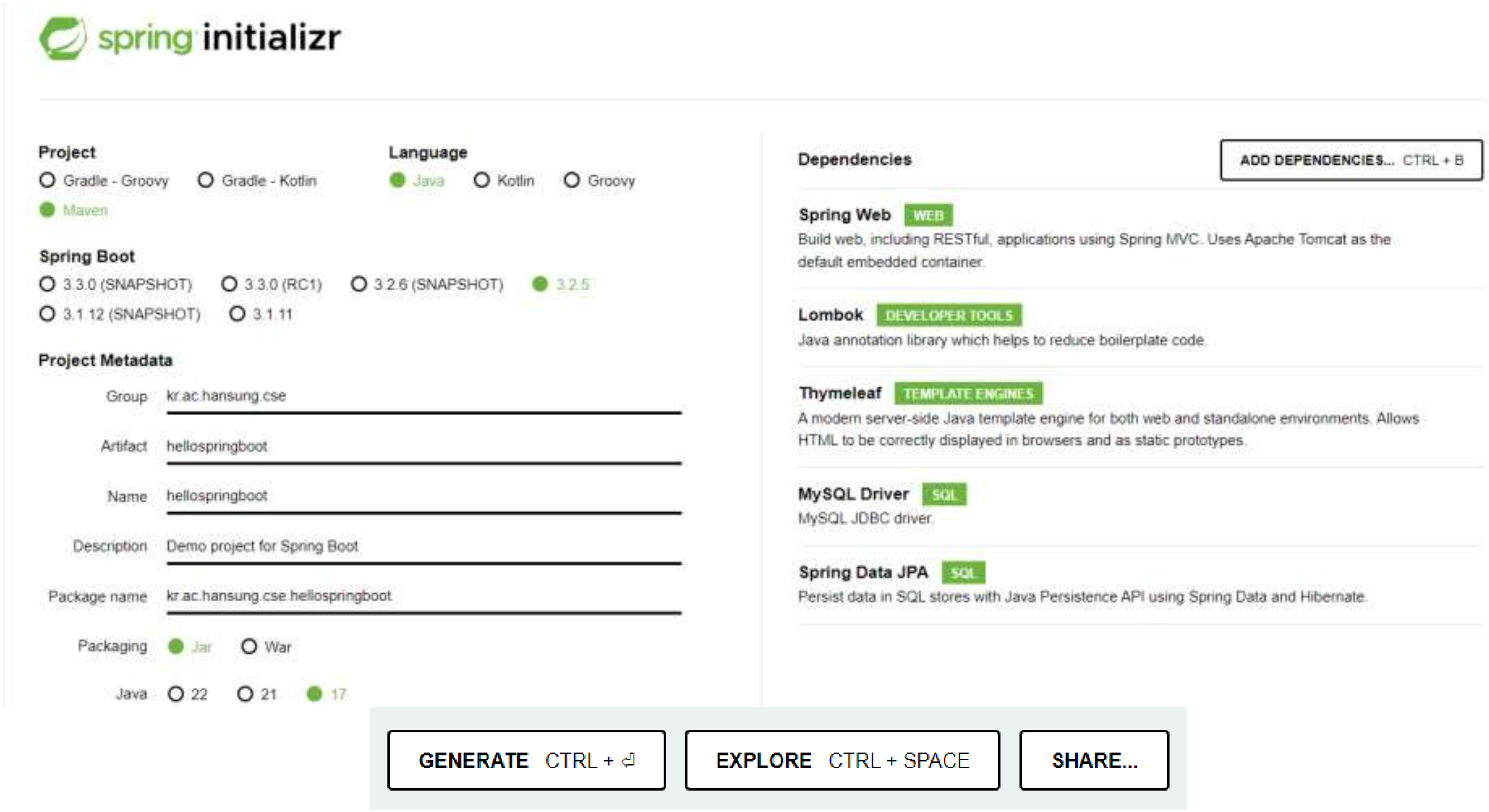
First SpringBoot Application

1. Spring Initializr

스프링 부트 애플리케이션을 생성한다

jar로 패키징

http://start.spring.io
(Online Spring Boot application generator)



spring initializr

Project
○ Gradle - Groovy ○ Gradle - Kotlin ● Maven

Language
● Java ○ Kotlin ○ Groovy

Spring Boot
○ 3.3.0 (SNAPSHOT) ○ 3.3.0 (RC1) ○ 3.2.6 (SNAPSHOT) ● 3.2.5
○ 3.1.12 (SNAPSHOT) ○ 3.1.11

Project Metadata

Group	kr.ac.hansung.cse
Artifact	hellospringboot
Name	hellospringboot
Description	Demo project for Spring Boot
Package name	kr.ac.hansung.cse.hellospringboot
Packaging	● Jar ○ War
Java	○ 22 ○ 21 ● 17

Dependencies ADD DEPENDENCIES... CTRL + B

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

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

Thymeleaf TEMPLATE ENGINES
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

MySQL Driver SQL
MySQL JDBC driver.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Spring Initializr

- Configure project at Spring Initializr website
- Generate the project
  - the zip file was downloaded
- Unzip the file
- Import Maven project into our IDE

설정, 생성, 압축풀기, 임포트

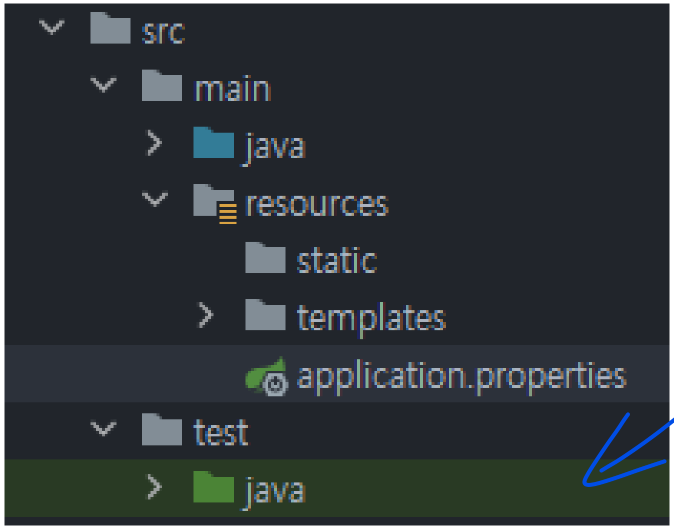
Using Intellij IDEA

- Using Intellij IDEA
  - You can create a Spring Boot project from Intellij IDEA by selecting File ➤ New ➤ Project ➤ Spring Initializr

참고 : 인텔리제이 ultimate 버전에서는 그게
자동으로 지원된다

구조

Project Structure



```
src
  main
    java
    resources
      static
      templates
      application.properties
  test
    java
```

테스트를 위한
코드도 있음

Directory	Description
/src/main/java	Java Source Code
/src/main/resources	Properties, html, css, images
/src/test/java	Test code

resources/static folder is used for serving web static content such as css, js, image
resources/templates folder is a place where you put all the thymeleaf templates

Project Structure

war 파일이 아닌 jar 파일로 패키징한다면,
$\boxed{}$ 표시된 경로를 쓰지 마시오

WARNING

Do not use the ***/src/main/webapp*** directory
if your application is packaged as a JAR

Although this is a standard Maven directory,
it works only with WAR packaging

It is silently ignored by most build tools if you generate a JAR

2. Looking at SpringBoot Project (pom.xml)

버전을 설정할 필요가 없다
↳ 부모 pom.xml을 상속받기 때문

```
<groupId>kr.ac.hansung.cse</groupId>
<artifactId>helloSpringBoot</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<name>helloSpringBoot</name>
<description>Demo project for Spring Boot</

<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.5</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>

<properties>
    <java.version>17</java.version>
</properties>
```

When no packaging is declared, "jar" is the default packaging type.

We don't need to specify the version for all the starter dependencies and other supporting libraries

```xml
<dependencies>
    <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>

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

    <dependency>
            <groupId>org.projectlombok</
            <artifactId>lombok</artifactId
            <optional>true</optional>
    </dependency>

    <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
    </dependency>
</dependencies>
```

Adding views using thymeleaf instead of jsp

The **spring-boot-starter-web** by default configures the **DispatcherServlet** to url-pattern "/" and adds Tomcat as embedded Servlet container which runs on port **8080**

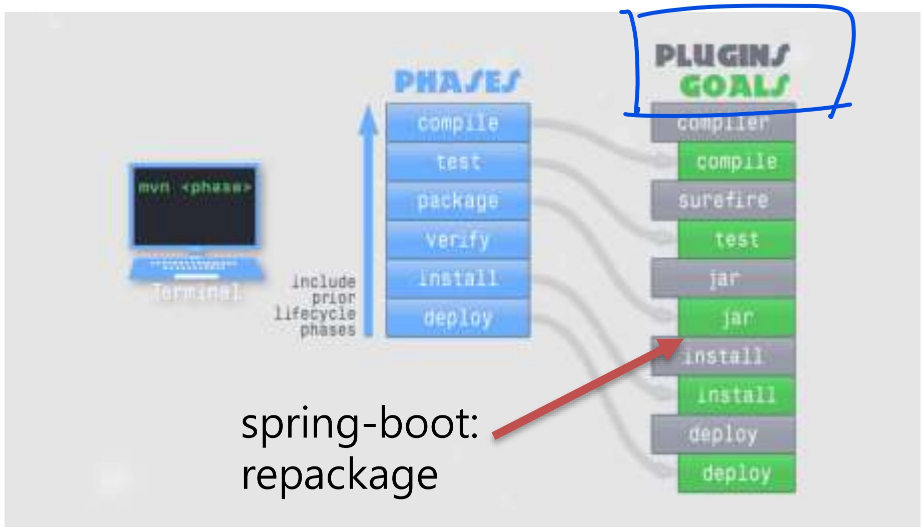
plugin의 등장, plugin ≒ 프로그램, goal ≒ 함수
Plugin은 maven의 2번째 장점

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

The Spring Boot Maven Plugin provides Spring Boot support in Maven, letting you 1) package executable jar or war archives and 2) run spring boot applications

# mvn package

when we execute *mvn package*, the *spring-boot:repackage* will be automatically executed



Spring Boot Maven Plugin

The plugin provides several goals
to work with a Spring Boot application

repackage 에 다양한 goal 들이 있다

- repackage: create a jar or war file that is auto-executable.
  It can replace the regular artifact
- run: run your Spring Boot application
- start and stop: ...
- build-info: ...

Running from the Command-Line

커맨드 사용하는 것

Two options for running the app

- Option 1: Using Executable JAR

```
java –jar .₩target₩helloSpringBoot-0.0.1-SNAPSHOT.jar
```

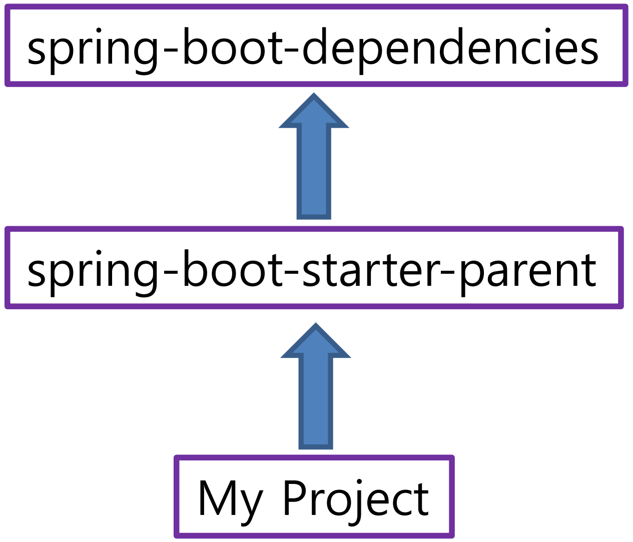
- Option 2: Use Spring Boot Maven plugin

```
mvn spring-boot:run
```

Pom.xml

Dependency Management 한테도 상속받고,

Plugin Management 한테도 상속받는다



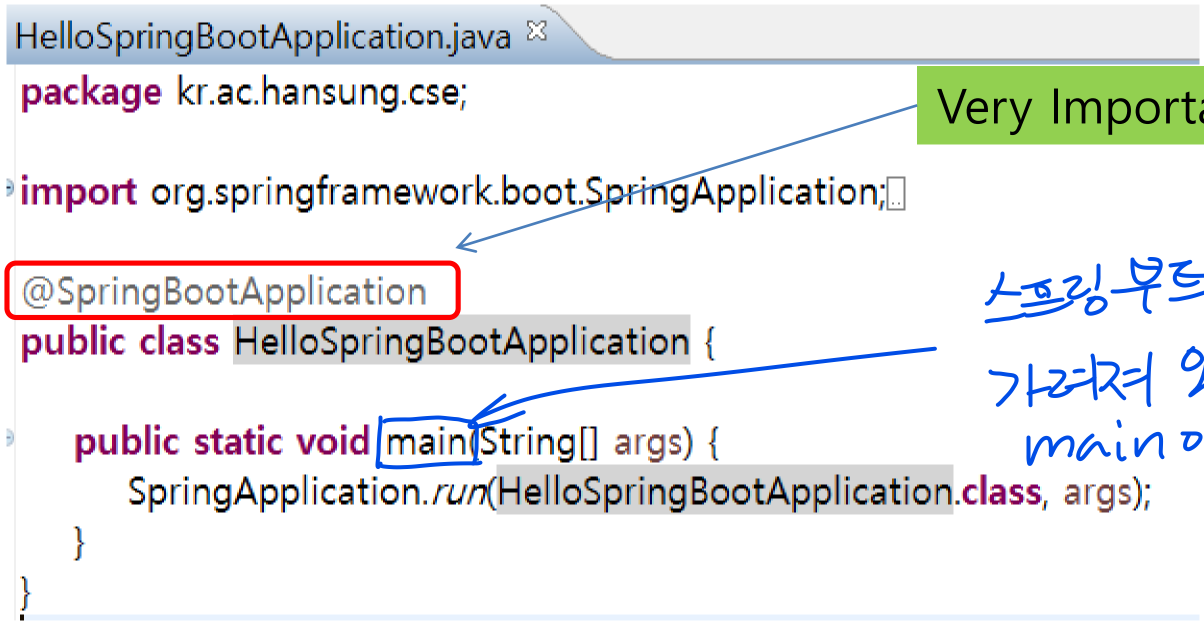
Dependency Management

Plugin Management

maven-compiler-plugin,
maven-jar-plugin
maven-surefire-plugin,
maven-war-plugin
spring-boot-maven-plugin,
...

∴ 내가 설정해야할 것들은 줄여든다.

3. Application EntryPoint Class



HelloSpringBootApplication.java

```java
package kr.ac.hansung.cse;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class HelloSpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(HelloSpringBootApplication.class, args);
    }
}
```

Very Important!!!

필수 →

스프링부트에서는, 가려져 있던 main이 드러난다

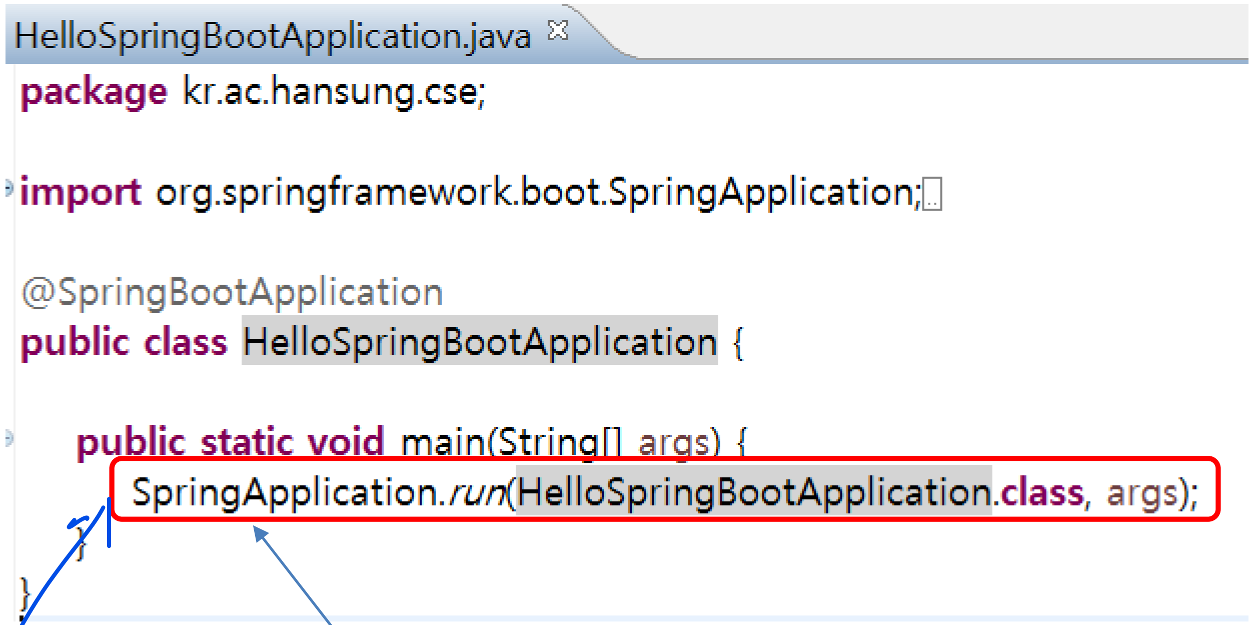
@SpringBootApplication annotation is a composed annotation

- @EnableAutoConfiguration enables SpringBoot's auto-configuration support
- @ComponentScan enables component scanning of current package
  Also recursively scans sub-packages
- @Configuration indicates that this class is a Spring configuration class

→ 자동 설정 활성화

→ 패키지와 서브패키지 스캔하고 컨테이너에 bean 추가해줌

Application EntryPoint Class



HelloSpringBootApplication.java

```
package kr.ac.hansung.cse;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class HelloSpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(HelloSpringBootApplication.class, args);
    }
}
```

bootstrap and launch a Spring application

run 은 bean을 등록하고, tomcat 실행까지 해준다

Behind the scenes

- Creates application context and registers all beans
- Starts the embedded server(tomcat), …

Application EntryPoint Class

- WebApplicationType
  - NONE : the application should not run as a web application and should not start an embedded web server
  - REACTIVE : the application should run as a reactive web application and should start an embedded reactive web server
  - $\boxed{\text{SERVLET}}$: the application should run as a servlet-based web application and should start an embedded servlet web server

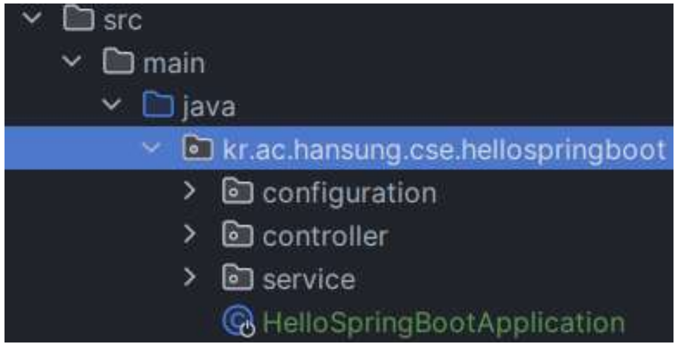
주로 사용

application.properties

```
spring.main.web-application-type=servlet
```

4. More on Component Scanning

- Default scanning is fine if everything is under the root package



```
src
  main
    java
      kr.ac.hansung.cse.hellospringboot
        configuration
        controller
        service
        HelloSpringBootApplication
```

- But what about my other packages?

explicitly list base packages to scan

외부 패키지 → 스캔하는법 (비추천)

```
@SpringBootApplication(
    scanBasePackages={"kr.ac.hansung.cse.hellospringboot",
                      "com.mypackage.springapp",
                      "kr.ac.hansung.iot"}  )
public class HelloSpringBootApplication {
    public static void main(String[] args) {
        SpringApplication.run(HelloSpringBootApplication.class, args);
    }
}
```

More on Component Scanning

```
@SpringBootApplication
@ComponentScan(basePackages={“kr.ac.hansung.cse.hellospringboot”,
                             “com.mypackage.springapp”,
                             “kr.ac.hansung.iot”} )
public class HelloSpringBootApplication {
    public static void main(String[] args) {
        SpringApplication.run(HelloSpringBootApplication.class, args);
    }
}
```

It is highly recommended that you put the main entry point class
in the root package, say in kr.ac.hansung.cse.helloSpringBoot,
so that the @EnableAutoConfiguration and @ComponentScan annotations
will scan for Spring beans, JPA entities, etc.,
in the root and all of its sub-packages automatically

5. Spring MVC

HomeController.java

```
@Controller
public class HomeController {

    // @RequestMapping(value="/", method = RequestMethod.GET).
    @GetMapping("/")
    public String home(Model model) {
            model.addAttribute("message", "hello world");
            return "index";
    }
}
```

컨트롤러

뷰

src/main/resources/templates/index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">

<head>
<meta charset="utf-8" />
</head>
<!-- th:text replaces the body of a tag -->
 <body>
<div  th:text  =  "${message}"> </div>
</body>

</html>
```

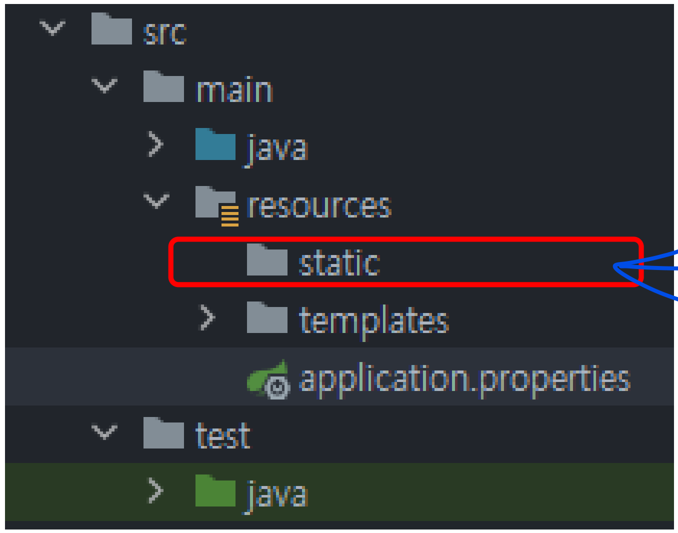
Thymeleaf

jsp 대신 쓸거임

Output

```
<!-- th:text replaces the body of a tag -->
 <body>
    <div>hello world</div>
</body>
```

Static Content



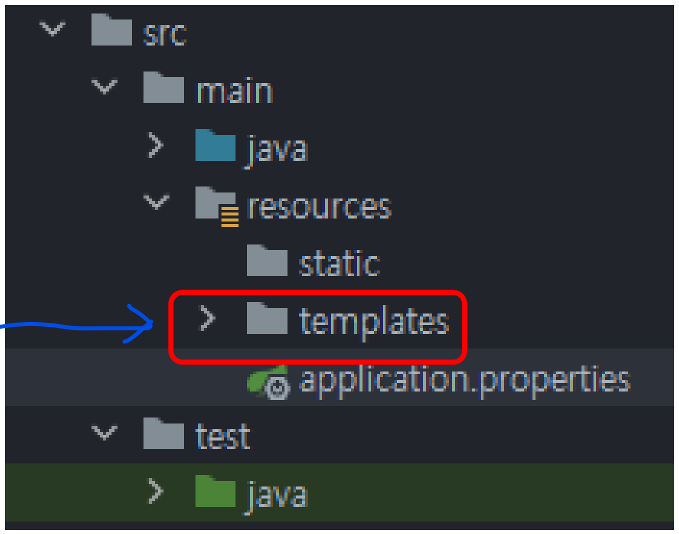
By default,
Spring Boot will load static resources
from "/static" and "/public" directory

Examples of static resources:

CSS, JavaScript, images, etc, ...

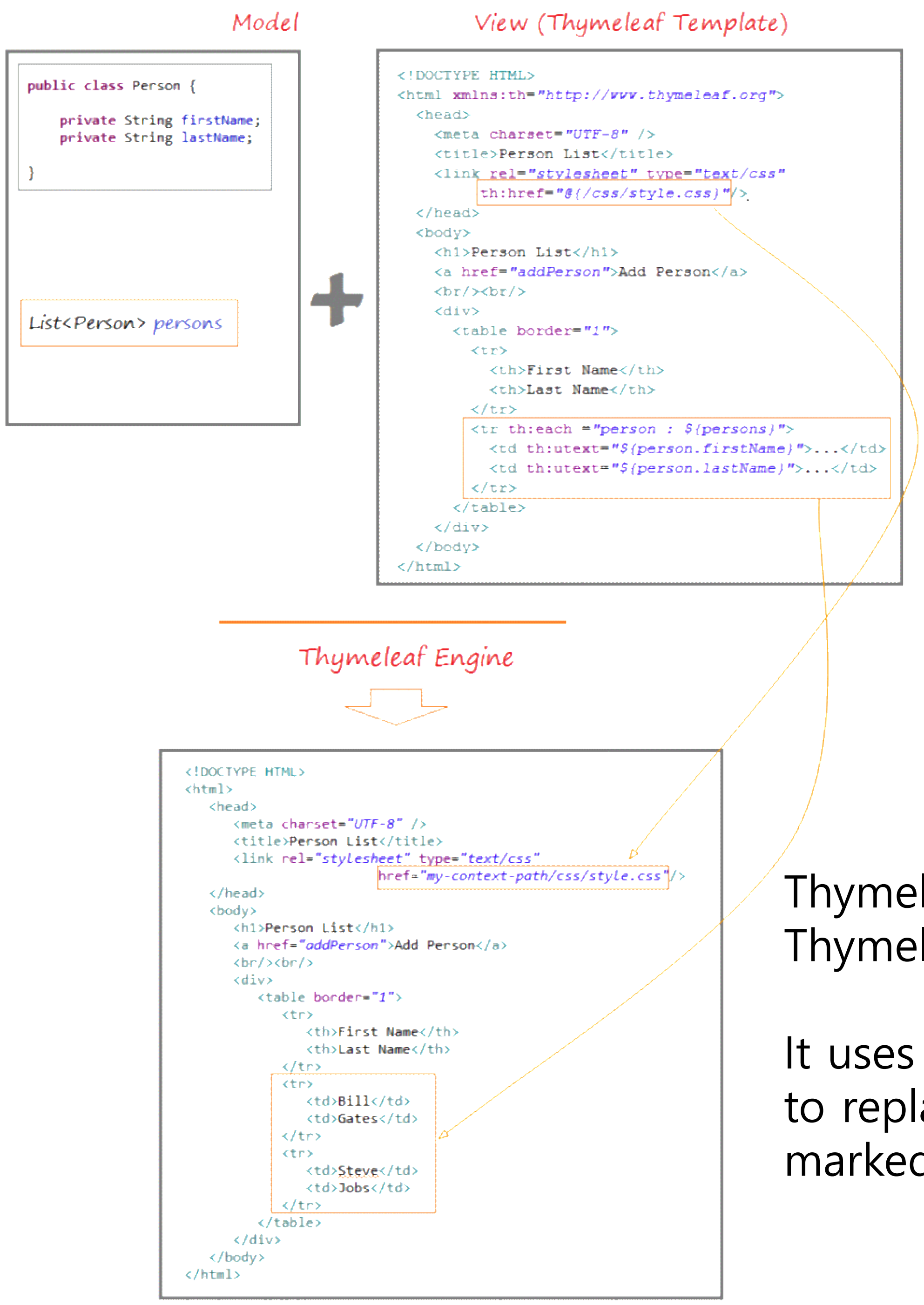
Templates

- Spring Boot includes auto-configuration for following template engines
  - Thymeleaf
  - FreeMarker
  - Mustache



Thymeleaf is a popular template engine

By default, Spring Boot will load templates from “/templates” directory



```
public class Person {

    private String firstName;
    private String lastName;

}
```

```
List<Person> persons
```

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
  <head>
    <meta charset="UTF-8" />
    <title>Person List</title>
    <link rel="stylesheet" type="text/css"
          th:href="@{/css/style.css}"/>
  </head>
  <body>
    <h1>Person List</h1>
    <a href="addPerson">Add Person</a>
    <br/><br/>
    <div>
      <table border="1">
        <tr>
          <th>First Name</th>
          <th>Last Name</th>
        </tr>
        <tr th:each ="person : ${persons}">
          <td th:utext="${person.firstName}">...</td>
          <td th:utext="${person.lastName}">...</td>
        </tr>
      </table>
    </div>
  </body>
</html>
```

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Person List</title>
    <link rel="stylesheet" type="text/css"
          href="my-context-path/css/style.css"/>
  </head>
  <body>
    <h1>Person List</h1>
    <a href="addPerson">Add Person</a>
    <br/><br/>
    <div>
      <table border="1">
        <tr>
          <th>First Name</th>
          <th>Last Name</th>
        </tr>
        <tr>
          <td>Bill</td>
          <td>Gates</td>
        </tr>
        <tr>
          <td>Steve</td>
          <td>Jobs</td>
        </tr>
      </table>
    </div>
  </body>
</html>
```

Thymeleaf Engine will parse Thymeleaf Template.

It uses Java data(model) to replace the positions marked on the Thymeleaf Template

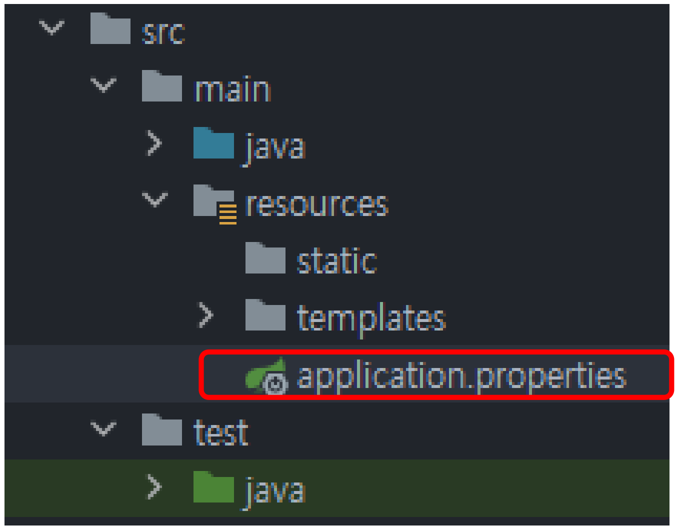
속성값

6. Application Properties

- By default, Spring Boot will load properties from **application.properties**

http://localhost:9000/helloSpringBoot/customer

src/main/resources/application.properties



```
# Can add Spring Boot properties
server.port=9000

# set context path
server.servlet.context-path=/helloSpringBoot

# logging.level.<logger-name>=<level>
logging.level.kr.ac.hansung=debug

# add our own custom properties
app.professor=Namyun Kim
app.course=Web Framework
```

Application Properties

객체에 바인딩

how to bind properties to an object

```
#configure my props

app.professor= Namyun Kim
app.course=Web Framework
```

```
@RestController
public class HelloWorldController {

    @Value("${app.professor}")
    private String professorName;

    @Value("${app.course}")
    private String courseName;

    ...
}
```

1보다 2가 더 편리하다

Application Properties

application.properties

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/test
jdbc.username=root
jdbc.password=secret
```

Method 1

```
@Configuration
public class AppConfig
{
        @Value("${jdbc.driver}")
        private String driver;

        @Value("${jdbc.url}")
        private String url;

        @Value("${jdbc.username}")
        private String username;

        @Value("${jdbc.password}")
        private String password;
…
…
}
```

Method 2

```
@Configuration
@ConfigurationProperties(prefix="jdbc")
public class DataSourceConfig
{
        private String driver;
        private String url;
        private String username;
        private String password;
        //setters and getters
}
```

automatically bind the properties that start with jdbc.*
to a bean's properties