의 대규모 모듈 분산 운영
  - 클라우드 상에서 유연한 확장성과 서비스 독립성이 필요한 구조

7. 요약 정리

구분	핵심 목적
Spring	자바 애플리케이션의 기본 아키텍처를 설계하고 구축함
Spring Boot	Spring을 빠르고 쉽게 쓸 수 있도록 함
Spring Cloud	마이크로서비스 시스템을 손쉽게 구성하고 운영하게 함

프로젝트 생성 방법

- # Spring Initializr 사용법

1. Spring Initializr란?

Spring Initializr는 Spring Boot 기반 애플리케이션을 **손쉽게 생성**할 수 있도록 도와주는 도구이다. 복잡한 설정 없이 웹 UI 또는 IDE를 통해 필요한 구성 요소를 선택하면, 자동으로 프로젝트 뼈대를 생성해준다.

- 공식 웹사이트: https://start.spring.io
- IntelliJ, VS Code, Eclipse 등 대부분의 IDE에서 통합 지원
- Gradle/Maven, Java/Kotlin/Groovy 등 다양한 옵션 선택 가능

2. 접근 방법

2.1 웹 UI 사용

1. 브라우저에서 https://start.spring.io 접속
2. 아래 항목을 순서대로 설정
   - **Project**: Maven Project 또는 Gradle Project

- **Language**: Java, Kotlin, Groovy
- **Spring Boot Version**: 안정화 버전 선택 (예: 3.2.x)
- **Project Metadata**:
  - **Group**: 예) `com.example`
  - **Artifact**: 예) `demo`
  - **Name**: 프로젝트 이름
  - **Description**: 간단한 설명
  - **Package Name**: 기본 패키지명
  - **Packaging**: Jar 또는 War
  - **Java Version**: Java 17, 21 등
- **Dependencies**:
  - Web: `Spring Web`, `Spring Reactive Web`
  - Data: `Spring Data JPA`, `MySQL Driver`
  - Dev Tools: `Spring Boot DevTools`, `Lombok`
  - Security, Mail, Validation 등 추가 가능

3. **GENERATE** 버튼 클릭 → zip 파일 다운로드
4. 다운로드 받은 프로젝트를 IDE에서 열기

2.2 IntelliJ IDEA에서 생성

1. **File** → **New** → **Project...**
2. **Spring Initializr** 선택
3. 초기 설정은 웹과 동일 (Project, Language, Group, Artifact 등)
4. 의존성 추가 화면에서 필요한 Starter 선택
5. **Finish** 클릭 → 자동으로 프로젝트 생성 및 설정 완료

2.3 VS Code (Spring Boot Extension Pack 필요)

1. `Ctrl + Shift + P` → `Spring Initializr: Generate a Maven Project`
2. 터미널에서 인터랙티브 방식으로 설정
3. zip 다운로드 및 자동 압축 해제
4. `pom.xml` 또는 `build.gradle` 기반 프로젝트 열기

3. 프로젝트 구조 설명 (Maven 기준)

```
demo/
├── src/
|   ├── main/
|   |   ├── java/com/example/demo/
|   |   |   └── DemoApplication.java      ← 메인 클래스 (SpringBootApplication)
|   |   └── resources/
|   |       ├── application.properties    ← 환경설정 파일
|   |       ├── static/                   ← 정적 리소스 (CSS, JS, 이미지)
|   |       └── templates/                ← Thymeleaf 등의 템플릿 파일
|   └── test/
|       └── java/com/example/demo/
|           └── DemoApplicationTests.java
├── pom.xml                               ← Maven 빌드 설정
```

4. 생성 직후 실행 방법

Maven

```
./mvnw spring-boot:run
```

Gradle

```
./gradlew bootRun
```

또는 IDE에서 `DemoApplication.java` 클래스의 `main()` 함수 실행

5. 의존성 선택 시 고려사항

기능	Starter 의존성
웹 개발	spring-boot-starter-web
REST API	spring-boot-starter-web + Jackson 포함
JPA + DB 연결	spring-boot-starter-data-jpa + mysql-driver
보안	spring-boot-starter-security
테스트	spring-boot-starter-test
템플릿 엔진	spring-boot-starter-thymeleaf
캐시	spring-boot-starter-cache + caffeine/redis
메시징	spring-boot-starter-amqp (RabbitMQ 등)

6. 자주 묻는 질문 (FAQ)

Q1. `Lombok`은 꼭 선택해야 하나요?

- 필수는 아니지만, **코드 간결성(boilerplate 제거)**에 매우 유리하므로 추천

Q2. `DevTools`는 무엇인가요?

- 코드 수정 시 자동 재시작, 템플릿 Live Reload, 캐시 무효화 등을 제공하는 **개발 편의성 도구**

Q3. 프로젝트 생성 시 Gradle과 Maven 중 무엇을 선택해야 하나요?

- Maven: 안정적, 기업에서 널리 사용됨
- Gradle: 빠른 빌드 속도, 설정 유연성, Kotlin DSL 지원

7. 유의사항

- `application.properties` → `application.yml`으로 구조화 가능
- `Java 17 이상`은 Spring Boot 3.x 사용 시 필수
- Spring Boot 버전과 의존성 호환성 확인 필요 (특히 Data, Security 관련)

8. 요약

항목	설명
사용 목적	Spring Boot 프로젝트 생성 자동화
주요 기능	빌드 도구 선택, 메타정보 설정, 의존성 자동 선택
실행 방법	zip 다운로드 → IDE 또는 CLI 실행
장점	빠른 개발 시작, 안정된 구성, 학습 곡선 완화
통합 도구	IntelliJ, Eclipse, VS Code, Spring CLI 등

- # `pom.xml` vs `build.gradle`

1. 개요

항목	**Maven (`pom.xml`)**	**Gradle (`build.gradle`)**
빌드 도구	Apache Maven	Gradle
설정 언어	XML (정형적)	Groovy 또는 Kotlin DSL (스크립트형)
빌드 설정 파일	`pom.xml`	`build.gradle` 또는 `build.gradle.kts`
빌드 속도	전체 빌드, 느린 편	증분 빌드, 빠른 편
설정 복잡도	구조는 명확하지만 길어지기 쉬움	유연하고 간결함

항목	Maven (`pom.xml`)	Gradle (`build.gradle`)
생명주기	정형화된 빌드 생명주기 (compile, test 등)	생명주기 커스터마이징 가능
의존성 선언	`<dependency>` XML 구조	`implementation` 방식 선언적 구조
멀티 모듈	복잡하고 정적	유연하고 동적, DSL 기반 구조화 용이
커뮤니티 및 생태계	안정적, 레거시 시스템 많음	현대적, 점차 확산 중

2. Maven 설정 예시 (`pom.xml`)

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>1.0.0</version>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.5</version>
  </parent>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
  </dependencies>

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

3. Gradle 설정 예시 (`build.gradle`)

```groovy
plugins {
    id 'org.springframework.boot' version '3.2.5'
    id 'io.spring.dependency-management' version '1.1.4'
    id 'java'
}
```

```
6
7  group = 'com.example'
8  version = '1.0.0'
9  sourceCompatibility = '17'
10
11 repositories {
12     mavenCentral()
13 }
14
15 dependencies {
16     implementation 'org.springframework.boot:spring-boot-starter-web'
17     testImplementation 'org.springframework.boot:spring-boot-starter-test'
18 }
19
20 tasks.named('test') {
21     useJUnitPlatform()
22 }
```

---

4. 선택 기준 요약

조건	추천 도구
기존 프로젝트, 안정성 우선	Maven
빠른 빌드, 유연한 설정	Gradle
구조적이고 정형화된 관리 필요	Maven
DSL 기반의 선언적 설정 선호	Gradle
멀티모듈 프로젝트 구성	Gradle

---

5. 정리

Maven은 전통적으로 많이 사용되어온 빌드 도구로 구조가 명확하고 예측 가능하지만, 설정이 장황할 수 있다. 반면 Gradle은 선언형 DSL을 기반으로 보다 간결하고 유연한 빌드 구성이 가능하며, 캐시 기반 빌드로 성능이 뛰어나다. 최근 Spring Boot에서는 Gradle이 점차 기본값으로 자리잡는 추세이다.

원한다면 Gradle의 Kotlin DSL 예시, 멀티모듈 설정 방법, 속도 비교 등으로 주제를 확장할 수 있다. 다음 항목으로 이어가고 싶으면 "계속"이라고 입력하면 된다.

자동 설정(Auto Configuration)

1. 동작 원리 요약

Spring Boot의 자동 설정은 아래와 같은 흐름으로 작동한다.

1. `@SpringBootApplication` → 내부적으로 `@EnableAutoConfiguration` 포함
2. `@EnableAutoConfiguration` → `AutoConfigurationImportSelector` 실행

3. `spring.factories` 또는 `META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports` 파일에서 자동설정 클래스 목록 로딩
4. 조건에 맞는 Configuration 클래스만 Bean으로 등록 (`@ConditionalOnXxx`)

---

2. 관련 주요 애노테이션

애노테이션	설명
`@EnableAutoConfiguration`	자동 설정 활성화 (보통 생략됨, `@SpringBootApplication`에 포함)
`@ConditionalOnClass`	특정 클래스가 존재할 때만 설정 적용
`@ConditionalOnMissingBean`	특정 Bean이 없을 때만 설정 적용
`@ConditionalOnProperty`	프로퍼티가 특정 값일 때만 설정 적용
`@ConditionalOnWebApplication`	웹 환경일 때만 적용

---

3. 자동 설정 예시

예: Spring Web 의존성이 있을 경우

`spring-boot-starter-web`을 추가하면 다음 설정들이 자동 적용된다.

- `DispatcherServlet` 자동 등록
- `WebMvcConfigurer` 기본 설정
- `MessageConverters`, `HandlerMappings`, `ViewResolvers` 등 자동 구성

자동 구성 클래스를 보면 다음과 같은 구조다.

```
@Configuration
@ConditionalOnClass({ Servlet.class, DispatcherServlet.class })
@ConditionalOnWebApplication
@EnableConfigurationProperties(WebMvcProperties.class)
public class WebMvcAutoConfiguration {
    // ...
}
```

→ 클래스패스에 `DispatcherServlet`이 존재하고, 웹 환경이면 자동 구성된다.

---

4. spring.factories / spring-autoconfigure-metadata

자동 설정 대상은 `META-INF/spring.factories` (Spring Boot 2.x 이하) 또는 `META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports` (Spring Boot 3.x 이상)에 명시되어 있다.

예 (Spring Boot 2.x 기준):

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.web.servlet.WebMvcAutoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration,\
org.springframework.boot.autoconfigure.orm.jpa.HibernateJpaAutoConfiguration
```

---

5. 자동 설정 확인 방법

1. `@SpringBootApplication`이 실제로 무엇을 포함하는지 살펴보기

```
@SpringBootApplication
// 내부적으로 다음 애노테이션 포함
@Configuration
@EnableAutoConfiguration
@ComponentScan
```

1. **실행 시 어떤 자동 설정이 적용됐는지 확인하려면:**

```
--debug
```

혹은 `application.properties`에 다음과 같이 작성

```
debug=true
```

→ 자동 설정된 항목과 제외된 항목을 콘솔에 출력해준다.

---

6. 자동 설정 제외 방법

특정 자동 설정을 비활성화하고 싶은 경우, 다음과 같이 `exclude` 옵션을 사용한다.

```
@SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })
public class MyApp {
    public static void main(String[] args) {
        SpringApplication.run(MyApp.class, args);
    }
}
```

또는 application.yml에서 속성 조건을 조절할 수도 있다.

---

7. 사용자 정의 자동 설정 만들기

커스텀 라이브러리를 만들거나 재사용 가능한 모듈에서 자동 설정을 만들고 싶을 때는 아래처럼 한다.

1. 구성 클래스에 `@Configuration`, `@ConditionalOnClass` 등으로 조건 설정
2. `resources/META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports` 파일 생성
3. 구성 클래스 이름을 등록

```
com.example.autoconfig.MyCustomAutoConfiguration
```

---

8. 장점

- 설정 최소화: 수동 설정 없이도 대부분의 구성 가능
- 빠른 개발: 설정 파일 없이도 애플리케이션 빠르게 구성 가능
- 조건 기반 구성: 클래스 존재 여부, 환경 설정 등에 따라 유동적으로 설정 적용

---

9. 주의사항

- 자동 설정은 **개발자에게 보이지 않게 구성되기 때문에**, 동작 원리를 잘 모르면 디버깅이 어려울 수 있다.
- 자동 설정과 수동 설정이 충돌하면, 보통 수동 설정이 우선한다.

결론

Spring Boot의 자동 설정은 Spring 애플리케이션의 진입 장벽을 크게 낮춘 핵심 기능이다. 기본 설정을 기반으로 빠르게 애플리케이션을 구성할 수 있으며, 필요시 유연하게 오버라이딩도 가능하다. 이를 이해하고 적극적으로 활용하면 Spring Boot의 생산성을 극대화할 수 있다.

다음으로 `starter` 구조, `application.properties vs application.yml`, `@SpringBootApplication` 구조 분석 등을 원한다면 알려주면 된다.
계속해서 전체 흐름을 따라가고 싶다면 `"계속"`이라고 입력해주면 된다.

의존성 관리 (Spring Boot Starter)

1. Starter란 무엇인가?

Spring Boot Starter는 **특정 기능을 빠르게 시작할 수 있도록 관련 라이브러리를 하나의 패키지로 묶은 의존성 집합**이다.
예를 들어, 웹 개발에 필요한 라이브러리들을 모두 포함한 `spring-boot-starter-web`,
데이터베이스 접근을 위한 `spring-boot-starter-data-jpa` 등이 존재한다.

---

2. 동작 구조

Spring Boot Starter는 내부적으로 여러 개의 실제 라이브러리들을 **transitive(전이적)** 하게 포함하고 있다.

예를 들어:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

이 하나만 추가하면 다음과 같은 라이브러리가 함께 포함된다:

- spring-web

- spring-webmvc
- jackson-databind
- tomcat-embed-core
- spring-boot-starter
- slf4j (로깅)

즉, 여러 라이브러리를 일일이 명시하지 않고도 웹 애플리케이션 구성이 가능하다.

---

3. 대표 Starter 목록

Starter 이름	설명
`spring-boot-starter`	공통 의존성 (logging, validation 등 포함)
`spring-boot-starter-web`	웹 MVC + REST API 개발 (Tomcat 포함)
`spring-boot-starter-data-jpa`	Spring Data JPA + Hibernate
`spring-boot-starter-thymeleaf`	서버 사이드 템플릿 Thymeleaf 지원
`spring-boot-starter-security`	Spring Security 기반 인증/인가
`spring-boot-starter-test`	JUnit5, Mockito 등 테스트 통합 구성
`spring-boot-starter-actuator`	헬스체크, 메트릭, 모니터링 기능
`spring-boot-starter-validation`	Bean Validation (javax.validation)
`spring-boot-starter-logging`	기본 로깅 (SLF4J + Logback)

---

4. `spring-boot-starter-parent`

의존성 관리를 위한 부모 프로젝트로, 모든 Spring Boot 프로젝트는 보통 이를 부모로 상속받는다.

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.2.5</version>
</parent>
```

기능

- 공통 의존성 버전 관리
- Maven 플러그인 기본 설정
- 기본 인코딩, Java 버전, 리포지토리 설정 등 포함

이 덕분에 `pom.xml`에서 개별 버전을 명시하지 않아도 되고, 버전 충돌도 방지된다.

---

5. BOM (Bill of Materials)

Spring Boot는 BOM을 통해 의존성 버전을 통제한다.
예를 들어, `spring-boot-dependencies`는 다음과 같이 모든 하위 라이브러리의 버전을 정의하고, 이 설정을 통해 하위 라이브러리들의 버전을 통일시킨다.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-dependencies</artifactId>
      <version>3.2.5</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Gradle에서는 `platform` 키워드를 사용해 BOM을 import할 수 있다.

---

6. 직접 의존성과 Starter 의존성 비교

직접 의존성 예시 (비권장)

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
</dependency>
```

Starter 의존성 예시 (권장)

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

→ 후자의 방식은 유지보수와 버전 충돌 방지에 훨씬 유리하다.

---

7. 장점 요약

- 일관된 의존성 버전 관리
- 빠른 설정 및 초기화
- 의존성 충돌 감소

- 학습 부담 감소
- 유지보수 용이

---

8. 주의할 점

- Starter는 내부적으로 많은 라이브러리를 포함하므로, 불필요한 스타터를 무분별하게 추가하면 프로젝트 크기가 커지고 보안 이슈가 생길 수 있다.
- 어떤 라이브러리가 포함되어 있는지 **Maven Dependency Tree** 또는 `gradle dependencies` 명령어로 확인할 수 있다.

결론

Spring Boot의 Starter 구조는 개발자에게 일관된 환경과 간결한 설정을 제공하며, 생산성과 유지보수성을 크게 향상시킨다.
프로젝트의 목적에 맞는 Starter만 선택하여 사용하는 것이 핵심이다.