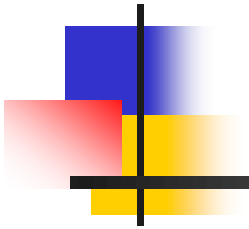
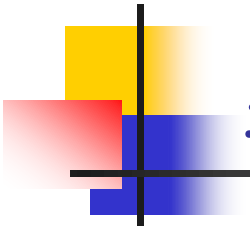


스프링부트

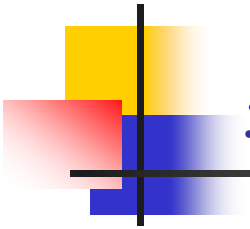
양 명 숙

[now4ever7@gmail.com]



스프링부트란

- 스프링 부트란?
  - 스프링 프레임워크를 사용하는 프로젝트를 아주 간편하게 설정할 수 있는 스프링 프레임웍의 서브 프로젝트라고 할 수 있다.
  - Spring Boot makes it easy to create stand-alone.
    - 단독실행 가능한 스프링애플리케이션을 생성한다.
  - Most Spring Boot applications need very little Spring configuration.
    - Spring Boot는 최소한의 초기 스프링 구성으로 가능한 한 빨리 시작하고 실행할 수 있도록 설계되었다.
  - 웹 컨테이너를 내장하고 있어 최소한의 설정으로 쉽게 웹 어플리케이션을 만들 수 있다.



스프링부트란

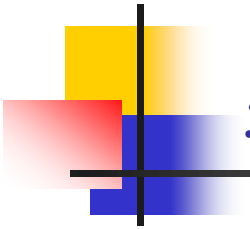
- 내장 Tomcat
  - 스프링부트는 웹 개발을 위해 자주 사용되는 Spring의 Component들과 Tomcat, Jetty 등의 경량 웹 어플리케이션 서버를 통합한 경량의 웹개발 프레임워크이다.
  - 즉 별도의 웹 어플리케이션 서버 없이 SpringBoot를 통해 프레임워크와 웹 어플리케이션 서버를 통합했다고 생각하면 된다.

스프링부트란

- **장점 : 스프링 부트를 왜 사용해야 하나?**
  - 스프링 프레임 워크를 사용하면 많은 XML 설정 파일등을 작성하는 등 설정하는 방법이 어려운 편
  - 스프링 부트는 반복되는 개발환경 구축을 위한 코드 작성등의 노력을 줄여주고 **쉽고 빠르게 프로젝트를 설정할 수 있도록 도와**준다.
  - 매우 빠르게 모든 스프링 개발에 관한 경험에 광범위한 접근을 제공한다.
  - 프로젝트 환경 구축에서 큰 영역을 차지하는 비기능적인 기능들을 기본적으로 제공한다.
    - (내장형 서버, 시큐리티, 측정, 상태 점검, 외부 설정)
  - Spring Boot Project는 개발을 진행하는 데 있어,**필수적인 설정들의 처리가 되어 있거나, 정말 간편하게 설정이 가능**하기 때문에 비즈니스 로직(사용자의 요구를 처리하는 로직)의 작성에 더욱 집중할 수 있는 환경을 제공해 줌

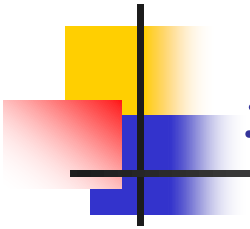
스프링부트란

- 스프링 부트는 templates 폴더, static 폴더, application.properties 파일이 기본적으로 생성됨
- templates
  - 기존의 스프링은 HTML 내에 자바 코드를 삽입하는 방식의 JSP를 사용
  - 디렉터리의 위치도 웹 디렉터리에 해당하는 src/main/webapp 안에 존재
  - 하지만, 이러한 방식은 war 파일로 패키지화되었을 경우에만 정적 리소스를 정상적으로 사용할 수 있다
  - 그러한 이유로 스프링 부트는 src/main/resources 디렉터리 내에서 화면과 관련된 파일을 관리
  - 스프링 부트는 타임리프(Thymeleaf) 템플릿 엔진의 사용을 권장
  - 타임리프는 JSP와 마찬가지로 HTML 내에서 데이터를 처리하는 데 사용됨



스프링부트란

- static
  - 해당 폴더에는 css, fonts, images, plugin, scripts 등의 정적 리소스 파일이 위치
- application.properties
  - 해당 파일은 웹 애플리케이션을 실행하면서 자동으로 로딩되는 파일
  - 예를 들어 톰캣(Tomcat)과 같은 WAS(포트 번호, 콘텍스트 패스 등)의 설정이나, 데이터베이스 관련 정보 등 각각으로 분리되어 있는 XML 또는 자바 기반의 설정을 해당 파일에 Key-Value 형식으로 지정해서 처리할 수 있다.

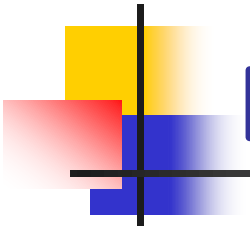


스프링부트란

- 스프링 부트에서도 여러 가지 뷰를 사용 할 수 있다.
  - JSP/JSTL
  - Thymeleaf
  - FreeMarker
  - Velocity
  - Groovy Template Engine
  - Tiles 등
- src/main/resources/[static] 폴더
  - 정적 리소스들을 추가 (css, images, js 등)
- src/main/resources/ [templates] 폴더
  - Thymeleaf(.html), Velocity(.vm)등과 관련된 파일만 동작하고 jsp 파일은 추가하여도 작동하지 않는다

스프링부트란

- ※ 폴더 구조
- src
  └── main
    └── java (java 파일)
    └── resources
      └── templates (View: Thymeleaf, Groovy, Velocity 등)
      └── static (정적 컨텐츠 : html, css, js, image 등)

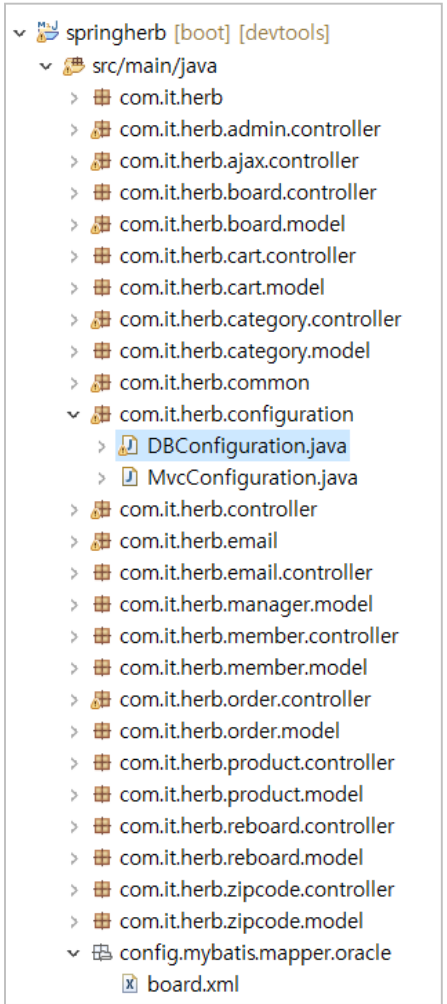


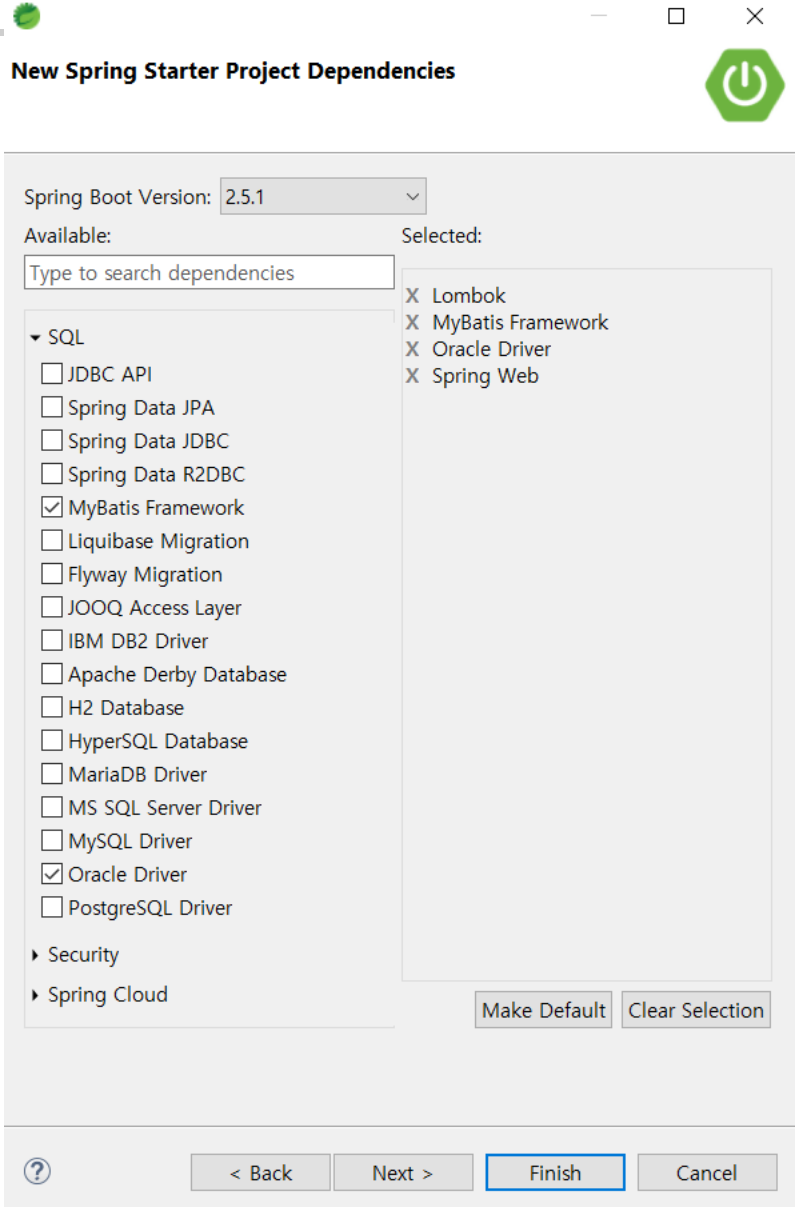
HTTP 응답

- HTTP 응답 - 정적 리소스, 뷰 템플릿
- 스프링(서버)에서 응답 데이터를 만드는 방법 3가지.
  - [1] 정적 리소스
    - 예) 웹 브라우저에 정적인 HTML, css, js을 제공할 때는, 정적 리소스를 사용한다.
  - [2] 뷰 템플릿 사용
    - 예) 웹 브라우저에 동적인 HTML을 제공할 때는 뷰 템플릿을 사용한다.
  - [3] HTTP 메시지 사용
    - HTTP API를 제공하는 경우에는 HTML이 아니라 데이터를 전달해야 하므로, HTTP 메시지 바디에 JSON 같은 형식으로 데이터를 실어 보낸다.

STS 에서 프로젝트 생성

- 프로젝트 생성
  - File – New – Spring Starter Project





정적 컨텐츠

application.properties

```
server.port = 9091

devtools.livereload.enabled=true
```

- src/main/resources/static/test_static.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>test_static.html</h1>
<h2>정적 컨텐츠 입니다</h2>

<!-- http://localhost:9091/test_static.html -->

</body>
</html>
```

MVC와 템플릿 엔진 (thymeleaf 템플릿 엔진)

- src/main/resources/templates/hello.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head> <title>Hello</title>
 <meta http-equiv="Content-Type"
 content="text/html; charset=UTF-8" />
</head>
<body>
<h1>thymeleaf 뷰페이지 - hello.html</h1>
<p th:text="'안녕하세요! ' + ${name}" >안녕하세요 guest님</p>

<!-- http://localhost:9091/hello -->
</body>
</html>
```

application.properties

```
server.port = 9091

# thymeleaf 수정 사항이 생길 때 수정을 하면
#브라우저 새로고침시 수정사항 반영을 위해
spring.thymeleaf.cache=false
spring.thymeleaf.check-template-location=true

devtools.livereload.enabled=true
```

동적 파일들의 파일 변경을 자동으로 반영

▶ pom.xml

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

▶ application.properties

```
devtools.livereload.enabled=true
```

MVC와 템플릿 엔진 (thymeleaf 템플릿 엔진)

```
package com.it.spboot.controller;

@Controller
public class HomeController {
	@GetMapping("/hello")
	public String hello(Model model) {
		model.addAttribute("name", "홍길동");

		return "hello";
	}
}
```

API

- [1] @ResponseBody 문자 반환
  - @ResponseBody 를 사용하면 뷰 리졸버(viewResolver)를 사용하지 않음
  - 대신에 HTTP의 BODY에 문자 내용을 직접 반환

```
package com.it.herb.controller;

@Controller
public class HomeController {
		@GetMapping("/api_string")
		@ResponseBody
		public String apiString(@RequestParam("name") String name) {
			return "hello " + name;

			//http://localhost:9091/api_string?name=hong
		}
```

결과

hello hong

API

- [2] @ResponseBody 객체 반환
  - @ResponseBody 를 사용하고, 객체를 반환하면 객체가 JSON으로 변환됨

```
@Controller
public class HomeController {
		@GetMapping("/api_object")
		@ResponseBody
		public Person apiObject(@RequestParam("name") String name) {
				Person p = new Person();
				p.setName(name);
				return p;

				//http://localhost:9091/api_object?name=hong
		}
}
class Person {
		private String name;
		public String getName() {
				return name;
		}
		public void setName(String name) {
				this.name = name;
		}
}
```

결과
{"name":"hong"}

jsp를 사용하는 방법

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

- spring-boot-starter-web 에는 tomcat이 포함되어 있지만, JSP 엔진은 포함하고 있지 않다.
  - 간단한 설정만 해주면 JSP view를 사용 가능
- jsp를 사용하는 방법
- [1] pom.xml
  - jasper, jstl을 의존성에 추가해야 JSP파일의 구동이 가능

jsp를 사용하는 방법

```
<dependency>
        <groupId>org.apache.tomcat.embed</groupId>
        <artifactId>tomcat-embed-jasper</artifactId>
        <scope>provided</scope>
</dependency>

<!-- jstl 라이브러리 -->
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
</dependency>
```

▶ build.gradle

```
compile('org.apache.tomcat.embed:tomcat-embed-jasper')
compile('javax.servlet:jstl:1.2')
```

jsp를 사용하는 방법

server.port = 9091
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp

- [2] JSP 경로 설정(디렉토리 생성)
  - WEB-INF/views/
  - (/src/main/webapp/WEB-INF/views/)
  - 톰캣기반 자바 웹어플리케이션에서는 보안상 jsp 위치를 URL로 직접 접근할수 없는 WEB-INF폴더 아래 위치
- [3] application.properties
  - ※ Spring 애플리케이션 시작시 application.properties 파일에 정의된 내용을 로드한다.
  - (스프링부트의 AutoConfiguration을 통해 자동 설정한 속성값들이 존재하며, application.properties의 해당 값들은 오버라이드 한다.)
  - server.port
    - 별다른 설정을 하지 않으면 default 포트는 8080
    - Spring Boot에 기본적으로 내장되어있는 Tomcat과 Jetty와 같은 WAS의 포트 번호를 임의로 변경 할 수 있다.
  - prefix/suffix
    - jsp 페이지를 처리하기 위한 prefix와 suffix를 application.properties에 추가.

jsp를 사용하는 방법

- [4] Controller 작성
- [5] jsp파일 서버 재시작 없이 바로 적용하기
  - 스프링 부트는 스프링 프로젝트와 다르게, 동적 파일들의 파일 변경을 자동으로 반영하지 않는다.
  - 기존 스프링과 동일하게 자동 반영하기 위해선 다음과 같은 설정을 추가하여 주면 된다.
  - Spring Boot 2.x 버전 기준

▶ pom.xml

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

▶ application.properties

```
devtools.livereload.enabled=true
```

- [6] Application 실행

jsp 파일 만들고 실행하기

- pom.xml에서 DB와 관련된 의존 항목들은 주석 처리
  - 스프링 부트 실행할 시 데이터베이스 기본 설정 정보(dataSource 등)가 없다면 실행 오류가 발생하기 때문
- Controller, jsp 파일 만든 후
- 프로젝트 우클릭 -> Run As-> Spring Boot App
- applicaion.properties에서 설정한 port번호로 접속 했을때 Hello Spring Boot!가 출력 되면 성공
  - http://localhost:9091/

jsp 파일 만들고 실행하기

```
<!-- <dependency>
                <groupId>org.mybatis.spring.boot</groupId>
                <artifactId>mybatis-spring-boot-starter</artifactId>
                <version>2.2.0</version>
        </dependency>

        <dependency>
                <groupId>com.oracle.database.jdbc</groupId>
                <artifactId>ojdbc8</artifactId>
                <scope>runtime</scope>
        </dependency>
        -->
```

jsp 파일 만들고 실행하기

```
package com.it.herb.controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class HomeController {
	@RequestMapping(value="/")
	public String index() {
		return "index";
	}
}
---------
<%@ page language="java" contentType="text/html; charset=UTF-8"    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Insert title here</title>
</head>
<body>
    <h1>Hello Spring Boot!!!!!!</h1>
</body>
</html>
```

MVC와 템플릿 엔진 (thymeleaf 템플릿 엔진)

- thymeleaf
  - Thymeleaf 홈페이지
    - https://www.thymeleaf.org/index.html
- Thymeleaf + Spring
  - https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html
  - [1] Dependency 추가

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

 - build.gradle
implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
```

  - [2] 파일 기본 경로 및 html 생성
    - 경로 : /src/main/resources/templates/thymeleaf
    - hello.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head> <title>Hello</title>
 <meta http-equiv="Content-Type"
 content="text/html; charset=UTF-8" />
</head>
<body>
<h1>thymeleaf 뷰페이지</h1>
<p th:text="'안녕하세요. ' + ${name} + '님'" >안녕하세요. guest님</p>
</body>
</html>
```

```
<!-- http://localhost:9091/hello -->
```

- [3]application.properties 설정
  - JSP와 같이 사용할 경우 뷰 구분을 위해 컨트롤러가 뷰 이름을 반환할 때 thymeleaf/ 로 시작하면 타임리프로 처리하도록 view-names 지정

```
spring.thymeleaf.view-names=thymeleaf/*
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
```

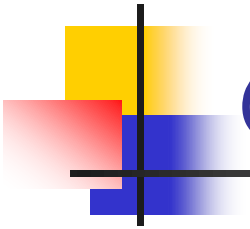
  - thymeleaf를 사용하다 수정 사항이 생길 때 수정을 하면 재시작을 해줘야 한다. 이를 무시하고 브라우저 새로고침시 수정사항 반영을 위해 cache=false 설정(운영시는 true)

```
spring.thymeleaf.cache=false
spring.thymeleaf.check-template-location=true
```

MVC와 템플릿 엔진 (thymeleaf 템플릿 엔진)

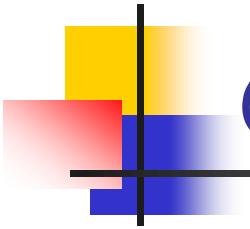
- [4] Controller 작성

```
@Controller
public class HelloController {
        @GetMapping("/hello")
        public String hello(Model model) {
                model.addAttribute("name", "홍길동");
                return "thymeleaf/hello";
        }
}
```



@Mapper

- @Mapper
  - 기존의 스프링은 DAO(Data Access Object) 클래스에 @Repository를 선언해서 해당 클래스가 데이터베이스와 통신하는 클래스임을 나타냈다
  - 마이바티스는 인터페이스에 @Mapper만 지정해주면 **XML Mapper에서 메서드의 이름과 일치하는 SQL 문을 찾아 실행**
  - Mapper 영역은 데이터베이스와의 통신, 즉 SQL 쿼리를 호출하는 것이 전부이며, 다른 로직은 전혀 필요하지 않다.



@Mapper

```
package com.it.herb.board.model;


@Mapper
public interface BoardDAO {
    public int insertBoard(BoardVO vo);
    public List<BoardVO> selectAll(SearchVO searchVo);
    public int selectTotalRecord(SearchVO searchVo);
    public int updateReadCount(int no);
    public BoardVO selectByNo(int no);
    public int updateBoard(BoardVO vo);
    public int deleteBoard(BoardVO vo);
    public List<BoardVO> selectMainNotice();

}
```

board.xml

```
<mapper namespace="com.it.herb.board.model.BoardDAO">

    <insert id="insertBoard" parameterType="boardVO">
        <selectKey keyProperty="no" resultType="int" order="BEFORE">
            select board_seq.nextval from dual
        </selectKey>

        insert into board(no,name, pwd, title, email, content)
        values(#{no} ,#{name}, #{pwd}, #{title}, #{email},
            #{content})
    </insert>
```

application.properties

포트 변경은
src/main/resources/application.properties
파일에서
server.port=9091
로 설정하면 톰캣이 9091번 포트로 기동

```
server.servlet.context-path=/herb
server.port=9093

# JSP Path (ViewResolver)
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp

# DataBase
#spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
#spring.datasource.url=jdbc:oracle:thin:@DESKTOP-K2C44OV:1521:xe
#spring.datasource.username=herb
#spring.datasource.password=herb123

#mapper location settings
#mybatis.config-location=classpath:/config/mybatis/oracle/mybatis-config.xml
#mybatis.mapper-locations=classpath:/config/mybatis/mapper/oracle/*.xml
#mybatis.type-aliases-package=com.it.herb
```

application.properties

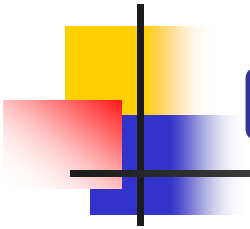
```
spring.datasource.hikari.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.hikari.jdbc-url=jdbc:oracle:thin:@DESKTOP-K2C44OV:1521:xe
spring.datasource.hikari.username=herb
spring.datasource.hikari.password=herb123
spring.datasource.hikari.connection-test-query=SELECT sysdate FROM dual

#MyBatis
mybatis.configuration.map-underscore-to-camel-case=true

#email
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=now4ever7@gmail.com
spring.mail.password=[비밀번호 입력]
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

STS 환경 설정하기

- sts 폴더에 있는 **SpringToolSuite4.ini** 파일 설정

```
-vm
C:□Java□jdk-15.0.2/bin/javaw.exe

-Xms2048m
-Xmx2048m
```

  - 다른 이름으로 저장 – **인코딩(Encoding)**을 **UTF-8**로 설정한 뒤에 저장
  - 경로는 **jdk 경로**와 일치해야 하며, 꼭 **-vmargs** 앞에 추가.
  - JVM이 사용하는 힙 메모리(Heap Memory)의 시작, 최대 사이즈
    - 두 값을 동일하게 설정하면 이클립스(STS) 특유의 버벅거림이 조금은 나아짐.
    - 8GB 램(RAM) 기준으로 1024가 적절하며, 램(RAM)이 4GB라면 512, 16GB라면 2048로 설정



Board2Application

```
package com.it.herb;

@SpringBootApplication
public class Board2Application {

	public static void main(String[] args) {
		SpringApplication.run(Board2Application.class, args);
	}

}
```

Board2Application

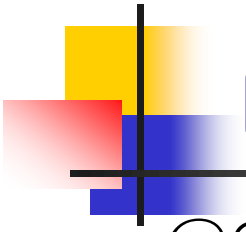
- main 메서드는 SpringApplication.run 메서드를 호출해서 웹 애플리케이션을 실행하는 역할
- 클래스 선언부에 선언된 @SpringBootAplication
  - 다음의 세 가지 애너테이션으로 구성되어 있다
    - @EnableAutoConfiguration
      - 스프링 부트는 개발에 필요한 몇 가지 필수적인 설정들의 처리가 되어 있는데, 해당 애너테이션에 의해 다양한 설정들의 일부가 자동으로 완료됨
    - @ComponentScan
      - 기존의 XML 설정 방식의 스프링은 빈(Bean)의 등록 및 스캔을 위해 수동으로 ComponentScan을 여러 개 선언하는 방식을 사용.
        스프링 부트는 해당 애너테이션에 의해 자동으로 컴포넌트 클래스를 검색하고, 스프링 애플리케이션 콘텍스트(IoC 컨테이너)에 빈(Bean)으로 등록함
    - @Configuration
      - 해당 애너테이션이 선언된 클래스는 자바 기반의 설정 파일로 인식됨
        스프링 4 버전부터 자바 기반의 설정이 가능하게 되었으며,
        XML 설정에 많은 시간을 소모하지 않아도 됨

DBConfiguration

```
package com.it.herb.configuration;

import javax.sql.DataSource;
import org.apache.ibatis.session.SqlSessionFactory;
import org.mybatis.spring.SqlSessionFactoryBean;
import org.mybatis.spring.SqlSessionTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.annotation.EnableTransactionManagement;

import com.it.herb.board.model.BoardVO;
import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;
```

```
@Configuration
@PropertySource("classpath:/application.properties")
@EnableTransactionManagement
public class DBConfiguration {
    @Autowired
    private ApplicationContext applicationContext;

    @Bean
    @ConfigurationProperties(prefix = "spring.datasource.hikari")
    public HikariConfig hikariConfig() {
        return new HikariConfig();
    }

    @Bean
    public DataSource dataSource() {
        return new HikariDataSource(hikariConfig());
    }
    @Bean
    public SqlSessionFactory sqlSessionFactory() throws Exception {
        SqlSessionFactoryBean factoryBean = new SqlSessionFactoryBean();
        factoryBean.setDataSource(dataSource());
    factoryBean.setMapperLocations(applicationContext.getResources("classpath:/mappers/**/*.xml"));
        factoryBean.setTypeAliasesPackage("com.it.herb");
        factoryBean.setConfiguration(mybatisConfg());
        return factoryBean.getObject();
    }
```

```
    @Bean
    public SqlSessionTemplate sqlSession() throws Exception {
            return new SqlSessionTemplate(sqlSessionFactory());
    }

    @Bean
    @ConfigurationProperties(prefix = "mybatis.configuration")
    public org.apache.ibatis.session.Configuration mybatisConfg() {
            return new org.apache.ibatis.session.Configuration();
    }

    //tx:annotation-driven 설정-@Transactional를 선언하여 트랜잭션 처리를 할 수 있다.
    @Bean
    public PlatformTransactionManager txManager() throws Exception{
            return new DataSourceTransactionManager(dataSource());
    }

}
```



DBConfiguration

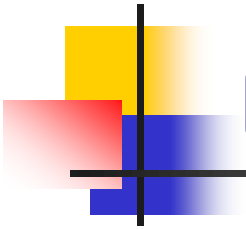
- @Configuration
  - 스프링은 @Configuration이 지정된 클래스를 자바 기반의 설정 파일로 인식.
- @PropertySource
  - 해당 클래스에서 참조할 properties 파일의 위치를 지정
- @Autowired
  - 빈(Bean)으로 등록된 인스턴스(이하 객체)를 클래스에 주입하는 데 사용
  - @Autowired 이외에도 @Resource, @Inject 등이 존재
- ApplicationContext
  - ApplicationContext는 **스프링 컨테이너(Spring Container)** 중 하나.
  - 컨테이너는 사전적 의미로 무언가를 담는 용기 또는 그릇을 의미
  - 스프링 컨테이너는 빈(Bean)의 생성과 사용, 관계, 생명 주기 등을 관리
  - 빈(Bean)은 객체이다

DBConfiguration

- @Bean
  - Configuration 클래스의 메서드 레벨에만 지정 가능
  - @Bean이 지정된 객체는 **컨테이너에 의해 관리되는 빈(Bean)으로 등록**
- @ConfigurationProperties
  - 인자에 **prefix** 속성을 지정할 수 있고, (prefix는 접두사)
  - prefix에 spring.datasource.hikari를 지정
  - @PropertySource에 지정된 파일(application.properties)에서 prefix에 해당하는 **spring.datasource.hikari**로 시작하는 설정을 모두 읽어 들여 해당 메서드에 매핑(바인딩)함
  - 해당 애너테이션은 메서드뿐만 아니라 클래스 레벨에도 지정할 수 있다.
- hikariConfig
  - 히카리CP 객체를 생성함
  - 히카리CP는 **커넥션 풀(Connection Pool) 라이브러리** 중 하나

DBConfiguration

- @dataSource
  - 데이터 소스 객체를 생성
  - 순수 JDBC는 SQL을 실행할 때마다 커넥션을 맺고 끊는 I/O 작업을 하는데, 이러한 작업은 상당한 리소스를 잡아먹는다
  - 이러한 문제의 해결책으로 커넥션 풀이 등장
  - 커넥션 풀은 커넥션 객체를 생성해두고, 데이터베이스에 접근하는 사용자에게 미리 생성해둔 커넥션을 제공했다가 다시 돌려받는 방법
  - 데이터 소스는 커넥션 풀을 지원하기 위한 인터페이스
- sqlSessionFactory
  - SqlSessionFactory 객체를 생성
  - SqlSessionFactory는 데이터베이스의 커넥션과 SQL 실행에 대한 모든 것을 갖는 중요한 역할을 함
  - SqlSessionFactoryBean은 마이바티스와 스프링의 연동 모듈로 사용되는데, 마이바티스 XML Mapper, 설정 파일 위치 등을 지정하고, SqlSessionFactoryBean 자체가 아닌, getObject 메서드가 리턴하는 SqlSessionFactory를 생성함



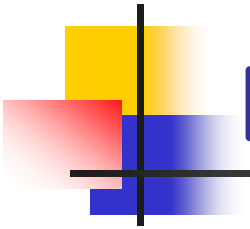
DBConfiguration

- sqlSession
  - sqlSession 객체를 생성
  - 1. SqlSessionTemplate은 마이바티스 스프링 연동 모듈의 핵심
  - 2. SqlSessionTemplate은 SqlSession을 구현하고, 코드에서 SqlSession을 대체하는 역할을 함
  - 3. SqlSessionTemplate은 쓰레드에 안전하고, 여러 개의 DAO나 Mapper에서 공유할 수 있다.
  - 4. 필요한 시점에 세션을 닫고, 커밋 또는 롤백하는 것을 포함한 세션의 생명주기를 관리한다.
  - SqlSessionTemplate은 SqlSessionFactory를 통해 생성되고, 데이터베이스의 커밋, 롤백 등 SQL의 실행에 필요한 모든 메서드를 갖는 객체

트랜잭션 처리

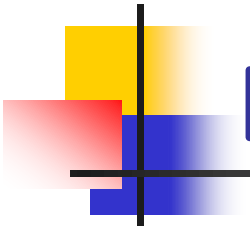
- tx:annotation-driven 설정-@Transactional를 선언하여 트랜잭션 처리를 할 수 있다.

```
@Configuration
@PropertySource("classpath:/application.properties")
@EnableTransactionManagement
public class DBConfiguration {

    @Autowired
    private ApplicationContext applicationContext;

    //tx:annotation-driven 설정-@Transactional를 선언하여 트랜잭션 처리를 할 수 있다.
    @Bean
    public PlatformTransactionManager txManager() throws Exception{
            return new DataSourceTransactionManager(dataSource());
    }

}
```



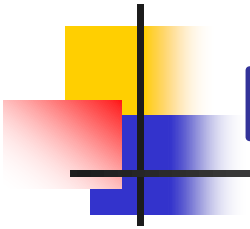
Hikari CP

- spring.datasource.hikari
  - Hikari : Connection pool의 일종
  - Hikari는 Database와의 Connection Pool을 관리해 준다
  - HikariCP는 미리 정해놓은 만큼의 커넥션을 Pool에 담아 놓는다
  - 요청이 들어오면 Thread가 커넥션을 요청하고, Hikari가 Pool 내에 있는 커넥션을 연결해 준다



HikariCP

- HikariCP는 Brett Wooldridge 가 2012년 경 개발한 매우 가볍고 빠르고 안정적인 JDBC Connection Pool
  - hikariCP는 스프링 부트 2.0부터 default JDBC connection pool이다.
  - "zero-overhead" - 엄청나게 높은 성능



HikariCP

- HikariCP가 해주는 역할은 Database와의 커넥션 풀을 관리해준다는 것
- 커넥션 풀을 관리해주는 것이 중요한 이유는 성능에 큰 영향을 미치기 때문
- JDBC 커넥션을 맺는 과정은 상당히 복잡할 뿐만 아니라 꽤나 자원을 많이 소모하는 작업
- 요청이 들어올 때 Thread가 Database와의 커넥션을 맺는다면 데이터베이스 뿐만 아니라 앱서버 입장에서도 굉장히 부하가 심하게 발생할 것임
- 그런데 HikariCP는 미리 정해놓은 만큼의 커넥션을 Pool에 담아 놓는다
- 요청이 들어오면 Thread가 커넥션을 요청하고, Hikari는 Pool내에 있는 커넥션을 연결해줌
- 그러면 Thread입장에서는 바로 쿼리를 날릴 수 있게 됨

```
package com.it.herb.configuration;

@Configuration
public class MvcConfiguration implements WebMvcConfigurer{

	@Override
	public void addInterceptors(InterceptorRegistry registry) {
		registry.addInterceptor(new LoginInterceptor())
		.addPathPatterns("/shop/cart/*",
	"/shop/order/*","/member/memberEdit.do","/member/memberOut.do");

		registry.addInterceptor(new AdminLoginInterceptor())
		.excludePathPatterns("/admin/login/adminLogin.do")
		.addPathPatterns("/admin/*/*", "/admin/*");
	}

	@Bean
	public CommonsMultipartResolver multipartResolver() {
		CommonsMultipartResolver multipartResolver
				= new CommonsMultipartResolver();
		multipartResolver.setDefaultEncoding("UTF-8"); // 파일 인코딩 설정
		multipartResolver.setMaxUploadSizePerFile(5 * 1024 * 1024); // 파일당 업로드 크기 제한 (5MB)
		return multipartResolver;
	}

}
```

Dependency 추가

```
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.6</version>
</dependency>

<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.3</version>
</dependency>

<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-mail</artifactId>
            <!-- <version>2.0.1.RELEASE</version> -->
</dependency>
```

Properties-객체화하여 사용하기

- application.properties가 아닌 기타 설정(데이터 소스, 계정정보, 특정 모듈 외부 입력 정보 등) 파일을 따로 분리하여 사용할 때가 많다.
- 그럴 경우에는 자동으로 properties가 등록되지 않아 사용할 수 없는데 이 정보들을 객체화 하여 사용하는 방법들 중 @PropertySource, @ConfigurationProperties가 있다
- @PropertySource
  - application.properties가 위치하는 classpath에 app-info.properties 파일을 하나 생성하여 app에 관련된 정보를 넣어주어 객체로 사용해 보자
  - @PropertySource 어노테이션 내부에 사용할 프로퍼티 위치를 입력하여 준뒤에 반드시 @Component로 등록하여 스프링에서 빈으로 관리되게 해주어야 함.

fileUpload.properties

[fileUpload.properties]

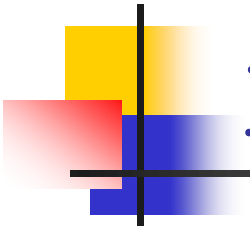
file.upload.path=pds_upload
file.upload.path.test=D:₩₩my₩₩sp2_ws₩₩Board2₩₩src₩₩main₩₩webapp₩₩pds_upload
imageFile.upload.path=pd_images
imageFile.upload.path.test=D:₩₩my₩₩sp2_ws₩₩Board2₩₩src₩₩main₩₩webapp₩₩resources₩₩pd_images
file.upload.type=test
#file.upload.type=deploy

FileUploadInfo

```
@Component
@PropertySource("classpath:/config/props/fileUpload.properties")
public class FileUploadInfo {
	@Value("${file.upload.path}")
	private String filePath;

	@Value("${file.upload.path.test}")
	private String filePathTest;

	@Value("${imageFile.upload.path}")
	private String imageFilePath;

	@Value("${imageFile.upload.path.test}")
	private String imageFilePathTest;

	@Value("${file.upload.type}")
	private String fileUploadType;

	public String getFilePath() {
		return filePath;
	}
	public void setFilePath(String filePath) {
		this.filePath = filePath;
```

```
@Component
public class FileUploadUtil {
	//자료실에서 사용하는지, 상품 등록에서 사용하는지 구분하는 상수
	public static final int PDS_TYPE=1;  //자료실에서 사용
	public static final int IMAGE_TYPE=2; //상품등록에서 사용

	//@Resource(name="fileUploadProperties")
	//private Properties fileUploadProps;

	@Autowired
	private FileUploadInfo fileInfo;

	private static final Logger logger
		=LoggerFactory.getLogger(FileUploadUtil.class);

	public  List<Map<String, Object>> fileUplaod(HttpServletRequest request,
			int type) throws IllegalStateException, IOException {
		MultipartHttpServletRequest multiRequest
			= (MultipartHttpServletRequest) request;
.....
	}

	public String getUploadPath(int type, HttpServletRequest request) {
		String testGb=fileInfo.getFileUploadType();

		String upPath="";
```

```
        if(type==PDS_TYPE) {  //자료실에서 업로드
                if(testGb.equals("test")) {
                        upPath=fileInfo.getFilePathTest();
                }else {
                        upPath=fileInfo.getFilePath();
                        //=> pds_upload
                }
        }else if(type==IMAGE_TYPE) { //상품 등록시 상품 이미지 업로드
                if(testGb.equals("test")) {
                        upPath=fileInfo.getImageFilePathTest();
                }else {
                        upPath=fileInfo.getImageFilePath();
                        //=> pd_images
                }
        }

        if(!testGb.equals("test")) {
                upPath
                = request.getSession().getServletContext().getRealPath(upPath);

        }

        logger.info("파일 업로드 경로: {}", upPath);

        return upPath;
}
```



파일 다운로드 처리

- 파일 다운로드 처리시 기존 소스 그대로 이용 가능
  - In Spring Boot, BeanNameViewResolver bean is registered by default, that means we can use a View's bean name as a view name by default.
  - In a plain Spring MVC application we have to explicitly register this bean ourselves

이메일

- 1. email 전송을 지원하는 스프링 모듈을 import

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

- 2. 스프링의 mail starter는 MailSender interface와 MailSenderImpl을 제공한다.
  - 1. 실제로 사용하는 인터페이스와 클래스는 편의기능을 더 추가한 JavaMailSender, JavaMailSenderImpl이 된다.
  - 2. 어차피 @Autowired로 주입하기 때문에 구현 클래스에 대해서 알 필요가 없다.

```
@Component
public class EmailSender {
		@Autowired
		private JavaMailSender mailSender;
```

이메일

- 3. application.properties에 아래 내용 넣기

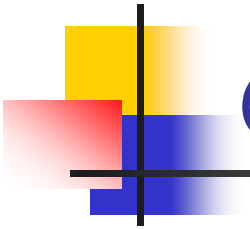
```
#email
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=이메일 주소
spring.mail.password=비밀번호
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

@RestController

```
@RestController
public class DemoApiController {

    @GetMapping("/demoapistring")
    public String demoapistring() {
        return "데모 스트링 타입 리턴";
    }

    @GetMapping("/demoapi")
    public Map<String, Object> demoapi() {
        Map<String, Object> map = new HashMap<>();
        map.put("name", "홍길동");
        map.put("birthday", 15920505);
        return map;
    }
}
```

@RestController 을 추가한 경우
return 타입이 Object 인 경우 해당 object 에 맞게 json 형식으로 표시

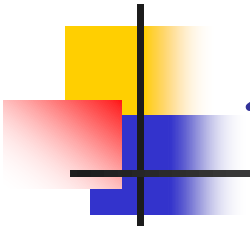
@GetMapping, @PostMapping
- Spring 4.3버전 이후로 추가



@RestController

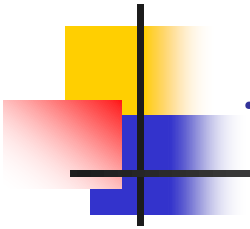
- @RestController
  - @Controller 대신에 @RestController 애노테이션을 사용하면, 해당 컨트롤러에 모두 @ResponseBody 가 적용되는 효과가 있다
  - 따라서 뷰 템플릿을 사용하는 것이 아니라, HTTP 메시지 바디에 직접 데이터를 입력한다. 이름 그대로 Rest API(HTTP API)를 만들 때 사용하는 컨트롤러이다.
  - @ResponseBody 는 클래스 레벨에 두면 전체 메서드에 적용되는데, @RestController 에노테이션 안에 @ResponseBody 가 적용되어 있다.
- @GetMapping, @PostMapping

```
@Controller
public class MemberController {
    private final MemberService memberService;
    @Autowired
    public MemberController(MemberService memberService) {
        this.memberService = memberService;
    }

    @GetMapping(value = "/members/new")
    public String createForm() {
        return "members/createMemberForm";
    }

    @PostMapping(value = "/members/new")
    public String create(MemberForm form) {
        Member member = new Member();
        member.setName(form.getName());
        memberService.join(member);
        return "redirect:/";
    }
    @GetMapping(value = "/members")
    public String list(Model model) {
        List<Member> members = memberService.findMembers();
        model.addAttribute("members", members);
        return "members/memberList";
    }
}
```



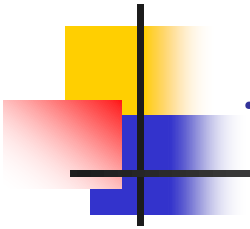
싱글톤으로 등록

- 스프링은 스프링 컨테이너에 스프링 빈을 등록할 때, 기본으로 싱글톤으로 등록한다(유일하게 하나만 등록해서 공유한다) 따라서 같은 스프링 빈이면 모두 같은 인스턴스다. 설정으로 싱글톤이 아니게 설정할 수 있지만, 특별한 경우를 제외하면 대부분 싱글톤을 사용한다



롬복

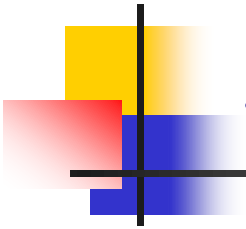
- Lombok(롬복) 이란?
  - 자동으로 기본적인 상용구를 생성 해 줌으로써 코드량을 줄이는데 도움을 주는 Java 어노테이션 라이브러리
  - 객체 클래스에 @Data 어노테이션만 달면, Getter/Setter/Equals/ToString과 같은 메서드를 자동으로 생성/연결해 줌
- Lombok 다운로드 하기
- [1]사이트에서 직접 다운 받는 경우
  - https://projectlombok.org/download 에서 다운로드
  - Lombok.jar 파일을 확인
- [2] Dependency를 직접 입력하여 Maven을 통해 다운 받는 경우
  - https://mvnrepository.com/artifact/org.projectlombok/lombok
  - 원하는 버전을 선택



롬복

- Maven 탭의 dependency 내용을 복사
- Maven Project의 <dependencies> 태그 하위에 복사한 내용을 붙인다
- 프로젝트 우클릭 → Maven → Project Update → Ok

- [3] Springboot 프로젝트 생성 과정에서 Maven Dependency를 통해 다운 받는 경우
  - STS 우클릭 → New → Spring Starter Project
  - 생성할 프로젝트의 내용 입력 → Dependencies 단계에서 Available에 lombok 검색 및 결과 체크 → Finish



롬복

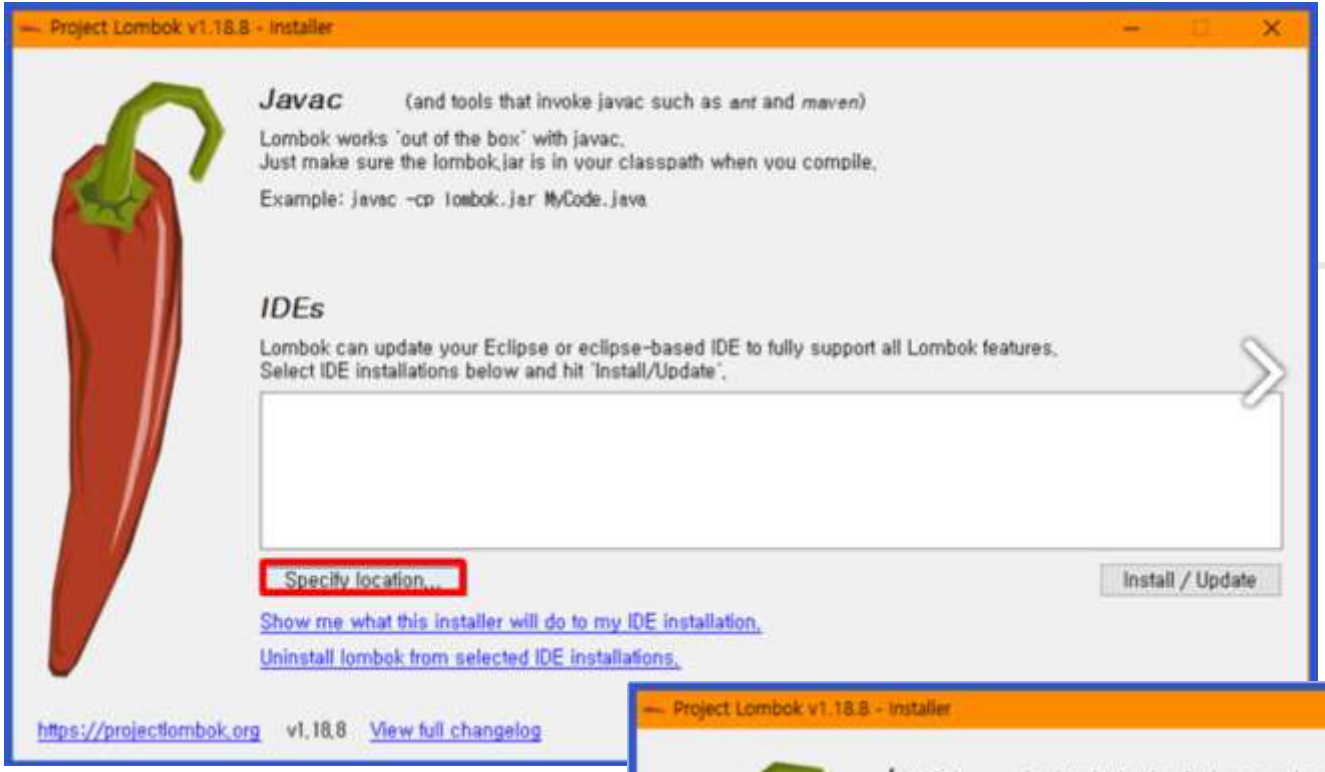
- 다운로드 한 Lombok 설치하기
  - STS(eclipse)가 켜져있다면 종료
  - 1. lombok.jar가 있는 디렉토리로 이동
  - 2. lombok.jar 더블클릭 (또는 cmd나 powershell에 java -jar lombok.jar 명령어 실행)
    - (lombok-1.18.8.jar 와 같이 버전이 있는 경우, java -jar lombok-1.18.8.jar)
  - 3. IDEs 목록에 사용하는 STS(eclipse)를 선택 (없다면 Specify location 버튼으로 찾기)
  - 4. IDE가 선택된 상태로 Install/Update 버튼 클릭.
  - 5. Install successful 이 나오면 설치가 완료 된 것
  - 6. STS(eclipse)를 켜서 테스트 객체 클래스에 @Data 어노테이션을 달고 테스트

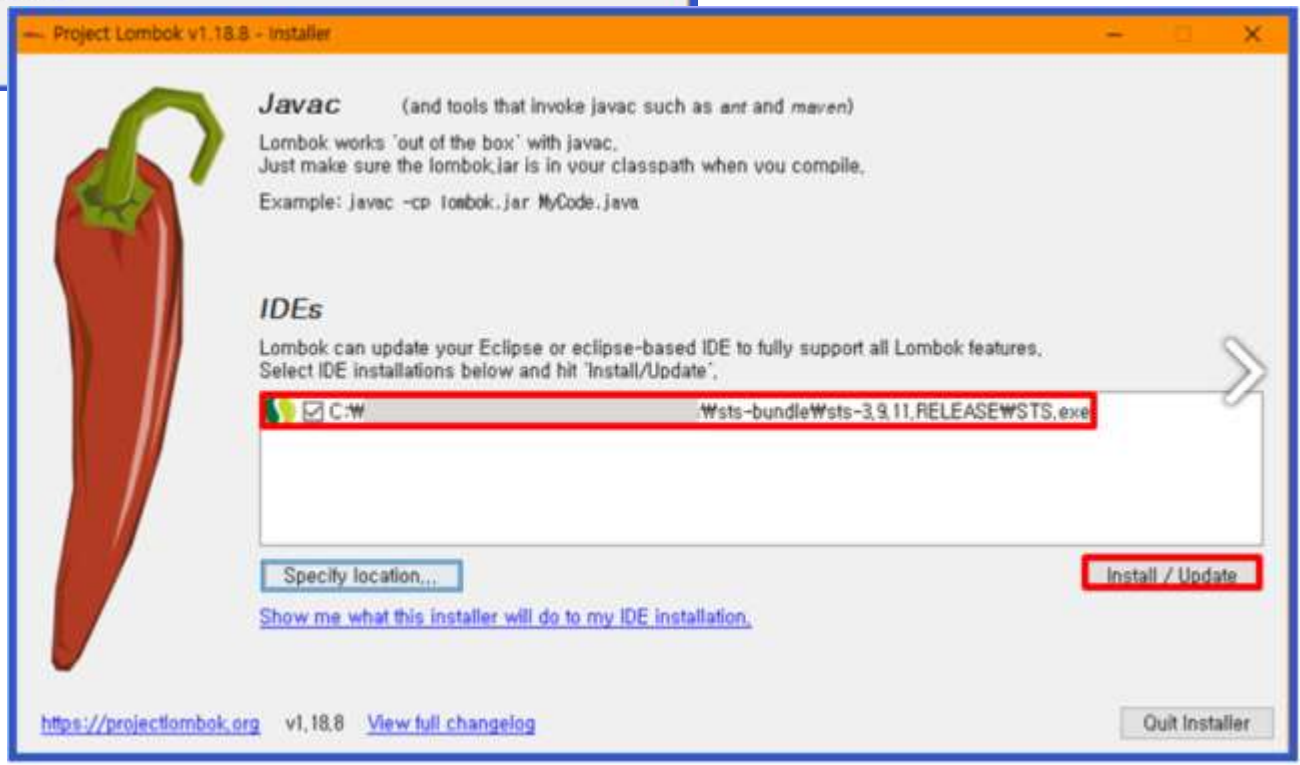
롬복

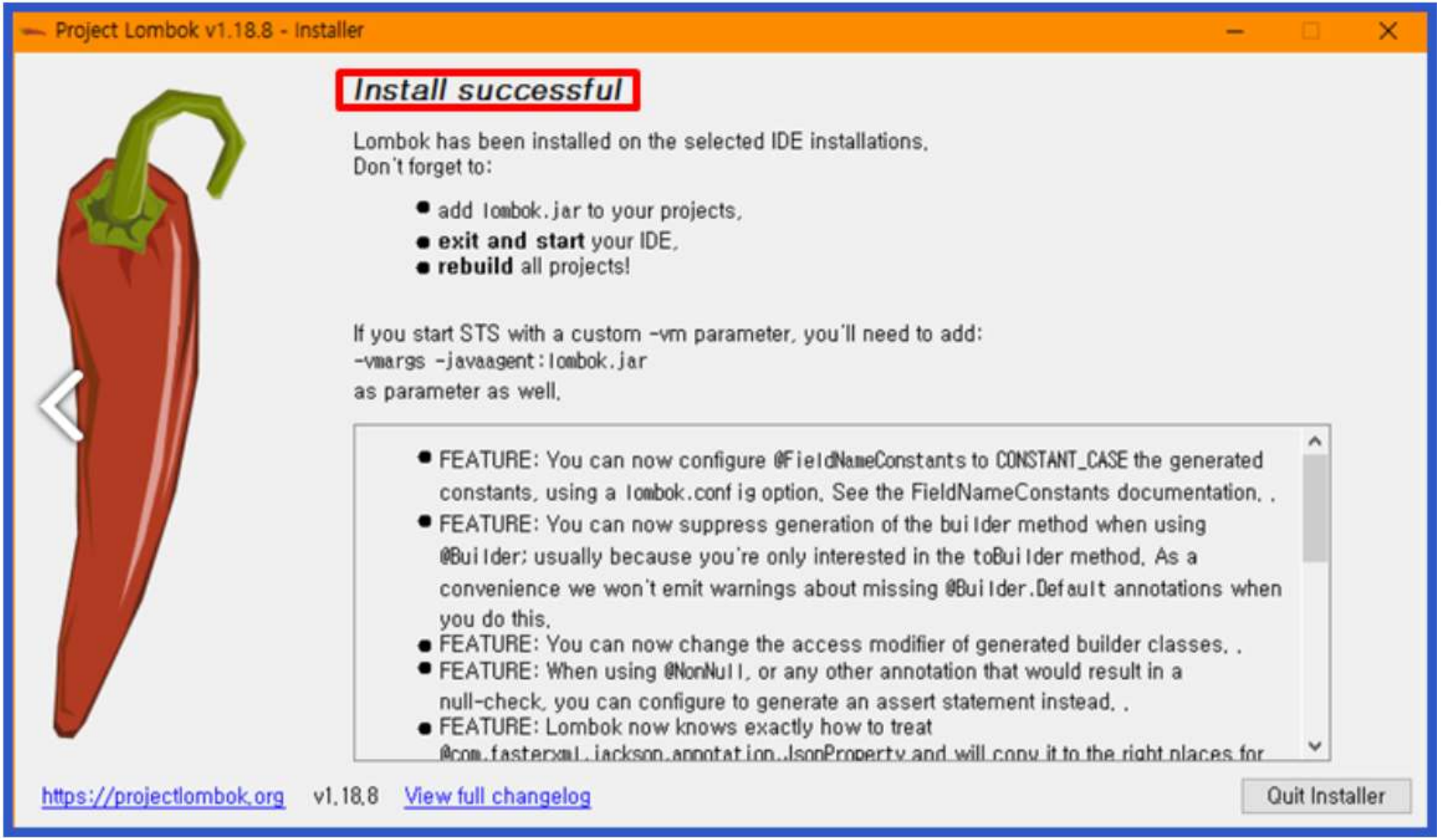
```
import lombok.Data;

@Data
public class User {
		String name;
		int age;
}
```

- @Data 어노테이션을 객체클래스에 입력하는 것 만으로도 getName(), getAge(), setName(String name), setAge(int age), toString() 등이 내부적으로 자동생성되므로 따로 작성해 줄 필요 없이도 다음과 같은 메서드 접근이 가능해짐

```
User user = new User();
user.setName("홍길동");
user.setAge(20);

System.out.println(user.toString());
```


Project Lombok v1.18.8 - Installer
Javac (and tools that invoke javac such as ant and maven)
Lombok works 'out of the box' with javac.
Just make sure the lombok.jar is in your classpath when you compile.
Example: javac -cp lombok.jar MyCode.java
IDEs
Lombok can update your Eclipse or eclipse-based IDE to fully support all Lombok features.
Select IDE installations below and hit 'Install/Update'.
Specify location...
Install / Update
Show me what this installer will do to my IDE installation.
Uninstall lombok from selected IDE installations.
https://projectlombok.org
v1.18.8
View full changelog


Project Lombok v1.18.8 - Installer
Javac (and tools that invoke javac such as ant and maven)
Lombok works 'out of the box' with javac.
Just make sure the lombok.jar is in your classpath when you compile.
Example: javac -cp lombok.jar MyCode.java
IDEs
Lombok can update your Eclipse or eclipse-based IDE to fully support all Lombok features.
Select IDE installations below and hit 'Install/Update'.
C:₩ ₩sts-bundle₩sts-3.9.11.RELEASE₩STS.exe
Specify location...
Install / Update
Show me what this installer will do to my IDE installation.
https://projectlombok.org
v1.18.8
View full changelog
Quit Installer


Project Lombok v1.18.8 - Installer
Install successful
Lombok has been installed on the selected IDE installations.
Don't forget to:
add lombok.jar to your projects,
exit and start your IDE,
rebuild all projects!
If you start STS with a custom -vm parameter, you'll need to add:
-vmargs -javaagent:lombok.jar
as parameter as well.
FEATURE: You can now configure @FieldNameConstants to CONSTANT_CASE the generated constants, using a lombok.config option. See the FieldNameConstants documentation. .
FEATURE: You can now suppress generation of the builder method when using @Builder; usually because you're only interested in the toBuilder method. As a convenience we won't emit warnings about missing @Builder.Default annotations when you do this.
FEATURE: You can now change the access modifier of generated builder classes. .
FEATURE: When using @NonNull, or any other annotation that would result in a null-check, you can configure to generate an assert statement instead. .
FEATURE: Lombok now knows exactly how to treat
https://projectlombok.org
v1.18.8
View full changelog
Quit Installer

롬복

```
@Controller
@RequestMapping("/basic/items")
@RequiredArgsConstructor
public class BasicItemController {
    private final ItemRepository itemRepository;

    @GetMapping
    public String items(Model model) {
        List<Item> items = itemRepository.findAll();
        model.addAttribute("items", items);
        return "basic/items";
    }
}
```

@RequiredArgsConstructor - final 이 붙은 멤버변수만 사용해서 생성자를 자동으로 만들어준다.

```
public BasicItemController(ItemRepository itemRepository) {
    this.itemRepository = itemRepository;
}
```

생성자가 1개만 있으면 스프링이 해당 생성자에 @Autowired 로 의존관계를 주입해준다.
따라서 final 키워드를 빼면 안된다, 그러면 ItemRepository 의존관계 주입이 안된다