1. 개요 및 개발 환경 설정

Spring 프레임워크의 역사와 철학

1. 스프링 프레임워크의 탄생 배경

1.1 EJB(Enterprise JavaBeans)의 한계

- 1990년대 후반~2000년대 초반, Java EE(EJB 기반)의 엔터프라이즈 애플리케이션은 복잡하고 무거운 구조였다.
- 개발자는 단순한 기능을 구현하기 위해 수많은 XML 설정과 무거운 컨테이너 설정을 요구받았다.
- **테스트가 어려웠고, 비즈니스 로직과 인프라 로직이 밀접하게 결합**되어 있었으며, **재사용성과 유지보수성**이 매우 낮았다.

1.2 Rod Johnson의 문제 제기

- 2002년, **Rod Johnson**은 EJB의 복잡성과 비효율성을 비판하면서 자바 기반의 경량 프레임워크를 제안함.
- 그의 저서 *"Expert One-on-One J2EE Design and Development"*에서 EJB 대신 **POJO(Plain Old Java Object)** 기반 설계를 제안했고, 거기에 사용된 코드가 바로 **Spring의 초기 버전**이 되었다.
- 이후 2003년 Rod Johnson을 중심으로 오픈소스 프로젝트 **Spring Framework**가 시작되었다.

---

2. 버전별 발전 역사

버전	출시년도	주요 특징
1.x	2004	DI, AOP, JDBC 템플릿 등 초기 핵심 컴포넌트 도입
2.x	2006	XML 기반 설정 강화, `@Autowired` 등 어노테이션 도입
3.x	2009	Java 5 지원, SpEL(Expression Language), REST 지원 시작
4.x	2013	Java 8 람다 지원, WebSocket 지원, Groovy 통합
5.x	2017	Java 8 완전 지원, Reactive Streams (WebFlux) 도입
6.x	2022~	Java 17 이상, Jakarta EE 9+ 기반 API로 완전 전환

---

3. Spring의 설계 철학

3.1 POJO 기반 개발

- 스프링은 **비침투성(non-invasive)**을 철저히 지향한다.
- 개발자는 특정 프레임워크나 상속 구조에 종속되지 않고, 일반 자바 객체(POJO)로만도 애플리케이션을 설계할 수 있다.
- 프레임워크가 아닌 **도메인 중심 설계(DDD)**에 집중할 수 있다.

3.2 제어의 역전(IoC)과 의존성 주입(DI)

- 스프링은 객체 생명주기와 의존성 관리를 개발자가 아닌 **컨테이너가 관리**한다.
- 이를 통해 컴포넌트 간 결합도를 낮추고, 테스트와 유지보수를 쉽게 한다.

3.3 관점 지향 프로그래밍(AOP)

- 공통 기능(로깅, 보안, 트랜잭션 등)은 핵심 비즈니스 로직과 분리된 **관점으로 구현**한다.
- 코드 중복 없이, 핵심 로직에 집중할 수 있다.

3.4 선언적 프로그래밍

- 설정 파일 또는 어노테이션을 통해 **로직을 선언적으로 구성**한다.
- 명령형 프로그래밍보다 코드의 의도를 더 명확히 표현할 수 있다.
- 대표 예: `@Transactional`, `@Cacheable`

3.5 개방-폐쇄 원칙(Open-Closed Principle)

- 기존 코드를 변경하지 않고도 **새로운 기능을 확장**할 수 있도록 구조화되어 있다.
- 커스터마이징 및 확장성이 뛰어나다.

3.6 모듈화 구조

- 스프링은 **느슨하게 결합된 모듈들의 집합**이다.
- 필요에 따라 **IoC, AOP, Web, Data, Security 등**을 독립적으로 선택해 사용할 수 있다.

---

4. 스프링의 성공 요인

4.1 개발자 중심의 설계

- Java EE의 복잡한 구조와 비교해 **학습 곡선이 낮고**, 개발에 집중할 수 있는 환경을 제공했다.
- 테스트 가능성과 구조적 유연성은 대규모 프로젝트에 적합했다.

4.2 커뮤니티 중심의 발전

- 수많은 개발자들이 GitHub, Stack Overflow, 블로그 등에서 사용사례와 해결책을 공유하며 **지속적인 진화를 견인**했다.

4.3 Spring Boot의 등장

- 스프링 부트는 "설정보다 관례(Convention over Configuration)" 철학을 바탕으로 개발 초기 진입장벽을 혁신적으로 낮췄다.
- 수많은 스타터, 자동 구성, 내장 톰캣 지원 등은 실무에서 **즉시 사용 가능한 생산성을 제공**했다.

---

5. 현대적 가치와 미래

5.1 클라우드 네이티브 아키텍처

- Spring Cloud를 통해 마이크로서비스, 서비스 디스커버리, API 게이트웨이, 분산 트레이싱 등 지원

5.2 Reactive Programming

- Spring WebFlux를 통해 **비동기, 논블로킹 프로그래밍** 모델 도입
- Reactor 기반 Flux/Mono 지원

5.3 Kotlin 및 GraalVM 통합

- 최신 언어 및 런타임 기술에 적극 대응

5.4 Jakarta EE 전환

- Java EE에서 Jakarta EE로 패키지 전환 (`javax` → `jakarta`)
- Spring Framework 6부터 완전 적용

---

6. 요약 정리

항목	내용
프레임워크 철학	IoC, DI, AOP, POJO 기반, 선언적 구성
주요 장점	모듈성, 비침투성, 테스트 용이성, 생산성
역사적 기원	Rod Johnson이 EJB의 복잡성 극복을 위해 고안
개발 방향	개발자 생산성 중심 → 자동화 → 클라우드 네이티브
핵심 가치	유연성, 확장성, 현대성, 테스트 용이성

Spring vs Spring Boot vs Spring Cloud

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

Java EE와의 차이점

1. 개념 정의

Java EE (현재 Jakarta EE)

- **Oracle**이 주도하던 **자바 공식 엔터프라이즈 플랫폼 표준**
- EJB, Servlets, JSP, JMS, JPA, JAX-RS 등을 포함하는 **명세(specification) 기반 아키텍처**
- Java 11 이후 Eclipse 재단으로 이관되어 명칭이 **Jakarta EE**로 변경됨
- **서버 벤더(TomEE, Payara, WildFly 등)**가 해당 명세를 구현

Spring Framework

- **Rod Johnson**이 주도하여 **경량 자바 애플리케이션 개발**을 목표로 개발된 **오픈소스 프레임워크**
- Java EE의 복잡성과 무거움을 해결하고자 등장
- IoC, AOP, DI, MVC, Data Access, Security 등을 모듈화하여 제공
- **비표준(Pojo 기반)** 방식으로 애플리케이션의 구조를 유연하게 구성 가능

---

2. 철학과 설계 원칙

항목	Java EE (Jakarta EE)	Spring Framework
구조	명세 중심, 구현체는 WAS(웹 애플리케이션 서버)	개발자 친화적 프레임워크
중심 개념	EJB, 컨테이너, 표준 API	POJO, IoC, DI, AOP, 선언적 구성

항목	Java EE (Jakarta EE)	Spring Framework
코드 제어 흐름	서버 컨테이너가 주도	개발자가 프레임워크와 코드 흐름을 구성
API 제공 방식	JCP에서 정의된 표준 API	오픈 커뮤니티 기반의 유연한 API
프레임워크 침투성	높음 (EJB 등 인터페이스, 상속 필수)	낮음 (POJO 중심, 프레임워크 독립적 코드 가능)
개발 철학	표준화, 규격 위주	생산성, 유연성, 테스트 용이성 중심

---

3. 주요 기술 구성 요소 비교

기능 구분	Java EE (Jakarta EE)	Spring Framework
의존성 주입	`@Inject`, `@EJB`	`@Autowired`, `@Inject`, `@Bean`
ORM	JPA (javax.persistence)	JPA + Spring Data JPA
웹 프레임워크	Servlet, JSP, JSF	Spring MVC
REST API	JAX-RS (`@Path`, `@GET`)	`@RestController`, `@GetMapping`
트랜잭션	`@Transactional`	`@Transactional` (Spring 자체 구현)
보안	JAAS	Spring Security
비동기 처리	`@Asynchronous`	`@Async`, Reactor (WebFlux)
배치 처리	없음 (벤더 종속)	Spring Batch
메시징	JMS	Spring Integration, Spring Kafka
웹소켓	javax.websocket API	Spring WebSocket

---

4. 개발 및 배포 방식의 차이

항목	Java EE	Spring Framework
배포 방식	EAR 또는 WAR 파일 → WAS에 배포	실행 가능한 JAR 파일 (Spring Boot)
서버 필요 여부	독립 실행 불가, WAS 필요	내장 Tomcat 등으로 **독립 실행 가능**
의존성 관리	수동, 서버 벤더에 따라 상이함	Maven/Gradle 기반 자동 관리
개발 초기 진입 장벽	높음	낮음
설정 방식	XML 위주, 표준화된 설정 필요	JavaConfig, 어노테이션 기반 설정 가능

---

5. 생산성과 유지보수성

항목	Java EE	Spring Framework
초기 개발 속도	느림	빠름
유지보수 용이성	상대적으로 어려움	쉬움 (계층 구조 분리, 테스트 편의)
테스트	Mock 및 단위 테스트 어려움	단위 테스트 및 MockBean 구성 쉬움
커스터마이징	벤더 종속적, 한계 있음	매우 유연, 설정/구조 변경 쉬움

---

6. 발전 방향 및 시장 흐름

항목	Java EE (Jakarta EE)	Spring (Boot 포함)
커뮤니티 지원	Eclipse 재단 관리, 기업 위주	대규모 글로벌 커뮤니티, 오픈소스 중심
주요 사용처	공공기관, 금융기관, 레거시 시스템	스타트업, 대기업, 클라우드 환경 전반
클라우드 대응	미약 (일부 Jakarta EE 10 시도 중)	Spring Cloud, WebFlux 등 매우 활발
도구 통합성	낮음	높은 IDE 및 클라우드 통합 (DevTools 등)

---

7. 대표적인 프레임워크 비교 (기술 스택 관점)

기술 목적	Java EE	Spring
ORM	JPA	Hibernate + Spring Data JPA
웹 애플리케이션	JSF, JSP, Servlets	Spring MVC, Thymeleaf, REST API
보안	JAAS	Spring Security
트랜잭션 관리	EJB, UserTransaction	`@Transactional` via AOP
메시징	JMS	Spring AMQP, Kafka
설정	`web.xml`, `application.xml`	`application.yml`, Java Config, Annotation

---

8. 요약

항목	Java EE (Jakarta EE)	Spring Framework
제어 주체	컨테이너 중심	개발자 중심
설정 복잡성	높음	낮음
확장 유연성	벤더 종속적	매우 유연

항목	Java EE (Jakarta EE)	Spring Framework
실행 구조	외부 WAS 필요	독립 실행 가능 (Spring Boot)
생산성	낮음 (특히 과거)	매우 높음
커뮤니티 및 생태계	느리게 변화	활발하고 빠른 진화

---

9. 결론

- Spring은 Java EE의 **대안 또는 보완재**로 출발하여, 현재는 **사실상의 표준 프레임워크**로 자리잡았다.
- Java EE는 **표준화, 안정성**에 강점을 가지며, Spring은 **생산성과 유연성, 실용성**에 집중한다.
- 현대 개발에서는 대부분 **Spring Boot + Spring Cloud** 조합을 통해 클라우드 네이티브 환경에 대응하고 있으며, Java EE는 **특정 기업 환경이나 레거시 시스템 유지보수**에 주로 활용된다.

프로젝트 생성 방식

- # Spring Initializr

1. 정의

Spring Initializr는 Spring Boot 애플리케이션을 빠르게 시작할 수 있도록 도와주는 **프로젝트 생성 자동화 도구**이다. 개발자가 복잡한 빌드 스크립트나 의존성 설정 없이 **기본 프로젝트 구조와 주요 의존성들이 포함된 프로젝트 뼈대**를 생성할 수 있도록 한다.

- 공식 URL: https://start.spring.io
- 다양한 방식 지원:
  - 웹 UI
  - IntelliJ IDEA, Eclipse, VS Code 등의 IDE 통합
  - REST API (백엔드 제공)
  - 커맨드라인 (Spring CLI, curl, HTTP Client)

---

2. 기본 원리

Spring Initializr는 다음과 같은 정보를 기반으로 프로젝트를 생성한다:

1. **메타데이터 입력** (Group, Artifact, Name, Description 등)
2. **의존성 선택** (`spring-boot-starter-web`, `data-jpa`, `security`, `lombok` 등)
3. **패키징 방식** (JAR or WAR)
4. **언어** (Java, Kotlin, Groovy)
5. **빌드 도구** (Maven, Gradle)
6. **Spring Boot 버전**

→ 이를 바탕으로 `.zip` 아카이브 형태로 **Maven 또는 Gradle 기반의 초기화된 프로젝트**를 생성해준다.

---

3. 웹 UI 사용법

https://start.spring.io에 접속 후:

3.1 Project

- `Maven Project`: XML 기반 빌드, 안정적
- `Gradle Project`: Groovy/Kotlin DSL 기반, 빠른 빌드

3.2 Language

- `Java`: 기본값, 가장 널리 사용됨
- `Kotlin`, `Groovy`: 함수형 스타일 지원 언어

3.3 Spring Boot Version

- 안정 버전 선택 (RELEASE)
- 최신 기능 테스트 시 SNAPSHOT 가능

3.4 Project Metadata

- **Group**: 예: `com.example`
- **Artifact**: 예: `demo`
- **Name**, **Description**, **Package name**
- **Packaging**: `jar` 또는 `war`
- **Java Version**: 8, 11, 17, 21 등

3.5 Dependencies

- 검색창을 통해 의존성 추가 가능
  - Web (`spring-boot-starter-web`)
  - JPA (`spring-boot-starter-data-jpa`, `H2`, `MySQL`)
  - DevTools, Lombok, Thymeleaf, Security 등
- 한 번에 여러 개 선택 가능

3.6 생성

- [GENERATE] 버튼 클릭 → `.zip` 파일 다운로드

---

4. 생성 결과 구조 (Maven 예시)

```
demo/
├── pom.xml                          ← Maven 빌드 설정
├── src/
|   ├── main/
|   |   ├── java/com/example/demo/
|   |   |   └── DemoApplication.java     ← main() 포함된 시작 클래스
|   |   └── resources/
|   |       ├── application.properties   ← 환경 설정 파일
|   |       ├── static/                  ← 정적 자원 (CSS, JS, 이미지)
|   |       └── templates/               ← Thymeleaf 템플릿 위치
|   └── test/
|       └── java/com/example/demo/
|           └── DemoApplicationTests.java ← 기본 테스트 클래스
```

---

5. IntelliJ IDEA에서 사용

1. **File > New > Project > Spring Initializr**
2. Initializr 서버 선택 (기본은 start.spring.io)
3. 프로젝트 메타정보, 의존성 선택
4. Finish → 프로젝트 자동 생성됨

---

6. CLI에서 사용

curl로 생성

```
curl https://start.spring.io/starter.zip \
  -d dependencies=web,data-jpa \
  -d type=maven-project \
  -d language=java \
  -d bootVersion=3.2.0 \
  -d baseDir=demo \
  -o demo.zip
```

압축 해제 후 빌드

```
unzip demo.zip
cd demo
./mvnw spring-boot:run
```

---

7. 초기 의존성 분석 예시

`spring-boot-starter-web` 선택 시 자동 포함되는 구성 요소:

- Spring MVC (`DispatcherServlet`)
- Jackson (JSON 직렬화/역직렬화)
- Validation (Hibernate Validator)
- 내장 Tomcat 서버
- SLF4J + Logback 로깅 설정

---

8. 생성된 메인 클래스 구조

```
@SpringBootApplication // @Configuration + @EnableAutoConfiguration + @ComponentScan
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

---

9. 고급 사용법

application.yml 로 구성

```
server:
  port: 8081

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/mydb
    username: root
    password: 1234
```

Lombok 적용

- `@Getter`, `@Setter`, `@Builder`, `@NoArgsConstructor` 등 활용
- IDE에 플러그인 설치 필수

---

10. 주의사항 및 팁

- `Lombok` 사용 시 IntelliJ 설정에서 Annotation Processor 활성화 필요
- `spring-boot-starter-test`는 JUnit5, Mockito, AssertJ 포함
- `application.properties` → `.yml`로 구조화하는 것이 가독성과 계층 구조상 유리
- Spring Boot 3.x 이상 사용 시 JDK 17 이상 필요

---

11. 요약

항목	설명
목적	Spring Boot 프로젝트 자동 생성
접근 방식	웹 UI / IDE / CLI / REST API
지원 언어	Java, Kotlin, Groovy
빌드 도구	Maven, Gradle
주요 장점	빠른 시작, 표준화된 구조, 반복 작업 제거

- # Spring CLI

1. 개요

Spring CLI는 Spring Boot 애플리케이션을 **명령줄에서 빠르게 생성, 실행, 테스트**할 수 있도록 해주는 **커맨드라인 도구**이다. 간단한 REST API 서버나 Groovy 기반 스크립트를 **Java 없이도 실행**할 수 있는 개발용 툴로, **Maven이나 Gradle 설정 없이도 빠른 프로토타이핑**이 가능하다.

- 가볍고 빠른 개발
- Groovy 기반 스크립트로 애플리케이션 정의 가능
- Spring Initializr와 연동 가능
- 테스트, REST 요청 실행, BOM 관리 가능

---

2. 설치 방법

2.1 SDKMAN (권장 방식 - macOS, Linux)

```
curl -s "https://get.sdkman.io" | bash
source "$HOME/.sdkman/bin/sdkman-init.sh"
sdk install springboot
```

2.2 Homebrew (macOS)

```
brew tap spring-io/tap
brew install spring-boot
```

2.3 Chocolatey (Windows)

```
choco install springboot
```

2.4 수동 다운로드

- https://github.com/spring-projects/spring-boot/releases 에서 zip 파일 다운로드 후 환경 변수 설정

---

3. 주요 명령어 요약

명령어	설명
`spring`	전체 CLI 명령 목록
`spring init`	프로젝트 생성 (Spring Initializr 사용)
`spring run`	Groovy 스크립트 실행
`spring test`	Groovy 테스트 실행
`spring install` / `uninstall`	Spring CLI 자체 설치/제거
`spring version`	현재 설치된 버전 확인

---

4. `spring init` 명령어

Spring Initializr를 CLI로 사용하는 기능이다.

4.1 기본 사용

```
spring init my-app
cd my-app
./mvnw spring-boot:run
```

4.2 고급 사용 예

```
spring init \
  --build=maven \
  --java-version=17 \
  --dependencies=web,data-jpa,h2,lombok \
  --packaging=jar \
  --groupId=com.example \
  --artifactId=demo \
  demo
```

옵션 설명:

옵션	설명
`--build`	Maven 또는 Gradle 선택
`--java-version`	JDK 버전 (예: 17, 21)

옵션	설명
`--dependencies`	Starter 의존성 목록 (쉼표로 구분)
`--packaging`	jar 또는 war
`--groupId`	패키지 그룹 명
`--artifactId`	프로젝트 이름
`--boot-version`	Spring Boot 버전 명시 가능

---

5. `spring run` 명령어

Spring CLI는 Groovy 기반의 `.groovy` 파일을 즉시 실행할 수 있다.

5.1 Hello API 예제

```
// app.groovy
@RestController
class HelloController {
  @GetMapping("/")
  String home() {
    return "Hello, Spring CLI"
  }
}
```

```
spring run app.groovy
```

→ http://localhost:8080 에서 바로 서비스됨

5.2 Groovy Script 특징

- 클래스 정의 없이 바로 사용 가능
- Spring Boot 자동 설정 적용됨
- REST Controller, Service 등 어노테이션 인식 가능

---

6. `spring test` 예제

```
// test_app.groovy
@RestController
class Demo {
  @GetMapping("/ping")
  String ping() {
    return "pong"
  }
}
```

```groovy
// test_app_test.groovy
@SpringBootTest
class PingTests extends Specification {
  @Autowired Demo controller

  def "ping should return pong"() {
    expect:
    controller.ping() == "pong"
  }
}
```

```
spring test test_app.groovy test_app_test.groovy
```

> Spock 기반 Groovy 테스트를 바로 실행 가능

---

7. REST 클라이언트 통합

Spring CLI는 `httpie` 또는 `curl`과 함께 빠르게 REST 요청 테스트가 가능하며, WebTestClient 또는 `spring shell`과도 연동할 수 있다.

---

8. 사용 사례

사용 목적	Spring CLI 활용 예
빠른 프로토타이핑	`spring run *.groovy`
REST API 단기 테스트	`curl`과 함께 실행
학습 및 데모	복잡한 빌드 없이 실습
Live Coding 데모	Groovy 코드 즉시 실행

---

9. Spring CLI vs Spring Boot

항목	Spring CLI	Spring Boot 프로젝트
목적	빠른 테스트, 스크립트 실행	구조적 애플리케이션 개발
사용 언어	주로 Groovy	Java, Kotlin, Groovy
실행 방식	커맨드라인 실행	IDE 또는 빌드 도구 기반 실행
빌드 파일	불필요	pom.xml / build.gradle 필요
장점	설정 없이 즉시 실행	확장성, 팀 개발, 대규모 애플리케이션 적합

---

10. 요약

항목	설명
도구 이름	Spring CLI
주요 기능	프로젝트 생성, Groovy 스크립트 실행, 테스트
의존 도구	Groovy 필요
대표 명령어	`spring init`, `spring run`, `spring test`
용도	빠른 애플리케이션 실험, 교육, 테스트
설치 방법	SDKMAN, Homebrew, Chocolatey 등

빌드 도구 선택

- # Maven 설정

1. Maven이란?

Maven은 자바 기반 프로젝트의 **빌드(Build), 의존성 관리(Dependency Management), 프로젝트 설정(Configuration), 배포(Deployment)**를 자동화해주는 도구이다.

- "Convention over Configuration" 철학 기반
- pom.xml을 통한 선언적 설정
- 로컬/원격 저장소에서 라이브러리 자동 다운로드

---

2. Maven 설치 및 환경 변수 설정

2.1 설치 (macOS 예시)

```
brew install maven
```

2.2 환경 변수 설정 (Linux 예시)

```
export M2_HOME=/usr/local/apache-maven-3.9.5
export PATH=$PATH:$M2_HOME/bin
```

2.3 확인

```
mvn -v
```

---

3. Maven 기본 디렉토리 구조

Maven 프로젝트는 다음과 같은 구조를 기본으로 한다:

```
project-root/
├── pom.xml
└── src/
    ├── main/
    |   ├── java/              → 애플리케이션 소스
    |   └── resources/         → 설정, 프로퍼티 파일 등
    └── test/
        ├── java/              → 테스트 코드
        └── resources/
```

---

4. 핵심 설정 파일: `pom.xml`

예시

```xml
<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
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Demo</name>
  <description>Spring Boot Demo Project</description>

  <properties>
    <java.version>17</java.version>
    <spring-boot.version>3.2.0</spring-boot.version>
  </properties>

  <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>
```

---

5. 주요 태그 설명

태그	설명
`<groupId>`	조직 또는 도메인명 기반 패키지 그룹
`<artifactId>`	프로젝트 이름
`<version>`	프로젝트 버전
`<packaging>`	`jar`, `war`, `pom` 등
`<properties>`	재사용 가능한 변수 선언
`<dependencies>`	의존성 목록 (라이브러리)
`<build>`	빌드 전략 설정 (플러그인 포함)
`<plugin>`	Maven 플러그인 설정 (컴파일, 실행, 배포 등)

---

6. 의존성 관리

6.1 기본 의존성

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

6.2 의존성 범위(scope)

scope	용도
`compile`	기본값, 애플리케이션에 항상 포함
`provided`	WAS에서 제공되는 것 (ex: Servlet API)
`runtime`	실행 시에만 필요 (ex: JDBC Driver)
`test`	테스트 코드 전용 (ex: JUnit, Mockito)

---

7. 부모 POM과 BOM

7.1 Spring Boot Parent POM

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.2.0</version>
</parent>
```

→ Maven 설정, 인코딩, 버전 관리 등을 상속

7.2 BOM (Bill Of Materials)

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>2023.0.0</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

→ 여러 의존성 버전을 한 번에 통제

---

8. Maven Plugin 설정

8.1 Spring Boot Maven Plugin

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <executions>
        <execution>
          <goals>
            <goal>repackage</goal> <!-- JAR로 재패키징 -->
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

8.2 Compiler Plugin

```
<plugin>
  <artifactId>maven-compiler-plugin</artifactId>
  <configuration>
    <source>17</source>
    <target>17</target>
  </configuration>
</plugin>
```

---

9. 프로파일 설정

예시: 개발/운영 환경 분리

```
<profiles>
  <profile>
    <id>dev</id>
    <properties>
      <env>dev</env>
    </properties>
  </profile>
  <profile>
    <id>prod</id>
    <properties>
      <env>prod</env>
    </properties>
  </profile>
</profiles>
```

실행:

```
mvn clean package -Pdev
```

---

10. 멀티모듈 구조

루트 `pom.xml`

```
<modules>
  <module>core</module>
  <module>api</module>
</modules>
```

각 모듈은 독립적인 pom.xml 보유

---

11. 명령어 요약

명령어	설명
`mvn clean`	빌드 디렉토리 삭제
`mvn compile`	자바 컴파일
`mvn test`	테스트 수행
`mvn package`	JAR/WAR 생성
`mvn spring-boot:run`	Spring Boot 애플리케이션 실행
`mvn dependency:tree`	의존성 트리 확인
`mvn versions:display-dependency-updates`	라이브러리 업데이트 확인

---

12. Maven 저장소 구조

- **로컬 저장소**: `~/.m2/repository`
- **중앙 저장소**: `https://repo.maven.apache.org/maven2`
- 사설 저장소 연동: Nexus, Artifactory 등

---

13. Maven과 Spring Boot 통합 요점

통합 기능	설명
Parent POM 상속	공통 설정 재사용 (`spring-boot-starter-parent`)
의존성 최소화	`spring-boot-starter-*` 자동 구성
내장 플러그인 지원	실행 가능한 JAR 생성 자동화 (`spring-boot-maven-plugin`)
프로파일 설정	application-dev.yml 등과 연동 가능

---

14. 요약 정리

항목	내용
사용 목적	빌드, 의존성, 테스트, 배포 자동화
설정 방식	선언형 XML (`pom.xml`)
핵심 구성	dependencies, plugins, profiles, properties
강점	안정성, 표준화, 커뮤니티 지원

항목	내용
약점	유연성 부족, XML 구조의 가독성 한계
대안	Gradle (스크립트 기반 유연 빌드)

- # Gradle 설정

1. Gradle이란?

Gradle은 JVM 언어(주로 Java, Kotlin, Groovy 등)를 기반으로 한 **모던 빌드 자동화 도구**로, **스크립트 기반의 선언형 + 명령형 하이브리드 문법**을 사용한다.
기존 Maven의 단점을 보완하며, 빠른 빌드 성능과 **높은 유연성**을 제공한다.

- Groovy 또는 Kotlin DSL 기반
- 의존성 자동 다운로드
- 빌드 캐시, 병렬 빌드, 증분 빌드
- 안드로이드 공식 빌드 도구

---

2. 기본 파일 구조

```
project/
├── build.gradle          ← 핵심 빌드 설정 (Groovy 또는 Kotlin)
├── settings.gradle       ← 루트 프로젝트 및 서브 모듈 정의
├── gradle.properties     ← 전역 속성 설정
└── src/
    ├── main/java         ← Java 코드
    └── main/resources    ← 리소스 (설정, 템플릿 등)
```

---

3. `build.gradle` 예제 (Spring Boot 프로젝트 기준)

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.2.0'
    id 'io.spring.dependency-management' version '1.1.4'
}

group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '17'

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
```

```
    runtimeOnly 'com.h2database:h2'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

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

---

4. 주요 블록 설명

블록	설명
`plugins`	사용할 Gradle 플러그인 선언
`group`, `version`	프로젝트 메타데이터 (Java 패키지, 버전)
`sourceCompatibility`	Java 컴파일 버전
`repositories`	의존성 다운로드 위치 (mavenCentral, jcenter 등)
`dependencies`	필요한 라이브러리 목록
`tasks`	사용자 정의 태스크 또는 내장 태스크 구성

---

5. 의존성 범위

키워드	설명
`implementation`	컴파일 및 런타임 사용, 외부에 노출되지 않음
`api`	(라이브러리 개발 시) 외부에 노출됨
`compileOnly`	컴파일만 필요 (ex: lombok)
`runtimeOnly`	런타임 시 필요 (ex: DB Driver)
`testImplementation`	테스트 전용

---

6. settings.gradle

```
rootProject.name = 'demo'
```

멀티모듈 구성 시 다음처럼 작성:

```
rootProject.name = 'parent-project'
include 'core'
include 'api'
```

---

7. Gradle 실행 명령어

명령어	설명
`gradle build`	전체 빌드
`gradle bootRun`	Spring Boot 애플리케이션 실행
`gradle test`	테스트 수행
`gradle clean`	빌드 결과물 삭제
`./gradlew`	로컬 wrapper로 실행 (버전 고정)

---

8. Wrapper 사용 (`gradlew`)

```
gradle wrapper --gradle-version 8.5
```

→ 프로젝트에 `gradlew`, `gradlew.bat`, `gradle/wrapper/gradle-wrapper.properties` 생성됨
→ 팀원 간 **빌드 도구 버전 통일** 가능

---

9. 빌드 성능 최적화

기능	설정 방법
빌드 캐시	`--build-cache` 또는 `gradle.properties`에서 설정
병렬 빌드	`--parallel`
증분 빌드	입력 변경 감지 자동 적용
데몬 활성화	`org.gradle.daemon=true`

---

10. Kotlin DSL 버전 (선택적)

파일명: `build.gradle.kts`

```
plugins {
    java
    id("org.springframework.boot") version "3.2.0"
    id("io.spring.dependency-management") version "1.1.4"
}

group = "com.example"
version = "0.0.1-SNAPSHOT"
java.sourceCompatibility = JavaVersion.VERSION_17

repositories {
```

```
    mavenCentral()
}

dependencies {
    implementation("org.springframework.boot:spring-boot-starter-web")
    testImplementation("org.springframework.boot:spring-boot-starter-test")
}
```

---

11. Maven과의 차이점

항목	Maven	Gradle
설정 방식	XML 선언형	Groovy/Kotlin 스크립트 기반
유연성	낮음 (고정된 구조)	높음 (DSL 기반 자유로운 설정)
빌드 속도	느림	빠름 (캐시, 병렬 빌드 등)
커스터마이징	복잡함	쉬움 (조건부 빌드 등 가능)
의존성 버전 통제	dependencyManagement	platform 또는 BOM 사용 가능
멀티모듈	복잡함	자연스럽고 유연함

---

12. 추가 설정 예제

12.1 BOM 적용 (Spring Cloud)

```
dependencyManagement {
    imports {
        mavenBom "org.springframework.cloud:spring-cloud-dependencies:2023.0.0"
    }
}
```

12.2 프로파일 기반 실행

```
./gradlew bootRun --args='--spring.profiles.active=dev'
```

---

13. gradle.properties (공통 속성)

```
org.gradle.jvmargs=-Xmx2048m
org.gradle.daemon=true
org.gradle.parallel=true
org.gradle.caching=true
```

---

14. 멀티모듈 프로젝트 예시

settings.gradle

```
rootProject.name = 'multi-project'
include 'core', 'web', 'batch'
```

각 서브모듈의 build.gradle

```
dependencies {
    implementation project(":core")
}
```

---

15. 요약

항목	내용
사용 목적	자바 프로젝트의 빌드 자동화
주요 파일	`build.gradle`, `settings.gradle`, `gradle.properties`
장점	빠른 빌드, 유연한 설정, 멀티모듈 최적화
단점	DSL 학습 필요
Maven 대비	성능 우수, 확장성 높음, 설정 자유도 높음

개발 환경 구성

- # IntelliJ, Eclipse, VS Code

Spring 개발용 IDE 비교: IntelliJ vs Eclipse vs VS Code

항목	IntelliJ IDEA	Eclipse IDE	Visual Studio Code (VS Code)
제작사	JetBrains	Eclipse Foundation	Microsoft
언어 지원	Java, Kotlin, Groovy, Scala 등	Java 중심	모든 언어 (Java 지원은 확장 설치 필요)
Spring Boot 통합	최상급: 자동 완성, 구조 탐색, DevTools	Spring Tools (STS) 통해 통합 지원	확장 프로그램(Spring Boot Extension Pack)
Spring Initializr 지원	내장 GUI 지원	STS에서 GUI 지원	커맨드 팔레트 기반 지원
빌드 도구 (Maven/Gradle)	완벽 지원 (자동 동기화)	우수 (수동 리프레시 필요)	터미널 기반 사용 권장

항목	IntelliJ IDEA	Eclipse IDE	Visual Studio Code (VS Code)
코드 자동완성/힌트	매우 강력	중간	언어 서버 기반 (중간~우수)
디버깅 기능	강력한 UI 기반 디버거	Eclipse의 고전 디버거	상대적으로 약한 UI
단축키 효율성	고급 단축키 다수	전통적 단축키 체계	사용자 지정 편함
실무 프로젝트 적합성	대규모, 기업 환경에 적합	공공기관, 기존 Java 시스템에 익숙함	경량 환경, 개인 학습, 빠른 실행에 적합
UI/UX 반응성	무거우나 정교함	복잡하고 다소 느림	가볍고 빠름
커뮤니티/문서 지원	많음 (JetBrains 공식 및 StackOverflow)	많음 (과거 자료 위주)	많음 (VS Code 자체 + Java 확장 문서)
오픈소스 여부	❌ (Community는 무료, Ultimate 유료)	✅ 전면 오픈소스	✅ 전면 오픈소스
Spring Security 구성	자동 import, 구조 탐색 매우 우수	수동 설정 많음	Extension으로 일부 자동화
Docker 연동	내장 도구 또는 플러그인	외부 플러그인 필요	내장 터미널 기반 구성

---

1. IntelliJ IDEA

1.1 버전

- Community (무료): 기본 Java, Maven, Gradle, Git, Spring Boot 기본 기능
- Ultimate (유료): Spring MVC, Spring Security, JPA, Thymeleaf 완전 지원

1.2 장점

- **Spring Boot와의 통합 최상**
- `Ctrl + Click` 또는 `Cmd + B`로 빈 연결 추적, 선언 위치 빠르게 이동
- 자동으로 `application.yml` 설정 값 추론
- 실시간 코드 변경 → DevTools 자동 적용

1.3 단점

- 무거움 (많은 메모리 사용)
- Ultimate 유료 라이선스 필요 (약 6~9만원/년)

---

2. Eclipse IDE + STS(Spring Tools)

2.1 장점

- 오픈소스 완전 무료
- **Java EE + Spring 개발에 전통적으로 강함**
- 공공기관/정부 프로젝트에서 많이 사용됨
- Spring Tools 4 플러그인으로 Boot, Initializr, DevTools 지원

2.2 단점

- 복잡한 메뉴 구조
- 신규 개발자 입장에서 접근성 낮음
- Maven/Gradle 싱크가 수동인 경우 잦음
- 실행/디버깅 중 잦은 프로젝트 재빌드 필요

---

3. Visual Studio Code

3.1 주요 확장팩

- Spring Boot Extension Pack (by Microsoft + Pivotal)
  - Spring Boot Tools
  - Spring Initializr
  - Spring Boot Dashboard
  - Java IntelliSense
  - Spring Boot Actuator UI

3.2 장점

- **가볍고 빠름**
- JSON, YAML 자동 완성 가능
- 실시간 실행 결과 확인 용이
- 개발 외 REST, 프론트, Docker 등도 함께 작업 가능

3.3 단점

- Java 언어 서버가 부족할 수 있음 (JDT 기반)
- 대규모 프로젝트나 다중 모듈에는 불리함
- 코드 탐색, 계층 추적 기능은 IntellIJ보다 약함

---

4. 선택 가이드

개발 상황	추천 IDE
Spring Boot 학습, 개인 프로젝트	VS Code or IntelliJ Community
실무 업무, 대규모 프로젝트	IntelliJ Ultimate
기존 Java EE 프로젝트 유지보수	Eclipse + STS
메모리 제한 환경, 클라우드 IDE	VS Code
빠른 프로토타입, 경량 구조	VS Code
복잡한 Bean 구성/관계도 분석 필요	IntelliJ Ultimate

---

5. 공통 기능 비교

기능	IntelliJ	Eclipse	VS Code
Spring Initializr 지원	GUI 내장	플러그인 기반	명령팔레트 지원
YAML 자동완성 (`application.yml`)	최상급	중간	중간
Bean 탐색	클래스 그래프 제공	수동 탐색	거의 없음
DevTools 연동	자동 적용	수동 재시작	기본 지원
Git 통합	매우 우수	우수	매우 우수
Docker 연동	내장 또는 플러그인	외부 플러그인 필요	터미널 기반

---

6. 결론 및 추천

- **IntelliJ IDEA Ultimate**은 Spring 개발에 있어 가장 강력하고 완벽한 도구이며, 기업/팀 프로젝트에 적합하다.
- **Eclipse**는 레거시 Java 기반 시스템이나 정부 기관, 교육용 프로젝트에 여전히 많이 사용된다.
- **VS Code**는 클라우드 네이티브/경량 개발에 적합하며, **초보자 학습**이나 프론트엔드와의 통합 작업에 유리하다.

Lombok 플러그인 설치

Lombok이란?

Lombok은 Java 클래스에서 **반복적으로 작성해야 하는 보일러플레이트 코드**(getter/setter, constructor, builder 등)를 **어노테이션 기반으로 자동 생성**해주는 라이브러리이다.

- 예: `@Getter`, `@Setter`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor`, `@ToString`, `@EqualsAndHashCode`

---

1. Maven / Gradle 의존성 추가

Maven

```xml
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.30</version>
  <scope>provided</scope>
</dependency>
```

Gradle

```groovy
dependencies {
    compileOnly 'org.projectlombok:lombok:1.18.30'
    annotationProcessor 'org.projectlombok:lombok:1.18.30'
    testCompileOnly 'org.projectlombok:lombok:1.18.30'
    testAnnotationProcessor 'org.projectlombok:lombok:1.18.30'
}
```

---

2. IntelliJ IDEA에서 Lombok 설정

2.1 플러그인 설치

1. 메뉴: `File → Settings (Preferences) → Plugins`
2. 검색창에 "Lombok" 입력
3. `Lombok` 플러그인 설치 → IntelliJ 재시작

2.2 애너테이션 프로세서 활성화

1. 메뉴: `File → Settings → Build, Execution, Deployment → Compiler → Annotation Processors`
2. 오른쪽 체크박스:
   - [✔] Enable annotation processing
3. 적용 및 OK 클릭

2.3 프로젝트 재빌드

- 메뉴: `Build → Rebuild Project`

---

3. Eclipse(STS 포함)에서 Lombok 설정

3.1 Lombok JAR 다운로드

- https://projectlombok.org/download 에서 `lombok.jar` 다운로드

3.2 설치

```
java -jar lombok.jar
```

- 자동으로 Eclipse 설치 경로를 탐색함
- `eclipse.ini` 파일에 `-javaagent:lombok.jar` 항목 추가됨
- 수동으로 추가할 경우:

```
-javaagent:/path/to/lombok.jar
```

3.3 재시작 및 확인

- Eclipse 재시작 후 `@Getter`, `@Setter` 자동완성 확인
- 코드 탐색 가능 (`Ctrl + Click`)

---

4. VS Code에서 Lombok 설정

4.1 확장팩 설치

- Java Extension Pack 설치 (필수)
- 추천 확장:
  - `Lombok Annotations Support for VS Code`

4.2 `settings.json` 확인

```
"java.configuration.runtimes": [
  {
    "name": "JavaSE-17",
    "path": "/usr/lib/jvm/java-17-openjdk",
    "default": true
  }
],
"java.jdt.ls.vmargs": "-javaagent:${workspaceFolder}/lib/lombok.jar"
```

> `lombok.jar` 파일을 프로젝트에 포함시켜 `-javaagent`로 명시해야 컴파일 오류 없음

---

5. 테스트 예제

```
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class User {
    private String name;
    private int age;
}
```

사용 예

```
User user = new User();
user.setName("Alice");
System.out.println(user.getName()); // "Alice"
```

---

6. 주의사항 및 팁

항목	설명
annotationProcessor 미설정 시	컴파일은 되나 IDE에서 오류 표시
IntelliJ 재시작 후 적용 안될 경우	`Invalidate Caches / Restart` 실행
의존성 중복 확인	Spring Boot Starters가 자동 추가할 수 있음 (`provided` 유지)
Kotlin 프로젝트에서는 사용 금지	Kotlin은 Lombok 대신 `data class` 사용 권장

---

7. 요약

IDE	설치 방법 요약
IntelliJ	플러그인 설치 + Annotation Processing 활성화
Eclipse (STS)	lombok.jar 실행 또는 eclipse.ini 수정
VS Code	확장 설치 + `-javaagent` 직접 지정 필요

Spring Boot 버전 전략

- ## SNAPSHOT, M, RC, RELEASE

소프트웨어 라이브러리(특히 Java 생태계에서의 Maven/Gradle 프로젝트)는 **기능 안정성, 릴리스 시점, 변경 가능성**에 따라 아래와 같은 버전 명명을 사용한다.

---

1. RELEASE (Stable Release)

정의

- **정식 출시 버전**으로, 기능이 확정되고 테스트가 완료된 안정된 상태
- 외부 서비스 또는 실제 제품에 사용할 수 있는 **운영 환경용 버전**

예시

```
3.2.0
2.7.13
1.18.30
```

특징

- 절대 변경되지 않음 (immutable)
- 모든 의존성 안정화 완료
- 보안 및 버그 패치 외에는 변경 없음

---

2. RC (Release Candidate)

정의

- **출시 후보 버전**, 최종 RELEASE 직전의 안정성 테스트 목적 버전
- 심각한 버그가 발견되지 않으면 그대로 RELEASE로 승격됨

예시

```
3.2.0-RC1
3.2.0-RC2
```

특징

- 릴리스에 매우 가까움
- 사용자 피드백 수집 및 최종 회귀 테스트용
- 버그가 없으면 그대로 버전 넘버만 바뀌어 `3.2.0`이 됨

---

3. M (Milestone)

정의

- **중간 이정표 버전**, 개발 중이지만 주요 기능이 구현된 시점의 공개 버전
- 실험적 요소 또는 기능 설계에 대한 피드백 수집 목적

예시

```
3.2.0-M1
3.2.0-M2
```

특징

- 전체 릴리스 사이클 중 초기 또는 중기 단계
- 내부 구조가 바뀔 가능성 있음
- 실무에서 사용 금지 (불안정함)

---

4. SNAPSHOT

정의

- **개발 중인 버전**으로, 빌드할 때마다 결과물이 바뀌는 **비결정적 버전**
- 최신 코드 반영 목적이며, 계속 업데이트됨

예시

```
3.2.0-SNAPSHOT
```

특징

- Maven/Gradle은 SNAPSHOT을 항상 **최신 버전으로 다운로드**
- 시간이나 Git 커밋 기준으로 빌드됨
- 캐시를 우회하거나 강제로 최신 버전을 받는 용도

---

5. Maven/Gradle 저장소 동작 차이

항목	RELEASE	SNAPSHOT
저장소 위치	`/releases`	`/snapshots`
버전 변경	없음 (불변)	계속 갱신됨
캐시 처리	영구 저장	자동 갱신
다운로드 시	한 번만 받음	매 빌드마다 갱신 시도
빌드 안정성	매우 높음	낮음 (빌드 실패 위험)

---

6. Spring Boot 릴리스 전략과 예

Spring 팀은 Spring Release Train 전략을 사용하며, 다음 순서대로 진행됨:

```
3.2.0-M1 → 3.2.0-M2 → 3.2.0-RC1 → 3.2.0 → 3.2.1 → 3.3.0-SNAPSHOT
```

버전	의미
`3.2.0-SNAPSHOT`	개발 중, 매일 변경될 수 있음
`3.2.0-M1`	초기 마일스톤 공개
`3.2.0-RC1`	릴리스 직전 최종 테스트
`3.2.0`	정식 출시 버전
`3.2.1`	버그 픽스 (patch release)

---

7. 실무 권장 사항

상황	권장 버전
운영 환경 (Production)	RELEASE
팀 내부 테스트	RC, M 가능
빠른 실험 및 확인	SNAPSHOT
공공 API, 기업 시스템	RELEASE만 사용
논문, 발표용 데모	M 또는 RC 가능 (단, 명시해야 함)

---

8. 요약 비교표

버전 유형	설명	변경 가능성	실무 사용
`SNAPSHOT`	개발 중인 최신 코드	높음	❌ 위험
`M1`, `M2`	중간 이정표 (기능 단위 공개)	중간	⚠️ 실험 용도
`RC1`, `RC2`	최종 테스트 후보	낮음	⚠️ 신중히 사용
`RELEASE`	정식 출시 버전	없음	✅ 운영 환경 전용

디렉터리 구조 설명

- `src/main/java`

`src/main/java`는 **Maven**이나 **Gradle** 기반 Java 프로젝트에서 **표준 프로젝트 디렉터리 구조**의 핵심 경로이며, **실제 애플리케이션의 자바 소스 코드**가 위치하는 디렉터리다. Spring, Spring Boot, 일반 Java 프로젝트 모두 이 구조를 따른다.

1. `src/main/java`란?

- `src`: "source"의 약자, 소스코드 저장 디렉터리
- `main`: 메인 애플리케이션 코드 (운영 대상 코드)
- `java`: 실제 Java 클래스가 포함되는 폴더
  (패키지 구조에 따라 하위 디렉터리 생성됨)

```
src/
 └── main/
     └── java/
         └── com/
             └── example/
                 └── demo/
                     ├── DemoApplication.java
                     ├── controller/
                     ├── service/
                     ├── repository/
                     └── domain/
```

---

2. 주요 역할 및 위치 구분

디렉터리	역할 설명
`src/main/java`	운영 코드(애플리케이션 동작에 필요한 클래스)
`src/main/resources`	설정 파일, 템플릿, 정적 파일 등 (ex: `application.yml`, HTML 등)
`src/test/java`	테스트 코드 (JUnit, Mockito 등)
`src/test/resources`	테스트 설정, 테스트용 리소스

---

3. 패키지 구조 예시 (`com.example.demo`)

```
src/main/java/
└── com/example/demo/
    ├── DemoApplication.java         # main() 포함, @SpringBootApplication
    ├── controller/                  # @RestController 클래스
    ├── service/                     # @Service 클래스
    ├── repository/                  # @Repository 또는 Spring Data JPA 인터페이스
    └── domain/                      # Entity 클래스 (@Entity)
```

---

4. 빌드 도구에 따른 처리 방식

Maven

`pom.xml`에서 다음이 기본값으로 적용됨:

```
<build>
  <sourceDirectory>src/main/java</sourceDirectory>
</build>
```

Gradle

`build.gradle`에서 특별한 지정 없이도:

```
sourceSets {
    main.java.srcDirs = ['src/main/java']
}
```

→ Gradle/Maven은 `src/main/java` 디렉터리 내 모든 `.java` 파일을 컴파일 대상 소스로 간주함.

---

5. 관련 디렉터리 용도 비교

경로	용도
`src/main/java`	실제 애플리케이션 로직 (컨트롤러, 서비스, DAO 등)
`src/main/resources`	설정 파일, 정적 자원, 템플릿
`src/test/java`	단위 테스트, 통합 테스트
`src/test/resources`	테스트 설정, 테스트 데이터 (YAML, SQL 등)

---

6. 자주 쓰는 패키지 예시 (Spring 기준)

패키지 이름	역할
`controller`	HTTP 요청 처리 (`@RestController`, `@GetMapping` 등)
`service`	비즈니스 로직 (`@Service`)
`repository`	DB 접근 (`@Repository`, JPA 인터페이스)
`domain`, `entity`	Entity 클래스 (`@Entity`)
`dto`	데이터 전송 객체 (`@Data`, `@Builder`)
`config`	설정 관련 클래스 (`@Configuration`, `@Enable...`)

---

7. 팁: 패키지와 디렉터리 동기화

- Java 클래스의 패키지 선언은 반드시 디렉터리 구조와 일치해야 함:

```
package com.example.demo.controller;

public class UserController {
    // ...
}
```

→ 파일 경로: `src/main/java/com/example/demo/controller/UserController.java`

---

8. 요약

항목	설명
목적	실제 실행 가능한 Java 코드 저장
위치	`src/main/java`
사용 규칙	Java 패키지 구조와 디렉터리 구조 일치 필요
빌드 대상 포함 여부	Maven/Gradle에서 기본 컴파일 경로
기타 관련 경로	`src/main/resources`, `src/test/java` 등

- `src/main/resources`

`src/main/resources`는 **Java 프로젝트에서 설정 파일, 정적 자원, 템플릿 파일, 메시지 파일 등 비코드 리소스를 저장하는 표준 경로**다.
Spring/Spring Boot 프로젝트에서 이 디렉터리는 매우 중요한 역할을 하며, 런타임 시 클래스패스에 자동 포함된다.

---

1. `src/main/resources`란?

- `src/main/java`가 자바 **소스 코드**를 담는다면,
  `src/main/resources`는 **설정 파일, 정적 파일, 템플릿, 메시지 파일 등**을 담는다.
- 컴파일 시 `classes/` **디렉터리로 복사되어 클래스패스에 포함**된다.

```
src/
 └── main/
     ├── java/                ← 소스 코드
     └── resources/           ← 설정과 리소스 파일
```

---

2. Spring에서의 기본 용도

파일/폴더	용도
`application.properties`	Spring 설정 파일
`application.yml`	YAML 형식의 설정
`static/`	정적 리소스 (HTML, CSS, JS, 이미지)
`templates/`	서버 렌더링용 템플릿 (Thymeleaf, Freemarker 등)
`messages.properties`	국제화(i18n) 메시지 번들
`banner.txt`	Spring Boot 실행 시 출력될 배너
`schema.sql`, `data.sql`	DB 초기화용 SQL 스크립트

---

3. 주요 리소스 파일 설명

3.1 `application.properties` 또는 `application.yml`

- Spring Boot의 핵심 설정 파일
- 예:

```
server.port=8081
spring.datasource.url=jdbc:mysql://localhost:3306/test
```

또는

```
server:
  port: 8081
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/test
```

3.2 `static/`

- 정적 파일 제공 경로 (`/static`, `/public`, `/resources`, `/META-INF/resources` 중 하나)
- URL 매핑: `/static/hello.html` → `http://localhost:8080/hello.html`

3.3 `templates/`

- Thymeleaf, JSP, Mustache, Freemarker 등 템플릿 엔진이 사용하는 디렉터리
- 컨트롤러에서 반환하는 뷰 이름과 매칭

```
return "index"; // templates/index.html 렌더링
```

3.4 `messages.properties`

- 국제화(i18n) 다국어 메시지 파일
- 예:

```
greeting.hello=Hello
greeting.goodbye=Goodbye
```

3.5 `banner.txt`

- 애플리케이션 실행 시 출력되는 배너 커스터마이징

```
  ____                  _       _
 / ___| _ __ ___   __ _| |_ ___| |__
 \___ \| '_ ` _ \ / _` | __/ __| '_ \
  ___) | | | | | | (_| | || (__| | | |
 |____/|_| |_| |_|\__,_|\__\___|_| |_|
```

---

4. 클래스패스 포함과 접근

4.1 클래스패스에 포함됨

- Maven/Gradle 빌드 시 `src/main/resources/**` 는 `target/classes/` 또는 `build/classes/java/main/` 로 복사됨
- Java 코드 또는 Spring에서 다음과 같이 접근 가능:

```
InputStream in = getClass().getResourceAsStream("/myfile.txt");
Resource res = new ClassPathResource("config.yaml");
```

---

5. 설정 파일 다중 분리

Spring Boot는 다음 파일들을 자동 인식함:

파일명	우선순위
`application.properties`	기본값
`application.yml`	YAML 형식
`application-{profile}.yml`	프로파일 별 분리

예:

```
# application-dev.yml
server:
  port: 8081
```

실행 시:

```
--spring.profiles.active=dev
```

---

6. SQL 초기화 자동 실행

- 파일 이름: `schema.sql`, `data.sql`
- Spring Boot는 H2, PostgreSQL 등에서 **자동 실행**
- 실행 순서: `schema.sql` → `data.sql`
- 조건: `spring.datasource.initialization-mode=always` (Spring 2.x), `spring.sql.init.*` (Spring 3.x)

---

7. 리소스 로딩 방식 요약

방식	설명
`@Value("classpath:filename")`	프로퍼티, 메시지, 텍스트 파일 주입
`new ClassPathResource("a.txt")`	리소스 파일 객체로 접근
`application.yml` → 자동 설정	서버 포트, DB, Thymeleaf 등 구성 가능
정적 파일 → `/static`	자동으로 정적 파일로 제공됨

---

8. 빌드 시 동작

Maven

- `src/main/resources` → `target/classes`

Gradle

- `src/main/resources` → `build/resources/main`

→ 따라서 `classpath:` 로 불러오는 모든 리소스는 이 경로에 존재하게 됨

---

9. 요약

항목	설명
위치	`src/main/resources`
역할	설정 파일, 정적 자원, 템플릿, SQL 등 저장
빌드시 클래스패스 포함	자동 포함 (`classes/` 디렉터리로 복사됨)
Spring Boot 사용 용도	설정 자동 적용, HTML 렌더링, 정적 파일 제공, SQL 초기화 등

- # `application.properties` / `application.yml`

다음은 **Spring Boot 프로젝트의 핵심 설정 파일**인 `application.properties`와 `application.yml`에 대한 가장 자세하고 구조적인 설명이다.
두 포맷의 차이점, 사용 규칙, Spring이 이를 처리하는 방식, 주요 설정 예제, 계층적 구조 처리, 프로파일 분리까지 전부 포함하여 설명한다.

---

1. 개요

Spring Boot는 기본적으로 `src/main/resources` 디렉토리 아래에 있는 다음 설정 파일을 자동으로 인식하여 사용한다:

- `application.properties` : 전통적인 키-값 형식의 설정 파일
- `application.yml` : YAML 형식의 계층 구조 설정 파일

둘 다 기능은 동일하며, **동일한 설정을 표현하는 방식이 다를 뿐이다**.

---

2. 기본 위치 및 적용 우선순위

위치	설명
`src/main/resources/application.*`	프로젝트 내 기본 설정
`config/` 폴더 또는 외부 경로	운영 환경에서 별도 분리 가능
`application-{profile}.yml`	프로파일별 분리 설정 (예: dev, prod)

> Spring Boot는 다음 우선순위로 설정 파일을 로딩한다 (높을수록 우선):

1. 커맨드라인 인수 (`--server.port=8085`)
2. `application-{profile}.yml` 또는 `.properties`
3. `application.yml` 또는 `.properties` (기본)

---

3. 형식 비교

설정 항목	`.properties` 형식	`.yml` 형식
포트 설정	`server.port=8081`	`server:\n  port: 8081`
DB 설정	`spring.datasource.url=...`	`spring:\n  datasource:\n    url: ...`
복잡한 구조 설정 (비추천)	`a.b[0].c=value`	`a:\n  b:\n    - c: value`
멀티라인 지원	어렵고 제한적	자연스럽게 지원
가독성	간결, 익숙함	계층 구조 표현에 유리

---

4. 주요 설정 항목 예시

4.1 서버 설정

`application.properties`

```
server.port=8081
server.servlet.context-path=/api
```

`application.yml`

```
server:
  port: 8081
  servlet:
    context-path: /api
```

4.2 데이터베이스 설정

`application.properties`

```
spring.datasource.url=jdbc:mysql://localhost:3306/test
spring.datasource.username=root
spring.datasource.password=1234
spring.jpa.hibernate.ddl-auto=update
```

`application.yml`

```yaml
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/test
    username: root
    password: 1234
  jpa:
    hibernate:
      ddl-auto: update
```

4.3 로깅 설정

```
logging.level.org.springframework.web=DEBUG
logging.file.name=logs/app.log
```

```yaml
logging:
  level:
    org.springframework.web: DEBUG
  file:
    name: logs/app.log
```

4.4 Thymeleaf 설정

```
spring.thymeleaf.cache=false
spring.thymeleaf.prefix=classpath:/templates/
```

```yaml
spring:
  thymeleaf:
    cache: false
    prefix: classpath:/templates/
```

4.5 사용자 정의 설정

```
myapp.name=TestApp
myapp.version=1.0
```

```yaml
myapp:
  name: TestApp
  version: 1.0
```

→ 자바에서 접근:

```
@Value("${myapp.name}")
private String appName;
```

또는

```
@ConfigurationProperties(prefix = "myapp")
public class MyAppProperties {
    private String name;
    private String version;
}
```

---

5. 프로파일 별 설정 (환경 분리)

기본 파일

```
# application.yml
spring:
  profiles:
    active: dev
```

환경 별 파일 (자동으로 분기됨)

```
# application-dev.yml
server:
  port: 8081

# application-prod.yml
server:
  port: 8080
```

> 실행 시:

```
--spring.profiles.active=dev
```

---

6. 외부 설정 파일 적용

```
java -jar app.jar --spring.config.location=file:./config/
```

또는 운영 시스템에서 `.properties` 파일을 별도 제공

---

7. 주의사항

항목	설명
YAML 들여쓰기	공백 2칸 또는 4칸, **탭 금지**
값에 특수 문자 포함 시	큰따옴표 `" "`로 감싸기
배열 설정	`list[0]=a`, `list[1]=b` (properties) / `list: [a, b]` (yml)

항목	설명
동일 키 중복	마지막 값이 적용됨
주석	`.properties` : `#` 또는 `!` / `.yml` : `#` 만 지원

---

8. 결론

항목	`.properties`	`.yml`
형식	평면 키-값 구조	계층형 구조 (들여쓰기 기반)
가독성	간결하고 빠름	복잡한 구조에 강함
실무 채택률	여전히 많이 사용됨	Spring 공식 문서 및 대규모 시스템에서 선호
다중 설정 관리	어렵다 (`key.prefix=...`)	자연스럽다 (`prefix:\n  key: ...`)