



# chpt4

## 서비스의 자원

### 웹 서비스

- 사용자가 원하는 자원의 관리
- 자원 제공, 제작, 갱신

#### CRUD

Create, Read, Update, Delete -> 웹 서비스의 가장 기초

## Create, Read, Update, Delete

1. initial 파일

### Spring Boot

☐ 3.0.0 (SNAPSHOT)
 ☐ 3.0.0 (M1)
 ☐ 2.7.0 (SNAPSHOT)
 ☐ 2.7.0 (M1)
 ☐ 2.6.4 (SNAPSHOT)
 ☐ 2.6.3
 ☐ 2.5.10 (SNAPSHOT)
 ☒ 2.5.9

### Project Metadata

Group

dev.dongmin

Artifact

crud

Name

crud

Description

Demo project for Spring Boot

Package name

dev.dongmin:crud

Packaging

☒ Jar
 ☐ War

Java

☐ 17
 ☒ 11
 ☐ 8

### Dependencies

ADD ...

### Spring Web

WEB

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

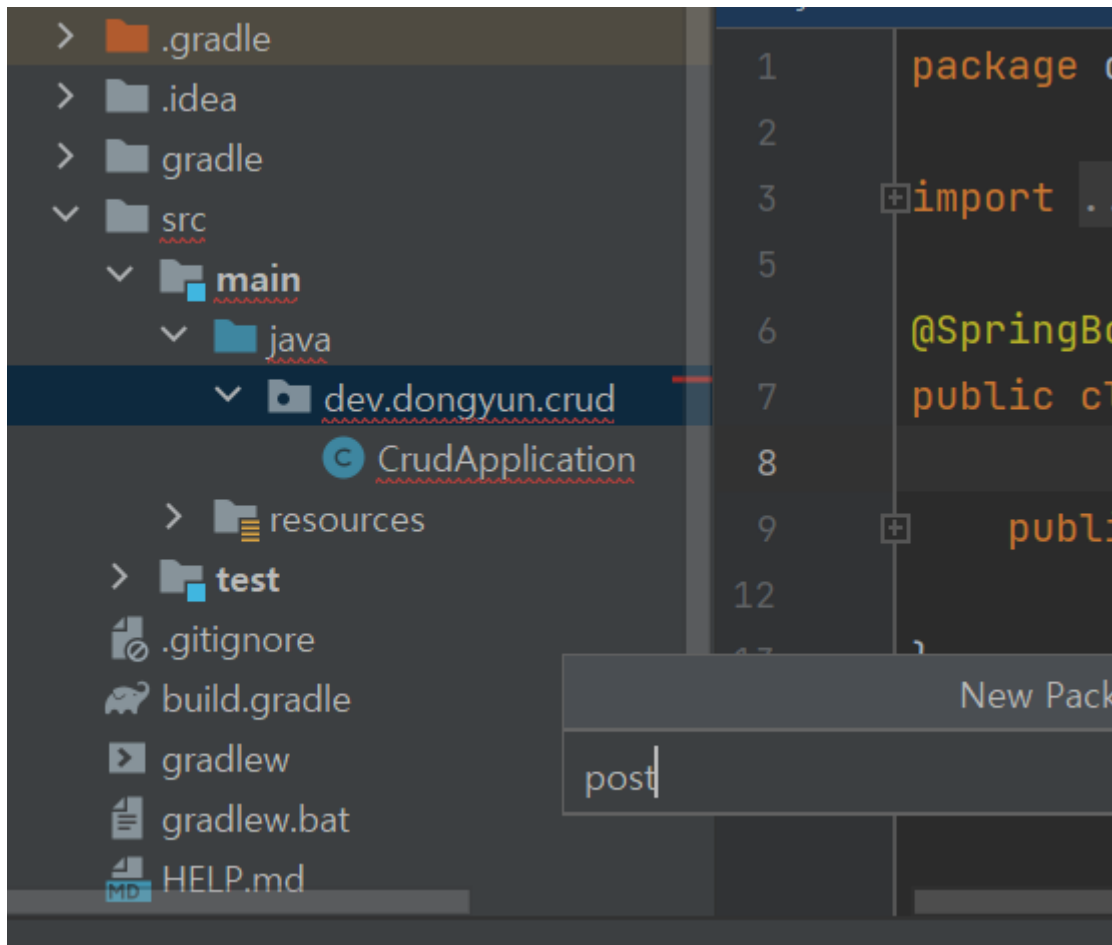
—

GENERATE

EXPLORE

SHARE...

2. 패키지 안에 Post 패키지 만들기



### 3. Post 패키지 안의 PostDto 클래스 선언

```
package dev.dongyun.crud.post;
//Data Transfer Object
//데이터를 주고 받는데 이용
public class PostDto {
    private String title;
    private String content;
    private String writer;

    public PostDto() {
    }

    public PostDto(String title, String content, String writer) {
        this.title = title;
        this.content = content;
        this.writer = writer;
    }

    public String getTitle() {
        return title;
    }

    public String getContent() {
```

```

        return content;
    }

    public String getWriter() {
        return writer;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public void setWriter(String writer) {
        this.writer = writer;
    }

    @Override
    public String toString() {
        return "PostDto{" +
            "title='" + title + '\'' +
            ", content='" + content + '\'' +
            ", writer='" + writer + '\'' +
            '}';
    }

    //Lombok
}

```

## 1. postcontroller생성

```

package dev.dongyun.crud.post;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import java.util.ArrayList;
import java.util.List;

@Controller
@ResponseBody
//이렇게 붙여놓으면
//클래스 안 모든 함수들이 responsebody가
// 붙은 형태로 함수 선언 완료

```

```

@RequestMapping("post")
public class PostController {
    private static final Logger logger = LoggerFactory.getLogger(PostController.class);
    private final List<PostDto> postList;

    public PostController() {
        postList = new ArrayList<>();
        //왜 위에는 list고 아래는 arraylist냐
        // list는 인터페이스, arraylist는 구현체
    }
    @PostMapping("create")
    public void createPost(@RequestBody PostDto postDto){
        logger.info(postDto.toString());
        this.postList.add(postDto);
    }
}

```

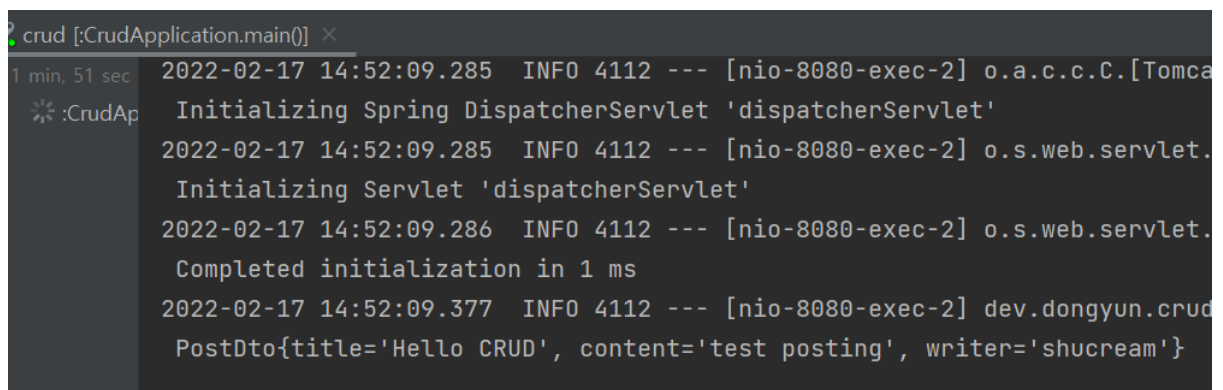
- POSTMAN에서 POST 요청 보내기

```

{
  "title": "Hello CRUD",
  "content": "test posting",
  "writer": "shucream"
}

```

## 로그확인



```

crud [CrudApplication.main()] ×
1 min, 51 sec 2022-02-17 14:52:09.285 INFO 4112 --- [nio-8080-exec-2] o.a.c.c.C.[Tomcat]
:CrudAp Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-02-17 14:52:09.285 INFO 4112 --- [nio-8080-exec-2] o.s.web.servlet.
Initializing Servlet 'dispatcherServlet'
2022-02-17 14:52:09.286 INFO 4112 --- [nio-8080-exec-2] o.s.web.servlet.
Completed initialization in 1 ms
2022-02-17 14:52:09.377 INFO 4112 --- [nio-8080-exec-2] dev.dongyun.crud
PostDto{title='Hello CRUD', content='test posting', writer='shucream'}

```

## CRUD 실습

```

package dev.dongyun.crud.post;

import org.slf4j.Logger;

```

```

import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;

@Controller
@ResponseBody
//이렇게 붙여놓으면
//클래스 안 모든 함수들이 responsebody가
// 붙은 형태로 함수 선언 완료

@RequestMapping("post")
public class PostController {
    private static final Logger logger = LoggerFactory.getLogger(PostController.class);
    private final List<PostDto> postList;

    public PostController() {
        postList = new ArrayList<>();
        //왜 위에는 list고 아래는 arraylist냐
        // list는 인터페이스, arraylist는 구현체
    }

    @PostMapping("create")
    public void createPost(@RequestBody PostDto postDto){
        logger.info(postDto.toString());
        this.postList.add(postDto);
    }

    @GetMapping("read-all")
    public List<PostDto> readPostAll(){
        logger.info("read all");
        return this.postList;
    }

    @GetMapping("read-one")
    public PostDto readPostOne(@RequestParam("id") int id){
        logger.info("read one");
        return this.postList.get(id);
    }

    @PatchMapping("update")
    public void updatePost(
        @RequestParam("id") int id,
        @RequestBody PostDto postDto //포스트 요청의 body
    ){
        PostDto targetPost = this.postList.get(id); //업데이트를 위한 목적 타겟
        if (postDto.getTitle()!=null){
            targetPost.setTitle(postDto.getTitle());
        }
        if (postDto.getContent()!=null){
            targetPost.setContent(postDto.getContent());
        }
        this.postList.set(id, targetPost);
    }

    @DeleteMapping("delete")

```

```

    public void deletePost(@RequestParam("id") int id){
        this.postList.remove(id);
    }
}

```

=> CRUD 기능 구현 완

## (+) PLUS

### 1)

#### | List, ArrayList의 관계

```

private final List<PostDto> postList;
public PostController() {
    postList = new ArrayList<>();
    //왜 위에는 list고 아래는 arrayList냐
    // list는 인터페이스, arrayList는 구현체
}

```

#### 출처블로그

=> List<Object> list = new ArrayList<Object>(); ArrayList<Object> list = new ArrayList<>();

대부분의 ArrayList는 아래보단 위와 같은 형태로 선언하여 사용된다.

ArrayList와 같은 구현체 클래스가 아닌, List라는 인터페이스로 선언하는 식이다.

왜 ArrayList를 주로 저렇게 업캐스팅해서 선언하는지 그 이유를 알아보았다.

요약 : 객체지향 프로그래밍의 일환으로, 다형성을 지원하기 위해서이다.

처음부터 변경에 유연한 구조로 미리 설계하는 방식이라고 할 수 있다

예를 들어.. ArrayList는 빠른 탐색에 유리하다는 장점이 있고,

마찬가지로 List인터페이스를 구현한 LinkedList는 삽입/삭제에 유리하다는 장점이 있다.

만약 ArrayList<Object> list = new ArrayList<>(); 와 같이 ArrayList라는 인스턴스로 선언하면,

나중에 데이터의 용도가 바뀌어 삽입/삭제가 유리한 LinkedList 자료구조로 변경해야 할 때 ArrayList로 선언된 모든 부분을 LinkedList로 변경해 주어야 한다.

또, ArrayList에서는 지원하지 않지만 LinkedList에서는 지원하지 않는 메소드를 사용했다면 그 메소드를 더 이상 사용할 수 없게 된다.

이는 변경에 유연하지 못한 구조라고 할 수 있다.

반면 List<Object> arrList = new ArrayList<>(); 와 같이 List라는 인스턴스로 선언하면,

똑같은 상황이 오더라도 선언부 외에 다른 부분을 변경할 필요가 없다. 이런 부분에서 이점이 있기 때문에 업캐스팅하여 선언하는 것이다.

추가 ) 호눅스의 답변

두 가지 이유가 있다.

대부분의 경우 ArrayList만이 제공하는 기능을 쓰지 않는다.

List를 쓰고 싶은데, List를 구현한 클래스 중에 ArrayList로 그냥 선언하는 것.

List로 선언해야 List에서 제공하는 메소드까지 사용 가능하기 때문이다.

다른 리스트로 바뀌어야 할 때 더 편하기 때문이다.

추가 ) StackOverFlow

이는 인터페이스의 특정 구현과 내 코드를 분리하기 위해서이다.

List list = new ArrayList();와 같이 업캐스팅하면,

나머지 코드는 이 데이터가 List형이라는 것만 알고 있다.

즉, 이렇게 선언한 뒤 코드를 짜면 list의 자료형이 List이므로 모든 코드가 List인터페이스를 따르게 코드를 작성할 수 있고, 따라서 List인터페이스를 구현한 다른 자료형 간에 쉽게 전환할 수 있게 된다.

결론 : 나중에 기존 코드를 바꾸지 않고 인터페이스 내에서 변경하기 쉽도록 하기 위해서이다.

추가 ) 이를 확장하면..

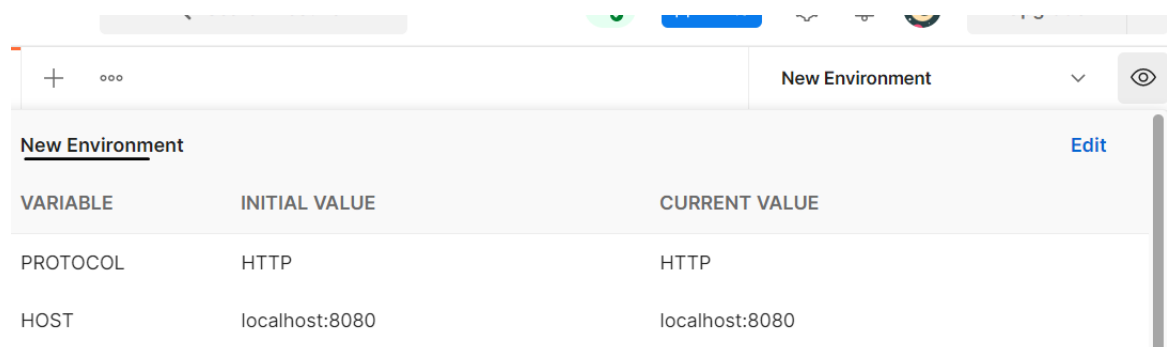
: 객체는 인터페이스를 사용해 선언하는 게 좋다는 결론에 도달할 수 있다.

매개변수 뿐 아니라 리턴값, 변수, 필드를 가능한 인터페이스 타입으로 선언할 수 있다. 적합한 인터페이스가 없다면, 클래스 계층구조 중 가장 상위 클래스를 사용하면 된다.

## 2)

### POSTMAN 환경변수 설정

#### 참조블로그



VARIABLE	INITIAL VALUE	CURRENT VALUE
PROTOCOL	HTTP	HTTP
HOST	localhost:8080	localhost:8080

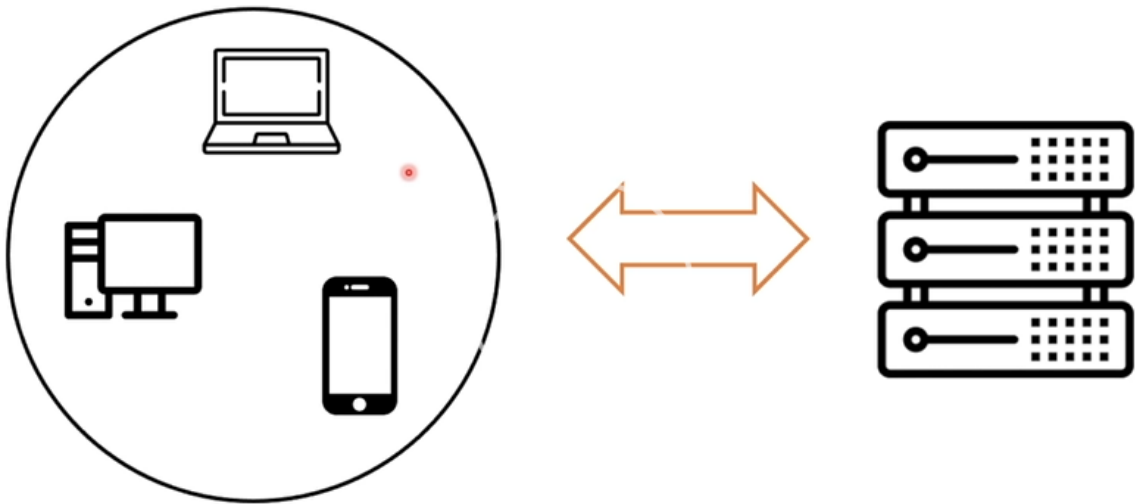


#### New Environment

	VARIABLE	TYPE ①	INITIAL VALUE ②	CURRENT VALUE ③
<input checked="" type="checkbox"/>	PROTOCOL	default	HTTP	HTTP
<input checked="" type="checkbox"/>	HOST	default	localhost:8080	localhost:8080
	Add a new variable			

## RESTful 이란?

- REpresentationl State Transfer



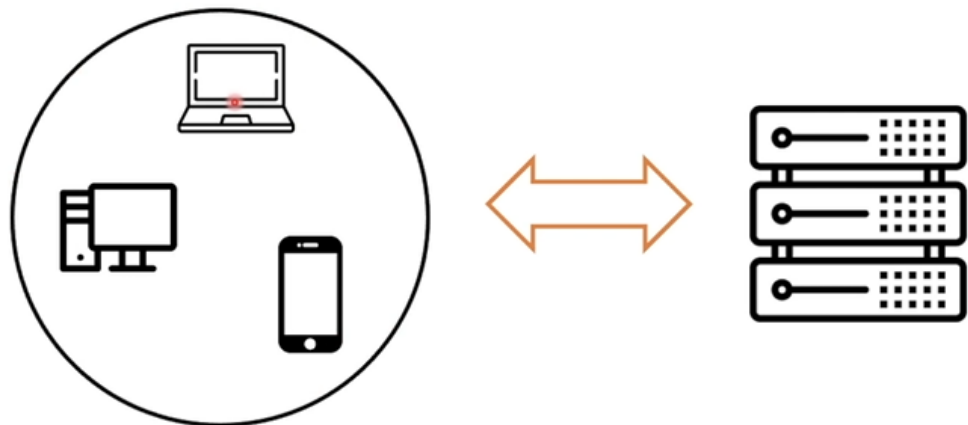
Client와 Server간의 결합성을 줄이기 위한 가이드

- 클라이언트와 서버 간의 결합성을 줄이는 가이드 (fe,be 따로 작업 가능할 수 있도록)
- 클라이언트가 사용할 api를 보기 좋게 만들 수 있는 가이드

## API를 RESTful하게 설계하는 방법

### 1) Client Server Architecture

## 1. Client Server Architecture

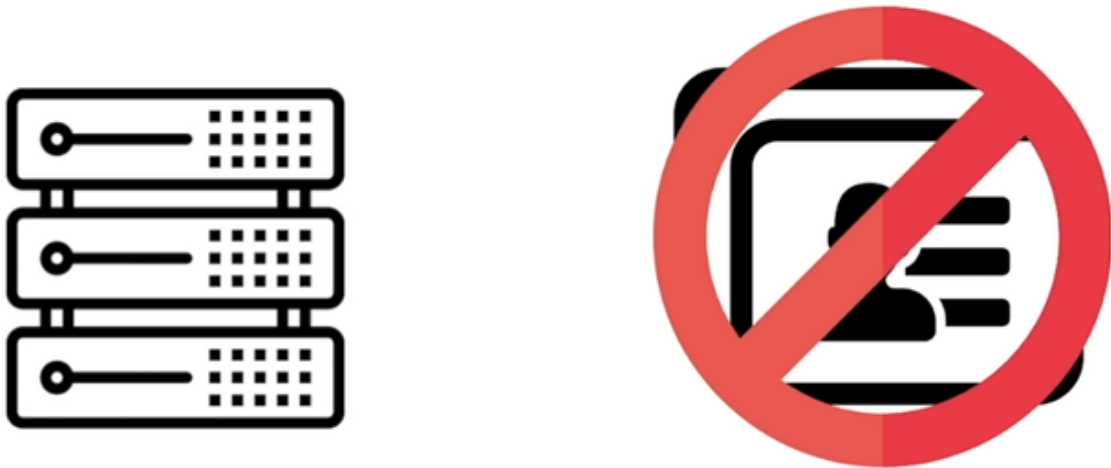


서버와 클라이언트의 분리

서로의 변화가 서로에게 영향을 주지 않는 형태가 되어야 한다.

- 서버와 클라이언트의 분리

## 2) Statelessness



- 서버 사용하는 대상이 있음
- 요청 받을 때마다 해당 사용자가 누군지 매번 체크해야 함
- 즉 사용자의 상태 저장하지 않음
- 리퀘스트 보내는 클라이언트는 **자신을 증명할 책임** 이 존재 (보안과 관련)
- 원하는 기능을 위한 상태는 client가 가져야 한다 (내가 글을 수정할 지, 삭제할 지 등)

### 3) Cacheability



#### 캐시 가능성

자원의 캐싱이 가능한지의 여부를 항상 표기해줘야 한다.

- 한번 정보 받으면 이 정보가 캐싱 가능한지 여부 체크 해야 한다
- 상태를 저장할 수 없으니깐 캐시로 정보를 저장하기

### 4) Layered System

- 서버와 클라이언트 간에 계층적인 구조 존재
- 하지만 클라이언트는 서버에 도달하기까지의 과정을 알 필요 없음

### 5) Uniformed interface

#### 5. Uniformed Interface



C Create  
R Read  
U Update  
D Delete



일관된 인터페이스

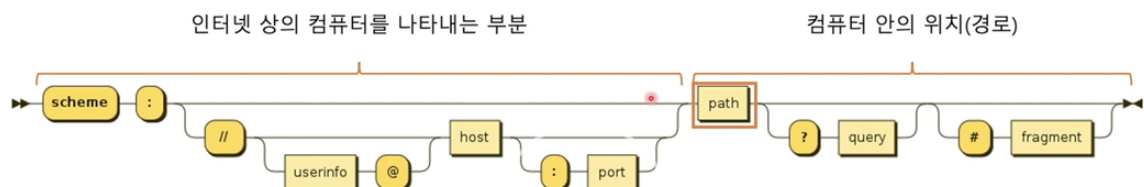
- 일관된 인터페이스  
=> 자원을 나타내기 위해서 사용해야 하는 인터페이스  
=> 돌려줄 땐 json을 돌려준다 등

## 6) Code on Demand (옵셔널 기능)

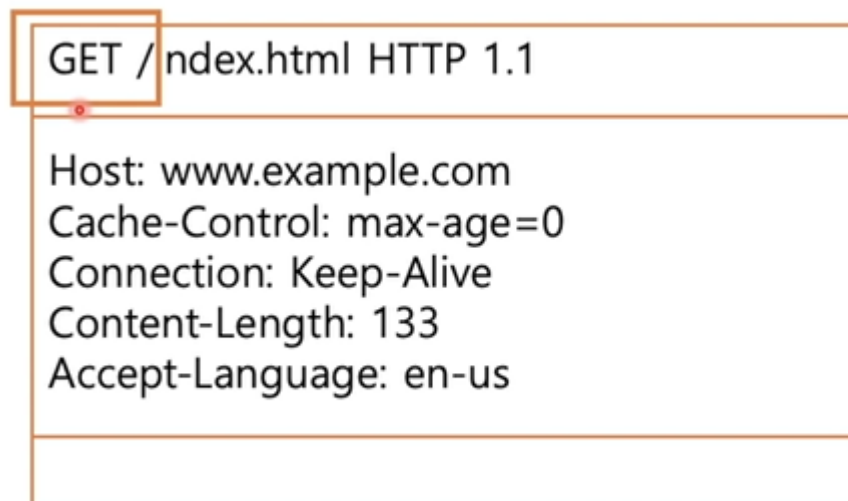
- 일시적 기능의 확장
- 사용 가능한 코드를 응답을 보내서 사용자 기능을 일시적으로 확장하기

## 고려해야 할 사항

### 1. uri



- uri를 통해 전달될 데이터는 아래의 형식으로
  - > 1) 경로를 통해 도달하고자 하는 자원을 지정
  - > 2) 방법을 통해 자원에 실행할 기능 지정



## RequestMapping 재구성

```

package dev.dongyun.crud.post;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;

@RestController
@RequestMapping("post")
public class PostRestController {
    private static final Logger logger = LoggerFactory.getLogger(PostRestController.class);
    private final List<PostDto> postList;

    public PostRestController(){
        this.postList = new ArrayList<>();
    }

    @PostMapping()
    public void createPost(@RequestBody PostDto postDto){
        logger.info(postDto.toString());
        this.postList.add(postDto);
    }

    @GetMapping()
    public List<PostDto> readPostAll(){
        logger.info("read-all");
        return this.postList;
    }

    //GET
    //GET /post?id=0

    @GetMapping("/{id}")
    public PostDto readPost(@PathVariable("id") int id){
        logger.info("in read post");
        return this.postList.get(id);
    }

    @PutMapping("/{id}")
    public void updatePost(
        @PathVariable("id") int id,
        @RequestBody PostDto postDto
    ){
        PostDto targetPost = this.postList.get(id); //업데이트를 위한 목적 타겟
        if (postDto.getTitle()!=null){
            targetPost.setTitle(postDto.getTitle());
        }
        if (postDto.getContent()!=null){
            targetPost.setContent(postDto.getContent());
        }
        this.postList.set(id, targetPost);
    }
}

```

```

@DeleteMapping("{id}")
public void deletePost(@PathVariable("{id}") int id){
    this.postList.remove(id);
}
}

```

## error - status code

[https://ko.wikipedia.org/wiki/HTTP\\_상태\\_코드](https://ko.wikipedia.org/wiki/HTTP_상태_코드) `@ResponseStatus(HttpStatus.CREATED)` 넣어 주기!

### 2xx (성공)

이 클래스의 상태 코드는 클라이언트가 요청한 동작을 수신하여 이해했고 승낙했으며 성공적으로 처리했음을 가리킨다.

200(성공): 서버가 요청을 제대로 처리했다는 뜻이다. 이는 주로 서버가 요청한 페이지를 제공했다는 의미로 쓰인다.

201(작성됨): 성공적으로 요청되었으며 서버가 새 리소스를 작성했다.

202(허용됨): 서버가 요청을 접수했지만 아직 처리하지 않았다.

203(신뢰할 수 없는 정보): 서버가 요청을 성공적으로 처리했지만 다른 소스에서 수신된 정보를 제공하고 있다.

204(콘텐츠 없음): 서버가 요청을 성공적으로 처리했지만 콘텐츠를 제공하지 않는다.

205(콘텐츠 재설정): 서버가 요청을 성공적으로 처리했지만 콘텐츠를 표시하지 않는다. 204 응답과 달리 이 응답은 요청자가 문서 보기를 재설정할 것을 요구한다(예: 새 입력을 위한 양식 비우기).

206(일부 콘텐츠): 서버가 GET 요청의 일부만 성공적으로 처리했다.

207(다중 상태, RFC 4918)

208(이미 보고됨, RFC 5842)

226 IM Used (RFC 3229)

### 4xx (요청 오류)

4xx 클래스의 상태 코드는 클라이언트에 오류가 있음을 나타낸다.

400(잘못된 요청): 서버가 요청의 구문을 인식하지 못했다.

401(권한 없음): 이 요청은 인증이 필요하다. 서버는 로그인이 필요한

페이지에 대해 이 요청을 제공할 수 있다. 상태 코드 이름이 권한 없음(Unauthorized)으로 되어 있지만 실제 뜻은 인증 안됨(Unauthenticated)에 더 가깝다.[2]

402(결제 필요): 이 요청은 결제가 필요합니다.

403(Forbidden, 금지됨): 서버가 요청을 거부하고 있다. 예를 들자면, 사용자가 리소스에 대한 필요 권한을 갖고 있지 않다. (401은 인증 실패, 403은 인가 실패라고 볼 수 있음)

404(Not Found, 찾을 수 없음): 서버가 요청한 페이지(Resource)를 찾을 수 없다. 예를 들어 서버에 존재하지 않는 페이지에 대한 요청이 있을 경우 서버는 이 코드를 제공한다.

405(허용되지 않는 메소드): 요청에 지정된 방법을 사용할 수 없다. 예를 들어 POST 방식으로 요청을 받는 서버에 GET 요청을 보내는 경우, 또는 읽기 전용 리소스에 PUT 요청을 보내는 경우에 이 코드를 제공한다.

406(허용되지 않음): 요청한 페이지가 요청한 콘텐츠 특성으로 응답할 수 없다.

407(프록시 인증 필요): 이 상태 코드는 401(권한 없음)과 비슷하지만 요청자가 프록시를 사용하여 인증해야 한다. 서버가 이 응답을 표시하면 요청자가 사용할 프록시를 가리키는 것이기도 한다.

408(요청 시간초과): 서버의 요청 대기가 시간을 초과하였다.

5xx (서버 오류) : 서버가 유효한 요청을 명백하게 수행하지 못했음을 나타낸다

500(내부 서버 오류): 서버에 오류가 발생하여 요청을 수행할 수 없다.

501(구현되지 않음): 서버에 요청을 수행할 수 있는 기능이 없다. 예를 들어 서버가 요청 메소드를 인식하지 못할 때 이 코드를 표시한다.

502 (Bad Gateway, 불량 게이트웨이): 서버가 게이트웨이나 프록시 역할을 하고 있거나 또는 업스트림 서버에서 잘못된 응답을 받았다.

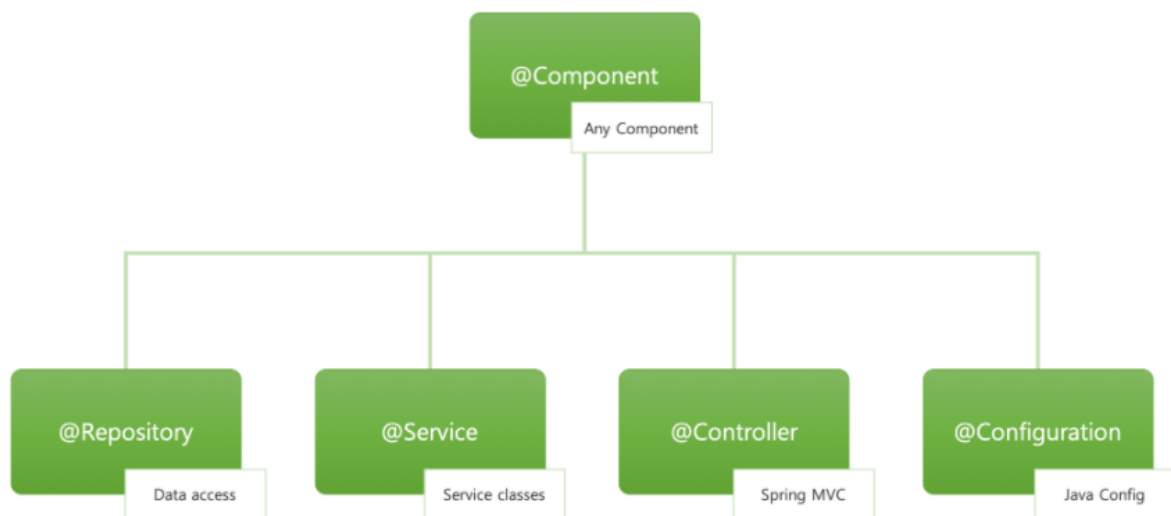
503(서비스를 사용할 수 없음): 서버가 오버로드되었거나 유지관리를 위해 다운되었기 때문에 현재 서버를 사용할 수 없다. 이는 대개 일시적인 상태이다.

504(게이트웨이 시간초과): 서버가 게이트웨이나 프록시 역할을 하고 있거나 또는 업스트림 서버에서 제때 요청을 받지 못했다.

505(HTTP 버전이 지원되지 않음): 서버가 요청에 사용된 HTTP 프로토콜 버전을 지원하지 않는다.

## Spring Stereotypes

아래의 이미지, 설명 출처 블로그



### StereoType이란?

스프링 컨테이너가 스프링 관리 컴포넌트로 식별하게 해주는 단순한 마커다. 즉, scan-auto-detection과 dependency injection을 사용하기 위해서 사용되는 가장 기본 어노테이션이다.

Stereo Type이 범용적으로 많이 사용하게 된 시키는 Spring 2.5 부터였다. 그 이전까지는 xml 파일에 bean을 등록하여 관리 하였다. 그러나 모든 bean들을 xml 파일로 관리 하다보니 다른 보일러플레이트 코드와 함께 xml 파일만 비대해지게 될 뿐이었다. 그래서 Spring 2.0 에서는 @Repository이 등장하였고 Spring 2.5 부터는 @Component, @Controller, @Service, @Configuration 등이 등장하면서 Stereo type에 대한 정의가 범용적으로 사용하게 되었다.



## @Component

빈으로 간주되어 DI 컨테이너에서 사용할 수 있게 하는 기본 어노테이션이다

## @Bean과 @Component는 어떤 차이가 있는가?

@Component는 클래스 상단에 적으며 그 default로 클래스 이름이 bean의 이름이 된다. 또한 spring에서 자동으로 찾고 (@ComponentScan 사용) 관리해주는 bean이다.

@Bean은 @Configuration으로 선언된 클래스 내에 있는 메소드를 정의할 때 사용한다. 이 메소드가 반환하는 객체가 bean이 되며 default로 메소드 이름이 bean의 이름이 된다.

## @Controller

Dispatcher Sevlet에서 제공되며 여기서 @RequestMapping 어노테이션을 사용하여 클라이언트 요청을 특정 컨트롤러에 전달할 수 있게 해준다

## @Service

Service annotation은 단순히 서비스 레이어에서 사용하는 bean이라는 것을 구분하기 위한 용도로 정의한다.

## @Repository

검사 되지 않은 예외(DAO 메소드에서 발생)를 Spring DataAccessException으로 변환 할 수 있게 해준다.

## @Configuration

한 개 이상의 @Bean 어노테이션으로 정의한 bean 들을 생성하려 할 때 클래스에 정의하는 어노테이션이다.

## @Bean은 언제 사용하는게 좋을까?

개발자가 직접 제어가 불가능한 라이브러리를 활용할 때 사용  
초기에 설정을 하기 위해 활용할 때 사용

- ioc 컨테이너가 만들어놓은 코드에서 빈을 검색해서 필요한 순간순간에 제공하는 것 => 스프링프레임워크의 핵심 기능
- 이때 이 빈들을 함수, 클래스 단위로 정의가능
- 클래스 단위로 정의하기 위한 것들이 구현돼있는 패키지 이름이 `Spring Stereotypes`

# Component란?

## 0. @SpringBootConfiguration

@EnableAutoConfiguration

@ComponentScan

- spring initializer로 만들어진 main함수 보게 되면 아래와 같음

```
@SpringBootApplication
public class CrudApplication {

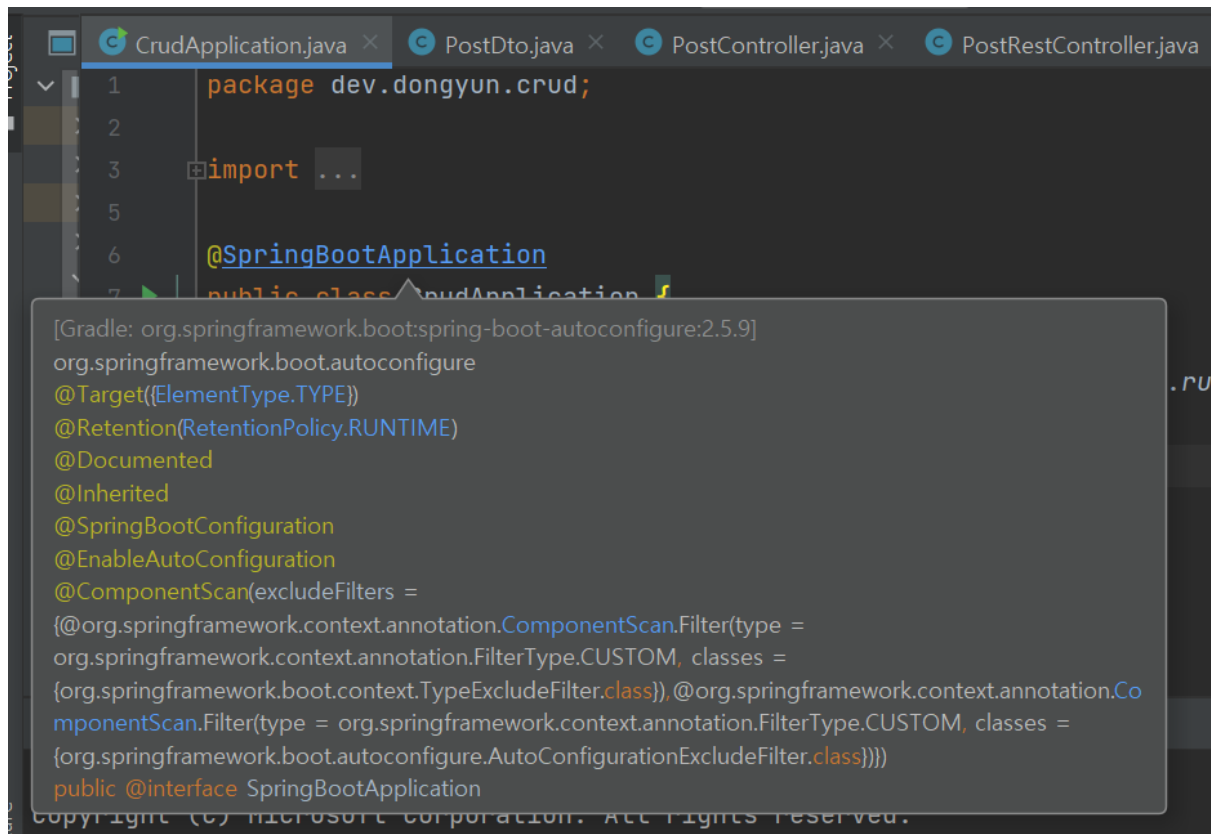
    public static void main(String[] args) {
        SpringApplication.run(CrudApplication.class, args);
    }

}
```

- > 이때

```
@SpringBootApplication
```

이라는 아이는 아래와 같은 설명을 가짐



!-> 이를 더 자세히 살펴본다면 아래와 같은데 이때 빨간색 밑줄 친 아이 두개를 같이 사용하게 된다면

```

@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = { @Filter(type = FilterType.CUSTOM, c
    @Filter(type = FilterType.CUSTOM, classes = AutoConfiguration
public @interface SpringBootApplication {

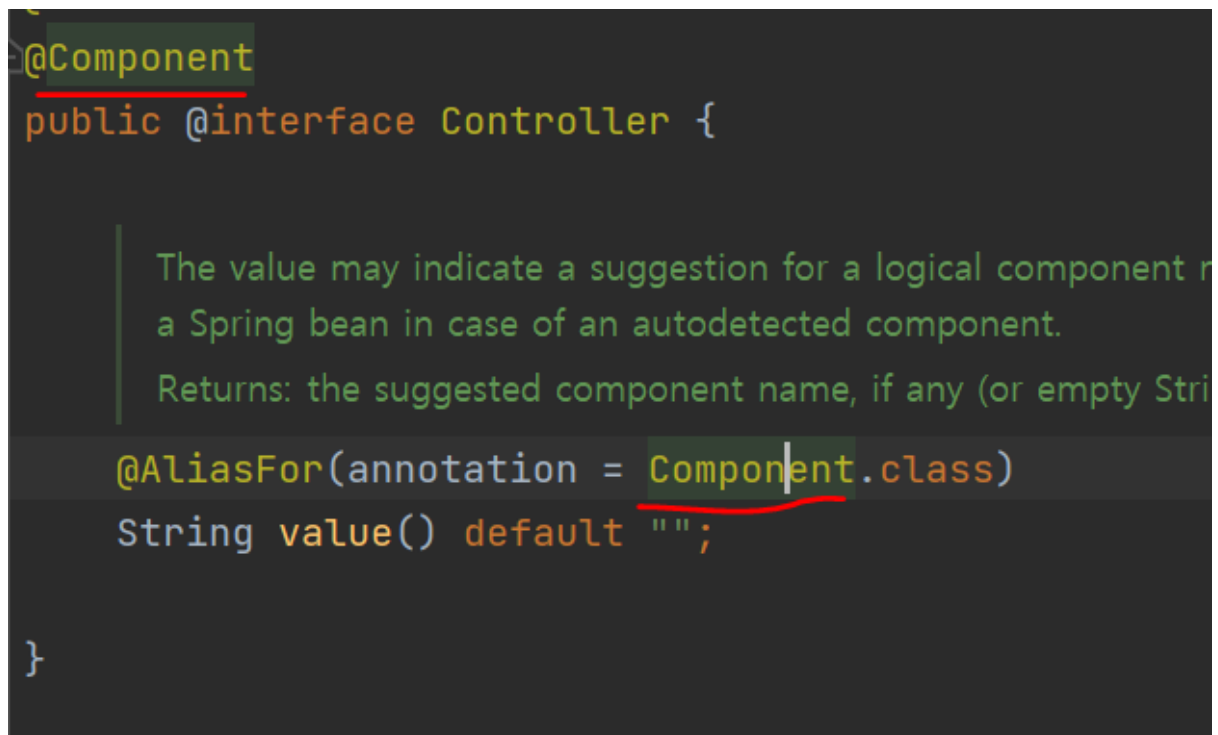
```

- > 여러 패키지가 존재하는데 이 중 어떤 패키지를 사용을 해서, 개네들을 ioc 컨테이너에 등록할 지 여부 지정해주는 annotation
- > 다만 스프링 부트에서는 마이크로 서비스 아키텍처, 도커 이미지화에 더 유리해서 굳이 검색하지 않을 패키지를 콤포넌트해줄 이유가 없어서 딱히 하지 않을 것

Component Scan : 안에 패키지를 검색할 수 있도록 지정해주는 것이다.

## 1. public @interface Component 살피기

- > 이 친구는 controller의 상위



- > stereotype ( `package org.springframework.stereotype;` )패키지 안에 자리잡고 있는 녀석이다
- 우리가 사용했던 Controller들도 사실상 Component의 일종이라고 볼 수 있는 것
- 또한 컨트롤러 말고도 다른 여러가지 만들기 가능

```

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Indexed
public @interface Component {

    The value may indicate a suggestion for a
    a Spring bean in case of an autodetected c
    Returns: the suggested component name, i

    String value() default "";

}

```

Component가 붙은 클래스들을 검색해서 애플 스프링컨테이너에 등록해주는 것이 Component 어노테이션의 역할, 스프링 컨테이너의 관리 하에 있게 해준다

(+) Restcontroller

```

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
@ResponseBody
public @interface RestController {

```

- 위와 같이 Controller와 ResponseBody 어노테이션을 가지고 있다

- 그래서 RestController 사용할 땐 굳이 responseBody 안넣어도 되는 것!!

=> 어플리케이션에서 어떤애가 무슨 역할 하는지는 모르지만 스프링 관리 하에 두고싶다고 한다면 component interface

=> request endpoint를 만들고 싶을 땐 controller interface

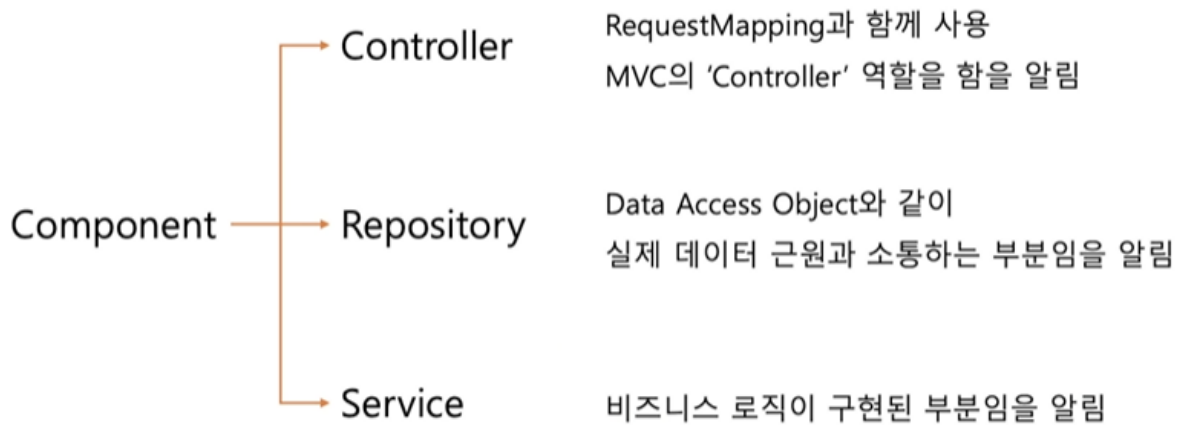
---

### 3. Spring 부트에서 Component는??



- ComponentScan을 이용해 사용할 Bean의 범위를 지정해준다  
=> 함수 단위 : @Bean  
=> 클래스 단위 : @Component
  - 위의 단위들을 붙여주게 된다면 Spring Ioc 컨테이너가 이 아이들을 관리해주게 되는 것
- 

### 4. Component의 아이들 (Controller, Repository, Service)



모든 Bean에 Component를 사용해도 작동하기는 한다.

- 물론 Controller, Repository, Service 다 컴포넌트의 일종이기 때문에 모든 Bean에 Component 사용해도 작동 가능 ○○ 그래도 위와 같이 세분화 시켜서 나눠준다!

## Service, Repository 사용하기

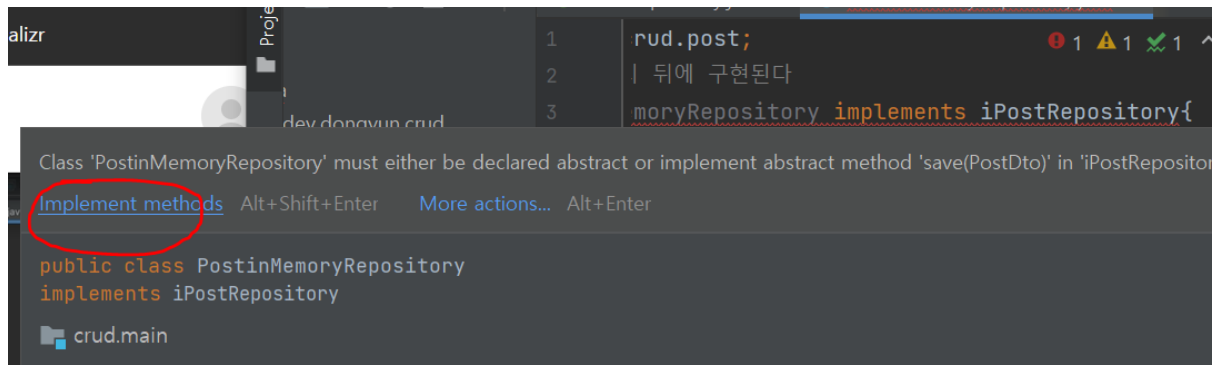
### 1) PostRepository Interface 선언

```
package dev.dongyun.crud.post;  
  
import java.util.List;  
  
public interface iPostRepository {  
    boolean save(PostDto dto);  
    PostDto findById(int id); //id를 주게 되면 PostDto가 돌아가게 된다  
    List<PostDto> findAll();  
    boolean update(PostDto dto);  
    boolean delete(int id);  
}
```

### 2) PostinMemoryRepository

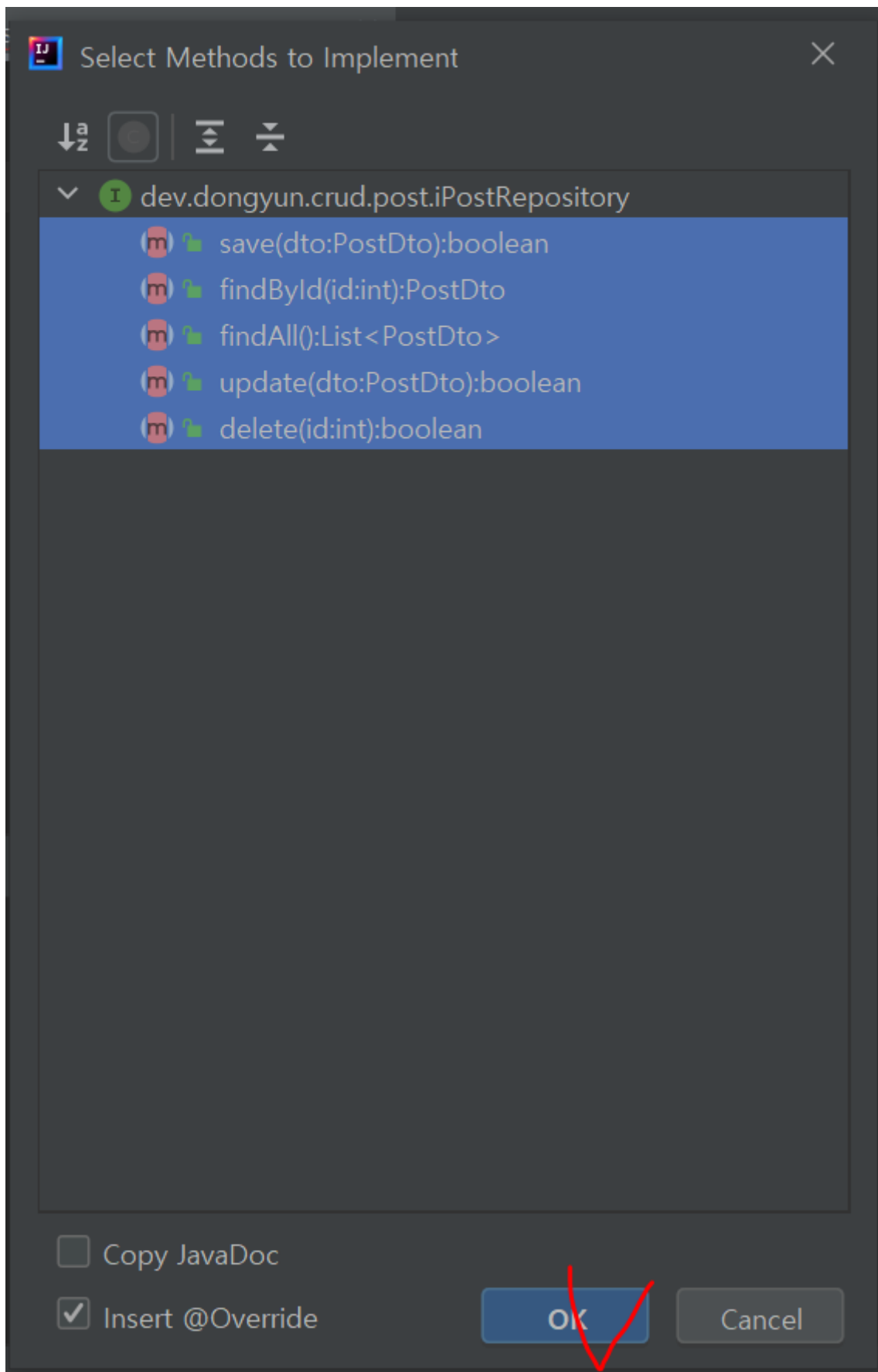
```
package dev.dongyun.crud.post;  
//포스트 메모리지만 메모리 뒤에 구현된다  
public class PostinMemoryRepository implements iPostRepository{  
}
```

- 우선 이렇게 하면 에러가 난다



- 이 implement methods를 통해 메소드를 만들어주기





```

package dev.dongyun.crud.post;

import java.util.List;

//포스트 메모리지만 메모리 뒤에 구현된다
public class PostinMemoryRepository implements iPostRepository{
    @Override
    public boolean save(PostDto dto) {
        return false;
    }

    @Override
    public PostDto findById(int id) {
        return null;
    }

    @Override
    public List<PostDto> findAll() {
        return null;
    }

    @Override
    public boolean update(PostDto dto) {
        return false;
    }

    @Override
    public boolean delete(int id) {
        return false;
    }
}

```

- 그리고 이제 @Repository annotaion 붙이고 crud 함수들 선언

```

package dev.dongyun.crud.post;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Repository;

import java.util.ArrayList;
import java.util.List;
@Repository
//포스트 메모리지만 메모리 뒤에 구현된다
public class PostinMemoryRepository implements iPostRepository{

    private static final Logger logger = LoggerFactory.getLogger(PostinMemoryRepository.class);
    private final List<PostDto> postList;

    public PostinMemoryRepository(){
        this.postList = new ArrayList<>();
    }
}

```

```

@Override
public boolean save(PostDto dto) {
    return this.postList.add(dto);
    //-> add가 애초에 boolean값을 돌려준
    //return true;
}

@Override
public PostDto findById(int id) {
    return this.postList.get(id);
}

@Override
public List<PostDto> findAll() {
    return this.postList;
}

@Override
public boolean update(int id, PostDto dto) {
    PostDto targetPost = this.postList.get(id); //업데이트를 위한 목적 타겟
    if (dto.getTitle()!=null){
        targetPost.setTitle(dto.getTitle());
    }
    if (dto.getContent()!=null){
        targetPost.setContent(dto.getContent());
    }
    this.postList.set(id, targetPost);

    return true;
}
//return 값은 성공 여부 알려주는 것
@Override
public boolean delete(int id) {
    this.postList.remove(id);
    return true;
}
}

```

### 3) PostService 만들기

#### 1. postservice 인터페이스 구현

레포지토리랑 서비스

데이터를 회수해서 그것을 확인하는 것 -> 레포지토리 (데이터 회수)

회수된 데이터에서 현재 요청을 보내는 사용자의 권한 여부 체크 -> 서비스

(+) 서비스는 실제 데이터 사용 전 검증 거칠 때 used

(+)아래 설명 출처 블로그

- @Controller 어노테이션을 붙이면 핸들러가 스캔할 수 있는 빈(Been) 객체가 되어 서블릿용 컨테이너에 생성됩니다. 마찬가지로 @Repository, @Service 어노테이션은 해당 클래스를 루트 컨테이너에 빈(Been) 객체로 생성해주는 어노테이션입니다.
- 둘 다 Bean 객체를 생성해주고 딱히 다른 기능을 넣어주는게 아니라서 뭘 써도 상관 없지만 명시적으로 구분해주기 위해 각자 분리해서 사용합니다. 부모 어노테이션인 @Component를 붙여줘도 똑같이 루트 컨테이너에 생성되지만 가시성이 떨어지기 때문에 잘 사용하지 않습니다.
- 참고로 객체 내에서 데이터 변경 작업이 있는 VO(DTO) 객체와 같은 경우는 동기화 문제로 인해 Bean 객체로 사용하지 않습니다. Bean 객체는 항상 데이터 변경이 없는 객체에 한해 사용하는 점에 유의해야 합니다.

## 컨트롤러 : @Controller (프레젠테이션 레이어, 웹 요청과 응답을 처리함)

- 로직 처리 : @Service (서비스 레이어, 내부에서 자바 로직을 처리함)
- 외부 I/O 처리 : @Repository (퍼시스턴스 레이어, DB나 파일같은 외부 I/O 작업을 처리함)

```
package dev.d~n.crud.post;

import java.util.List;

public interface PostService {
    void createPost(PostDto dto);
    List<PostDto> readPostAll();
    PostDto readPost(int id);
    void updatePost(int id, PostDto dto);
    void deletePost(int id);
}
```

### 1. postservice 인터페이스 사용할 postservicesimple 클래스 만들

```
package dev.dongyun.crud.post;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.List;

public class PostServiceimple implements PostService {
    private static final Logger logger = LoggerFactory.getLogger(PostServiceimple.class);
}
```

```

    private final iPostRepository postRepository;
    //레포지토리 in이 아닌
    //우리는 애 (인터페이스) 이후, 뒤의 구현이 어떻게 되어있는지 상관않겠다는 뜻

    public PostServiceimple(iPostRepository postRepository){
//여기에 autowired annotation을 붙여줘서 레포지토리 중 알맞은 레포지토리 찾게 해줘야 함
        this.postRepository = postRepository;
    }

    @Override
    public void createPost(PostDto dto) {

    }

    @Override
    public List<PostDto> readPostAll() {
        return null;
    }

    @Override
    public PostDto readPost(int id) {
        return null;
    }

    @Override
    public void updatePost(int id, PostDto dto) {

    }

    @Override
    public void deletePost(int id) {

    }
}

```

- 위와 같이 메소드 부르고 로그 선언하고 이제 메소드들 메꿔주기

```

package dev.dongyun.crud.post;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class PostServiceimple implements PostService {
    private static final Logger logger = LoggerFactory.getLogger(PostServiceimple.class);

    private final iPostRepository postRepository;

    public PostServiceimple(
        @Autowired iPostRepository postRepository
    )

```

```

    ) {
        this.postRepository = postRepository;
    }

    @Override
    public List<PostDto> readPostAll() {
        return this.postRepository.findAll();
    }

    @Override
    public void createPost(PostDto dto) {
        if(!this.postRepository.save(dto)){
            throw new RuntimeException(("save failed"));
        }
        this.postRepository.save(dto);
    }
    @Override
    public PostDto readPost(int id) {
        return this.postRepository.findById(id);
    }

    @Override
    public void updatePost(int id, PostDto dto) {
        this.postRepository.update(id, dto);
    }

    }

    @Override
    public void deletePost(int id) {
        this.postRepository.delete(id);
    }
}

```

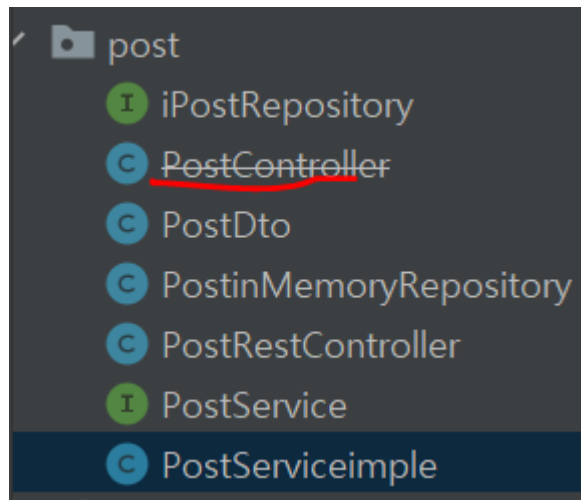
아래 3번 부분은 무시, Deprecated의 개념 용도만 체크

1. ~~그리고 이제 우리는 postcontroller가 아닌 restcontroller 사용할 예정~~  
~~-> 따라서 post rest controller 위에 @Deprecated 붙여주기~~

```

//Deprecated는 당장 지원은 해주지만 곧 쓰지 못할 버전이라고 알려주는 것
//오로지 응답을 받고 요청을 보내는 역할로만
@Deprecated

```



그럼 이렇게 빗금 처리됨 & 더 이상 아래 항목들 관리 진행하지 않음

```
public PostServiceimple(  
    @Autowired iPostRepository postRepository  
) {  
    this.postRepository = postRepository;  
}
```

```
public PostServiceimple(  
    @Autowired iPostRepository postRepository  
) {  
    this.postRepository = postRepository;  
}
```

- > iPostRepository인터페이스에 해당하는 아이들을 ioc가 찾아서 강제로 넣어주는 것- Dependency injection 이라는 특징이었지

1. 이제 service와 postcontroller 연결해서 사용

(+)참고 postservice interface

```
package dev.dongyun.crud.post;  
  
import java.util.List;  
  
public interface PostService {  
    void createPost(PostDto dto);  
    List<PostDto> readPostAll();  
    PostDto readPost(int id);  
}
```

```

    void updatePost(int id, PostDto dto);
    void deletePost(int id);
}

```

기존에 지정해뒀던 postlist -> postservice의 메소드 호출 방식으로 진행

```

package dev.dongyun.crud.post;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;
//Deprecated는 당장 지원은 해주지만 곧 쓰지 못할 버전이라고 알려주는 것
//오로지 응답을 받고 요청을 보내는 역할로만
@Controller
@ResponseBody
//이렇게 붙여놓으면
//클래스 안 모든 함수들이 responsebody가
// 붙은 형태로 함수 선언 완료

@RequestMapping("post")
public class PostController {
    private static final Logger logger = LoggerFactory.getLogger(PostController.class);
    private final PostService postservice;

    public PostController(
        @Autowired PostService postservice) {
        this.postservice = postservice;
        //왜 위에는 list고 아래는 arraylist냐
        // list는 인터페이스, arraylist는 구현체
    }
    @PostMapping("create")
    public void createPost(@RequestBody PostDto postDto){
        logger.info(postDto.toString());
        this.postservice.createPost(postDto);
    }

    @GetMapping("read-all")
    public List<PostDto>readPostAll(){
        logger.info("read all");
        return this.postservice.readPostAll();
    }
    @GetMapping("read-one")
    public PostDto readPostOne(@RequestParam("id") int id){
        logger.info("read one");
        return this.postservice.readPost(id);
    }

    @PatchMapping("update")

```



```

public void updatePost(
    @RequestParam("id") int id,
    @RequestBody PostDto postDto //포스트 요청의 body
){
    PostDto targetPost = this.postservice.readPost(id); //업데이트를 위한 목적 타겟
    if (postDto.getTitle()!=null){
        targetPost.setTitle(postDto.getTitle());
    }
    if (postDto.getContent()!=null){
        targetPost.setContent(postDto.getContent());
    }
    this.postservice.updatePost(id, targetPost);
}

@DeleteMapping("delete")
public void deletePost(@RequestParam("id") int id){
    this.postservice.deletePost(id);
}
}

```

## 1. RestController도 postlist 없애고 postservice 다루는 방향으로 고쳐봅세~~

```

package dev.dongyun.crud.post;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;

@RestController
@RequestMapping("post")
public class PostRestController {
    private static final Logger logger = LoggerFactory.getLogger(PostRestController.class);
    private final PostService postservice;
    public PostRestController(
        @Autowired PostService postservice
    ){
        this.postservice = postservice;
    }

    @PostMapping()
    @ResponseStatus(HttpStatus.CREATED)
    public void createPost(@RequestBody PostDto postDto){
        logger.info(postDto.toString());
        this.postservice.createPost(postDto);
    }

    @GetMapping()
    public List<PostDto> readPostAll(){

```

```

        logger.info("read-all");
        return this.postservice.readPostAll();
    }

    //GET
    //GET /post?id=0

    @GetMapping("/{id}")
    public PostDto readPost(@PathVariable("id") int id){
        logger.info("in read post");
        return this.postservice.readPost(id);
    }

    @PutMapping("/{id}")
    public void updatePost(
        @PathVariable("id") int id,
        @RequestBody PostDto postDto
    ){
        PostDto targetPost = this.postservice.readPost(id); //업데이트를 위한 목적 타겟
        if (postDto.getTitle()!=null){
            targetPost.setTitle(postDto.getTitle());
        }
        if (postDto.getContent()!=null){
            targetPost.setContent(postDto.getContent());
        }
        this.postservice.updatePost(id, targetPost);
    }

    @DeleteMapping("/{id}")
    public void deletePost(@PathVariable("/{id}") int id){
        this.postservice.deletePost(id);
    }
}

```

- 추가적으로 건네줄 때 HttpServlet도 넣기 가능  
=> 기존코드

```

    @PostMapping()
    @ResponseStatus(HttpStatus.CREATED)
    public void createPost(@RequestBody PostDto postDto){
        logger.info(postDto.toString());
        this.postservice.createPost(postDto);
    }

```

=> HttpServlet 추가적으로 넣기

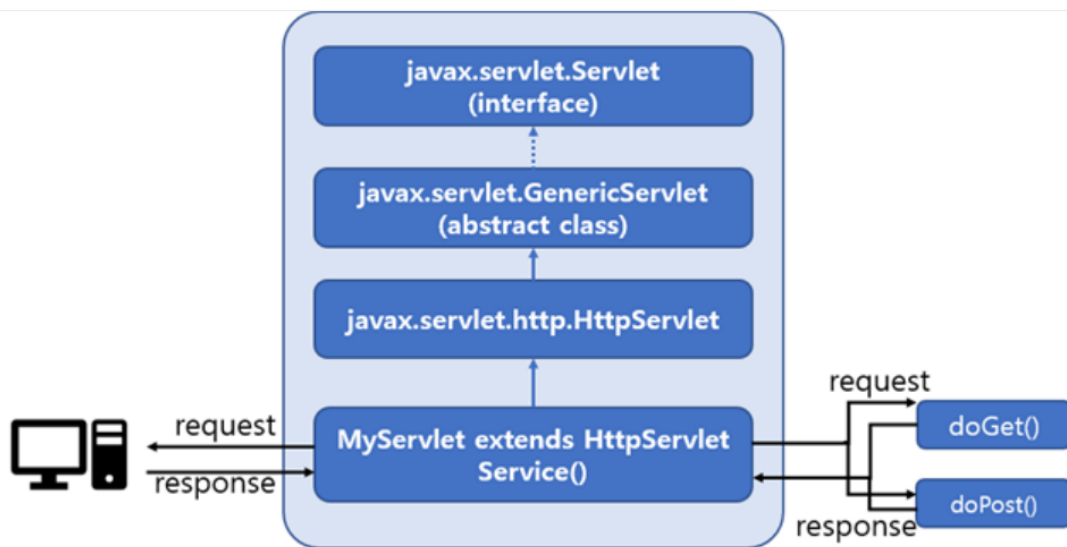
```

    public void createPost(@RequestBody PostDto postDto, HttpServletRequest request){
        logger.info(postDto.toString());
        this.postservice.createPost(postDto);
    }

```

=> HttpServlet의 구조는 아래와 같음 (설명, 이미지 출처)

- 클라이언트 요청에 따라 서블릿 컨테이너는 service() 메서드를 호출하고, service() 메서드는 요청이 GET인지 POST인지 구분하여 각각 doGet(), doPost() 메서드를 호출한다.



HttpServlet을 상속받은 서블릿 동작 구조

- http servlet test

```
@PostMapping()
@ResponseStatus(HttpStatus.CREATED)
public void createPost(@RequestBody PostDto postDto, HttpServletRequest request){
    logger.info("content");
    logger.info(postDto.toString());
    logger.info("full request");
    logger.info(String.valueOf(request));
    logger.info("header of request");
    logger.info(request.getHeader("Content-type"));
    this.postservice.createPost(postDto);
}
```

```
: content
: PostDto{title='request

: full request
: org.apache.catalina

: header of request
: application/json
```

=> 위와 같이 request는 json 형태로 온다는 걸 알 수 있고

```
logger.info("method of request");
logger.info(request.getMethod());
```

- getMethod를 통해서 request로부터 사용된 메소드 정보도 get 가능

```
: content
: PostDto{title='request

: full request
: org.apache.catalina

: method of request
: POST
```

그 외에도 request에서 얻어오는 함수들 (

[이미지, 설명 출처 블로그](#)

)

request 객체는 요청된 파라미터 값 외에도  
웹 브라우저와 웹 서버의 정보도 가져올 수 있음

메소드	설명
<i>String getProtocol()</i>	웹 서버로 요청 시 사용 중인 프로토콜 리턴
<i>String getServerName()</i>	서버의 도메인 이름 리턴
<i>String getMethod()</i>	요청에 사용된 요청 방식(GET, POST 등) 리턴
<i>String getQueryString()</i>	요청에 사용된 QueryString 리턴
<i>String getRequestURL()</i>	요청에 사용된 URL 주소 리턴
<i>String getRequestURI()</i>	요청에 사용된 URL로부터 URI값 리턴
<i>String getRemoteHost()</i>	웹 서버로 정보를 요청한 웹 브라우저의 host 이름 리턴
<i>String getRemoteAddr() : String</i>	웹 서버로 정보를 요청한 웹 브라우저의 ip 주소 리턴
<i>String getServerPort()</i>	웹 서버로 요청시 서버의 port 번호 리턴
<i>String getContextPath() : String</i>	해당 JSP페이지가 속한 웹 애플리케이션의 컨텍스트 경로 리턴 컨텍스트 경로는 웹 애플리케이션의 루트 경로
<i>String getHeader(name)</i>	웹 서버로 요청 시 HTTP 요청 header 이름인 name에 해당하는 속성값 리턴
<i>Enumeration getHeaderNames()</i>	HTTP 요청 header에 있는 모든 헤더 이름 리턴

## 에러 디버깅

1. Parameter 0 of constructor in dev.dongyun.crud.post.PostController required a bean of type 'dev.dongyun.crud.post.PostService' that could not be found.Action: Consider defining a bean of type 'dev.dongyun.crud.post.PostService' in your configuration.

=> bean이 지정되지 않았단다 -> 이는 적절한 어노테이션을 붙여주지 않았다는 말이지  
실제로 PostService 인터페이스를 상속받는 구현체아잉 들어가보니 어노테이션 부분  
이 깨끗..^^

```
Parameter 0 of constructor in dev.dongyun.crud.post.PostController required a bean of type 'dev.dongyun.crud.post.PostService'
that could not be found.

The injection point has the following annotations:
- @org.springframework.beans.factory.annotation.Autowired(required=true)

Action:
|
Consider defining a bean of type 'dev.dongyun.crud.post.PostService' in your configuration.
```

```
import java.util.List;

public class PostServiceimple implements PostService {
    private static final Logger logger = LoggerFactory.getLo
```

- @Service 붙여줘서 빈으로 생성 ok

#### 1. PostService와 같은 인터페이스 새로 선언 시에

=> `private final PostService postservice;` 코드 추가

```
@RestController
@RequestMapping("post")
public class PostRestController {
    private static final Logger logger = LoggerFactory.getLogger(PostRestController.cl
ass);
    private final List<PostDto> postList;
    private final PostService postservice;
    public PostRestController(){
        this.postList = new ArrayList<>();
    }
}
```

- > 그럼 해당 줄에 빨간 줄 뜰텐데, 이때 아래와 가팅 autowired 써서 생성자 수정해주면 됨

```
public class PostRestController {
    private static final Logger logger = LoggerFactory.getLogger(PostRestController.cl
ass);
    private final PostService postservice;
    public PostRestController(
        @Autowired PostService postservice
    ){
        this.postservice = postservice;
    }
}
```

(+)

postman에서 postcontroller 따로, restcontroller 따로 테스트해보고 싶을 땐  
requestmapping 죽이고 살리는거 선택함 따라서

## 관계형 데이터베이스와 ERD

관계형 데이터베이스 (Codd의 12규칙을 따르고자 하는 데베)

- 테이블 (관계) 의 형태로 데이터 저장
- 관계형 연산자로 테이블 형태로 데이터 반환

Id	Name	Price	Count
1	Hamburger	6000	50
2	Chicken	16000	10
3	Pizza	20000	5

Primary Key

- pk 값 : 하나의 row를 가리키고 있는 값, 이 pk 값을 통해서 우리는 데이터의 행을 데려올 수 있는 것

**erd : Entity-Relationship Diagram**

=> 다대다, 일대일, 일대다 등의 관계 有

## MySQL과 Workbench 설치하기

0) Docker check & mysql 다운로드 받기

- windows기준으로 git bash에 docker help 쳤을 때 뭔가가 짜라라 나오면 잘 설치된 것
- 아래와 같이 입력해준다면 (name 과 password는 따로 지정해조야 함!)

```
docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d -p 3306:3306 mysql:8
```

```
MINGW64:/c/Users/DONGYUN
11585aaf4aad: Verifying Checksum
11585aaf4aad: Download complete
3b19465b002b: Pull complete
7b0d0cfe99a1: Pull complete
5b5dc265cb1d: Verifying Checksum
5b5dc265cb1d: Download complete
9ccd5a5c8987: Pull complete
fd400d64ffec: Verifying Checksum
fd400d64ffec: Download complete
2dab00d7d232: Pull complete
5d726bac08ea: Pull complete
11bb049c7b94: Pull complete
7fcdd679c458: Verifying Checksum
7fcdd679c458: Download complete
7fcdd679c458: Pull complete
11585aaf4aad: Pull complete
5b5dc265cb1d: Pull complete
fd400d64ffec: Pull complete
Digest: sha256:e3358f55ea2b0cd432685d7e3c79a33a85c7a359b35fa87fc49
Status: Downloaded newer image for mysql:8
641da5dbf45fdb6e521ea68d2c45a167dedd0f49803d634d996ab2686794881f
DONGYUN@DESKTOP-HN4KK92 MINGW64 ~ (master)
```

- 다운로드 잘된당

```
docker ps로 확인
```

```
DONGYUN@DESKTOP-HN4KK92 MINGW64 ~ (master)
$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS
641da5dbf45f   mysql:8    "docker-entrypoint.s..." About a minute ago Up About a minute   0.0.0.0:3306->3306/tcp, 33060/tcp
some-mysql
```

- 방금 만든 계정 삭제



```
$ docker rm -f some-mysql  
some-mysql
```

```
docker rm -f some-mysql
```

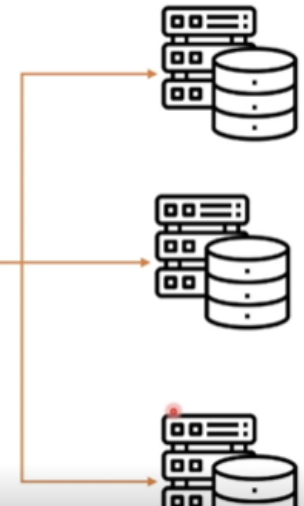
## 1) Mysql & WorkBench 설치



관계형 데이터베이스 서버  
서버 소프트웨어 + 물리 서버




서버에 접속하기 위한 클라이언트



- workbench는 미리 깔아둔게 있었기 때문에 그것을 사용

=> 설치 후

MySQL Connections  

# MySQL에 스키마 / 유저 생성

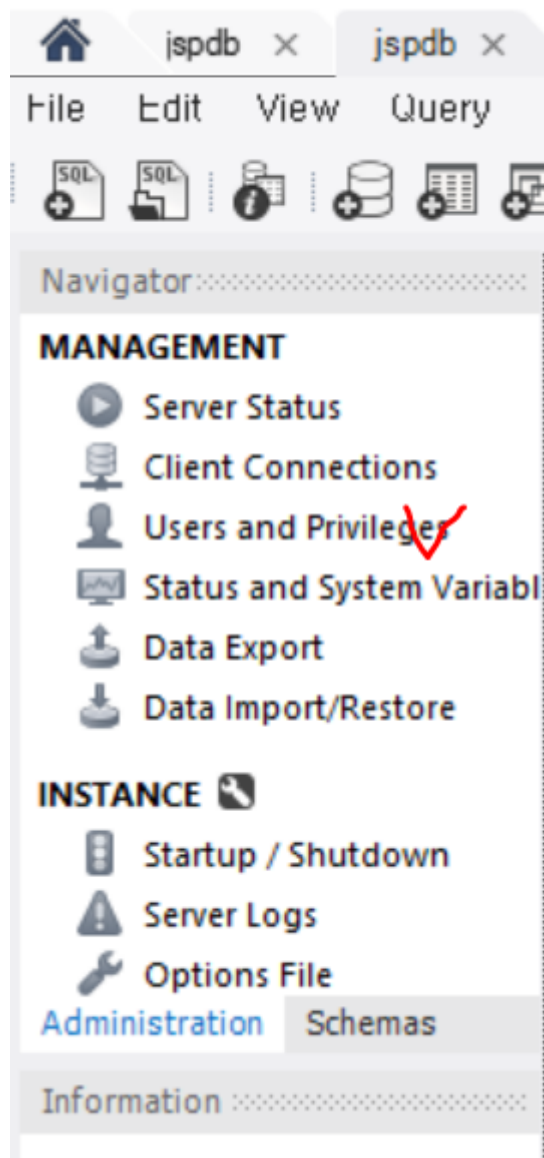
```
C:\Users\DONGYUN>cd ..
C:\Users>cd ..
C:\>cd "Program Files"
C:\Program Files>cd MySQL
C:\Program Files\MySQL>cd "MySQL Server 8.0"
C:\Program Files\MySQL\MySQL Server 8.0>cd bin
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql
ERROR 1045 (28000): Access denied for user 'ODBC'@'172.17.0.1' (using password: NO)
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u root -p
Enter password: 
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.0.28 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```



1. create scheme

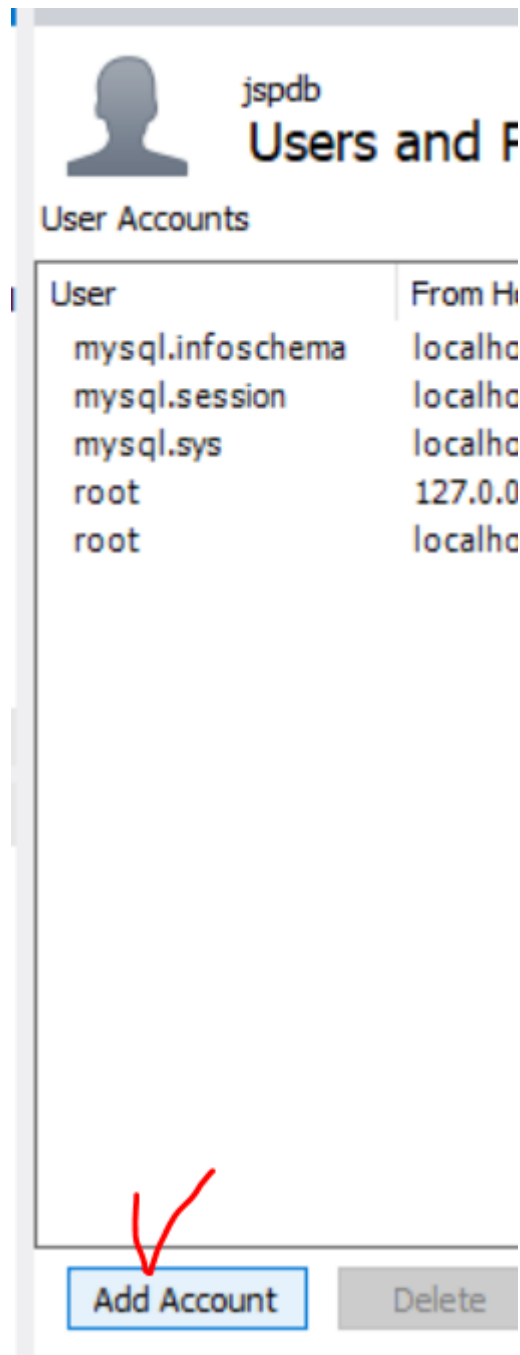
### Review the SQL Script to be Applied on the Database

Online DDL

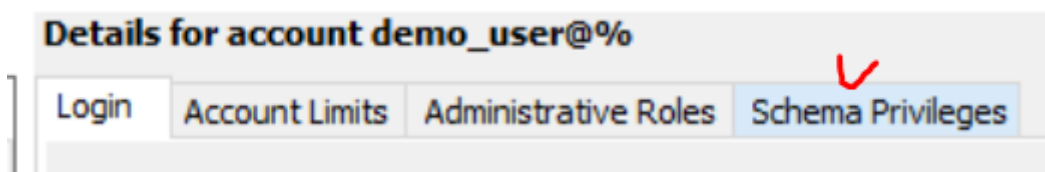
Algorithm:  Lock Type:

```
1 CREATE SCHEMA `demo_scheme` ;
2
```

## 2. user privilege & 유저 만들기



## 3. 권한 설정



**Details for account demo\_user@%**

Login Account Limits Administrative Roles Schema Privileges


CREATE, CREATE ROUTINE, CREATE TEMPORARY TABLES, CREATE VIEW, DELETE, DROP, EVENT, EXECUTE, GRANT

<

Schema and Host fields may use % and \_ wildcards.  
The server will match specific entries before wildcarded ones.

Revoke All Privileges Delete Entry

4. 해당 유저로 새로운 로컬호스트라는 db를 만들기

 Setup New Connection

Connection Name: localhost

Connection Method: Standard (TCP/IP)

Parameters SSL Advanced

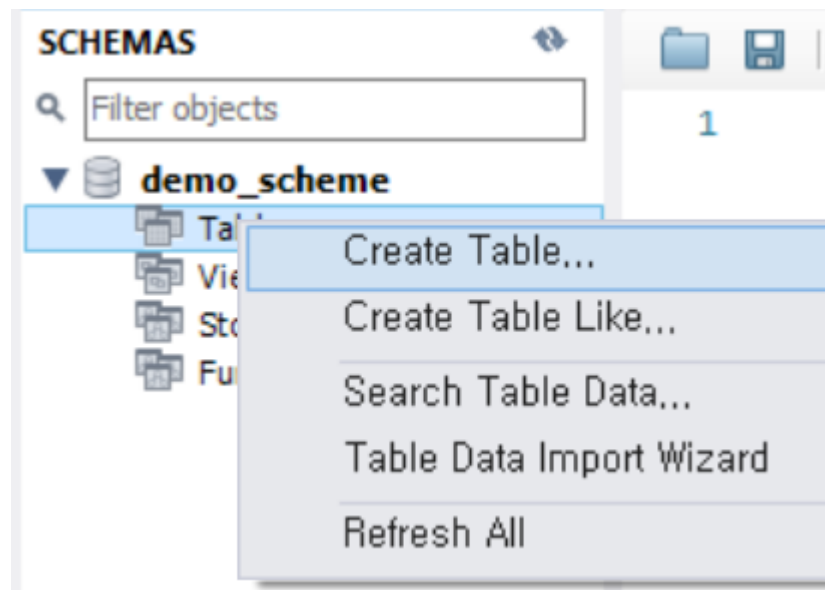
Hostname: 127.0.0.1 Port: 3306

Username: demo\_user


Password: Store in Vault ... Clear


Default Schema: demo\_scheme

5. 한번 여기서 테이블 만들어보기





Query 1    new\_table - Table    x


 Table Name:     Schema: **demo\_sc**  
 Charset/Collation:      Engine:   
 Comments:

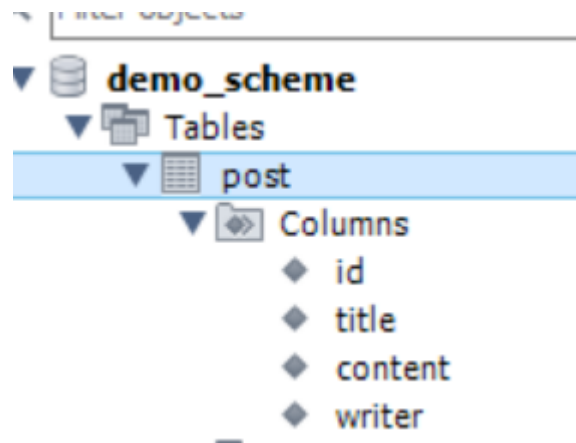
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
 idnew_table	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

---


 Table Name:     Schema: **demo\_s**  
 Charset/Collation:      Engine:   
 Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
 id	<input type="text" value="INT"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

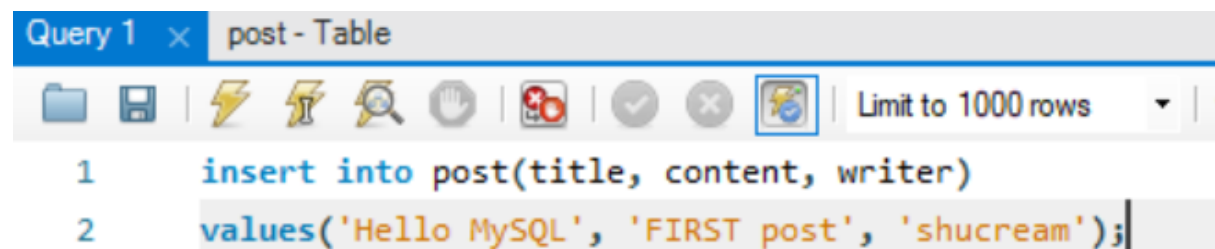
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
title	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
content	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
writer	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	



- 이렇게 설정한 테이블대로 잘 설정됨

## 기본적인 SQL 작성법

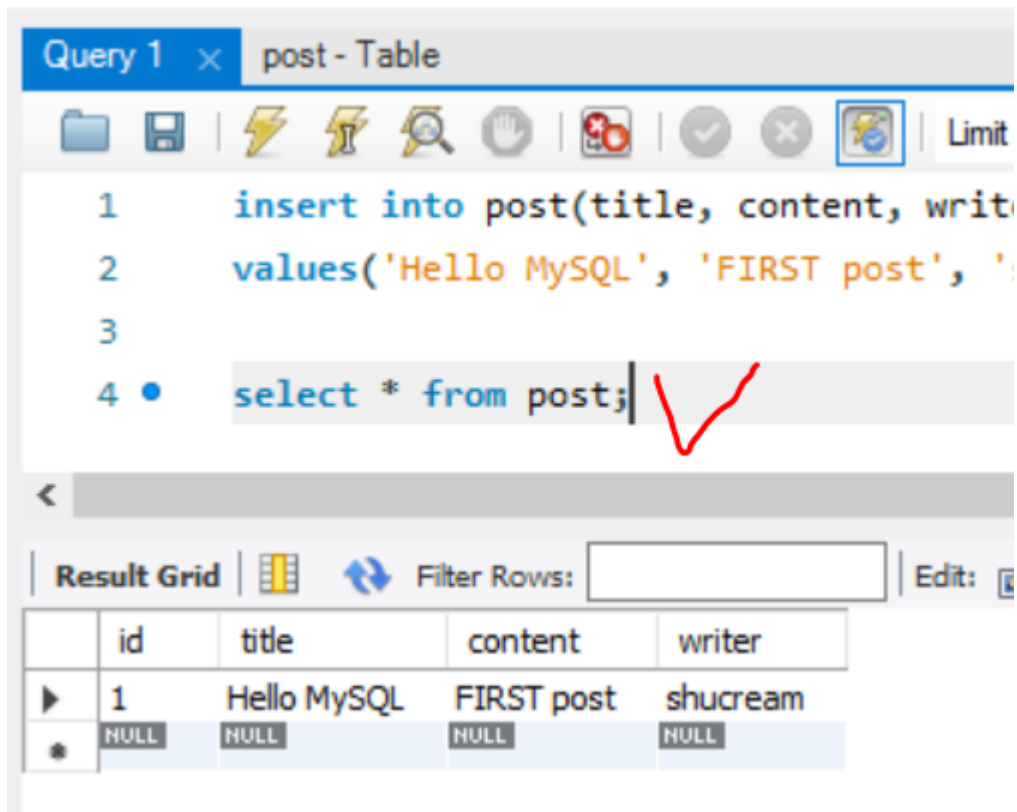
### 1) INSERT



- 위와 같이 INSERT 명령어를 통해서 데이터 집어넣기

Action Output			
#	Time	Action	Message
1	17:17:20	Apply changes to post	Changes applied
2	17:19:35	insert into post(title, content, writer) values('Hello MySQL', 'FIRST post', 'shucream')	1 row(s) affected ✓

### 2) SELECT



특정한 조건 걸어서 select 도 가능 (writer이 누구인지, writer의 이름이  
월로 시작하는지 등)

- 등호 사용



Query 1 x post - Table

Limit to 1000 rows

```

6 • insert into post(title, content, writer)
7   values('Hello MySQL', 'second post', 'shucream');
8
9 • select * from post where writer='shucream';
10

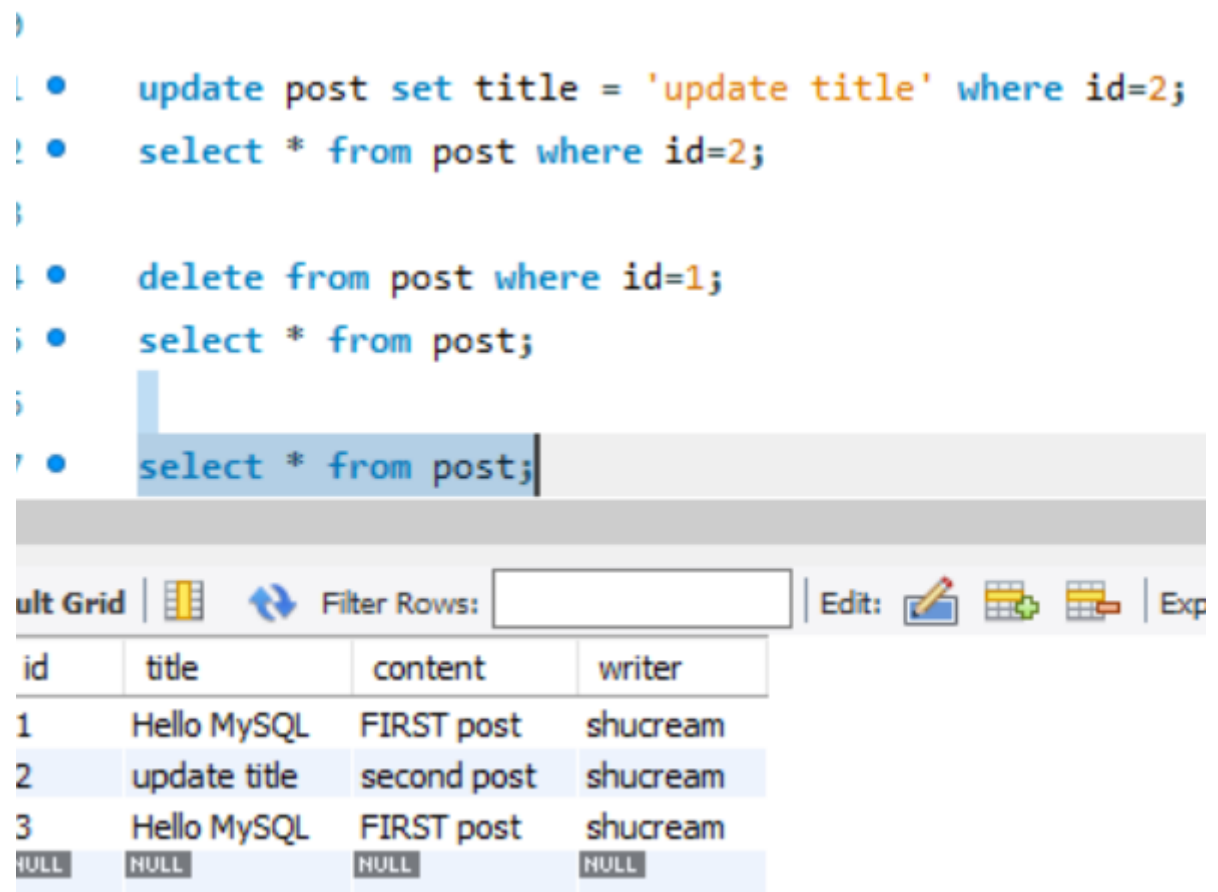
```

Result Grid

	id	title	content	writer
	1	Hello MySQL	FIRST post	shucream
	2	Hello MySQL	second post	shucream
+	NULL	NULL	NULL	NULL

- like 사용

### 3) update & delete



(+) innerjoin, outerjoin

(+) altertable

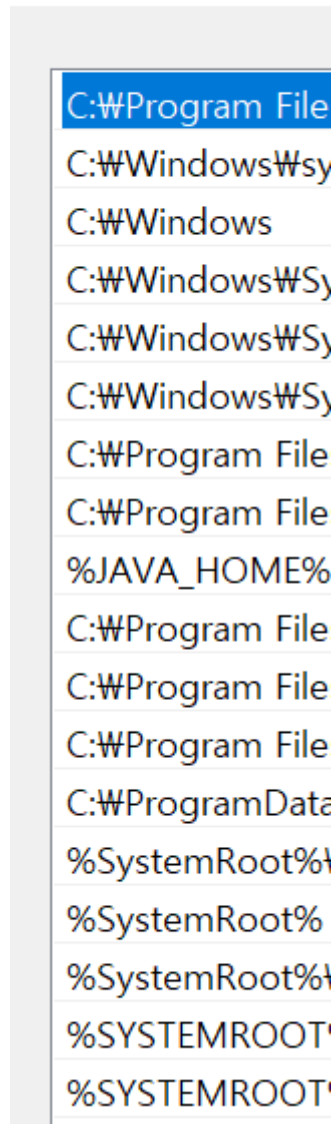
#### 4) truncate

- 아예 초기화
- delete는 id를 변화시키지 않음 id 한번 4인 애는 앞에 1,2,3 없어져도 4유지
- truncate 는 아예 id 값도 초기화

(+) MYSQL 환경변수 설치함

<https://crispypotato.tistory.com/44>

## 환경 변수 편집



내 환경변수에는 MYSQL 내용 존재하지 않았음

- mysql bin이 있는 경로 입력해줘야 함 (나 같은 경우는 아래와 같았음)

`C:\Program Files\MySQL\MySQL Server 8.0\bin` & 이를 확인 눌러서 저장

```
C:\Program Files\MySQL\MySQL Server 8.0\
C:\Program Files\MySQL\MySQL Server 8.0\bin
C:\Program Files\MySQL\MySQL Server 8.0\docs
C:\Program Files\MySQL\MySQL Server 8.0\etc
```

C:\Program Files\MySQL\MySQL Server 8.0\bin	
	확인
	취소