Spring Boot

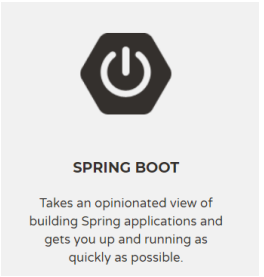
Spring Boot 개요 및 개발 환경 구축

Spring Boot 개요

- Spring Framework의 Sub Project
  - framewor의 사전적 의미 – 뼈대 및 구조

- Spring과 Boot의 합성어
  - Spring : 오픈 소스 프레임워크
  - Boot : '컴퓨터를 부팅한다', 즉 시스템에서 사용 가능한 상태로 만든다는 의미

Spring 단독 실행 즉 즉시 운영 가능한 애플리케이션 개발에 도움을 줌



Spring Boot 개요

- Spring Framework를 재구성
  - Spring의 핵심 기능은 수용하지만 많은 '엔터프라이즈'기능을 제거
  - 자바 기반의 REST(Representational State Transfer) 지향 마이크로서비스 프레임워크 재구성
  - 커맨드 도구 와 tomcat or 제티 같은 내장 서버를 통해 복잡한 설정과 실행 간소화

  - 기본적으로 설정된 Spring 플랫폼 및 서드 파티 library 사용을 통해 생산성을 증대시키고 최소한의 설정으로 애플리메이션 개발 및 구동이 가능

Spring Boot 개요

+

설정 자동화
WAS 내장
단위 테스트 강화

=

설정 자동화/간소화를 통해 개발 속도 향상

tomcat과 같은 WAS환경을 내장하고 실행 속도개선

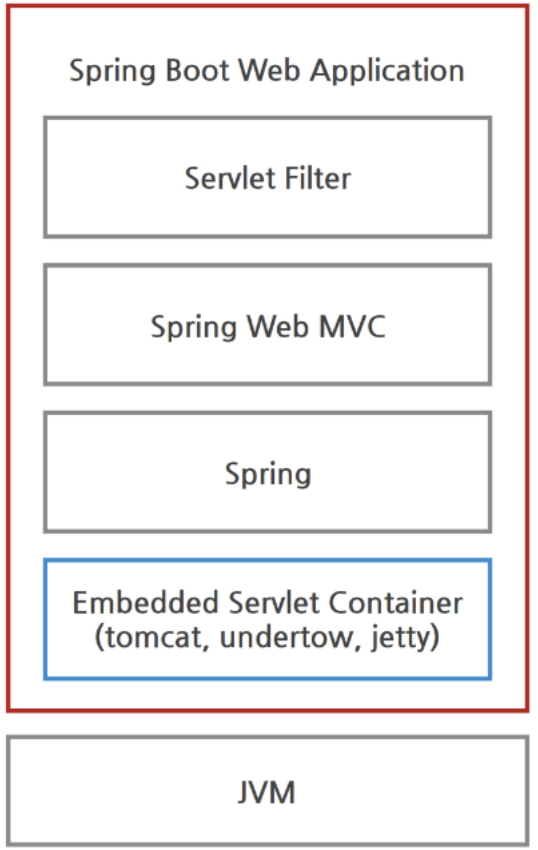
단위테스트를 강화해서 프로젝트의 안정성 강화

Spring Boot 특징

- UTF-8 인코딩을 위한 서블릿 필터가 등록되어 있음
- 정적 자원과 WebJars에 대한 위치 설정 및 요청 연결 템플릿 엔진 설정 및 기본 View Resolver를 구성하고 있음
- JSON/XML을 다룰 수 있는 메세지 변환기를 등록하였음

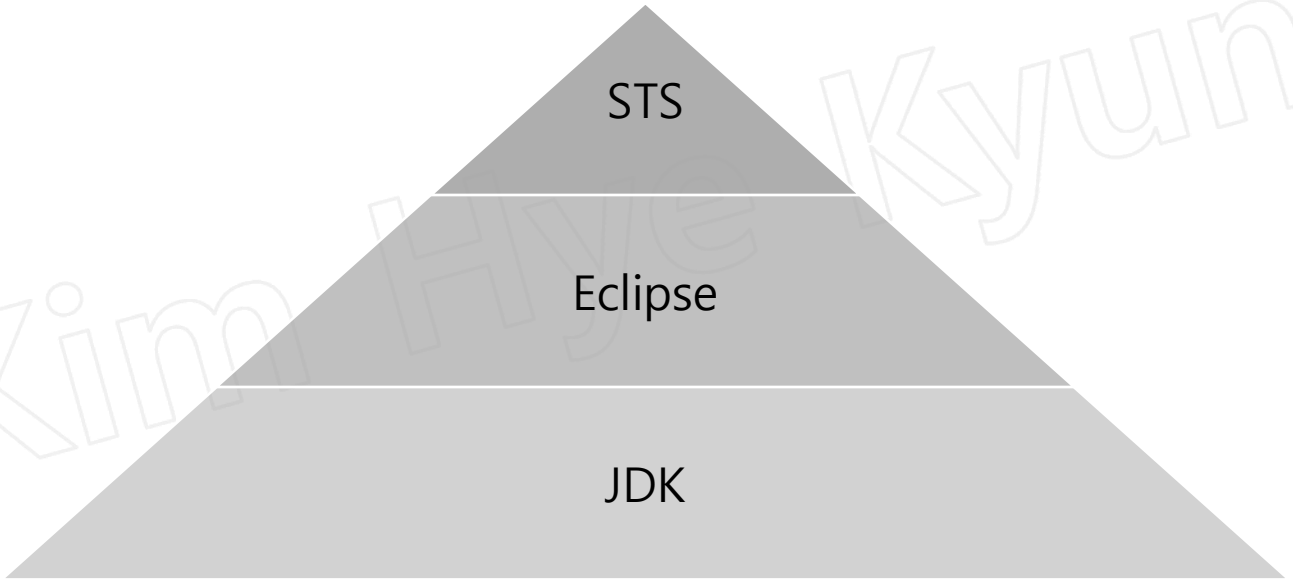
Spring Boot의 장점

- 라이브러리 관리 자동화
- 설정의 자동화
- 라이브러리 버전 자동 관리
- 테스트 환경과 내장 톰캣
- 독립적으로 실행 가능한 JAR



Spring Boot 개발 환경 구축

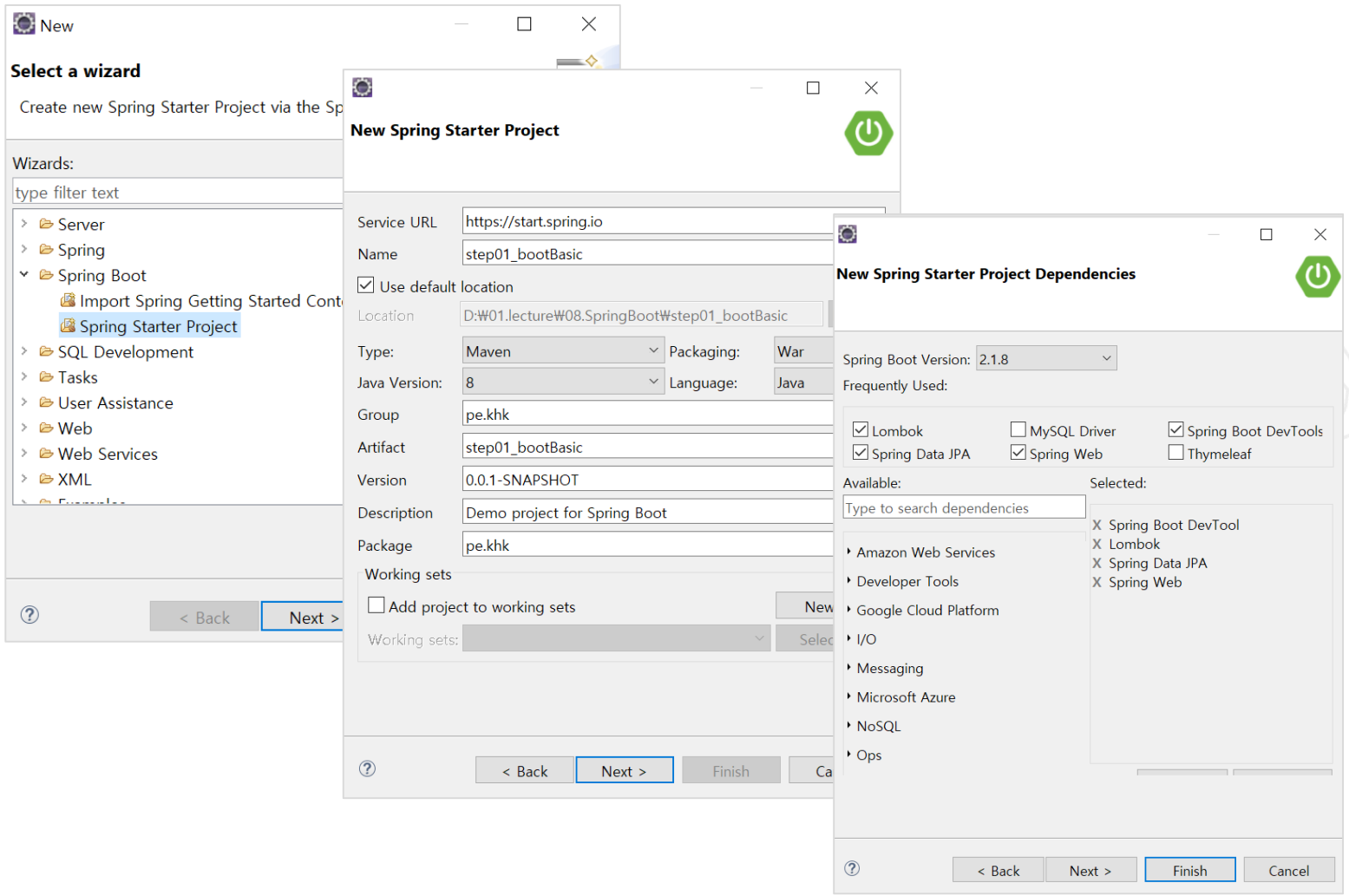
- Spring Boot 2.0 이상인 경우 반드시 JDK 8 이상 설치



Spring Boot 퀵스타트

- Spring Boot로 Project 생성시
  - Spring library 등의 librar들을 개발자가 신경쓸 필요 없음
    - 모든 library들을 자동으로 다운로드 및 관리
  - XML 환경 설정 파일 역시 작성하지 않음
    - Bean 설정을 위한 XML이 아닌 어노테이션 기반으로 처리

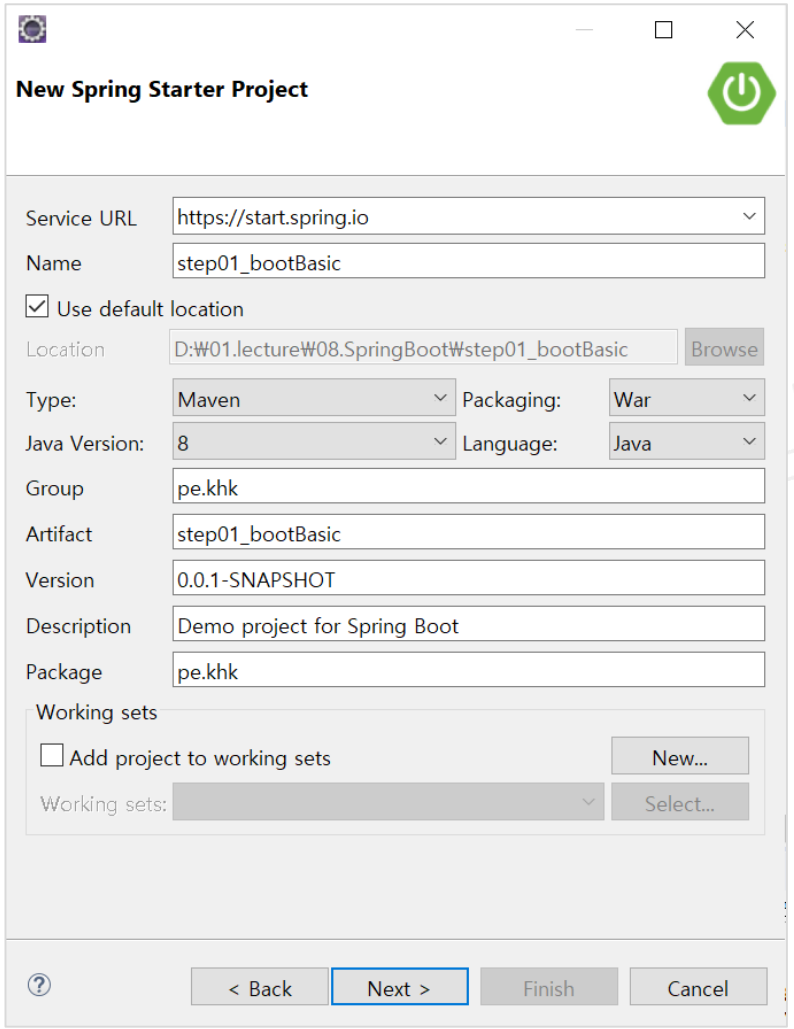
Spring Boot Project 생성 단계




```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.8.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>pe.khk</groupId>
<artifactId>step01_bootBasic</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
<name>step01_bootBasic</name>
<description>Demo project for Spring Boot</description>

<properties>
    <java.version>1.8</java.version>
</properties>

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

최적화된 tomcat 포함

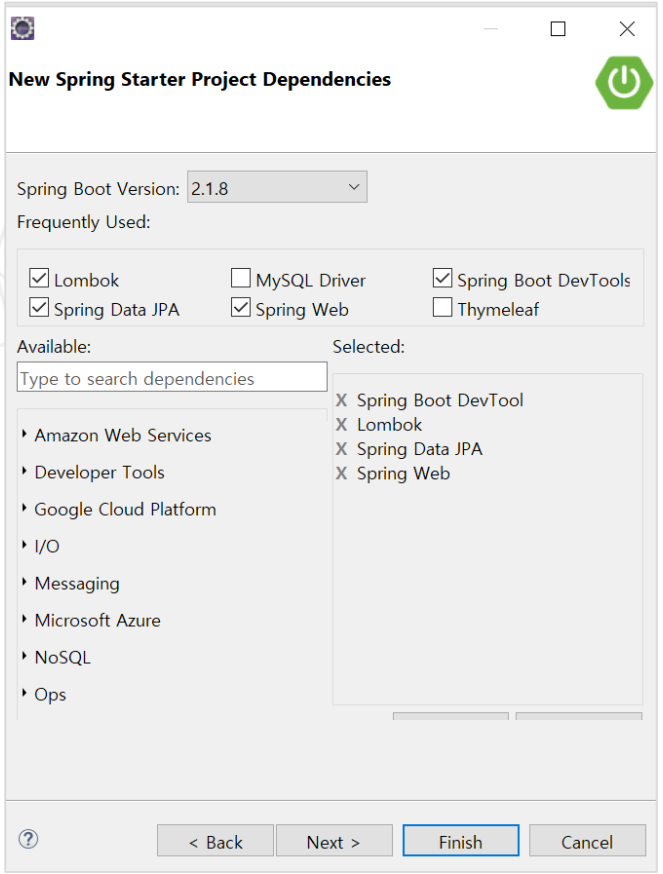
Spring Boot Project 생성 단계 - 참고



항목	설명
Servlet URL	Spring Boot의 정보를 받아오는 경로로 변경하지 않음
Name	생성할 프로젝트 이름
Type	library 관리 도구
Packaging	packaging파일 형식, jar or war
Group	프로젝트를 만들고 관리할 단체나 회사 정보(도메인명) 일반적으로 소속사
Artifact	프로젝트를 의미
Package	프로젝트 생성시 기본적으로 생성되는 package 경로

Spring Boot Project 생성 단계

- 의존성 설정
  - Spring Legacy 대비 설정이 간단해진 첫 번째 부분
  - 의존 librar들의 버전 관리를 설정만으로 사용

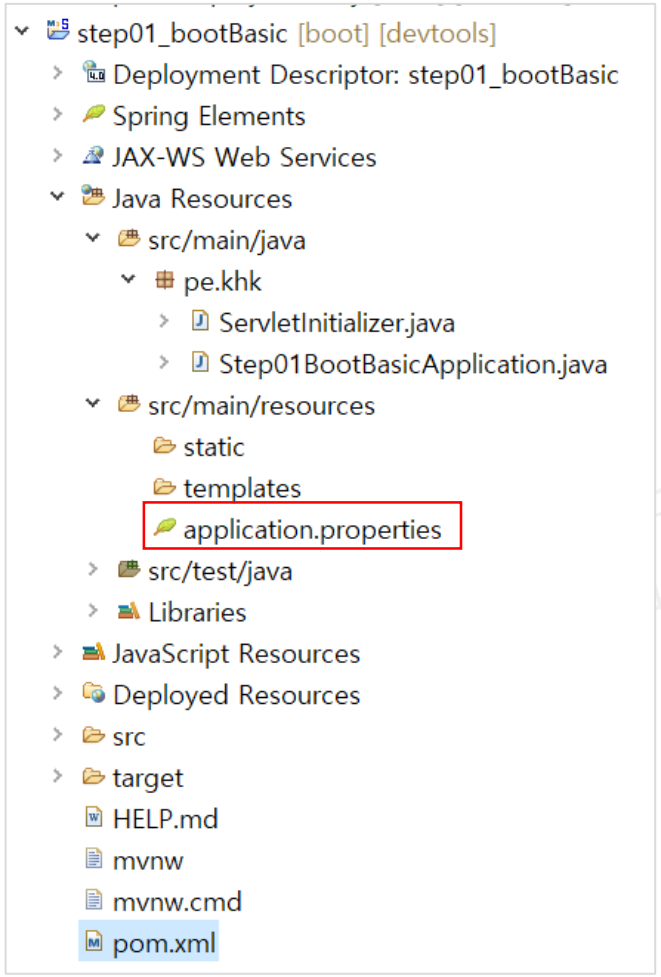


Spring Boot Project 생성후 pom.xml

- pom.xml file에 Unknown 에러 발생시 해결책
  - 에러가 있어도 정상 실행
  - 에러 삭제를 위한 추가 설정
  - <maven-jar-plugin.version>3.1.1</maven-jar-plugin.version>

```
17    <properties>
18        <java.version>1.8</java.version>
19
20        <!-- pom.xml에 발생된 unknown 에러 처리를 위한 추가 -->
21        <maven-jar-plugin.version>3.1.1</maven-jar-plugin.version>
22
23    </properties>
```

Spring Boot Project 구조



구조	설명
src/main/java	자바 소스
src/main/resources	자바 소스가 아닌 xml이나 properties 파일
src/main/resources/static	HTML과 같은 정적인 웹리소스 저장
src/main/resources/templates	타임리프와 같은 템플릿 기반의 웹소스
application.properties	프로젝트 전체에서 사용할 프로퍼티 정보들을 관리하고 설정하는 파일 필요한 프로퍼티들을 등록하면 복잡한 자바 코드를 수정하지 않고도 애플리케이션의 동작을 변경 할 수 있음
src/main/test	Junit 기반의 test 케이스

Spring Boot 주요 library – pom.xml

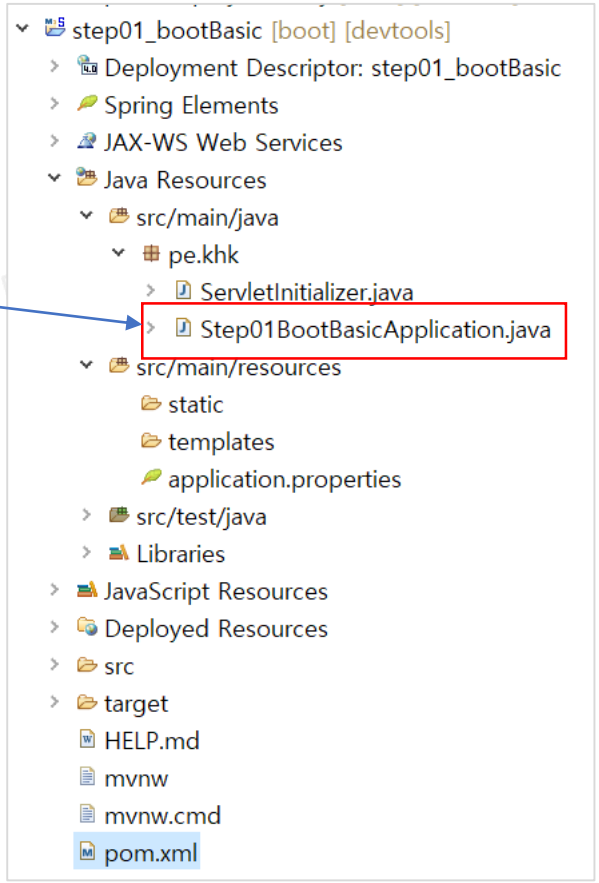
```xml
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.8.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>pe.khk</groupId>
    <artifactId>step01_bootBasic</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>
    <name>step01_bootBasic</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

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

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

library	설명
spring-boot-starter-web	웹 애플리케이션 개발에 필요한 Spring MVC 관련 library Spring Boot가 웹 프로젝트 환경에 최적화된 library들 등록
spring-boot-starter-test	Junit을 비롯한 test 관련 libtary

Spring Boot Application 실행

Spring Boot Application 실행

- main 클래스
  - 자동 생성되는 기본 클래스
    - Project명+Application.java
      - 웹 애플리케이션으로 실행
  - src/main/java 경로에 자동 생성
  - 실행시 내장 tomcat 자동 구동되면서 실행

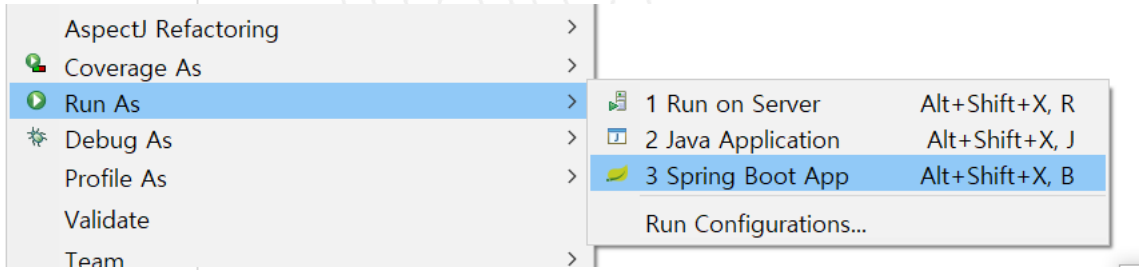


- 실행 방식
  - 방법 1 – 웹 애플리케이션 방식으로 실행
  - 방법 2 – 일반 자바 애플리케이션 방식으로 실행

Spring Boot 실행

- 방식 1 : 웹 애플리케이션 방식으로 실행
- @SpringBootApplication Spring Boot로 만든 애플리케이션의 시작 클래스

```java
package pe.khk;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class Step01BootBasicApplication {

    public static void main(String[] args) {
        SpringApplication.run(Step01BootBasicApplication.class, args);
    }

}
```




```java
//실행 방식 3 : WebApplicationType.SERVLET 설정으로 웹으로 실행
SpringApplication application = new SpringApplication(Step01BootBasicApplication.class);
application.setWebApplicationType(WebApplicationType.SERVLET);
application.run(args);
```

Spring Boot 실행

- 웹 애플리케이션 방식으로 실행

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.8.RELEASE)

2019-09-30 16:58:05.659  INFO 14948 --- [  restartedMain] pe.khk.Step01BootBasicApplication        : Starting Step01BootBasicApplication on HyeKyung with
2019-09-30 16:58:05.671  INFO 14948 --- [  restartedMain] pe.khk.Step01BootBasicApplication        : No active profile set, falling back to default profi
2019-09-30 16:58:05.697  INFO 14948 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devto
2019-09-30 16:58:05.697  INFO 14948 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting
2019-09-30 16:58:06.198  INFO 14948 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data repositories in DEFAULT mo
2019-09-30 16:58:06.214  INFO 14948 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 16ms. Fo
2019-09-30 16:58:06.414  INFO 14948 --- [  restartedMain] trationDelegate$BeanPostProcessorChecker : Bean 'org.springframework.transaction.annotation.Pro
2019-09-30 16:58:06.646  INFO 14948 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2019-09-30 16:58:06.662  INFO 14948 --- [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2019-09-30 16:58:06.663  INFO 14948 --- [  restartedMain] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.24]
2019-09-30 16:58:06.746  INFO 14948 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2019-09-30 16:58:06.746  INFO 14948 --- [  restartedMain] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed
2019-09-30 16:58:06.784  WARN 14948 --- [  restartedMain] ConfigServletWebServerApplicationContext : Exception encountered during context initialization
2019-09-30 16:58:06.784  INFO 14948 --- [  restartedMain] o.apache.catalina.core.StandardService   : Stopping service [Tomcat]
2019-09-30 16:58:06.784  INFO 14948 --- [  restartedMain] ConditionEvaluationReportLoggingListener :

Error starting ApplicationContext. To display the conditions report re-run your application with 'debug' enabled.
2019-09-30 16:58:06.800 ERROR 14948 --- [  restartedMain] o.s.b.d.LoggingFailureAnalysisReporter   :

***************************
APPLICATION FAILED TO START
***************************
```

Spring Boot 실행

- 방식 2 : 일반 자바 애플리케이션 방식으로 실행

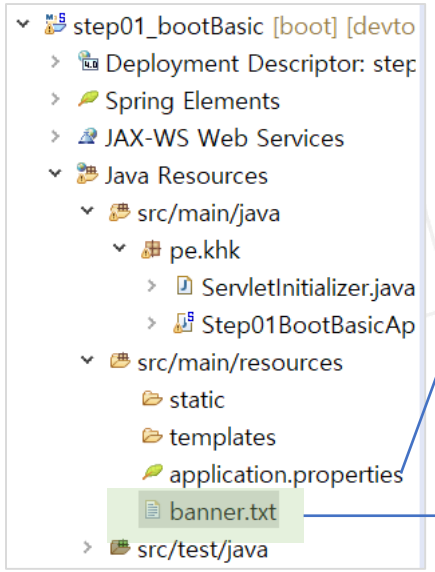
```java
package pe.khk;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.WebApplicationType;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Step01BootBasicApplication {

    public static void main(String[] args) {
        //실행 방식 1 : 웹으로 실행
        //SpringApplication.run(Step01BootBasicApplication.class, args);

        //실행 방식 2 : 자바 애플리케이션으로 실행
        SpringApplication application = new SpringApplication(Step01BootBasicApplication.class);
        application.setWebApplicationType(WebApplicationType.NONE);
        application.run(args);
    }

}
```

Spring Boot 실행

- 사용자 정의 banner로 대체하기
  - 사용자 정의 logo 메세지 출력을 위한 설정
  - Server의 Port 수정

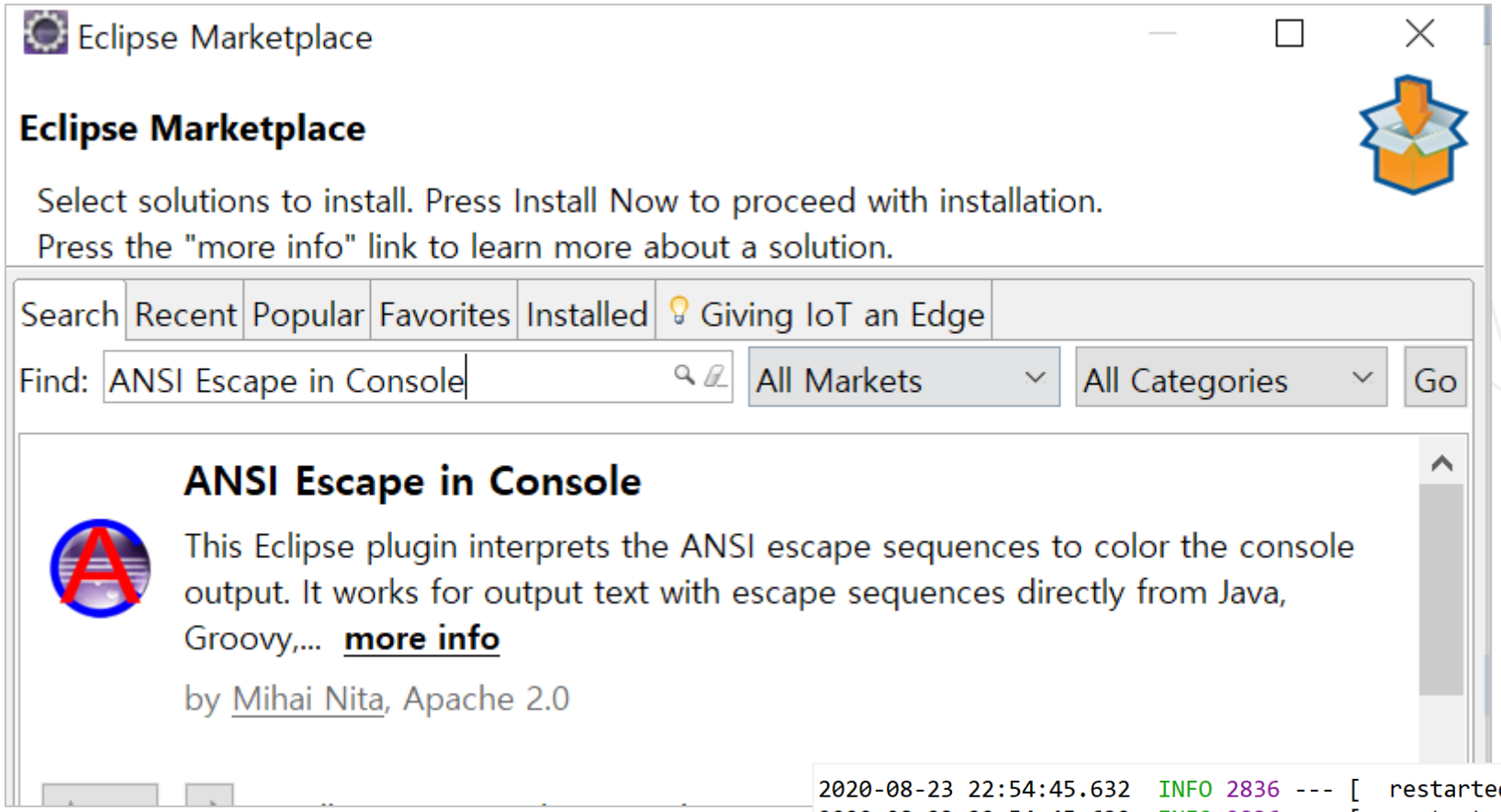
application.properties

```
server.port=8000
```




```
************************************

        사용자 정의 문구 구성이 가능합니다

************************************
2019-09-30 19:18:44.963  INFO 1668 --- [  restartedMain] pe.khk.Step01BootBasicApp
2019-09-30 19:18:44.963  INFO 1668 --- [  restartedMain] pe.khk.Step01BootBasicApp
2019-09-30 19:18:44.995  INFO 1668 --- [  restartedMain] .e.DevToolsPropertyDefaul
2019-09-30 19:18:44.995  INFO 1668 --- [  restartedMain] .e.DevToolsPropertyDefaul
2019-09-30 19:18:45.464  INFO 1668 --- [  restartedMain] .s.d.r.c.RepositoryConfig
2019-09-30 19:18:45.480  INFO 1668 --- [  restartedMain] .s.d.r.c.RepositoryConfig
2019-09-30 19:18:45.699  INFO 1668 --- [  restartedMain] trationDelegate$BeanPostP
2019-09-30 19:18:45.902  INFO 1668 --- [  restartedMain] o.s.b.w.embedded.tomcat.T
```

참고 : ANSI Escape in Console 플러그인 설치




```
2020-08-23 22:54:45.632  INFO 2836 --- [  restartedMain] c.e.demo.Step09BootBasicApplication      :
2020-08-23 22:54:45.632  INFO 2836 --- [  restartedMain] c.e.demo.Step09BootBasicApplication      :
2020-08-23 22:54:45.679  INFO 2836 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor :
2020-08-23 22:54:45.679  INFO 2836 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor :
2020-08-23 22:54:46.585  INFO 2836 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  :
2020-08-23 22:54:46.617  INFO 2836 --- [  restartedMain] o.apache.catalina.core.StandardService   :
```

레벨별 color

Level	Color	의 미
ERROR	Red	사용자 요청을 처리하는 중 문제
WARN	Yellow	처리 가능한 문제지만, 향후 시스템 에러의 원인이 될 수 있는문제
INFO	Green	로그인이나 상태 변경과 같은 정보성 메시지
DEBUG	Green	개발시 디버깅 목적으로 출력하는 메시지
TRACE	Green	DEBUG 레벨 보다 좀더 상세한 메시지

Spring Boot 의존성 관리와 자동 설정

의존성 관리

의존성 관리

Starter

- 특정 모듈을 쉽고 편리하게 사용할 수 있도록 관련된 library 의존성 해결

AutoConfiguration(자동 설정)

- 애플리케이션 운용에 필요한 빈들을 자동 초기화

Actuator

- Spring Boot로 개발된 시스템을 모니터링 할 수 있는 다양한 기능 제공

의존성 관리 : Start

- spring-boot-starter 를 이용하면

  특정 모듈과 관련된 의존성을

  package처럼 관리

  project에 새로운 모듈을 쉽게 등록 및 제거 가능

```
<properties>
    <java.version>1.8</java.version>
    <maven-jar-plugin.version>3.1.1</maven-jar-plugin.version>
</properties>

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
```

의존성 관리 : 자동설정

- 자동설정이란?
  - Spring Boot로 만든 프로젝트에는 애플리케이션 실행을 위한 메인 클래스가 기본적으로 제공
  - 실행시
    - 내장 tomcat 구동
    - spring 기반의 웹 애플리케이션 자동 실행
      - spring boot가 제공하는 자동 설정 기능이 동작하여 수많은 빈들이 등록되고 동작
      - **@SpringBootApplication**
        - 사용자가 작성한 빈과 자동설정 빈들을 모두 초기화

```java
package com.example.demo;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@ComponentScan(basePackages= {"controller"})
public class Step09BootBasicApplication {

    public static void main(String[] args) {
        SpringApplication.run(Step09BootBasicApplication.class, args);
    }

}
```

의존성 관리 : 자동설정

- 기능 수행을 위한 주요 애노테이션

주요 애노테이션	기 능
@SpringBootConfiguration	환경설정 빈 클래스를 표현하기 위해 사용하는 @Configuration과 동일
@EnableAutoConfiguration	spring container 초기화와 관련된 애노테이션
@ComponentScan	spring container 초기화와 관련된 애노테이션 @Configuration, @Controller, @Repository, @Service, @RestController 이 붙은 객체를 메모리에 올리는 기능

```
 */
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
        @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
        @Filter(type = FilterType.CUSTOM,
                classes = AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {

```

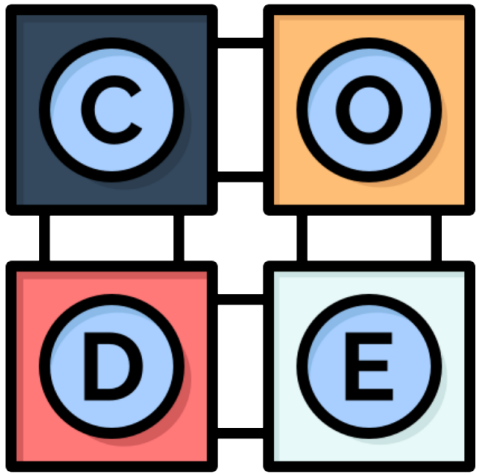
주요 Annotation

- @ComponentScan
  - @Compoent라는 어노테이션이 붙어있는 class를 빈으로 등록
  - @Compoent, @Configuration, @Repository, @Service, @Controller, @RestController ..

- @EnableAutoConfiguration
  - 중요
  - Spring Boot의 meta 파일을 읽어서, 미리 정의되어 있는 자바 설정 파일(@Configuration)들을 빈으로 등록하는 역할을 수행
  - spring.factories 라는 Spring Boot의 meta파일을 읽어들임

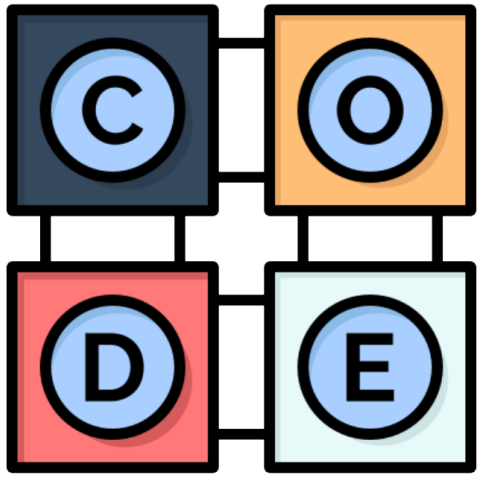
Spring Boot 테스트

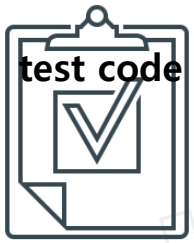
테스트 코드의 필요성

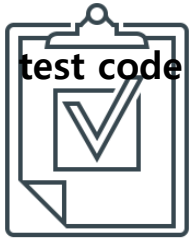


시스템의 안정성을 향상시키기 위해 필요한 작업은?

테스트 코드의 필요성





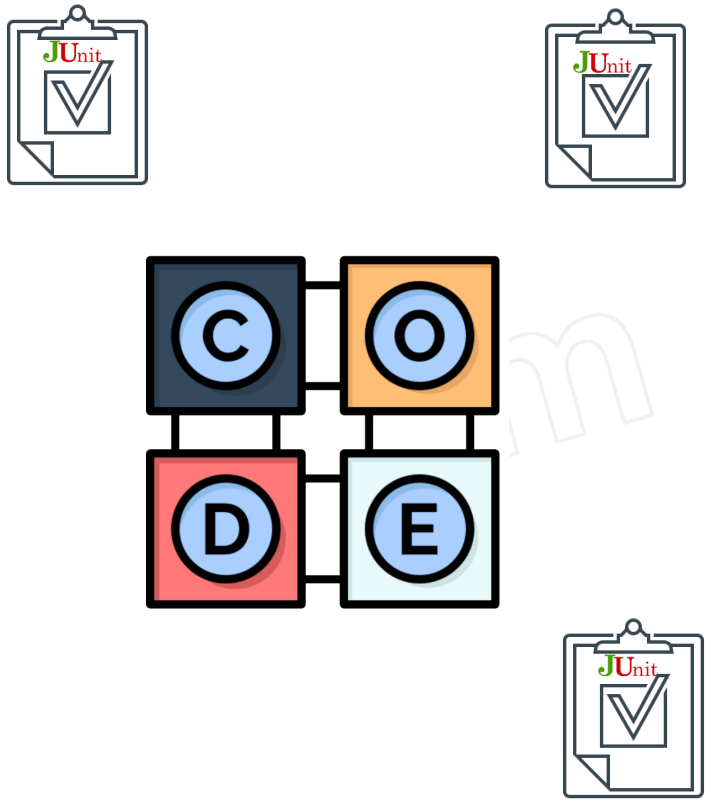


단위 테스트 필요

유지 보수 과정에서 재사용되어

시스템의 안정성을 보장하는 중요한 장치

테스트 코드 적용 방식



일관성 있는 단위 테스트를 진행

단위 test란?

작성한 클래스에 대한 test로서 test 단계 중 가장 기본

단위 test가 효율적으로 진행되기 위해서는 test 할 객체가 최대한 단순해야 함

선호하는 단위 테스트 framework - JUnit

단위 테스트 코드의 장점

- 개발 단계 초기에 문제를 발견하게 도와줌
- 개발자가 나중에 코드를 리팩토링하거나 라이브러리 업그레이드 등에서 기존 기능이 올바르게 작동하는지 확인 할 수 있음
- 기능에 대한 불확실성을 감소 시킬 수 있음
- 시스템에 대한 실제 문서를 제공(단위 테스트는 자체가 문서로 사용할 수 있음)

테스트 코드의 필요성

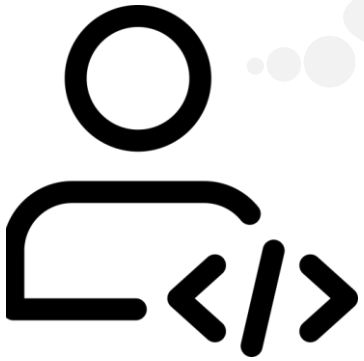
- 주의 사항

test에 대한 구체적인 가이드나 정책이 없는 상태에서 개발자에게 test를 위임할 경우

test에 대한 일괄성 유지가 어렵다!!!

비효율적인 test로 인해 test를 이해하거나
수정하는데 많은 시간이 걸린다!!!

테스트 코드의 필요성



웹 애플리케이션 개발시 controller가
정상적으로 동작하는지 확인하기 위해서는?

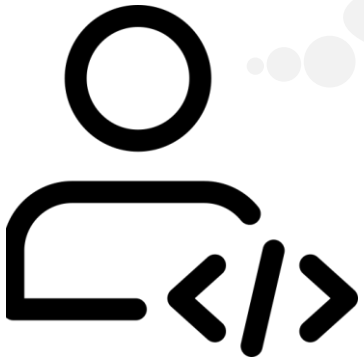
1단계 : Servlet Container가 구동되어야 함

2단계 : 브라우저로 요청/응답 결과 확인

고려 사항

- 수정시마다 매번 브라우저를 통해서 결과 확인은 번거로운 일
- 일반 Model등에서 Test 하는 것과 달리 Web상의 단위 Test는 HttpServletRequest와 HttpServletResponse의 내용 검증도 필요

테스트 코드의 필요성



웹 애플리케이션 개발시 controller가
정상적으로 동작하는지 확인하기 위해서는?

1단계 : Servlet Container가 구동되어야 함

2단계 : 브라우저로 요청/응답 결과 확인

고려 사항

- 수정시마다 매번 브라우저를 통해서 결과 확인은 번거로운 일

해결책

- **서버 구동없이 컨트롤러만 단독으로 테스트 하거나, 컨트롤러와 연관된 비즈니스 컴포넌트 실행 없이 컨트롤러만 독립적으로 테스트 할 수 있는 환경 필요 - MockMvc 라는 객체 필요**

Spring Boot 에서의 테스트 방식

테스트 스타터

MokeMvc

테스트 스타터 방식

Spring Boot에서 test 하기

- 기본 테스트 클래스 이해하기
  - Spring Boot를 이용하여 Project를 생성하면 test starter는 자동으로 추가

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

```java
package kr.pe.khk;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class SpringBootProjectApplicationTests {

    @Test
    void contextLoads() {
    }

}
```

Test 케이스가 실행될 때 test에 필요한 모든 설정과 빈들을 자동으로 초기화 하는 역할 수행

Spring Boot 기본 테스트 클래스 이해 및 활용

사용자가 작성한 빈과 자동설정 빈들을 모두 초기화

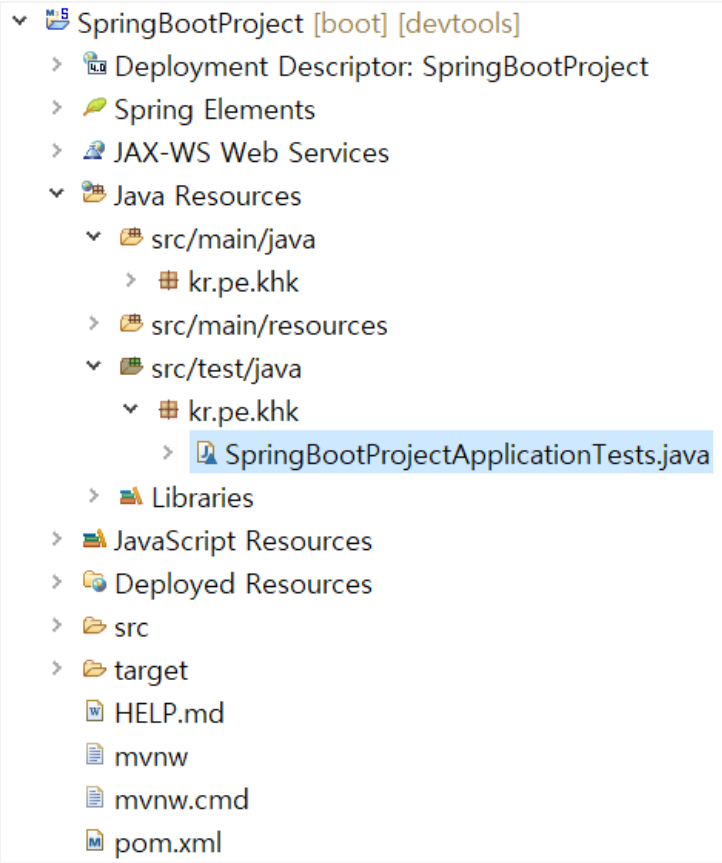
@SpringBootTest

- main 클래스에 선언된 @SpringBootApplication과 흡사한 기능
- **테스트 케이스가 실행될 때 테스트에 필요한 모든 설정과 빈들을 자동으로 초기화**
- 각 속성의 의미

속성	의미
properties	테스트가 실행되기 전에 테스트에 사용할 property 들을 key=value 형식으로 추가하거나, properties 파일에 설정된 property를 재정의
classes	테스트할 클래스들을 등록 classes 속성 생략시에 애플리케이션에 정의된 모든 빈들을 생성함
webEnvironment	애플리케이션이 실행 될 때, 웹과 관련된 환경을 설정 할 수 있음

Spring Boot 기본 테스트 클래스 이해 및 활용

- 기본 test 클래스 이해하기
  - Project 생성시 src/test/java 소스 폴더에 간단한 test 케이스 제공

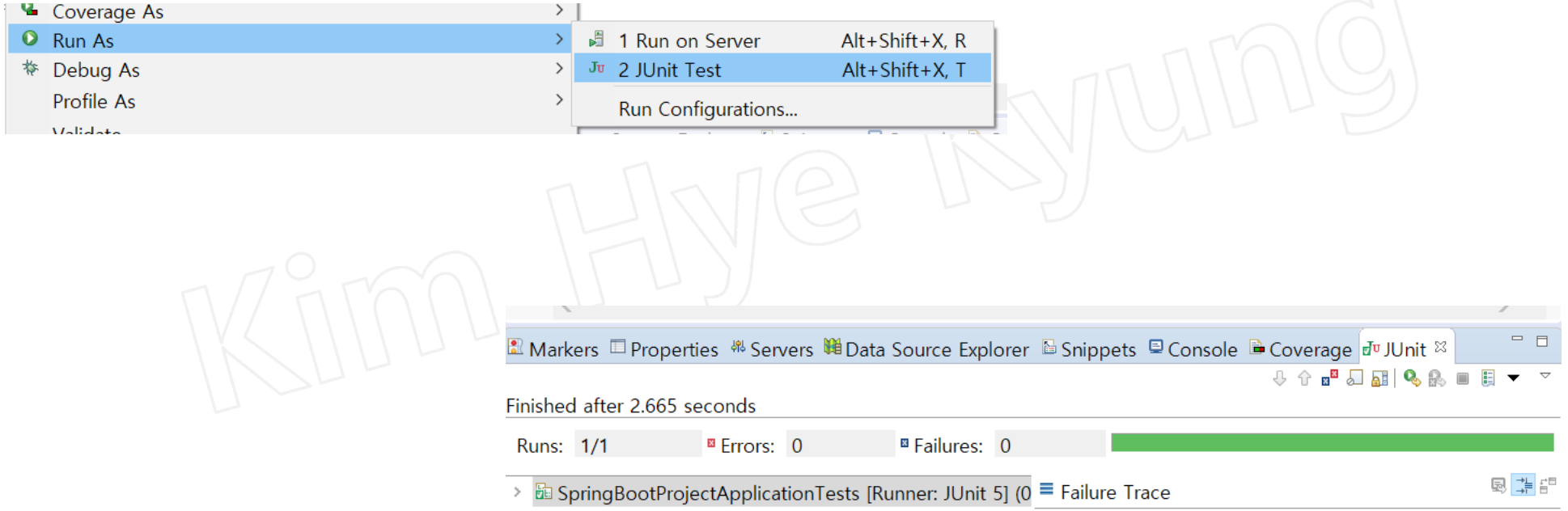



```java
package kr.pe.khk;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class SpringBootProjectApplicationTests {

    @Test
    void contextLoads() {
    }

}
```

Test 케이스가 실행될 때 test에 필요한 모든 설정과 빈들을 자동으로 초기화 하는 역할 수행

Spring Boot 기본 테스트 클래스 이해 및 활용

- 테스트 케이스 실행하기 & Junit 뷰를 통해서 결과 확인



Spring Boot 기본 테스트 클래스 이해 및 활용

@SpringBootTest

```
package kr.pe.khk;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class SpringBootProjectApplicationTests {

    @Test
    void contextLoads() {
    }

}
```

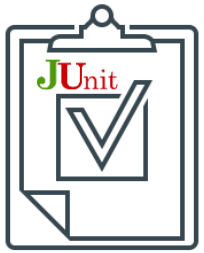
- 실행시 BootController 객체 생성

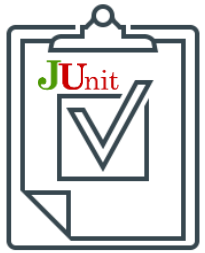
```
@RestController
public class BootController {

    public BootController() {
        System.out.println("***** BootController *****");
    }
```

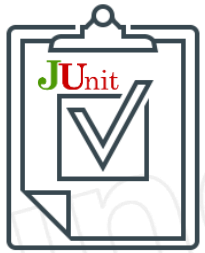
```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.7.RELEASE)

2020-06-08 16:42:51.704  INFO 1580 --- [
2020-06-08 16:42:51.712  INFO 1580 --- [
***** BootController *****
2020-06-08 16:42:52.792  INFO 1580 --- [
2020-06-08 16:42:53.061  INFO 1580 --- [
2020-06-08 16:42:53.273  INFO 1580 --- [ext
```

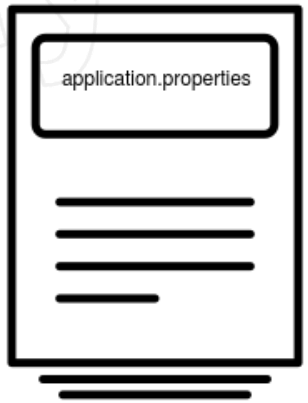
test에서 **공통으로 사용하는 데이터들이 존재할** 경우 효율적인 개발 방법은?

Spring Boot에서 test 하기



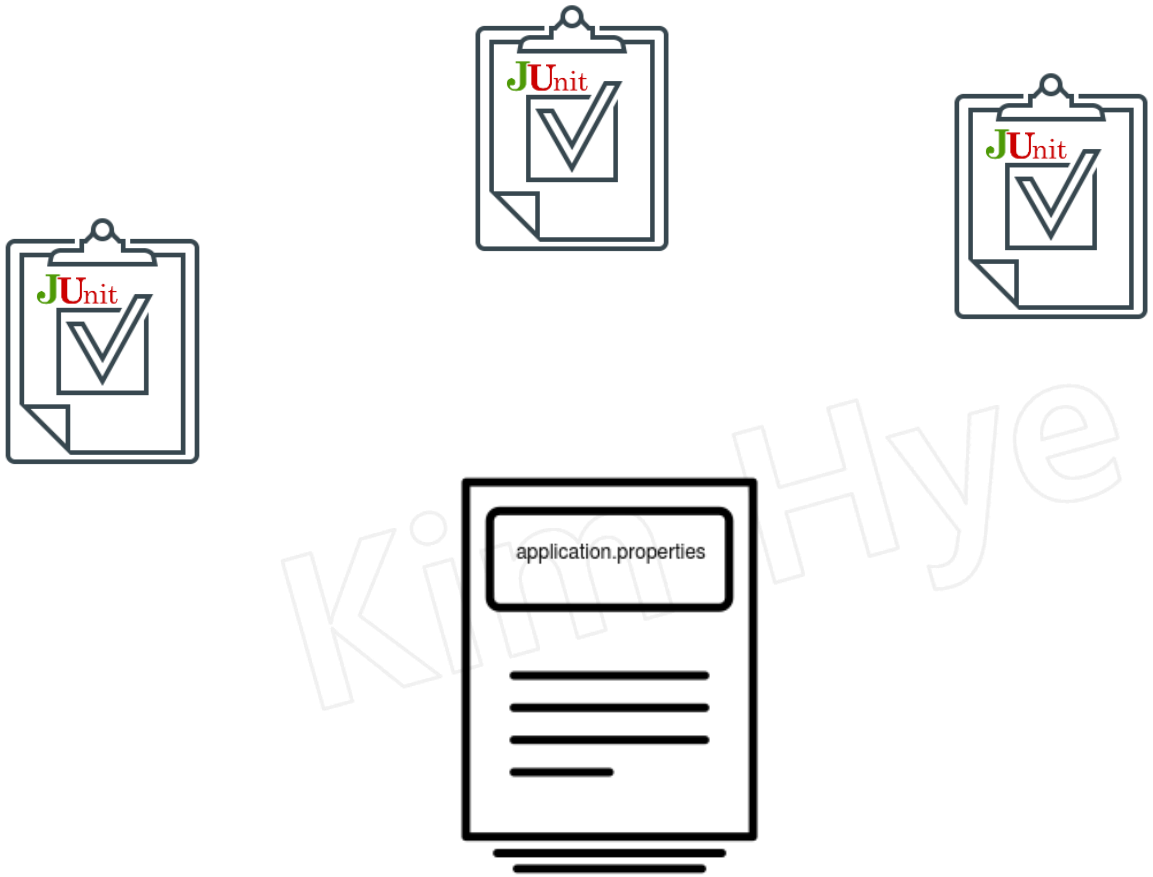






외부 properties file 사용

Spring Boot에서 test 하기



test에서 공통으로 사용하는 데이터

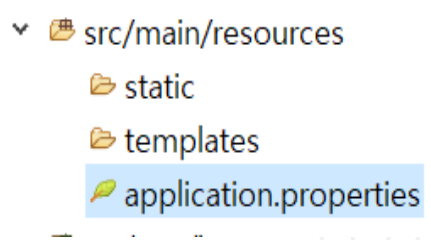
장점

- test 데이터를 재사용 하거나 변경이 용이

Spring Boot에서 test 하기

@SpringBootTest

- 외부 properties file을 사용한 개발




```
1 # public test data
2
3 author.name=khk
4 author.job=IT Man
5
```

```
9  @SpringBootTest
10 class SpringBootProjectApplicationTests {
11
12     @Autowired
13     Environment environment;
14
15     //@Test
16     //void contextLoads() {} //bean 객체 자동 생성
17
18     /* 1. bean 객체 자동 생성
19      * 2. application.properties 파일에 설정된 여러 test에서 공통적으로 사용되는 데이터를 활용
20      */
21     @Test
22     public void testMethod() {
23         System.out.println("이름 : " + environment.getProperty("author.name"));
24         System.out.println("직종 : " + environment.getProperty("author.job"));
25     }
26
27 }
```

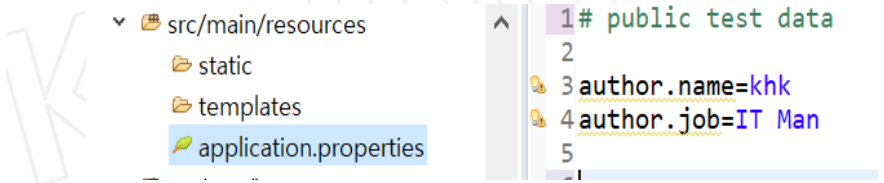
```
2020-06-11 11:19:33.432  INFO 12360 --- [
2020-06-11 11:19:33.432  INFO 12360 --- [
***** BootController *****
2020-06-11 11:19:34.494  INFO 12360 --- [
2020-06-11 11:19:34.767  INFO 12360 --- [
이름 : khk
직종 : IT Man
2020-06-11 11:19:34.986  INFO 12360 --- [ext
```

Spring Boot에서 test 하기

@SpringBootTest

- 외부 properties file 내용 재정의 & 새로운 property 추가

```
//classes - test할 class 지정
//properties - application.properties file 설정 재정의 & 새로운 property 추가
@SpringBootTest(classes=BootController.class,
                properties= {"author.name=Kim",
                             "author.job=IT Woman",
                             "author.nation=Korea"})
class SpringBootProjectApplicationTests {

    @Autowired
    Environment environment;


    /* 1. bean 객체 자동 생성
     * 2. application.properties 파일에 설정된 여러 test에서 공통적으로 사용되는 데이터들 활용
     */
    @Test
    public void testMethod() {
        System.out.println("이름 : " + environment.getProperty("author.name"));
        System.out.println("직종 : " + environment.getProperty("author.job"));
        System.out.println("직종 : " + environment.getProperty("author.nation"));
    }

}
```




```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.7.RELEASE)

2020-06-11 12:19:29.788  INFO 5040 --- [
2020-06-11 12:19:29.789  INFO 5040 --- [
***** BootController *****
2020-06-11 12:19:29.990  INFO 5040 --- [
이름 : Kim
직종 : IT Woman
직종 : Korea
```

Spring Boot에서 실행하기

@SpringBootApplication

- Spring Boot의 자동 설정, Spring bean 읽기와 생성을 모두 자동으로 설정
- 선언 위치
  - 애노테이션이 있는 위치부터 설정을 읽기 때문에 project의 최상단에 위치해야만 함

```
@SpringBootApplication
public class SpringBootTestApplication {

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

}
```

MockMvc 테스트 방식

Mock란?

Mock 이란?

'테스트를 위해 만든 모형'

Mocking 이란?

테스트를 위해 실제 객체와
비슷한 모의 객체를 만드는 것

Mock-up 이란?

Mocking한 객체를
메모리에서
얻어내는 과정

Web Application 테스트의 애로 사항

- 객체를 테스트 하기 위해서는 테스트 대상의 객체가 메모리에 있어야 함
- 생성하는 데 복잡한 절차가 필요하거나 많은 시간이 소요되는 객체의 테스트는 쉽지 않음
- 웹 애플리케이션의 컨트롤러처럼 WAS나 다른 SW의 도움이 필요한 객체의 테스트도 쉽지 않음
- 의존 관계가 복잡한 객체는 당연히 테스트 과정도 복잡하고 어려움

Web Application 테스트의 애로 사항 & **해결책**

- 객체를 테스트 하기 위해서는 테스트 대상의 객체가 메모리에 있어야 함
- 생성하는 데 복잡한 절차가 필요하거나 많은 시간이 소요되는 객체의 테스트는 쉽지 않음
- 웹 애플리케이션의 컨트롤러처럼 WAS나 다른 SW의 도움이 필요한 객체의 테스트도 쉽지 않음
- 의존 관계가 복잡한 객체는 당연히 테스트 과정도 복잡하고 어려움

- **테스트를 쉽게 하기 위한 해결책 :**
  - **테스트 하려는 실제 객체와 비슷한 가짜 객체를 만들어서 테스트에 필요한 기능만 가지도록 모킹 하면서 테스트**

Mocking 이란?

테스트를 위해 실제 객체와 비슷한 모의 객체를 만드는 것

Web Application 테스트 방식

- 웹 환경상에서 Controller를 Test 하려면?
  - 전제조건
    - Servlet Container 구동 & DispatcherServlet 객체 메모리에 로딩

- Servlet Container를 모킹한다는 의미?
  - 실제 Servlet Container가 아닌 테스트용 모형 Container를 사용하여 간단하게 테스트
  - 적용 기술
    - **MockMvc 객체 생성**
    - **브라우저나 WAS의 동작을 동일하게 처리해 줄 수 있는 환경 구성**
    - **브라우저에서 발생하는 요청을 가상으로 만들고 컨트롤러가 응답하는 내용을 기반으로 검증 수행**
    - @WebMvcTest 또는 @AutoConfigureMockMvc 사용

MockMvc 수행 방식

메소드	기능
perform	요청 즉 가상의 request 처리 요청은 MockHttpServletRequestsBuilder를 통해서 생성
expect	결과 즉 가상의 response에 대한 검증
do	테스트 과정에서 콘솔 출력 등 직접 처리 할 일을 작성

MockMvc 요청 만들기

메소드	기능
param/params()	요청 파라미터 설정
header/headers()	요청 헤더 설정
cookie()	쿠키 설정
requestAttr()	request scope에 attribute 설정
sessionAttr()	session scope에 attribute 설정
characterEncoding()	요청의 인코딩 설정
…	

메소드 chaining 문법으로 적용

MockMvc 응답 상태 코드 검증

- controller의 동작을 test 하기 위해서는 요청도 중요하지만 사실 controller가 어떤 결과를 전송했는지 검증하는 것이 가장 중요
- MockMvcResultMatchers
  - server의 응답 결과를 검증할 수 있는 메소드 제공

메소드	로 직
idOk()	응답 상태 코드가 정상인지 확인, 정상 응답 코드 : 200
isNotFound()	404 NotFound 여부 확인
isMethodNotAllowed()	405 메소드 불일치 여부 확인
isInternalServerError()	500 여부 확인
is(int status)	몇 번 응답 상태 코드가 설정되어 있는지 확인

MockMvc 실행하기

메소드	로 직
print()	실행 결과를 표준 출력을 이용해 출력

Web Application 테스트 방식

- controller 테스트시
  - servlet container를 mocking 하기 위해서는 @WebMvcTest 또는 @AutoConfigureMock 사용

@WebMvcTest	@AutoConfigureMock
@Controller & @RestController 애노테이션에 한해서 스프링빈으로 자동 등록	@SpringBootTest (webEnvironment=WebEnvironment.MOCK) 설정으로 모킹한 객체를 의존 받기 위한 애노테이션

@WebMvcTest

```java
package kr.pe.khk.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class WebMvcTestController {

    @GetMapping("/webMvcTest")
    public String webMvcTestProcess(@RequestParam String name) {
        System.out.println("이름 : " + name);
        return "요청 받고 응답하는 메소드의 데이터";
    }
}
```

```java
@RunWith(SpringRunner.class)
@WebMvcTest
public class WebMvcTestControllerTests {

    // 서블릿 컨테이너를 모킹
    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testHello() throws Exception{
        mockMvc.perform(get("/webMvcTest").param("name", "유재석"))
        .andExpect(status().isOk())
        .andDo(print());
    }
}
```

```
이름 : 유재석

MockHttpServletRequest:
      HTTP Method = GET
      Request URI = /webMvcTest
       Parameters = {name=[유재석]}
          Headers = []
             Body = <no character encoding set>
    Session Attrs = {}

Handler:
             Type = kr.pe.khk.controller.WebMvcTestController
           Method = kr.pe.khk.controller.WebMvcTestController#webMvcTestProcess(String)

Async:
    Async started = false
     Async result = null

Resolved Exception:
             Type = null

ModelAndView:
        View name = null
             View = null
            Model = null

FlashMap:
       Attributes = null

MockHttpServletResponse:
           Status = 200
    Error message = null
          Headers = [Content-Type:"text/plain;charset=UTF-8", Content-Length:"49"]
     Content type = text/plain;charset=UTF-8
             Body = 요청 받고 응답하는 메소드의 데이터
    Forwarded URL = null
   Redirected URL = null
          Cookies = []
```

@AutoConfigureMock

```java
@RestController
public class BoardController {

    @Autowired
    private BoardService boardService;

    @GetMapping("/hello")
    public String hello(String name) {
        return boardService.hello(name);
    }
}
```

```java
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment=WebEnvironment.MOCK)
@AutoConfigureMockMvc
public class BoardControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private BoardService boardService;

    @Test
    public void testHello() throws Exception {
        when(boardService.hello("유재석")).thenReturn("Hello : 유재석");

        //브라우저에서 서버에 URL 요청을 하듯 controller 실행
        //andExpect() : server의 응답 결과 검증
        //컨트롤러 동작 test도 중요하나 controller가 어떤 결과를 전송했는지 검증하는 것이 가장 중요
        mockMvc.perform(get("/hello").param("name", "유재석"))
        .andExpect(status().isOk())
        .andExpect(content().string("Hello : 유재석"))
        .andDo(print());
    }
}
```

```
MockHttpServletRequest:
      HTTP Method = GET
      Request URI = /hello
       Parameters = {name=[유재석]}
          Headers = []
             Body = null
    Session Attrs = {}

Handler:
             Type = kr.pe.khk.controller.BoardController
           Method = kr.pe.khk.controller.BoardController#hello(String)

Async:
    Async started = false
     Async result = null

Resolved Exception:
             Type = null

ModelAndView:
        View name = null
             View = null
            Model = null

FlashMap:
       Attributes = null

MockHttpServletResponse:
           Status = 200
    Error message = null
          Headers = [Content-Type:"text/plain;charset=UTF-8", Content-Length:"17"]
     Content type = text/plain;charset=UTF-8
             Body = Hello : 유재석
    Forwarded URL = null
   Redirected URL = null
          Cookies = []
```

응답 데이터 검증

```
    @Test
    public void testHello() throws Exception{
        mockMvc.perform(get("/webMvcTest").param("name", "유재석"))
        .andExpect(status().isOk())   //200 여부 검증
        .andExpect(content().string("유재석---")) //응답 본문의 내용을 검증, controller에서 유재석을 반환, 이 값이 맞는지 검증
        .andDo(print());
    }
}
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console | Coverage | JUnit | Terminal

hed after 2.164 seconds

ns: 1/1 | Errors: 0 | Failures: 1

WebMvcTestControllerTests [Runner: JUnit 5] (0.234 s)

testHello() (0.234 s)

Failure Trace

java.lang.AssertionError: Response content expected:<유재석---> but was:<유재석>

at org.springframework.test.util.AssertionErrors.fail(AssertionErrors.java:59)

at org.springframework.test.util.AssertionErrors.assertEquals(AssertionErrors.java:122)

at org.springframework.test.web.servlet.result.ContentResultMatchers.lambda$string$4(Co

at org.springframework.test.web.servlet.MockMvc$1.andExpect(MockMvc.java:196)

at kr.pe.khk.WebMvcTestControllerTests.testHello(WebMvcTestControllerTests.java:36)

at java.util.ArrayList.forEach(ArrayList.java:1257)

at java.util.ArrayList.forEach(ArrayList.java:1257)

응답된 데이터의 유효성 검증

Spring Boot 로깅

Spring Boot 로깅

- 로그

디버깅 할 때도 필요

실행중인 애플리케이션의
성능을 분석하거나
다양한 용도로 사용

Spring Boot 로깅

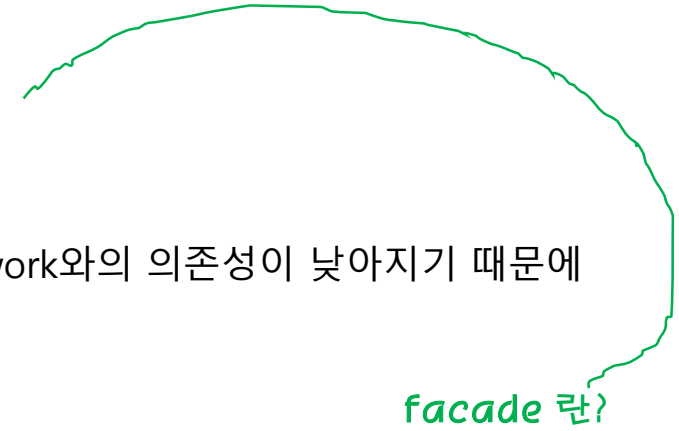
- Spring Boot 구동하면 다양한 레벨의 로그 메세지를 콘솔에 출력
  - 별도의 설정이 없을 경우 기본적으로 레벨은 INFO 로 처리
- 레벨별 의미

Level	의미
ERROR	사용자 요청을 처리하는 중 문제
WARN	처리 가능한 문제 향후 시스템 에러의 원인이 될 수 있는 문제
INFO	로그인이나 상태 변경과 같은 정보서 메세지
DEBUG	개발 시 디버깅 목적으로 출력하는 메세지
TRACE	DEBUG 레벨 보다 좀 더 상세한 메세지

특정 로그 레벨을 지정하면 상위 우선순위 로그가 모두 출력

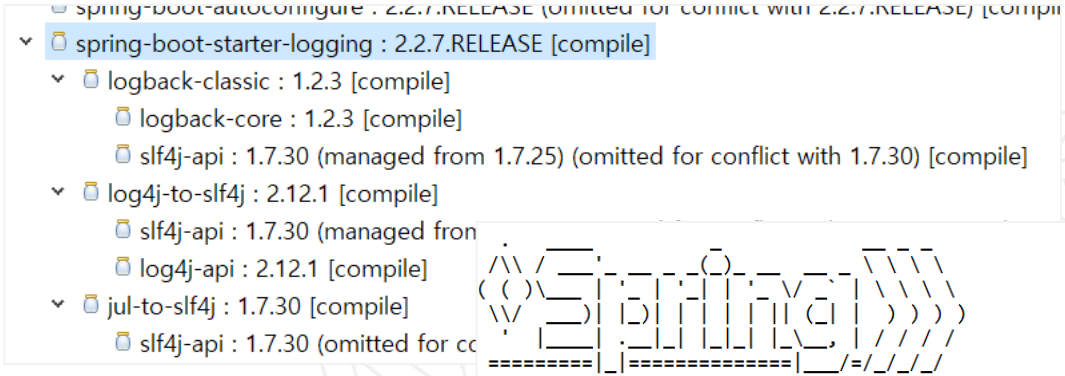
Spring Boot 로깅

- 사용되는 Log Framework
  - LogBack 사용
    - SLF4J(Simple Logging Facade for Java) 를 통해 LogBack 사용
  - SLF4J
    - 복잡한 로깅 프레임워크들을 쉽게 사용할 수 있도록 도와주는 facade
    - facade 이용시 장점
      - 복잡한 로깅 framework의 구조를 몰라도 쉽게 사용 할 수 있으며, framework와의 의존성이 낮아지기 때문에 쉽게 교체할 수 있음



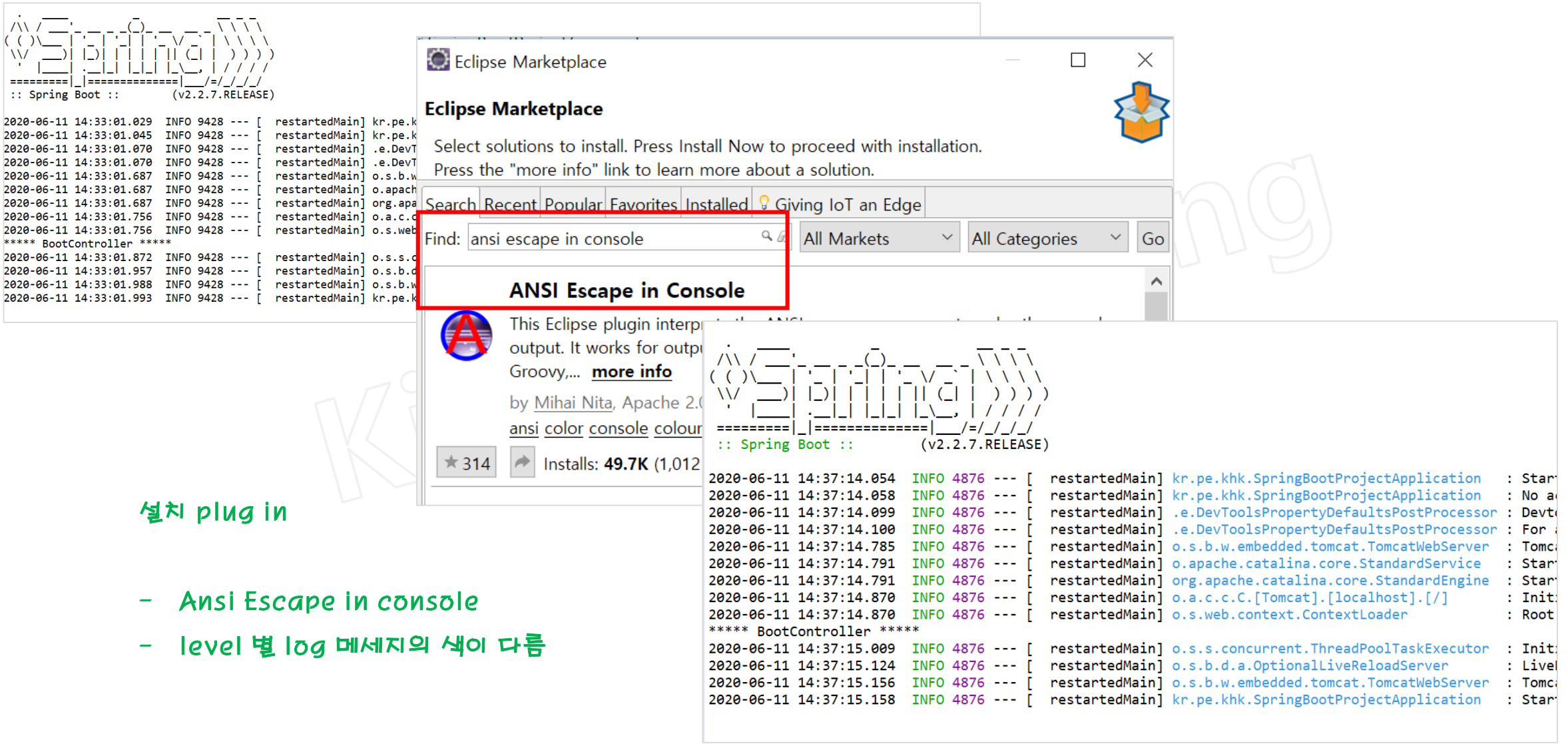
Spring Boot 로깅 library & 출력 형식

- Spring Boot로 생성한 Project에는 기본적으로 로깅 관련 library들이 내장되어 있음
- LogBack 을 이용하여 출력된 로그




```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.7.RELEASE)

2020-06-11 14:33:01.029  INFO 9428 --- [  restartedMain] kr.pe.khk.SpringBootProjectApplication   : Starting SpringBootProjectApplication on HyeKyung w
2020-06-11 14:33:01.045  INFO 9428 --- [  restartedMain] kr.pe.khk.SpringBootProjectApplication   : No active profile set, falling back to default prof
2020-06-11 14:33:01.070  INFO 9428 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devt
2020-06-11 14:33:01.070  INFO 9428 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting
2020-06-11 14:33:01.687  INFO 9428 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2020-06-11 14:33:01.687  INFO 9428 --- [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2020-06-11 14:33:01.687  INFO 9428 --- [  restartedMain] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.34]
2020-06-11 14:33:01.756  INFO 9428 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2020-06-11 14:33:01.756  INFO 9428 --- [  restartedMain] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization complete
***** BootController *****
2020-06-11 14:33:01.872  INFO 9428 --- [  restartedMain] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecut
2020-06-11 14:33:01.957  INFO 9428 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2020-06-11 14:33:01.988  INFO 9428 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context
2020-06-11 14:33:01.993  INFO 9428 --- [  restartedMain] kr.pe.khk.SpringBootProjectApplication   : Started SpringBootProjectApplication in 1.2 seconds
```

Spring Boot 로깅용 plug in



설치 plug in

- Ansi Escape in console
- level 별 log 메세지의 색이 다름

Spring Boot 레벨 수정

- 로그 레벨 수정
  - 별도의 설정이 없을 경우 기본적으로 레벨은 INFO 로 처리
  - application.properties
    - 설정 정보: logging.level.package명..=level

```
# Logging Setting
# logging.level.packagename...=level
logging.level.kr.pe.khk=debug
```

pakcage명

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.7.RELEASE)

2020-06-16 13:24:29.737  INFO 10532 --- [  restartedMain] kr.pe.khk.SpringBootProjectApplication   : Sta
2020-06-16 13:24:29.737 DEBUG 10532 --- [  restartedMain] kr.pe.khk.SpringBootProjectApplication   : Run
2020-06-16 13:24:29.737  INFO 10532 --- [  restartedMain] kr.pe.khk.SpringBootProjectApplication   : No
2020-06-16 13:24:29.888  INFO 10532 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tom
2020-06-16 13:24:29.889  INFO 10532 --- [  restartedMain] o.apache.catalina.core.StandardService   : Sta
2020-06-16 13:24:29.889  INFO 10532 --- [  restartedMain] org.apache.catalina.core.StandardEngine  : Sta
```

Spring Boot Application에 로그 활용 방법

- 로그 레벨별 메소드 활용 방법

```java
@Service
public class LoggingRunner implements ApplicationRunner {

    private Logger logger = LoggerFactory.getLogger(LoggingRunner.class);

    @Override
    public void run(ApplicationArguments args) throws Exception {
        logger.trace("TRACE log");
        logger.debug("DEBUG log");
        logger.info("INFO log");
        logger.warn("WARN log");
        logger.error("ERROR log");
    }
}
```

```
# Logging Setting
# logging.level.packagename...=level
logging.level.kr.pe.khk=debug
```

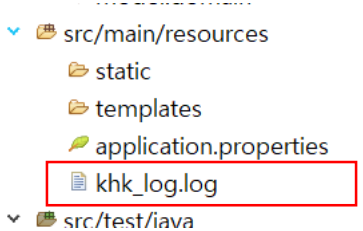
```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.7.RELEASE)

2020-06-16 13:41:38.842  INFO 13672 --- [  restartedMain] kr.pe.khk.SpringBootProjectApplication   : Starting SpringBo
2020-06-16 13:41:38.842 DEBUG 13672 --- [  restartedMain] kr.pe.khk.SpringBootProjectApplication   : Running with Spri
2020-06-16 13:41:38.842  INFO 13672 --- [  restartedMain] kr.pe.khk.SpringBootProjectApplication   : No active profile
2020-06-16 13:41:38.994  INFO 13672 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialize
2020-06-16 13:41:38.994  INFO 13672 --- [  restartedMain] o.apache.catalina.core.StandardService   : Starting service
2020-06-16 13:41:38.994  INFO 13672 --- [  restartedMain] org.apache.catalina.core.StandardEngine  : Starting Servlet
2020-06-16 13:41:39.024  INFO 13672 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spri
2020-06-16 13:41:39.024  INFO 13672 --- [  restartedMain] o.s.web.context.ContextLoader            : Root WebApplicati
2020-06-16 13:41:39.054  INFO 13672 --- [  restartedMain] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing Exe
2020-06-16 13:41:39.085  INFO 13672 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server
2020-06-16 13:41:39.093  INFO 13672 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started o
2020-06-16 13:41:39.093  INFO 13672 --- [  restartedMain] kr.pe.khk.SpringBootProjectApplication   : Started SpringBo
2020-06-16 13:41:39.093 DEBUG 13672 --- [  restartedMain] kr.pe.khk.LoggingRunner                  : DEBUG log
2020-06-16 13:41:39.093  INFO 13672 --- [  restartedMain] kr.pe.khk.LoggingRunner                  : INFO log
2020-06-16 13:41:39.093  WARN 13672 --- [  restartedMain] kr.pe.khk.LoggingRunner                  : WARN log
2020-06-16 13:41:39.093 ERROR 13672 --- [  restartedMain] kr.pe.khk.LoggingRunner                  : ERROR log
```

Spring Boot Logging 기록 출력 형식


```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.7.RELEASE)

2020-06-16 13:41:38.842  INFO 13672 --- [  restartedMain] kr.pe.khk.Spr
2020-06-16 13:41:38.842 DEBUG 13672 --- [  restartedMain] kr.pe.khk.Spr
2020-06-16 13:41:38.842  INFO 13672 --- [  restartedMain] kr.pe.khk.Spr
2020-06-16 13:41:38.994  INFO 13672 --- [  restartedMain] o.s.b.w.embed
2020-06-16 13:41:38.994  INFO 13672 --- [  restartedMain] o.apache.cata
2020-06-16 13:41:38.994  INFO 13672 --- [  restartedMain] org.apache.ca
2020-06-16 13:41:39.024  INFO 13672 --- [  restartedMain] o.a.c.c.C.[To
2020-06-16 13:41:39.024  INFO 13672 --- [  restartedMain] o.s.web.conte
2020-06-16 13:41:39.054  INFO 13672 --- [  restartedMain] o.s.s.concurr
2020-06-16 13:41:39.085  INFO 13672 --- [  restartedMain] o.s.b.d.a.Opt
2020-06-16 13:41:39.093  INFO 13672 --- [  restartedMain] o.s.b.w.embed
```

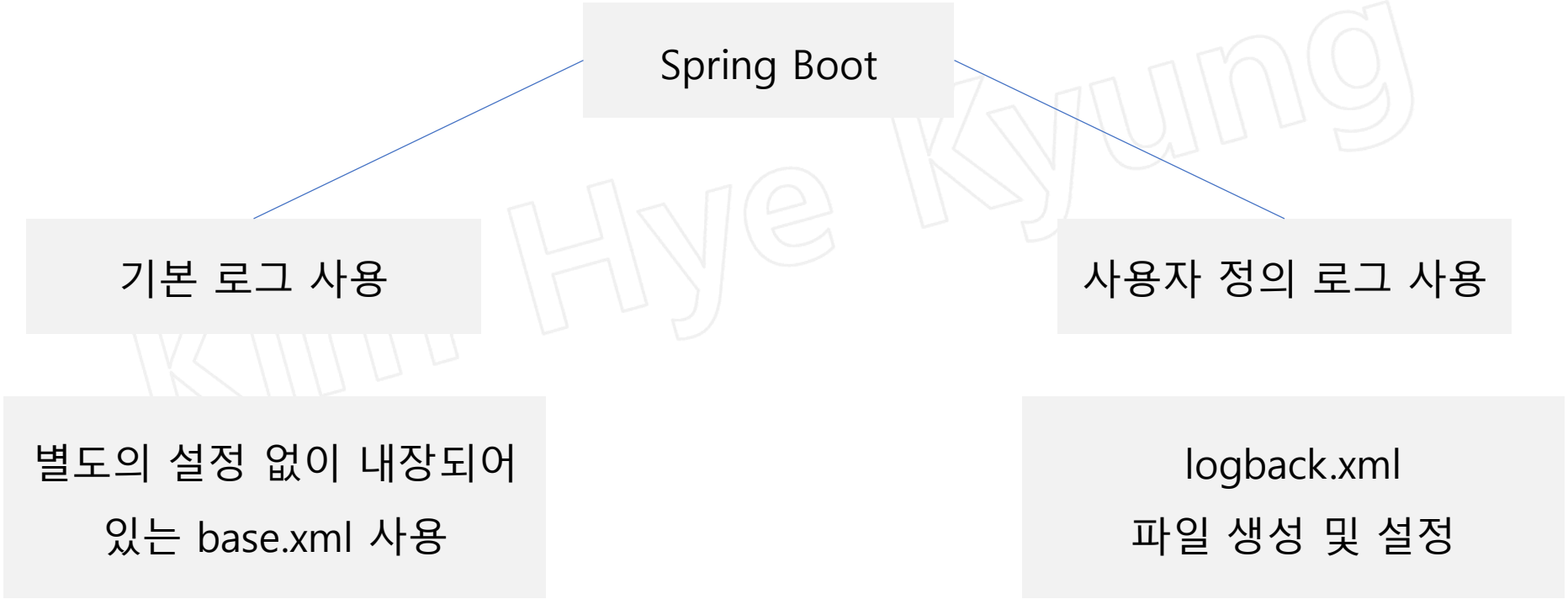
console에 기록

log file로 기록




```
 8
 9# Logging Setting
10# logging.level.packagename...=level
11logging.level.kr.pe.khk=debug
12
13# file output
14logging.file=src/main/resources/khk_log.log
```

Spring Boot Logging 수정



Spring Boot Logging : 사용자 정의 로그

- logback.xml

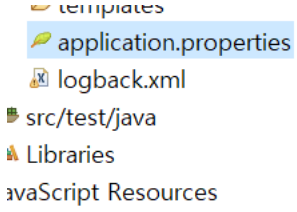
logback.xml

appender	logger
어디에 어떤 패턴으로 로그를 출력할 것인가?	작성할 애플리케이션에서 사용할 로거를 등록

```
<!-- fileAppender 라는 이름으로 RollingFileAppender 등록
    RollingFileAppender - 특정 파일에 로그를 출력하는 어펜더 -->
<appender name="fileAppender" class="ch.qos.logback.core.rolling.RollingFileAppender">

<!-- 콘솔에 로그 출력하는 어펜더 -->
<appender name="consoleAppender" class="ch.qos.logback.core.ConsoleAppender">
```

Spring Boot Logging : 사용자 정의 로그

- 사용자 정의 logback.xml 사용을 위한 개발




```
# Logging Setting
# logging.level.packagename...=level
#logging.level.kr.pe.khk=debug

# file output
#logging.file=src/main/resources/khk_log.log
```

pattern	의미
%d	시간
%date{format}	원하는 형태로 시간 정보 출력
%logger{length}	Logger 이름. {length}는 최대 자릿수, length에 따라 로거 이름이 축약됨
%thread	현재 thread명
%-5level	로그 레벨. 5는 출력 고정폭 값
%msg	로그 메시지
%n	개행(new line) 처리

```
<configuration>

    <!-- fileAppender 라는 이름으로 RollingFileAppender 등록
        RollingFileAppender - 특정 파일에 로그를 출력하는 어펜더 -->
    <appender name="fileAppender" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>src/main/resources/logs/khk_log.log</file>
        <!--
            1. <rollingPolicy> : 로깅 정책 설정
            2. TimeBasedRollingPolicy : 일정 시간 기준으로
            롤링되는 로그 파일 생성 3. <maxHistory> : 롤링 파일이 만들어지는 시간 기준 <maxHistory>1</maxHistory>
            정해진 시간이 지나면 기존의 로그 파일을 압축하고 새로운 로그 파일을 생성하라는 의미 -->

        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>
                src/main/resources/logs/khk_log.%d{yyyy-MM-dd}.log.gz
            </fileNamePattern>
            <maxHistory>1</maxHistory>
        </rollingPolicy>

        <!-- 출력할 로그의 패턴 지정 -->
        <encoder>
            <pattern>
                %d{yyyy:MM:dd HH:mm:ss.SSS} %-5level -- [%thread] %logger{30} : %msg %n
            </pattern>
        </encoder>
    </appender>

    <!-- 콘솔에 로그 출력하는 어펜더 -->
    <appender name="consoleAppender" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>
                %d{yyyy:MM:dd HH:mm:ss.SSS} %-5level -- [%thread] %logger{30} : %msg %n
            </pattern>
        </encoder>
    </appender>

    <logger name="kr.pe.khk" level="info" additivity="false">
        <appender-ref ref="consoleAppender" />
        <appender-ref ref="fileAppender" />
    </logger>

    <root level="info">
        <appender-ref ref="consoleAppender" />
    </root>

</configuration>
```

Spring Boot 빌드

빌드란?

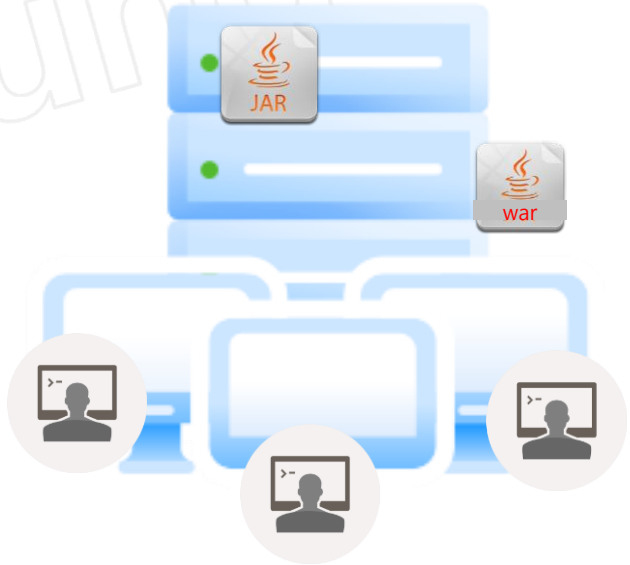
완성된 프로젝트에서 소스를 컴파일 하고

컴파일된 소스들을 적합한 위치에 취합하고

jar나 war 파일로 packaging 하여 운영서버에

배포하는 일련의 과정을 의미

빌드 process



빌드 file 종류

Jar ARchive

Standalone Java Project Packaging file

독립적으로 실행 가능한 애플리케이션을 빠르게 개발하는 것을 목표로 함

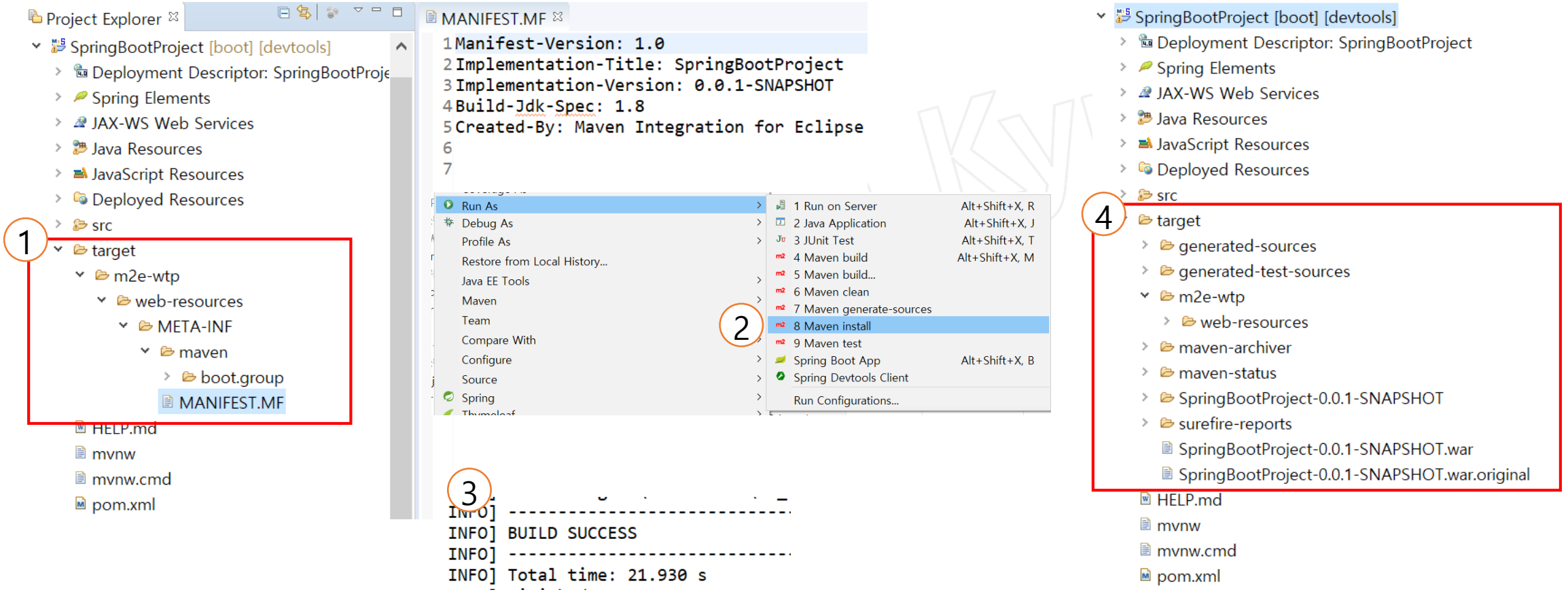
Web ARchive

Web Project Packaging file

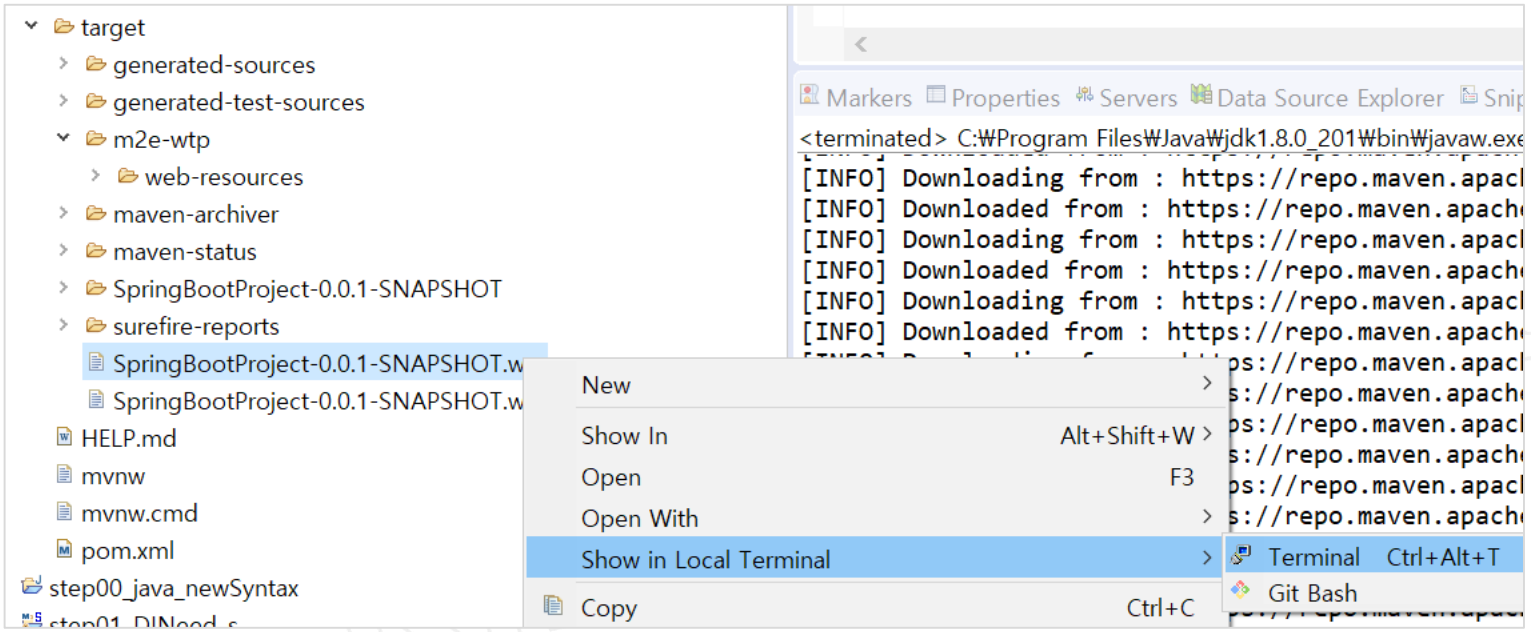
내부 디렉터리 구조와 복잡한 파일들의 위치도 신경 써야 함

따라서 war가 아닌 jar 파일로 패키징해서 사용 가능하게 지원

Project Packaging 하기



Project Packaging 파일 실행해 보기

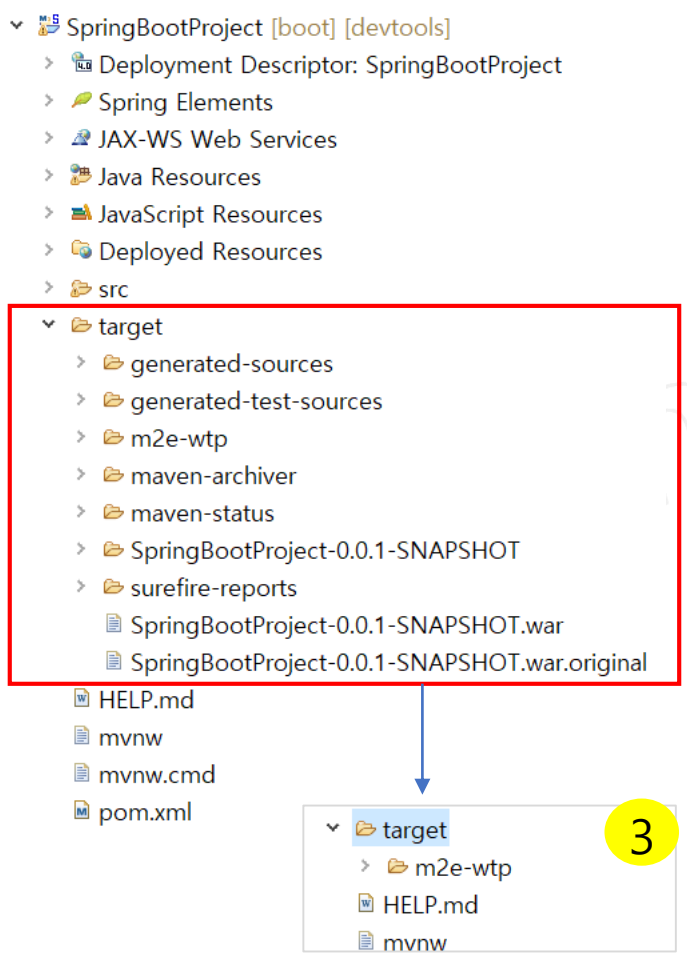


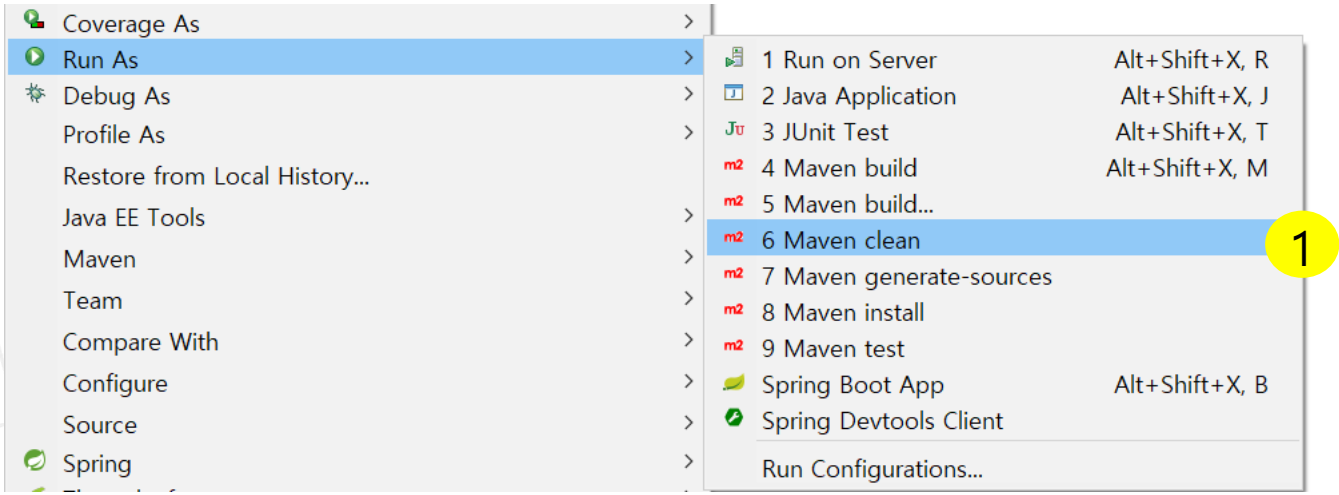
터미널 창 오픈 -> 다음 명령어 입력 -> 터미널 창 정상 실행 확인 -> 브라우저로 요청 및 응답 확인

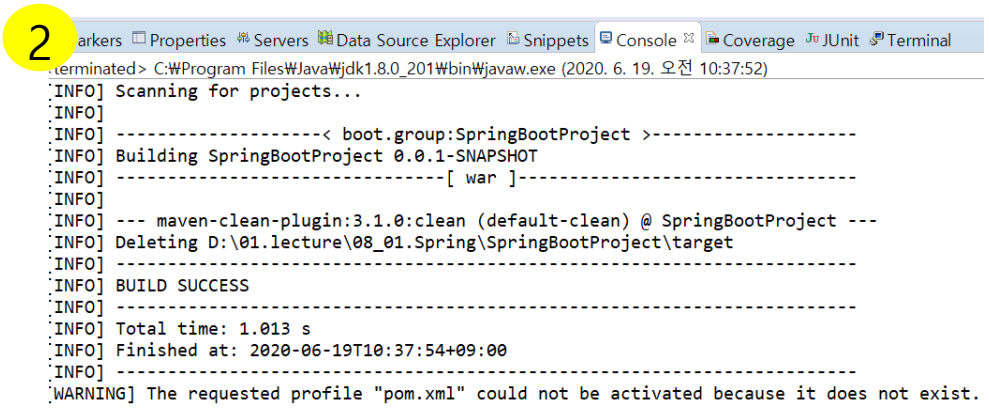
```
\SpringBootProject\target>java -jar SpringBootProject-0.0.1-SNAPSHOT.war
```

Packaging 결과 파일 삭제하기

- Packaging 결과 파일과 Packaging 과정에서 생성된 임시 파일과 폴더들 삭제







Spring Data JPA

Spring Data JPA란?

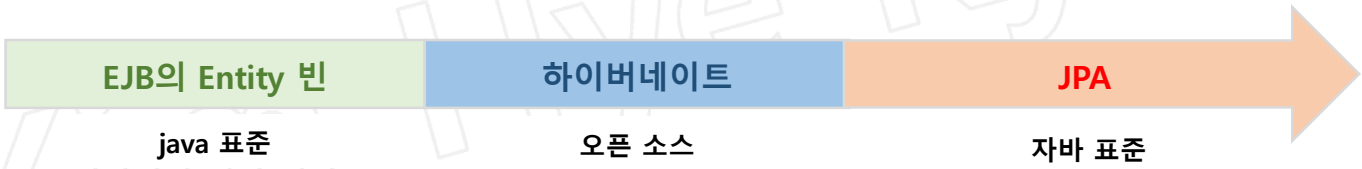
Spring Boot에서 JPA를 쉽게

사용할 수 있도록 지원하는 모듈

JPA History

- Java 기반의 DB연동 기술의 변천사

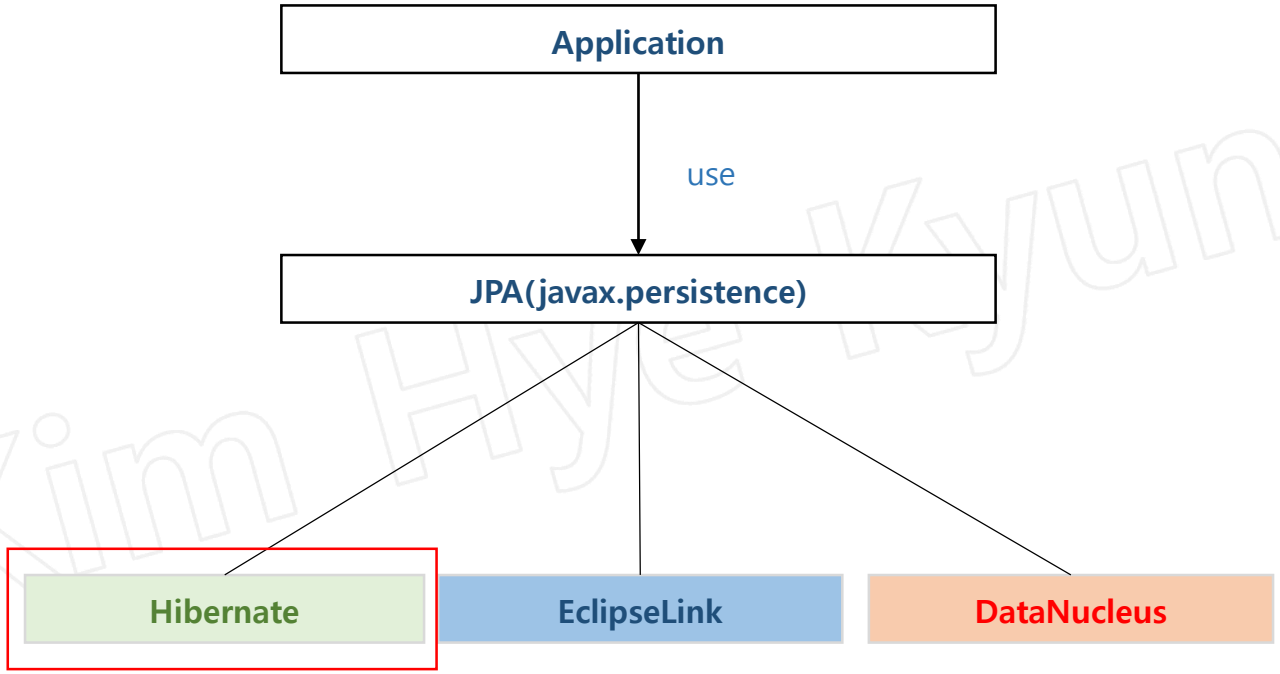
자바 표준은 아니지만 가장
많은 개발자들이 사용하는
ORM 기술



구현 및 배포의 어려움등의
여러가지 이슈

하이버네이트 개발자들이
중심이 되어 만든
ORM 표준 기술

JPA 구현체 종류



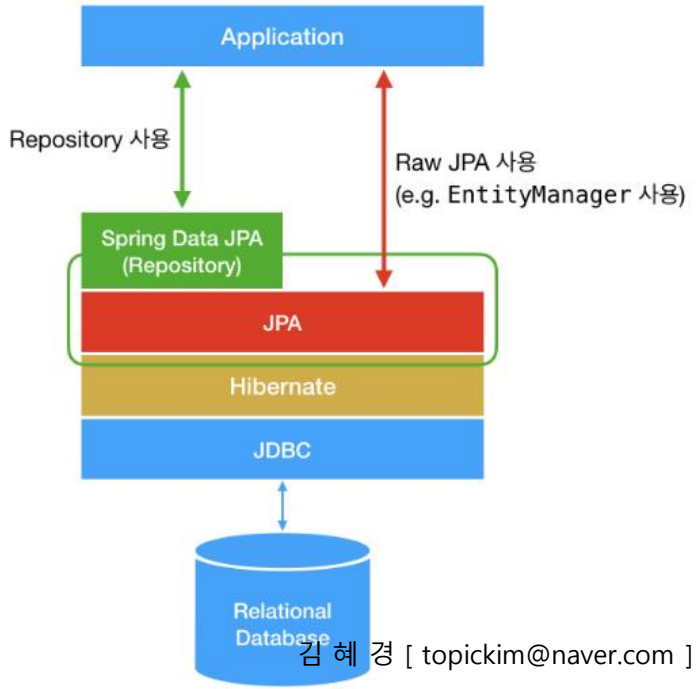
Spring Boot에서
사용하는 구현체

Spring Boot & JPA

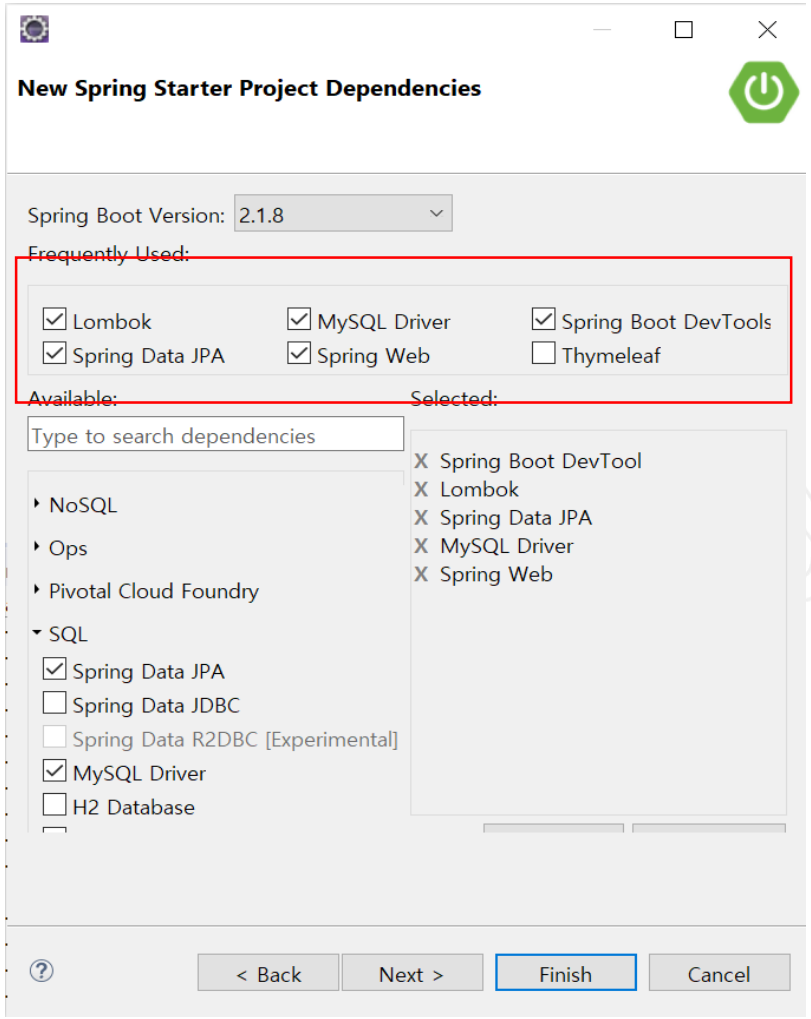
- JPA 연동에 필요한 library들과 복잡한 XML 설정을 자동으로 처리
  - 복잡한 JPA의 개념이나 동작 원리를 모르고도 쉽게 JPA 사용 가능
  - JPA 연동에 필요한 library들과 복잡한 XML 설정을 자동으로 처리하기 위해 JPA 스타터를 제공
  - Spring Data JPA를 사용하면 기본적으로 EntityManager가 활성화 되어 있는 상태

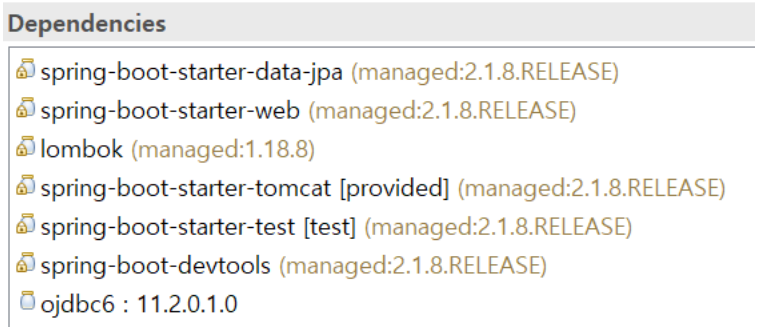
- 장점
  - 큰 어려움 없이 JPA 관련 의존성과 XML 설정 처리가 가능

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

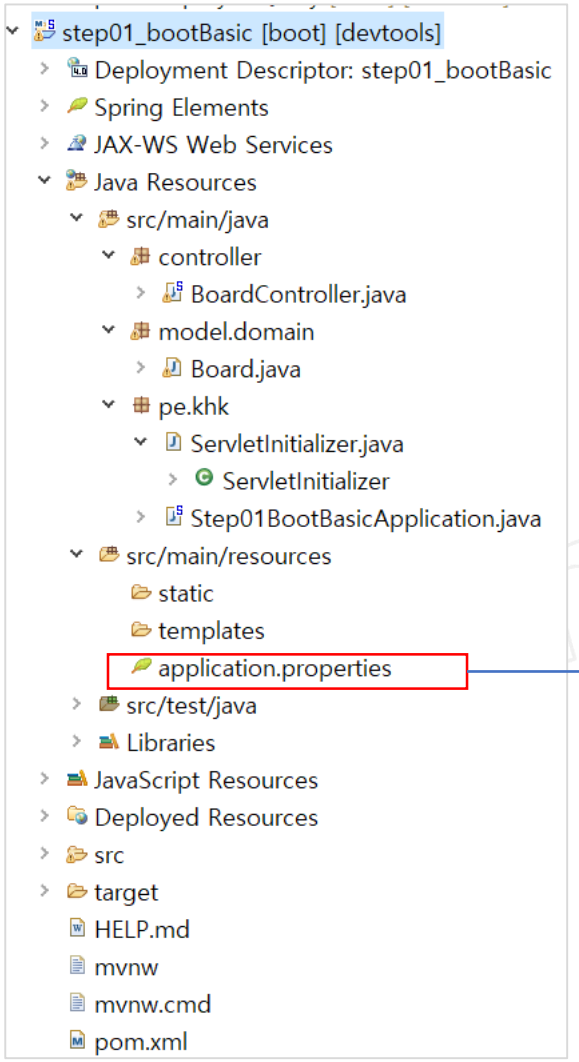


Spring Data JPA Project 생성 및 설정





Spring Data JPA 설정

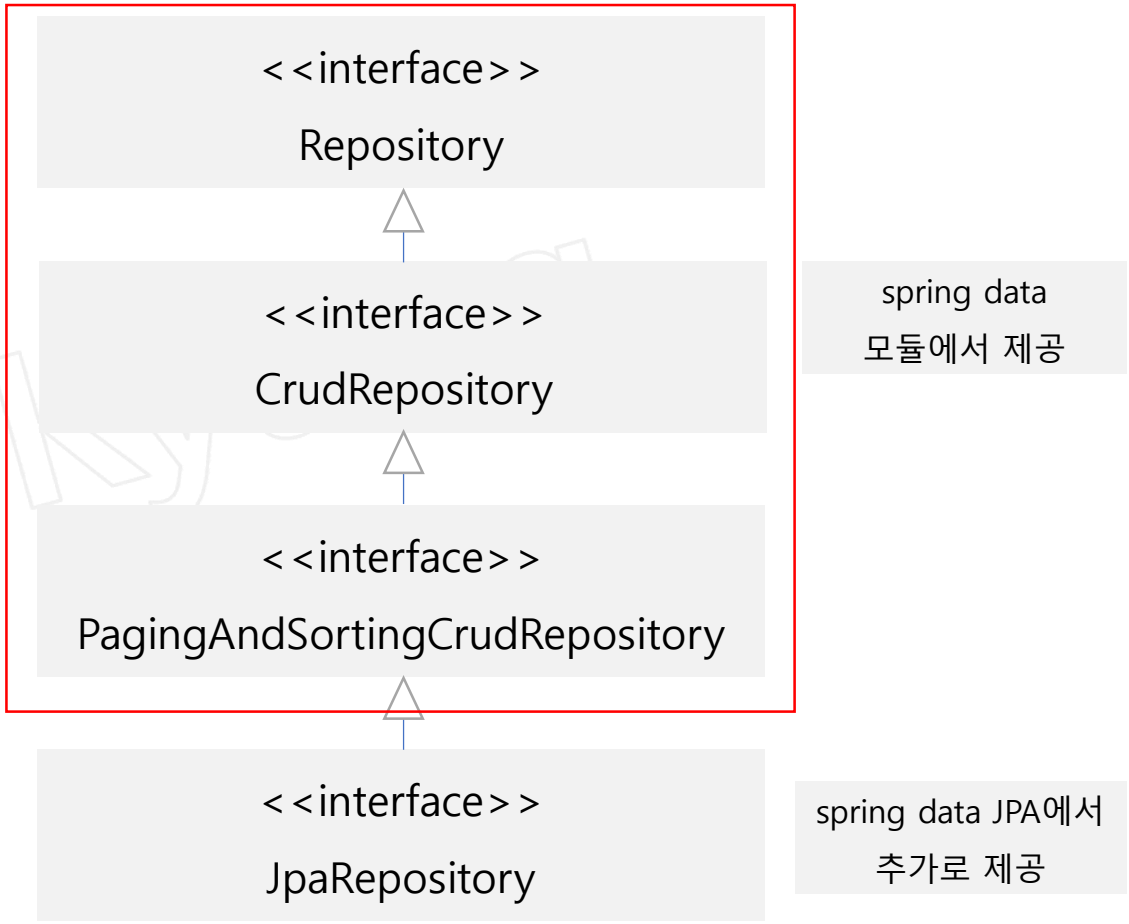



```
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@127.0.0.1:1521:xe
spring.datasource.username=SCOTT
spring.datasource.password=TIGER
spring.jpa.database-platform=org.hibernate.dialect.OracleDialect


spring.jpa.hibernate.ddl-auto=create
spring.jpa.generate-ddl=false
spring.jpa.show-sql=true
spring.jpa.database=oracle

logging.level.org.hibernate=info


server.port=8000
```

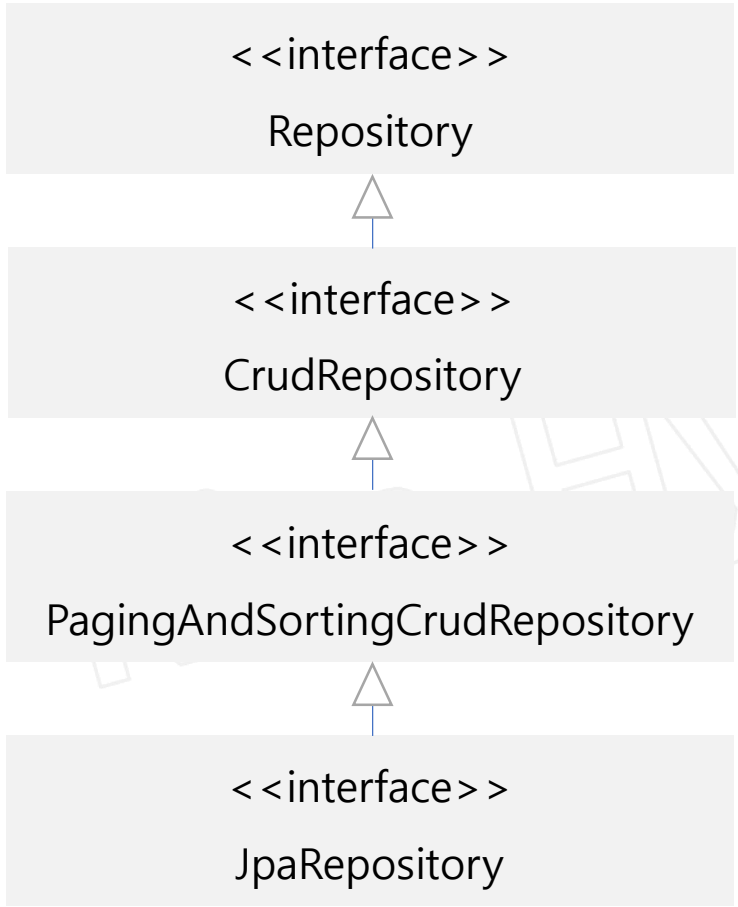
Spring Data JPA - Repository

- Repository interface 작성
  - 실제 DB와 연동
  - CRUD 기능을 처리할 Repository
  - DAO와 동일한 개념
  - 개발 방법
    - Spring에서 제공하는 Repository중 하나를 상속
    - Spring Data Jpa 사용시 별도의 하위 클래스 구현 없이 interface만 정의

Spring Boot가 내부적으로 interface에 대한 구현 객체를 자동으로 생성



Spring Data JPA - Repository



일반적으로 사용되는 interface

검색기능이 필요하고 검색 결과 화면에 대해 paging처리를 하고자 할 경우 사용

JPA에서 추가하는 기능을 사용하고자 할 경우

Spring Data JPA - Repository

- interface에 공통적으로 적용할 두가지 제네릭 타입

```
package model.dao;

import java.util.List;

import org.springframework.data.repository.CrudRepository;

import model.domain.Board;

public interface BoardRepository extends CrudRepository<Board, Long> {
    List<Board> findBoardByTitle(String titleData);
    List<Board> findBoardByTitleContaining(String titleData);
}
```

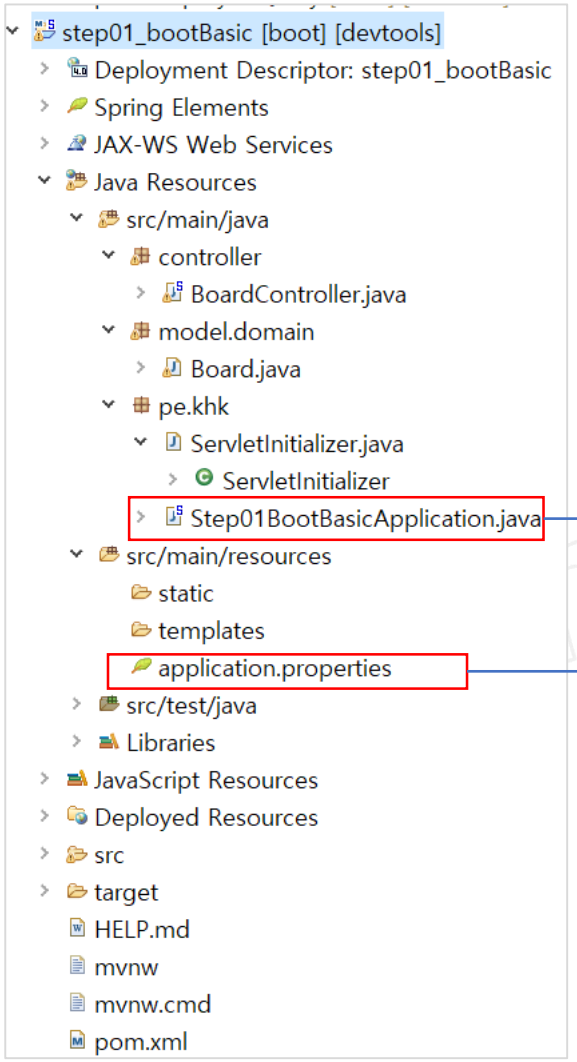
entity 식별 클래스 타입

식별자 타입

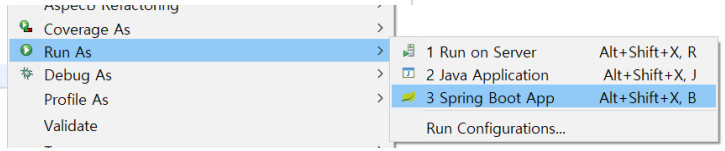
@Id로 매핑한 식별자 변수의 타입

Spring Boot 웹 애플리케이션 작성하기

Spring Boot 기본 예제 1

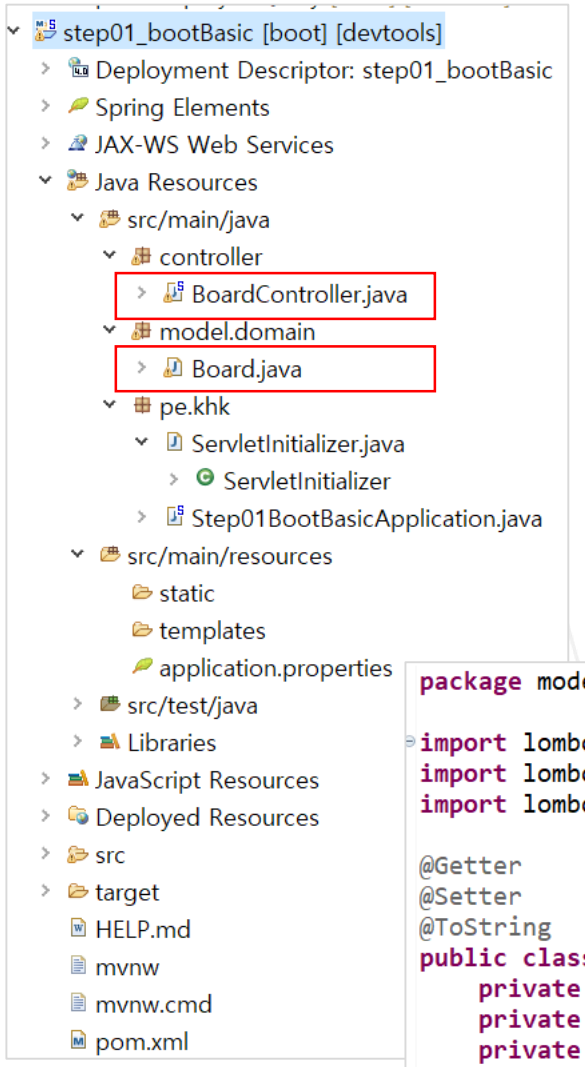



```java
package pe.khk;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@ComponentScan(basePackages= {"pe.khk", "controller"})
public class Step01BootBasicApplication {

    public static void main(String[] args) {
        SpringApplication.run(Step01BootBasicApplication.class, args);
    }

}
```



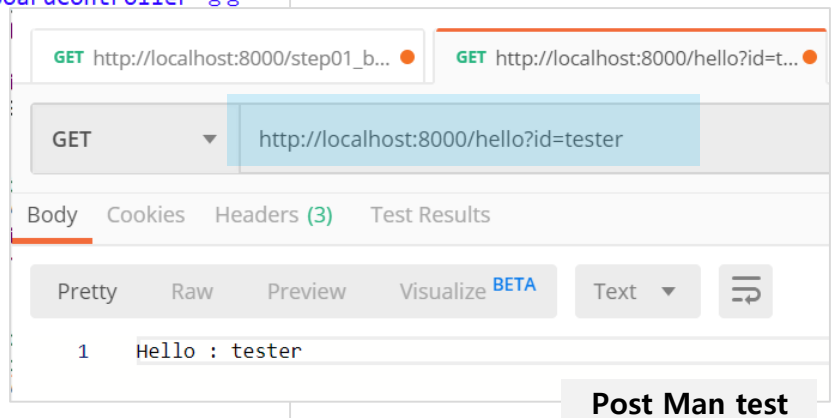

```
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@127.0.0.1:1521:xe
spring.datasource.username=SCOTT
spring.datasource.password=TIGER
spring.jpa.database-platform=org.hibernate.dialect.OracleDialect


spring.jpa.hibernate.ddl-auto=create
spring.jpa.generate-ddl=false
spring.jpa.show-sql=true
spring.jpa.database=oracle

logging.level.org.hibernate=info


server.port=8000
```

Spring Boot 기본 예제 1

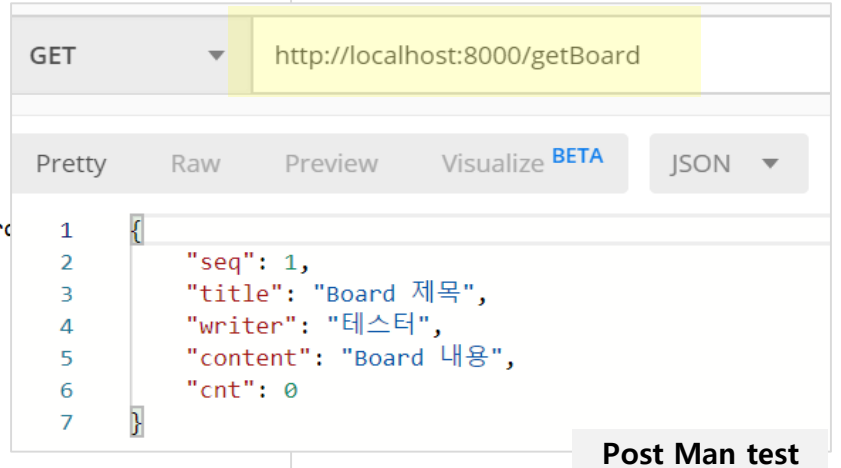



```java
package model.domain;

import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Getter
@Setter
@ToString
public class Board {
    private int seq;
    private String title;
    private String writer;
    private String content;
    private int cnt;
}
```

```java
@RestController
public class BoardController {

    public BoardController() {
        System.out.println("********************* BoardController 생성 ***
    }

    //http://localhost:8000/hello?id=test
    @GetMapping("/hello")
    public String hello(String id) {
        return "Hello : " + id;
    }

    //http://localhost:8000/getBoard
    //json 형태로 응답
    @GetMapping("/getBoard")
    public Board getBoard() {
        Board board = new Board();
        board.setSeq(1);
        board.setTitle("Board 제목");
        board.setWriter("테스터");
        board.setContent("Board 내용");
        board.setCnt(0);
        return board;
    }

    //http://localhost:8000/getBoardList
    @GetMapping("/getBoardList")
    public List<Board> getBoardList() {
        List<Board> boardList = new ArrayList<Boar
        for (int i = 1; i <= 10; i++) {
            Board board = new Board();
            board.setSeq(i);
            board.setTitle("제목" + i);
            board.setWriter("테스터");
            board.setContent(i + "번 내용입니다.");
            board.setCnt(0);
            boardList.add(board);
        }
        return boardList;
    }
}
```



Post Man test



Post Man test

Spring Boot 기본 예제 2

- Building REST services with Spring
  - https://spring.io/guides/tutorials/rest/

TUTORIAL

Building REST services with Spring

REST has quickly become the de-facto standard for building web services on the web because they're easy to build and easy to consume.

There's a much larger discussion to be had about how REST fits in the world of microservices, but - for this tutorial - let's just look at building RESTful services.

Why REST? REST embraces the precepts of the web, including its architecture, benefits, and everything else. This is no surprise given its author, Roy Fielding, was involved in probably a dozen specs which govern how the web operates.

What benefits? The web and its core protocol, HTTP, provide a stack of features:

- Suitable actions (`GET` , `POST` , `PUT` , `DELETE` , ...)
- Caching
- Redirection and forwarding
- Security (encryption and authentication)

nonrest/src/main/java/payroll/Employee.java

```
package payroll;

import lombok.Data;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Data
@Entity
class Employee {

  private @Id @GeneratedValue Long id;
  private String name;
  private String role;

  Employee() {}

  Employee(String name, String role) {
    this.name = name;
    this.role = role;
  }
}
```

Spring Boot 웹 애플리케이션 작성

- 컨트롤러 빈 등록
- 자동 컴포넌트 스캔
- DevTool 사용
- Lombok library 사용

Spring Boot 웹 애플리케이션 작성

컨트롤러 빈 등록

- @Controller
- @RestController

  REST 방식의 응답을 처리하는 Controller

  리턴되는 문자열이 응답

  별도의 view 화면 구성이 불필요

자동 컴포넌트 스캔

DevTool 사용

Lombok library 사용

Spring Boot 웹 애플리케이션 작성

컨트롤러 빈 등록

자동 컴포넌트 스캔 — @ComponentScan

DevTool 사용

Lombok library 사용

```
@ComponentScan(basePackages=
{ "kr.pe.khk", "..." })
```

Spring Boot 웹 애플리케이션 작성

컨트롤러 빈 등록

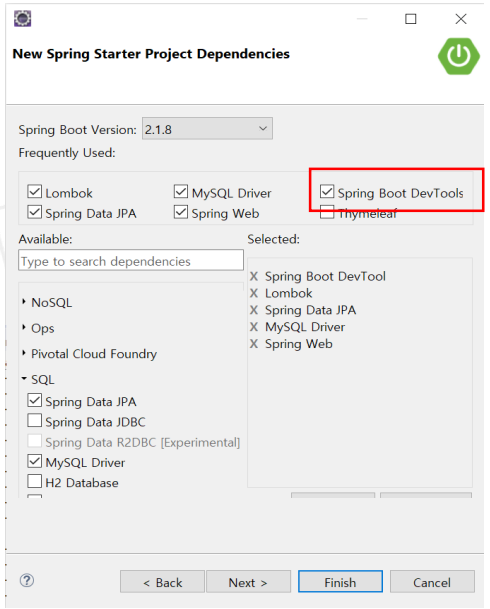
자동 컴포넌트 스캔

DevTool 사용

Lombok library 사용

애플리케이션 재실행 없이
수정된 로직 자동 적용

project에 DevTool추가



pom.xml 에 library 추가

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
```

Spring Boot 웹 애플리케이션 작성 – Lombok 설치

컨트롤러 빈 등록

자동 컴포넌트 스캔

DevTool 사용

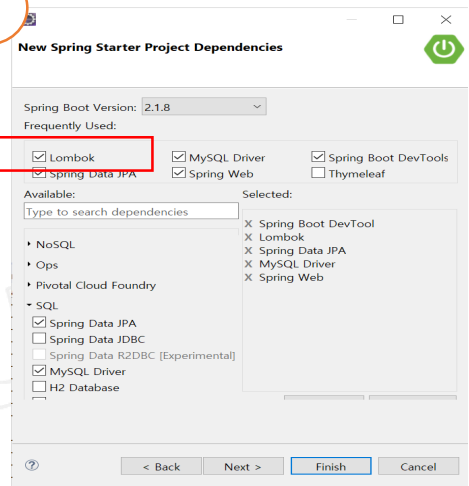
Lombok library 사용

쉽고 빠르게 도메인 클래스 개발

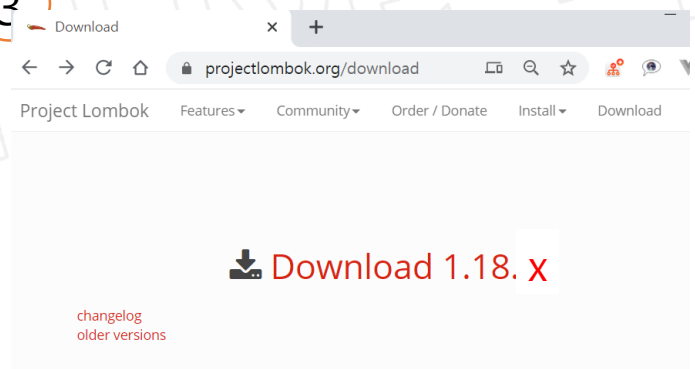
1

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

2



3



4

Lombok이 있는 경로에서 명령어 실행 후 eclipse 선택

>java –jar lombok.jar

설치 완료 후 eclipse 폴더 확인

lombok.jar 추가되어 있음

Spring Boot 다양한 이슈

프로젝트 생성후 로직 없이 실행시 발생되는 문제 & 해결책

- 실행시 발생된 문제

```
2020-06-03 16:57:49.732  INFO 10428 --- [  restartedMain] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/
2020-06-03 16:57:49.841  INFO 10428 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicat
2020-06-03 16:57:49.842  INFO 10428 --- [  restartedMain] o.s.web.context.ContextLoader            : Root WebApplicationContext: initializati
2020-06-03 16:57:49.891  WARN 10428 --- [  restartedMain] ConfigServletWebServerApplicationContext : Exception encountered during context ini
2020-06-03 16:57:49.893  INFO 10428 --- [  restartedMain] o.apache.catalina.core.StandardService   : Stopping service [Tomcat]
2020-06-03 16:57:49.923  INFO 10428 --- [  restartedMain] ConditionEvaluationReportLoggingListener :

Error starting ApplicationContext. To display the conditions report re-run your application with 'debug' enabled.
2020-06-03 16:57:49.929 ERROR 10428 --- [  restartedMain] o.s.b.d.LoggingFailureAnalysisReporter   :

***************************
APPLICATION FAILED TO START
***************************

Description:

Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource could be configured.

Reason: Failed to determine a suitable driver class


Action:

Consider the following:
        If you want an embedded database (H2, HSQL or Derby), please put it on the classpath.
        If you have database settings to be loaded from a particular profile you may need to activate it (no profiles are currently active).
```

- 해결책
  - application.properties 파일에 다음 내용 추가 후 해결

```
3 #DataSource Setting
4 spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
5 spring.datasource.url=jdbc:oracle:thin:@127.0.0.1:1521:xe
6 spring.datasource.username=SCOTT
7 spring.datasource.password=TIGER
```

```
    <dependency>
        <groupId>com.jslsolucoes</groupId>
        <artifactId>ojdbc6</artifactId>
        <version>11.2.0.1.0</version>
    </dependency>
</dependencies>
<repositories>
    <repository>
        <id>oracle</id>
        <name>ORACLE JDBC Repository</name>
        <url>https://maven.atlassian.com/3rdparty/</url>
    </repository>
</repositories>
```

pom.xml에 driver 추가

jsp 활용을 위한 실행시 발생된 문제

```
java.lang.ClassNotFoundException: org.eclipse.jdt.internal.compiler.env.INameEnvironment
        at java.net.URLClassLoader.findClass(URLClassLoader.java:382) ~[na:1.8.0_201]
        at java.lang.ClassLoader.loadClass(ClassLoader.java:424) ~[na:1.8.0_201]
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:349) ~[na:1.8.0_201]
        at java.lang.ClassLoader.loadClass(ClassLoader.java:357) ~[na:1.8.0_201]
        at java.lang.ClassLoader.defineClass1(Native Method) ~[na:1.8.0_201]
        at java.lang.ClassLoader.defineClass(ClassLoader.java:763) ~[na:1.8.0_201]
        at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142) ~[na:1.8.0
        at java.net.URLClassLoader.defineClass(URLClassLoader.java:468) ~[na:1.8.0_201]
```

- 해결책
  - https://stackoverflow.com/questions/55501858/getting-error-500-while-using-springmvc-with-jsp-page

```
<dependency>
    <groupId>org.eclipse.jdt.core.compiler</groupId>
    <artifactId>ecj</artifactId>
    <version>4.6.1</version>
    <scope>provided</scope>
</dependency>
```