

# Spring Boot

시작하기

# 개발 환경

JDK 17 / Tomcat 9

H2 DataBase

STS 4.15.3

Spring Boot 3.3

# JDK/Tomcat

<https://www.oracle.com/java/technologies/downloads/#jdk17-windows>

<https://tomcat.apache.org/>

# H2

<https://www.h2database.com>



[Translate](#)

**Search:**

**Home**

[Download](#)

[Cheat Sheet](#)

**Documentation**

[Quickstart](#)

[Installation](#)

[Tutorial](#)

[Features](#)

[Security](#)

[Performance](#)

## H2 Database Engine

Welcome to H2, the Java SQL database. The main features of H2 are:

- Very fast, open source, JDBC API
- Embedded and server modes; in-memory databases
- Browser based Console application
- Small footprint: around 2.5 MB jar file size

### Download

Version 2.2.220 (2024-07-15)



[Windows Installer \(6.7 MB\)](#)



[All Platforms \(zip, 3.3 MB\)](#)

[All Downloads](#)

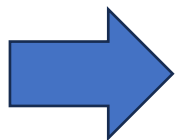
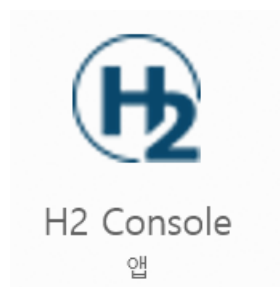
### Support

[Stack Overflow \(tag H2\)](#)

[Google Group](#)

For non-technical issues, use:  
[dbsupport at h2database.com](#)

# H2



English ▼ 설정 도구 도움말

로그인

저장한 설정: Generic H2 (Embedded) ▼

설정 이름: Generic H2 (Embedded) 저장 삭제

---

드라이버 클래스: org.h2.Driver

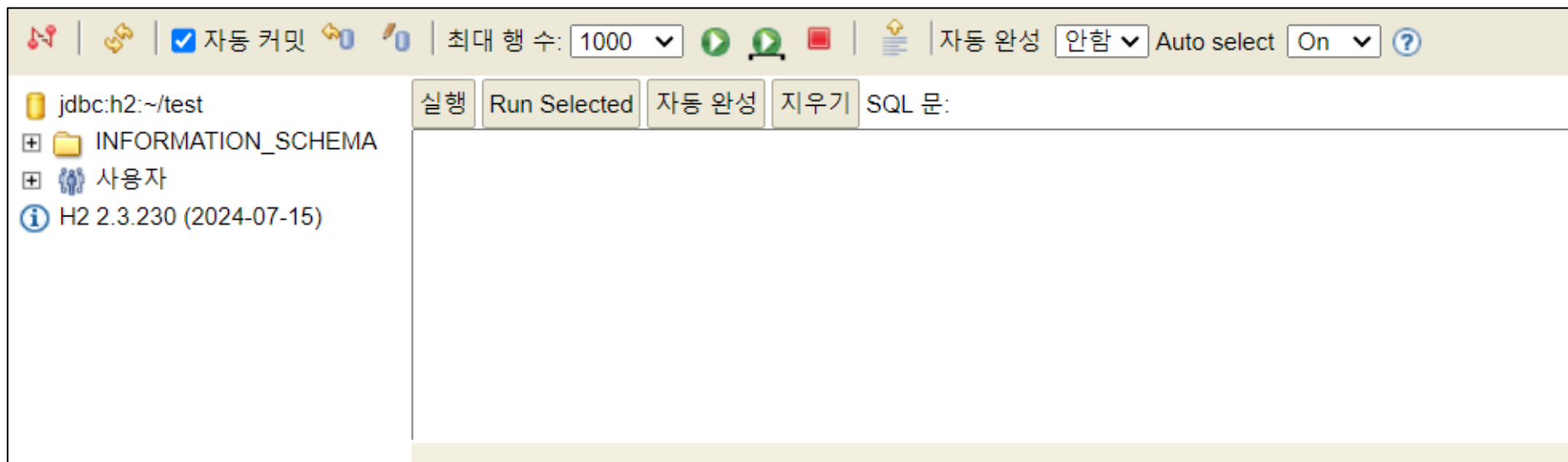
JDBC URL: jdbc:h2:~/test

사용자명: sa

비밀번호:

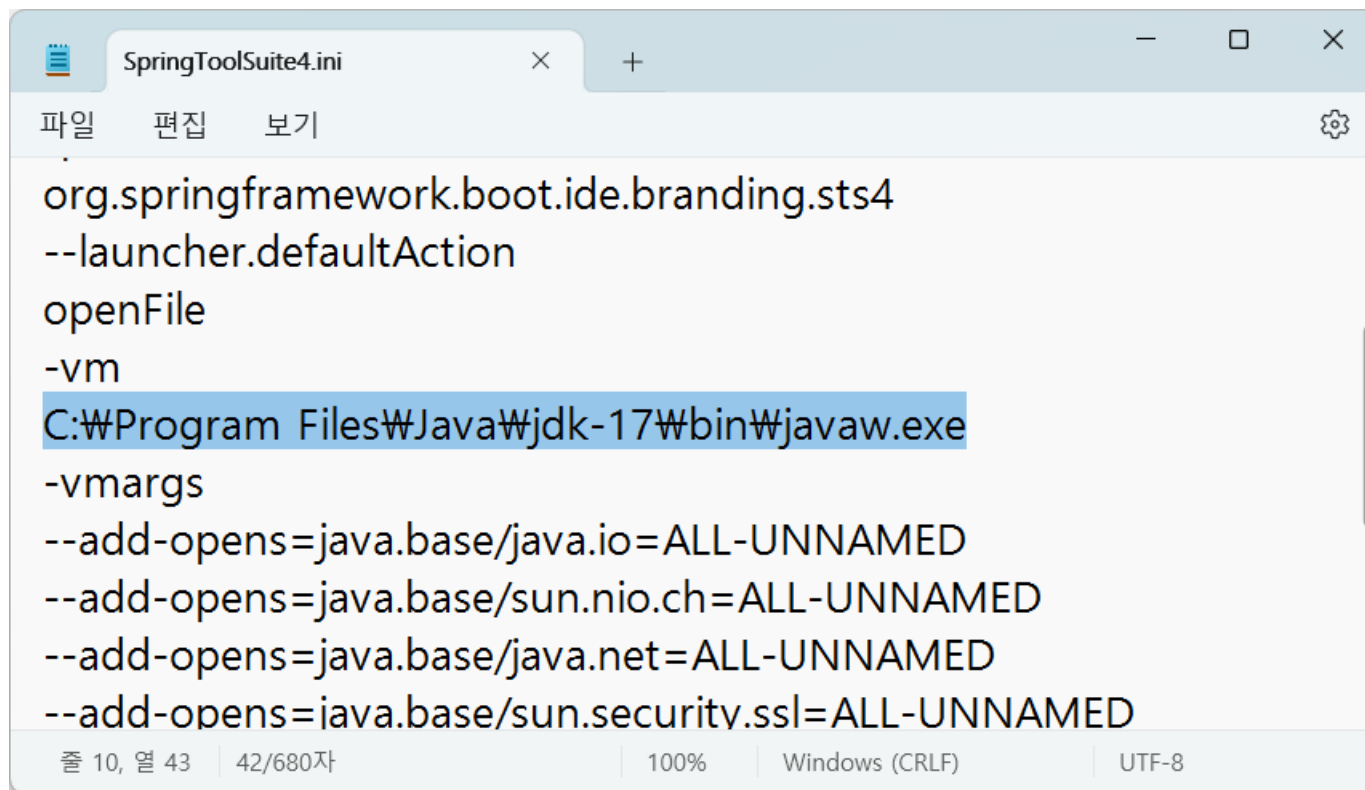
연결 연결 시험

# H2



# STS

- <https://spring.io/tools>  
다운로드
- 압축 풀기
- SpringToolSuite4.ini 파일에  
javaw.exe 경로 지정

A screenshot of a text editor window titled 'SpringToolSuite4.ini'. The window has a menu bar with '파일' (File), '편집' (Edit), and '보기' (View). The main text area contains the following content:

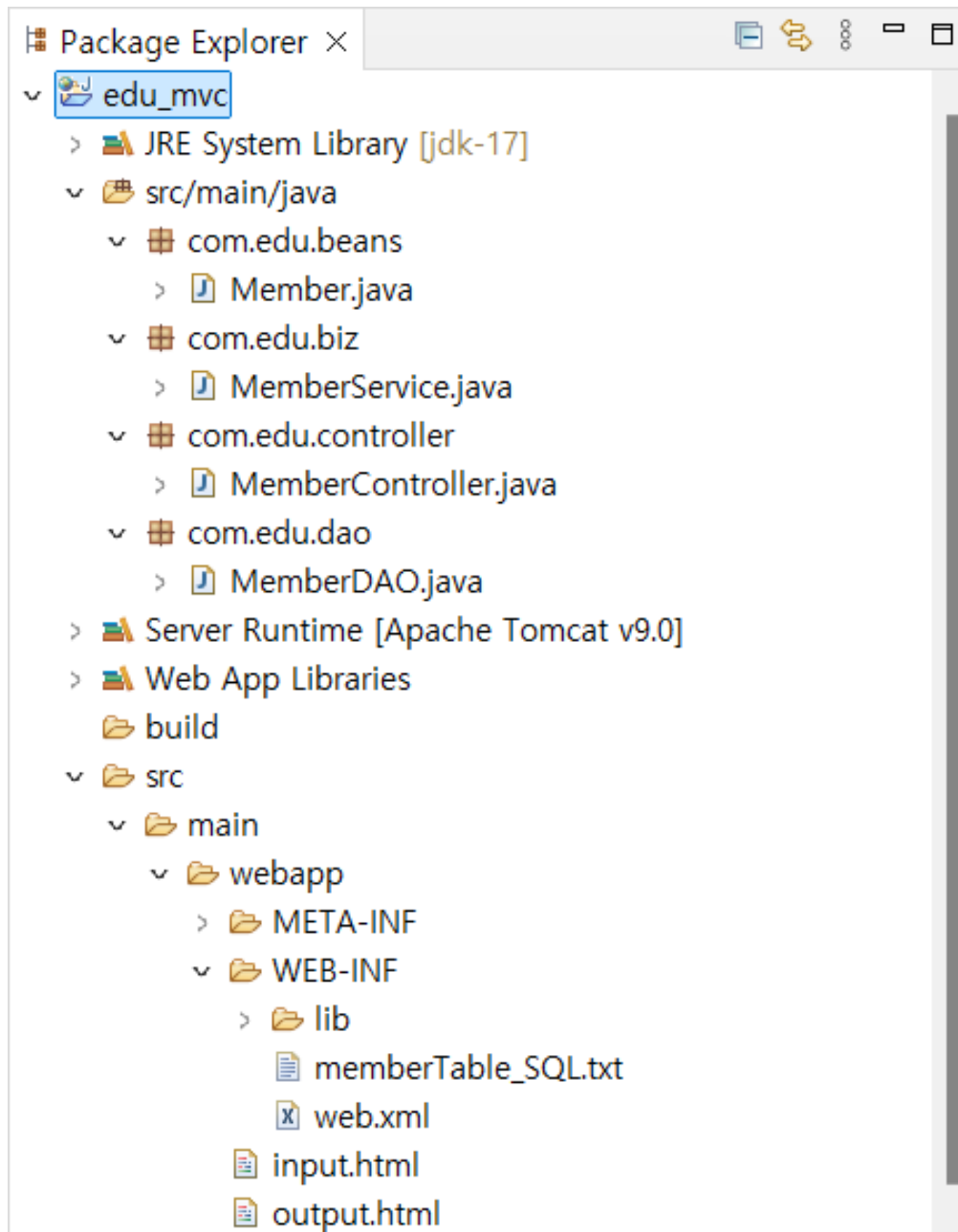
```
org.springframework.boot.ide.branding.sts4
--launcher.defaultAction
openFile
-vm
C:\Program Files\Java\jdk-17\bin\javaw.exe
-vmargs
--add-opens=java.base/java.io=ALL-UNNAMED
--add-opens=java.base/sun.nio.ch=ALL-UNNAMED
--add-opens=java.base/java.net=ALL-UNNAMED
--add-opens=java.base/sun.security.ssl=ALL-UNNAMED
```

The path 'C:\Program Files\Java\jdk-17\bin\javaw.exe' is highlighted in blue. The status bar at the bottom shows '줄 10, 열 43' (Line 10, Column 43), '42/680자' (42/680 characters), '100%' zoom, 'Windows (CRLF)' encoding, and 'UTF-8' file type.

# MVC

---

- edu\_mvc.zip





# Spring

---

- edu\_spring.zip

```

v edu_spring
  v src/main/java
    v com.edu.member
      > Member.java
      > MemberController.java
      > MemberDAO.java
      > MemberService.java
    v com.edu.test
      > AppleSpeaker.java
      > BeanFactory.java
      > LgTV.java
      > SamsungTV.java
      > SonySpeaker.java
      > Speaker.java
      > TV.java
      > TVUser.java
      > TVUser2.java
  v src/main/resources
    META-INF
    applicationContext.xml

```

```

> src/test/java
> src/test/resources
> JRE System Library [jdk-17]
> Maven Dependencies
> Server Runtime [Apache Tomcat v9.0]
v src
  v main
    v webapp
      resources
      v WEB-INF
        classes
        v spring
          v appServlet
            servlet-context.xml
            root-context.xml
        views
          output.html
          web.xml
          input.html
      test
  target
  pom.xml

```

# 생성자 주입(Constructor Injection)

```
package com.edu.test;

import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component("tv")
public class SamsungTV implements TV {
    private Speaker speaker;

    public SamsungTV() {
        System.out.println("==> SamsungTV(1) 객체 생성");
    }

    @Autowired
    public SamsungTV(Speaker speaker) {
        System.out.println("==> SamsungTV(2) 객체 생성");
        this.speaker = speaker;
    }

    public void powerOn() {
        System.out.println("SamsungTV---전원 켜다");
    }
}
```

```
    public void powerOff() {
        System.out.println("SamsungTV---전원 끈다");
    }

    public void volumeUp() {
        speaker.volumeUp();
    }

    public void volumeDown() {
        speaker.volumeDown();
    }
}
```

# 생성자 주입(Constructor Injection)

```
package com.edu.test;

import org.springframework.stereotype.Component;

@Component
public class SonySpeaker implements Speaker{

    public SonySpeaker() {
        System.out.println("==> SonySpeaker 객체 생성");
    }

    public void volumeUp() {
        System.out.println("SonySpeaker---소리 올린다");
    }

    public void volumeDown() {
        System.out.println("SonySpeaker---소리 내린다");
    }
}
```

# 생성자 주입(Constructor Injection)

```
package com.edu.test;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;

public class TVUser2 {

    public static void main(String[] args) {
        AbstractApplicationContext factory =
            new GenericXmlApplicationContext("applicationContext.xml");
        TV tv = (TV) factory.getBean("tv");
        tv.powerOn();
        tv.volumeUp();
        tv.volumeDown();
        tv.powerOff();
        factory.close();
    }
}
```

```
==> SonySpeaker 객체 생성
==> SamsungTV(2) 객체 생성
SamsungTV---전원 켜다
SonySpeaker---소리 올린다
SonySpeaker---소리 내린다
SamsungTV---전원 끈다
```

# 세터 주입(Setter Injection)

```
package com.edu.test;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component("tv")
public class SamsungTV implements TV {
    private Speaker speaker;

    public SamsungTV() {
        System.out.println("==> SamsungTV(1) 객체 생성");
    }

    @Autowired
    public void setSpeaker(Speaker speaker) {
        System.out.println("==> setSpeaker() 호출");
        this.speaker = speaker;
    }

    public void powerOn() {
        System.out.println("SamsungTV---전원 켜다");
    }
}
```

~ 생략 ~

```
==> SamsungTV(1) 객체 생성
==> SonySpeaker 객체 생성
==> setSpeaker() 호출
SamsungTV---전원 켜다
SonySpeaker---소리 올린다
SonySpeaker---소리 내린다
SamsungTV---전원 끈다
```

# 필드 주입(Field Injection)

```
@Component("tv")  
public class SamsungTV implements TV {
```

```
@Autowired  
private Speaker speaker;
```

```
public SamsungTV() {  
    System.out.println("==> SamsungTV(1) 객체 생성");  
}  
public void powerOn() {  
    System.out.println("SamsungTV---전원 켜다");  
}  
public void powerOff() {  
    System.out.println("SamsungTV---전원 끈다");  
}  
public void volumeUp() {  
    speaker.volumeUp();  
}  
public void volumeDown() {  
    speaker.volumeDown();  
}  
}
```

```
==> SamsungTV(1) 객체 생성  
==> SonySpeaker 객체 생성  
SamsungTV---전원 켜다  
SonySpeaker---소리 올린다  
SonySpeaker---소리 내린다  
SamsungTV---전원 끈다
```

# 의존성 주입(Dependency Injection) 장점

## 코드의 유연성과 재사용성 향상

의존성 분리: 객체가 직접 의존성을 생성하지 않고 외부에서 주입받기 때문에, 코드의 결합도가 낮아진다. 이는 객체를 재사용하고, 다른 환경이나 문맥에서 재배포하는데 유리하다.

구성의 유연성: 설정 파일이나 어노테이션을 통해 의존성을 주입함으로써, 다양한 구성과 설정을 유연하게 적용할 수 있다.

## 테스트 용이성

단위 테스트: 의존성을 쉽게 모의 객체(Mock Object)로 대체할 수 있어 단위 테스트를 수행하기 쉬워진다. 이를 통해 테스트의 독립성과 신뢰성을 높일 수 있다.

독립성 보장: 객체 간의 강한 결합이 없으므로, 특정 객체를 테스트할 때 해당 객체에 의존하는 다른 객체들을 모킹(mocking)하거나 스텝(stubbing)할 수 있다.

## 유지보수성 향상

코드의 가독성: DI를 통해 객체의 생성과 초기화 코드가 제거되어, 비즈니스 로직에 집중할 수 있고 코드의 가독성이 향상된다.

변경 용이성: 의존성이 외부에서 주입되기 때문에, 의존성의 변경이 필요할 때 객체 내부의 코드 수정 없이 설정만 변경하여 쉽게 적용할 수 있다.

## 중복 코드 감소

공통 기능의 중앙 관리: DI 컨테이너에서 공통으로 사용되는 객체들을 관리하고 주입하기 때문에, 중복 코드가 줄어들고 일관된 방식으로 관리할 수 있다.

# Spring Boot란

---

- Spring Boot = Spring + Boot
- Spring : 오픈 소스 웹 프레임워크
- Boot : '컴퓨터를 부팅한다' 즉 시스템을 사용가능한 상태로 만듦
- Spring Boot : 스프링 프레임워크를 사용 가능한 상태로 만들어 주는 도구

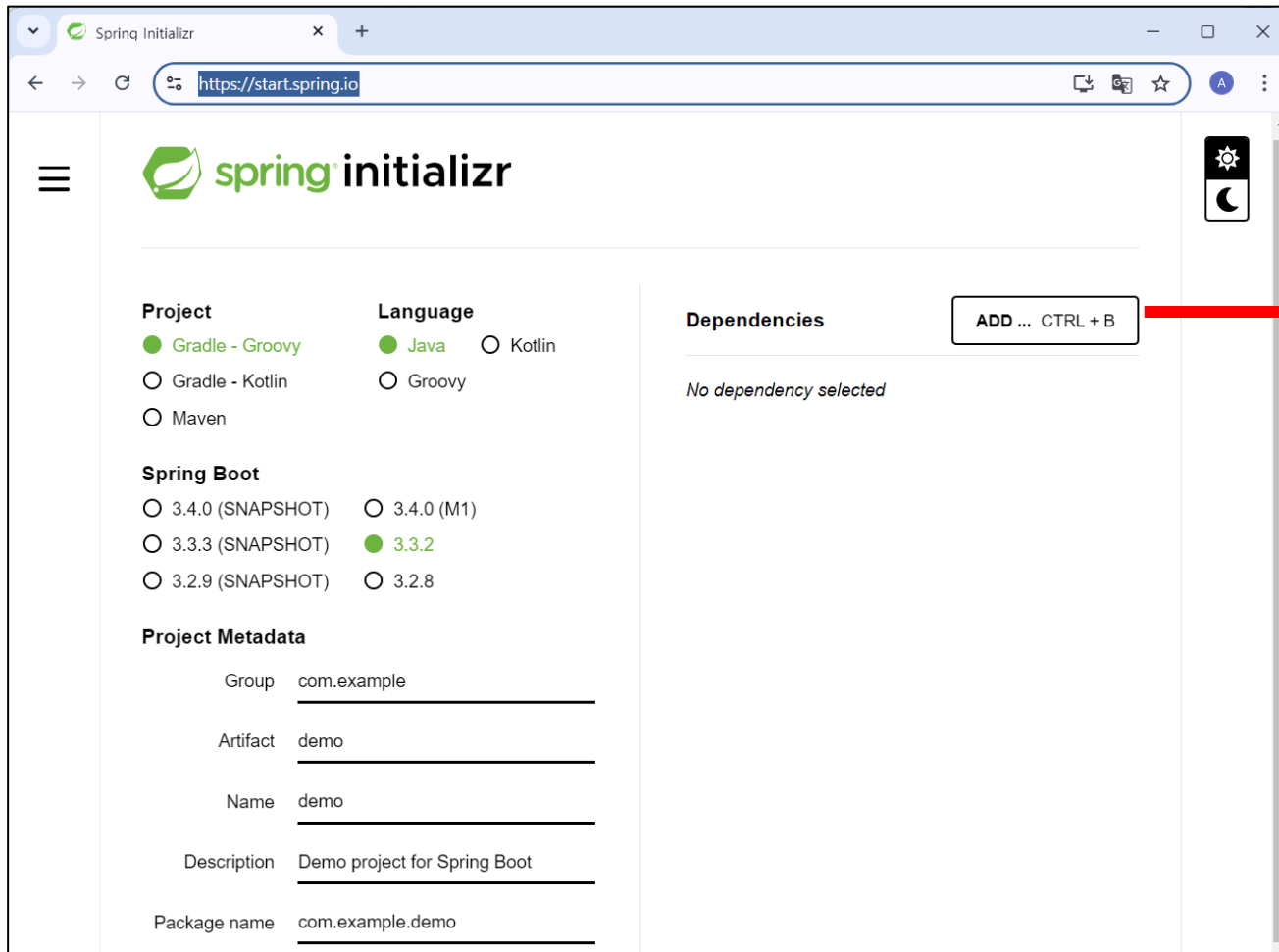


# Spring Boot 장점

---

1. 라이브러리 관리 자동화
2. 설정의 자동화
3. 라이브러리 버전 자동 관리
4. 테스트 환경과 내장 톰캣
5. 독립적으로 실행 가능한 JAR

# IntelliJ Community Edition - <https://start.spring.io/>



The screenshot shows the Spring Initializr web application. The browser address bar displays <https://start.spring.io/>. The page features a sidebar with a hamburger menu icon and the Spring Initializr logo. The main content area is divided into three sections: Project, Language, and Dependencies. The Project section includes options for Gradle (Groovy, Kotlin, Maven) and Spring Boot versions (3.4.0 (SNAPSHOT), 3.3.3 (SNAPSHOT), 3.2.9 (SNAPSHOT), 3.4.0 (M1), 3.3.2, 3.2.8). The Language section has radio buttons for Java (selected), Kotlin, and Groovy. The Dependencies section shows 'No dependency selected' and an 'ADD ... CTRL + B' button. The Project Metadata section includes fields for Group (com.example), Artifact (demo), Name (demo), Description (Demo project for Spring Boot), and Package name (com.example.demo).

**Project**

- ☒ Gradle - Groovy
- ☐ Gradle - Kotlin
- ☐ Maven

**Language**

- ☒ Java
- ☐ Kotlin
- ☐ Groovy

**Spring Boot**

- ☐ 3.4.0 (SNAPSHOT)
- ☐ 3.3.3 (SNAPSHOT)
- ☐ 3.2.9 (SNAPSHOT)
- ☒ 3.4.0 (M1)
- ☒ 3.3.2
- ☐ 3.2.8

**Project Metadata**

Group:

Artifact:

Name:

Description:

Package name:

**Dependencies**

No dependency selected

## Dependencies

### Lombok **DEVELOPER TOOLS**

Java annotation library which helps to reduce boilerplate code.

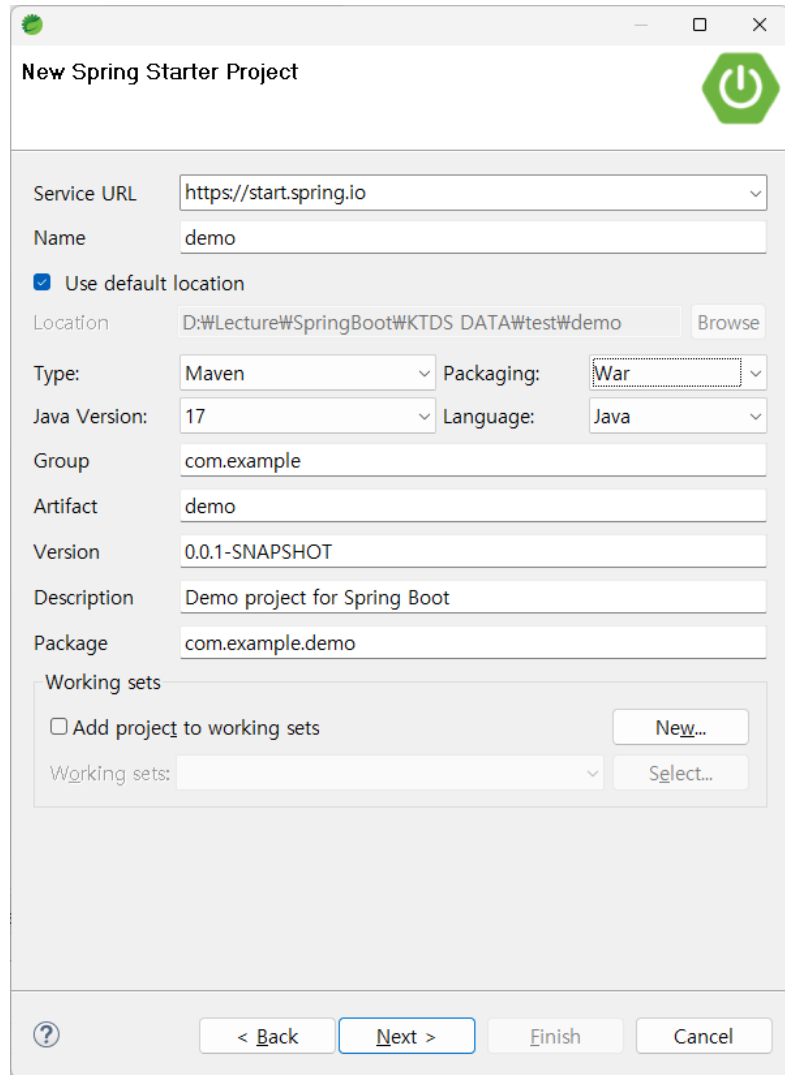
### Spring Web **WEB**

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

demo.zip 압축풀기

# SpringBoot Project

- Ctrl+N → Spring Boot → Spring Starter Project



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

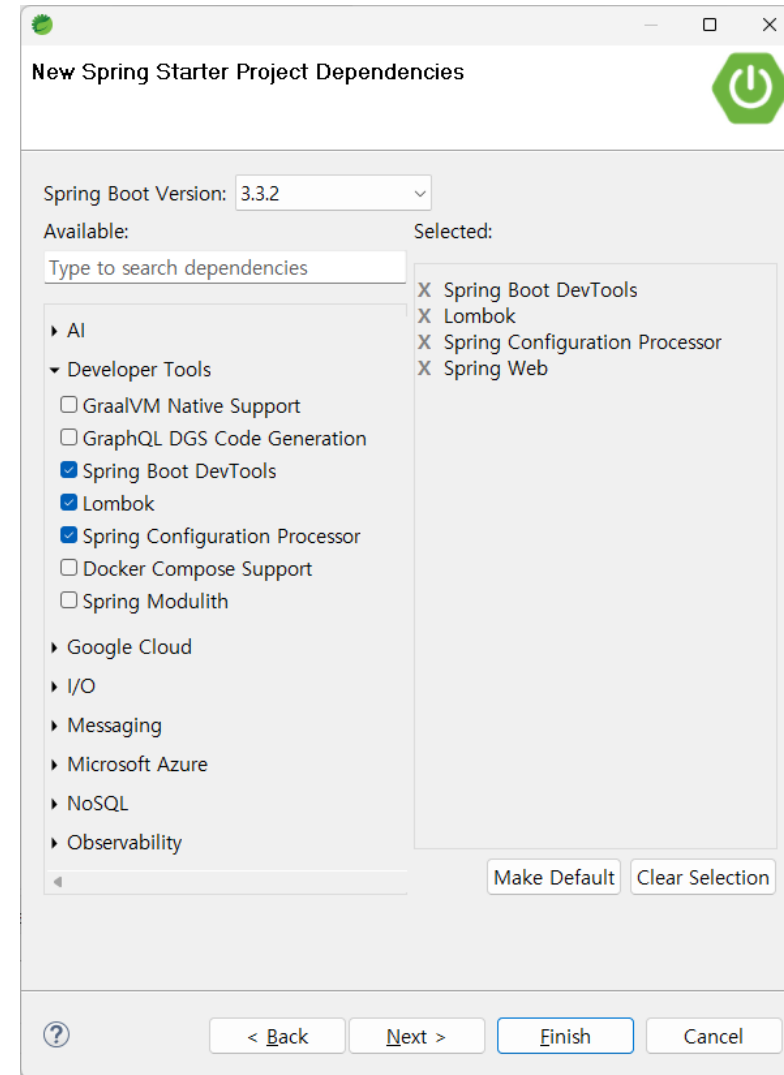
Description:

Package:

Working sets

☐ Add project to working sets

Working sets:



New Spring Starter Project Dependencies

Spring Boot Version:

Available:

Type to search dependencies

Selected:

- ☒ Spring Boot DevTools
- ☒ Lombok
- ☒ Spring Configuration Processor
- ☒ Spring Web

# Spring Boot Starter

---

- 스프링 부트 프로젝트에서 다양한 기능을 손쉽게 추가하고 설정할 수 있도록 도와주는 의존성 모음이다. 특정 기능을 구현하는데 필요한 모든 필수 라이브러리와 설정을 포함하고 있어, 개발자가 복잡한 설정을 신경 쓰지 않고도 빠르게 애플리케이션을 개발할 수 있게 해준다.
- 주요 특징
  - 편리한 종속성
    - 스프링 부트 스타터는 특정 기능을 구현하기 위해 필요한 여러 라이브러리를 하나의 의존성으로 통합한다. 개발자는 프로젝트의 **pom.xml** 또는 **build.gradle** 파일에 필요한 스타터 의존성을 추가하는 것만으로 해당 기능을 사용할 수 있다.
  - 자동 설정
    - 스프링 부트 스타터는 자동 설정 기능을 제공하여, 특정 라이브러리가 포함된 경우 자동으로 적절한 설정을 구성한다. 예를 들어, 데이터베이스 스타터를 사용하면 데이터베이스 연결 설정이 자동으로 구성된다.
  - 간편한 통합
    - 스타터 의존성을 통해 쉽게 다양한 스프링 생태계의 프로젝트와 통합할 수 있다. 이는 개발자가 필요한 기능을 쉽게 추가하고, 표준화된 방법으로 설정할 수 있게 해준다.

# 주요 Spring Boot Starter

---

- **spring-boot-starter-web**

- 웹 애플리케이션을 개발하는 데 필요한 모든 라이브러리를 포함한다. 내장 톰캣 서버, 스프링 MVC, Jackson 등을 포함하여 RESTful 웹 서비스를 쉽게 개발할 수 있게 해준다

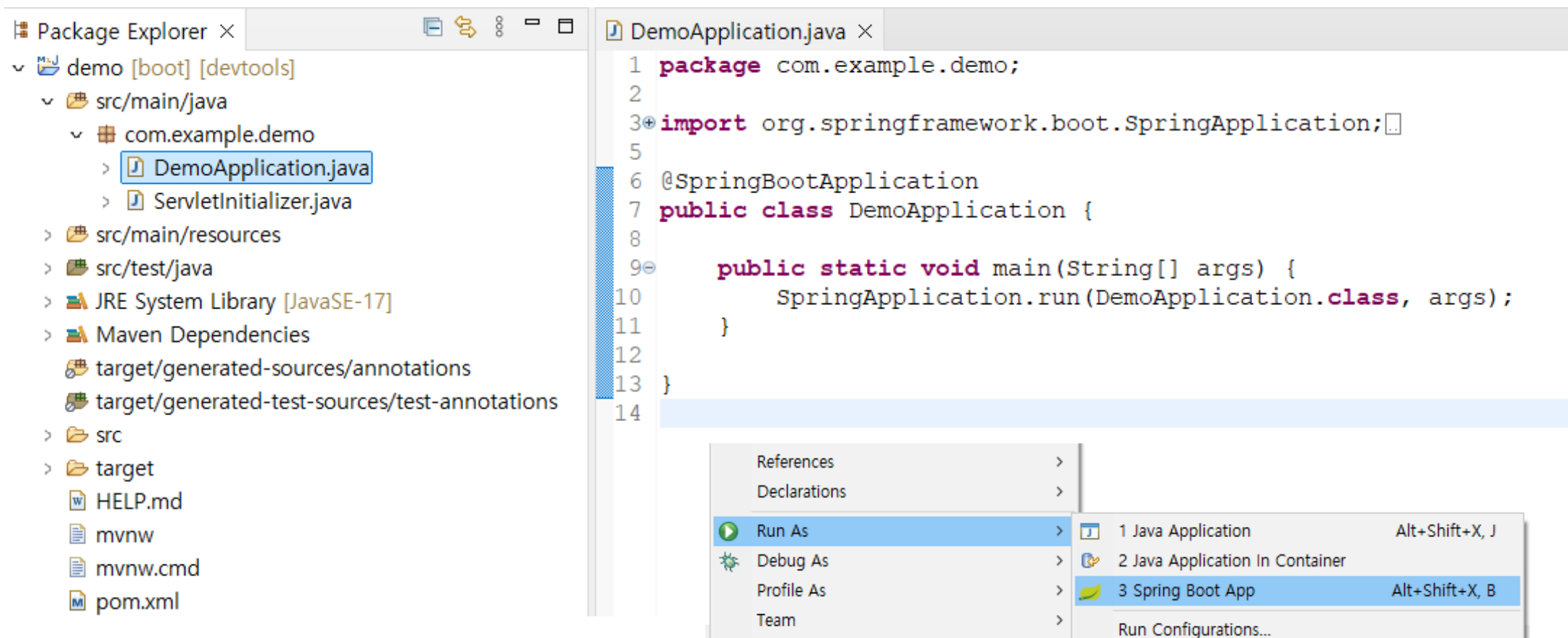
```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

- **spring-boot-starter-data-jpa**

- JPA(Java Persistence API)를 사용한 데이터베이스 액세스를 지원한다. Hibernate와 스프링 데이터 JPA를 포함하여, 데이터베이스 연동을 간단하게 설정할 수 있다.

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

# Spring Starter Project





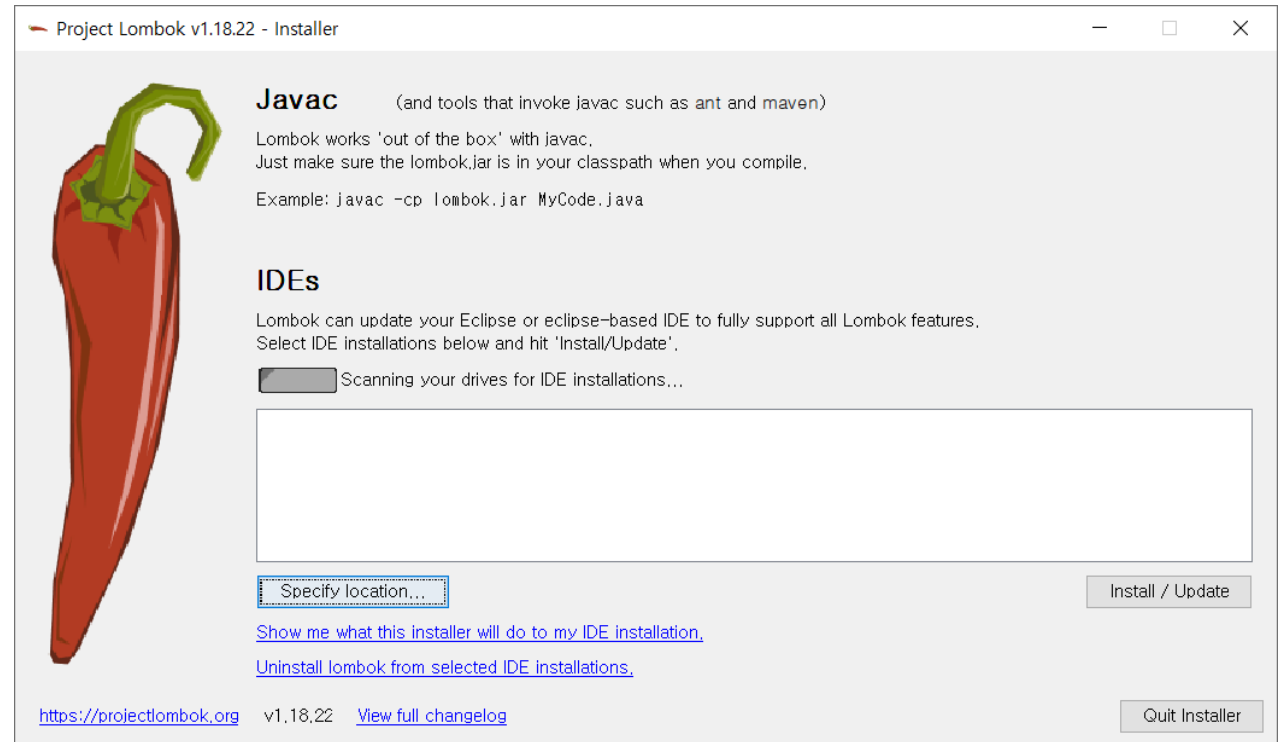
# @SpringBootApplication 구성요소

- @SpringBootConfiguration
  - 스프링 부트 설정 클래스를 지정하며 Spring Boot의 자동 설정 메커니즘과 결합되어 동작하며, 애플리케이션 컨텍스트를 설정하는 데 사용된다.
- @EnableAutoConfiguration
  - 스프링 부트의 자동 설정 기능을 활성화함. 클래스패스에 있는 다양한 설정 파일을 기반으로 애플리케이션의 설정을 자동으로 구성. 예를 들어 DB 의존성이 포함된 경우 자동으로 datasource 관련 설정이 이루어진다.
- @ComponentScan
  - @Component, @Service, @Repository, @Controller 등으로 마킹된 빈을 자동으로 검색하고 등록한다



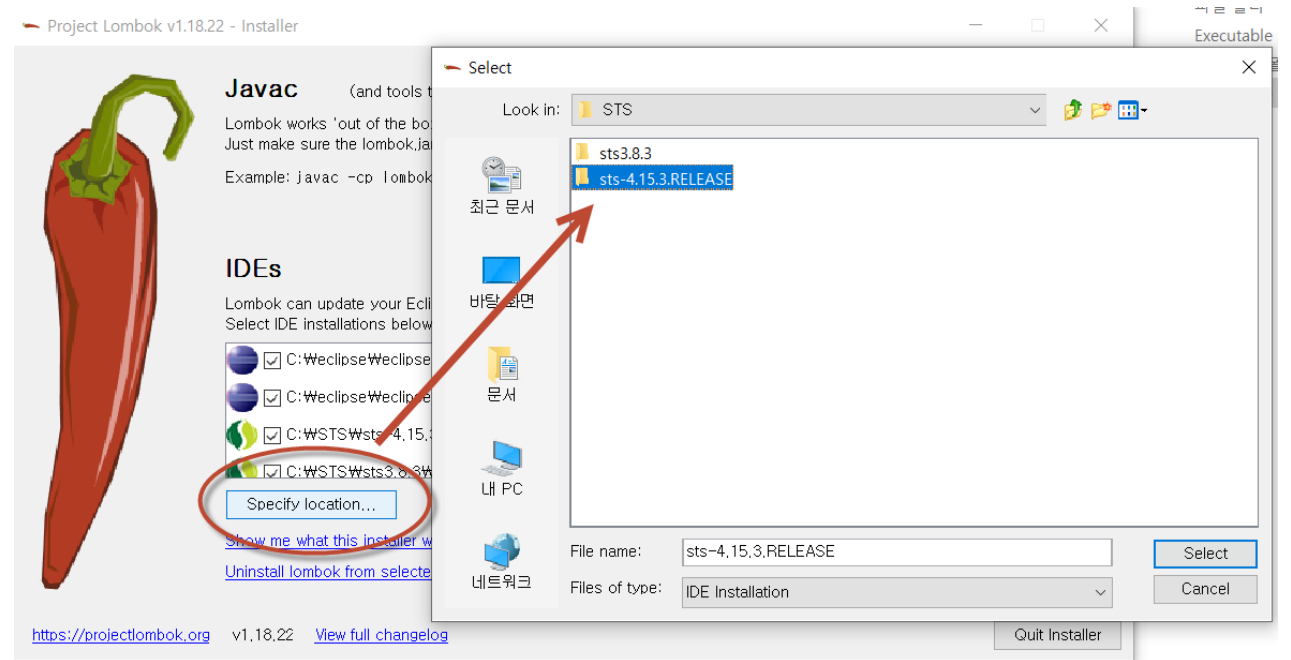
# Lombok

lombok.jar 더블클릭 →



# Lombok

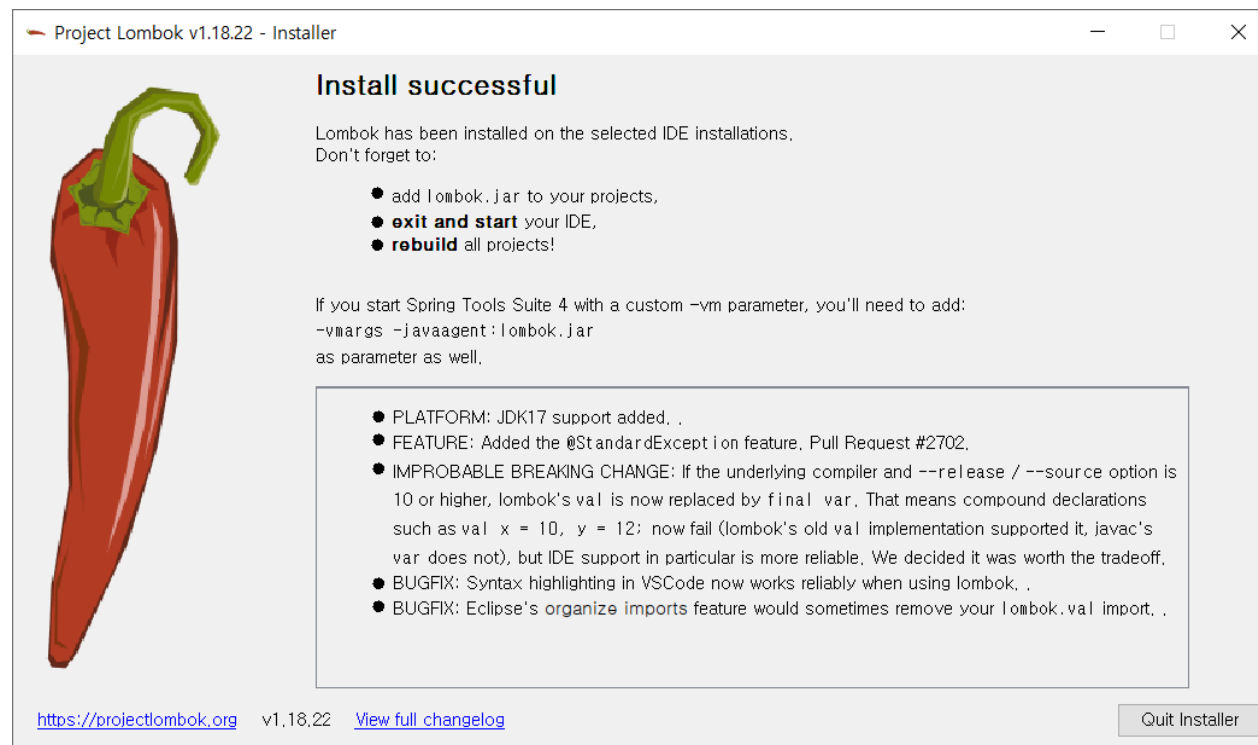
이클립스 위치 선택 →  
[Install/Update] 선택



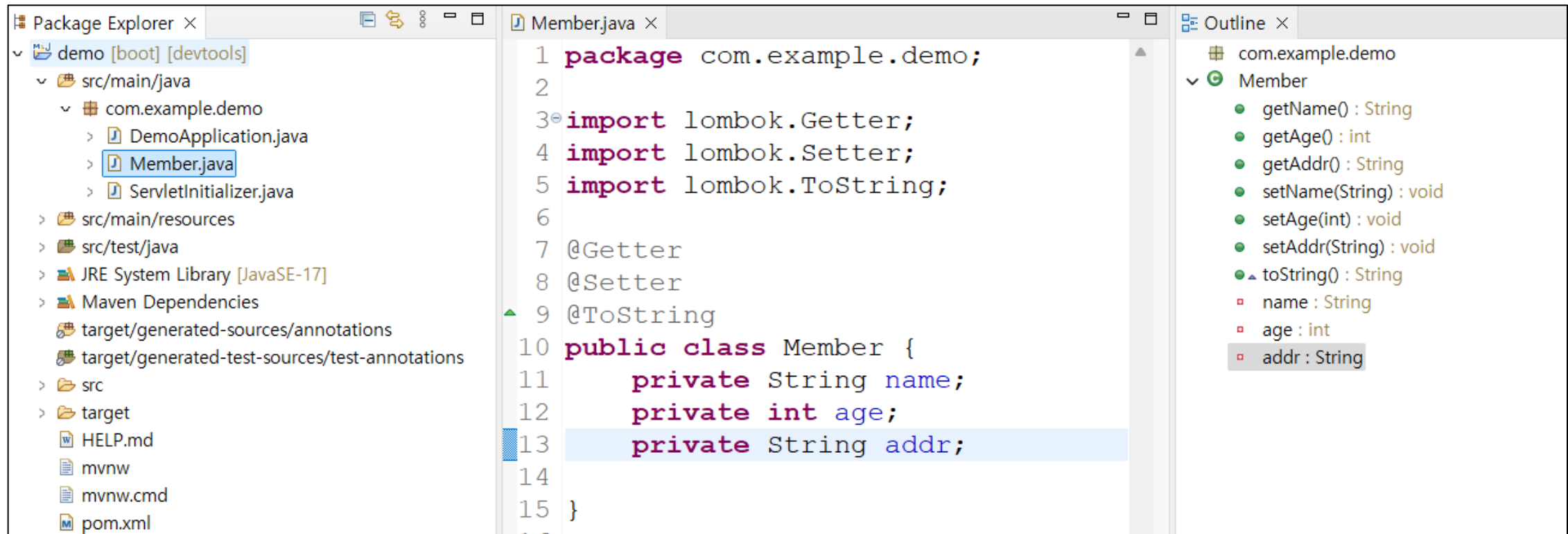
# Lombok

[Quit Installer] 선택 →

- Maven → Update Project
- Gradle → Refresh Gradle Project



# Lombok Test



# Getter/Setter 자동 생성

The image shows a screenshot of an IDE with two panels. The left panel displays the source code for a Java class named `Member` in the package `com.example.demo`. The code uses Lombok annotations `@Getter` and `@Setter` to generate getters and setters for three private fields: `name` (String), `age` (int), and `addr` (String). The right panel shows the Outline view, which lists the package and the `Member` class, along with the methods and fields generated by Lombok.

```
1 package com.example.demo;
2
3 import lombok.Getter;
4 import lombok.Setter;
5
6 @Getter
7 @Setter
8 public class Member {
9     private String name;
10    private int age;
11    private String addr;
12 }
```

Outline:

- com.example.demo
  - Member
    - getName() : String
    - getAge() : int
    - getAddr() : String
    - setName(String) : void
    - setAge(int) : void
    - setAddr(String) : void
    - name : String
    - age : int
    - addr : String

# 생성자 자동 생성

The image shows a screenshot of an IDE with two panels. The left panel displays the source code for `Member.java`, and the right panel shows the Outline view.

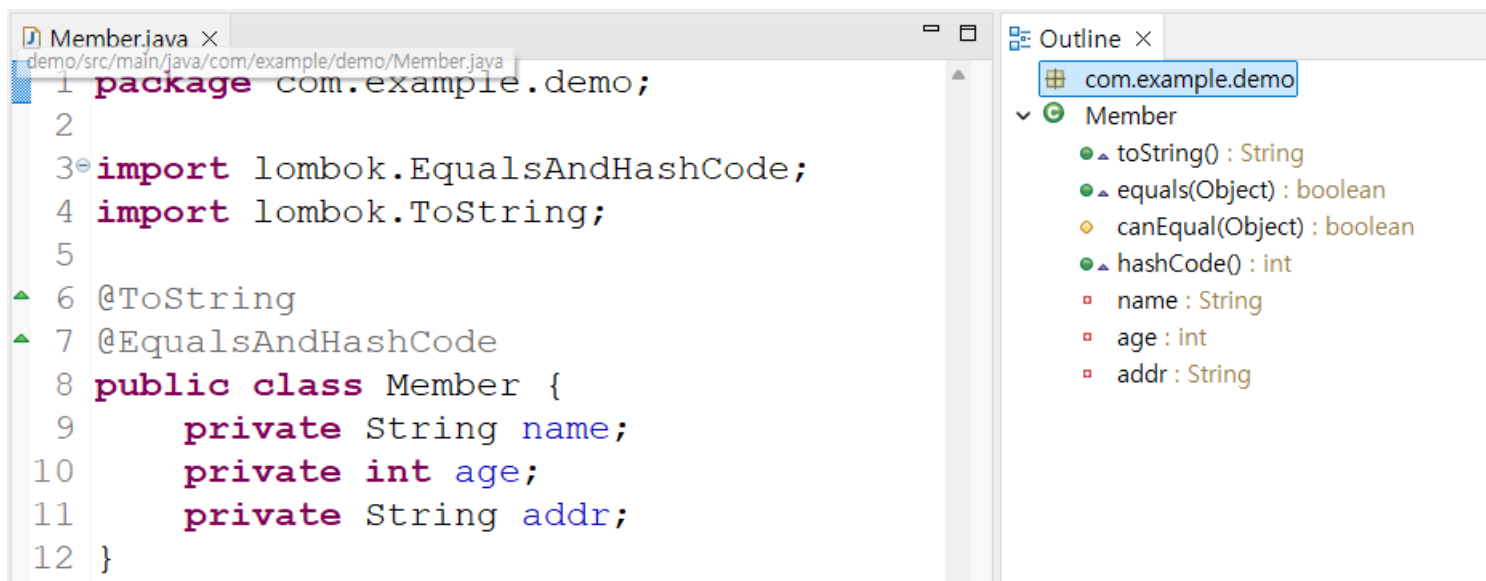
**Member.java**

```
1 package com.example.demo;
2
3 import lombok.AllArgsConstructor;
4 import lombok.NoArgsConstructor;
5
6 @NoArgsConstructor
7 @AllArgsConstructor
8 public class Member {
9     private String name;
10    private int age;
11    private String addr;
12 }
```

**Outline**

- com.example.demo
  - Member
    - Member()
    - Member(String, int, String)
    - name : String
    - age : int
    - addr : String

# toString(), equals(), hashCode() 자동 생성:



# 빌더 패턴

```
public class User {
    private String name;
    private int age;
    private String address;

    private User(UserBuilder builder) {
        this.name = builder.name;
        this.age = builder.age;
        this.address = builder.address;
    }

    @Override
    public String toString() {
        return "User [name=" + name + ", age=" + age +
            ", address=" + address + "];"
    }

    public static void main(String[] args) {
        User user = new User.UserBuilder()
            .name("홍길동")
            .age(30)
            .address("서울 강남구")
            .build();

        System.out.println(user);
    }
    // Getter methods here...
```

```
public static class UserBuilder {
    private String name;
    private int age;
    private String address;

    public UserBuilder name(String name) {
        this.name = name;
        return this;
    }

    public UserBuilder age(int age) {
        this.age = age;
        return this;
    }

    public UserBuilder address(String address) {
        this.address = address;
        return this;
    }

    public User build() {
        return new User(this);
    }
}

} //end of User Class
```



# 빌더 패턴

The screenshot displays an IDE with two panels. The left panel shows the source code for `Member.java`, and the right panel shows the Outline view.

**Source Code (Member.java):**

```
1 package com.example.demo;
2
3 import lombok.Builder;
4
5 @Builder
6 public class Member {
7     private String name;
8     private int age;
9     private String addr;
10 }
11
12
```

**Outline View:**

- com.example.demo
  - Member
    - Member(String, int, String)
    - builder() : MemberBuilder
  - MemberBuilder
    - name : String
    - age : int
    - addr : String
    - MemberBuilder()
    - name(String) : MemberBuilder
    - age(int) : MemberBuilder
    - addr(String) : MemberBuilder
    - build() : Member
    - toString() : String
  - name : String
  - age : int
  - addr : String

**Usage Example:**

```
Member member = Member.builder()
    .name("John Doe")
    .age(30)
    .addr("123 Main St")
    .build();
```