



It's Your Life

with





# 서블릿의 MVC 패턴



Model



Controller



View





Client

Get 방식의 Request

/login?username=tetz

Tomcat  
Server

LoginServlet

@WebServlet(/login)

```
@Override ① Tetz +1
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String username = request.getParameter(s: "username");
    String password = request.getParameter(s: "password");
}
```

<http://localhost:8080/login?username=tetz>

로그인 성공!

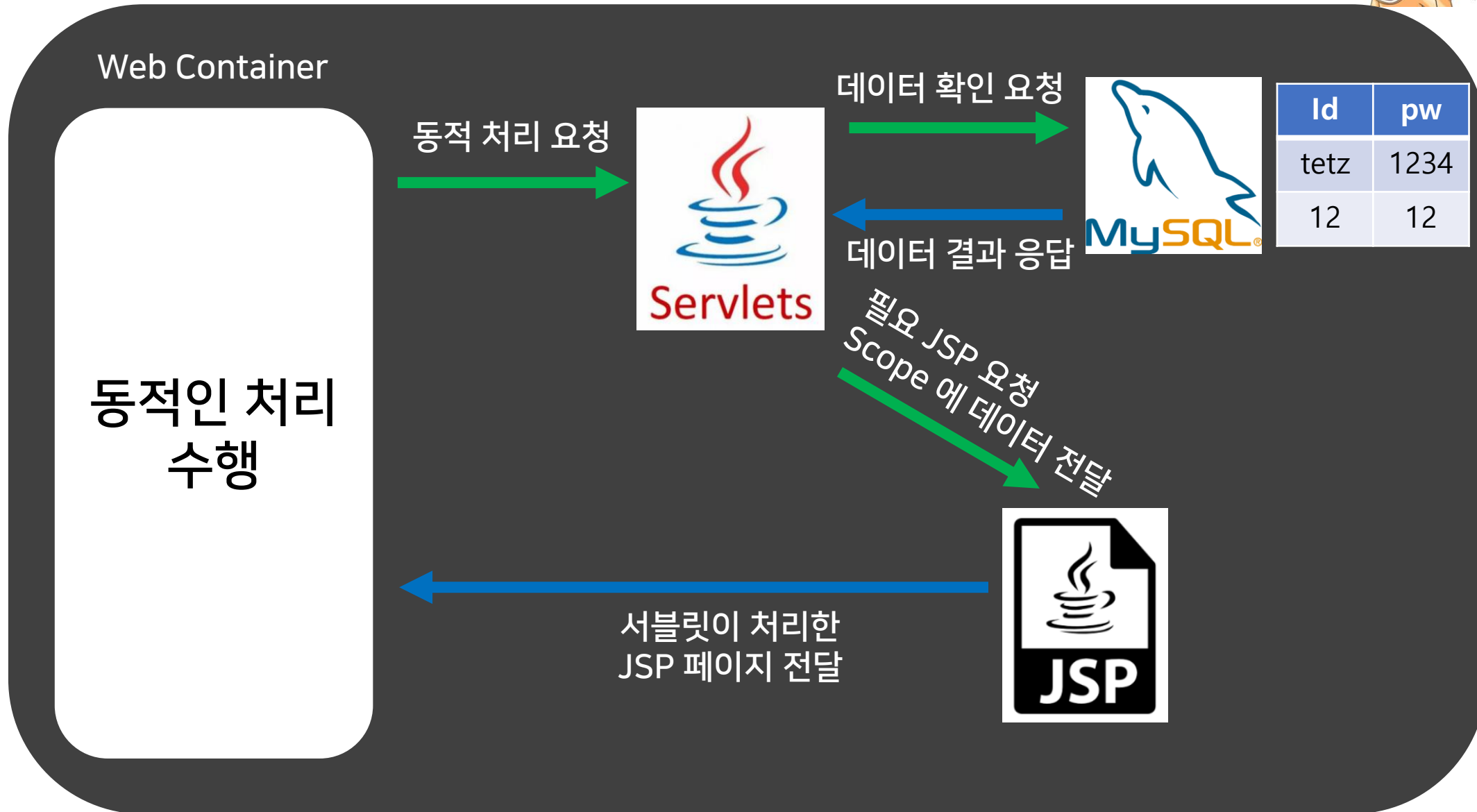
서버에서 보내는  
Response

```
<html>
<h1>로그인 성공!</h1>
</html>
```

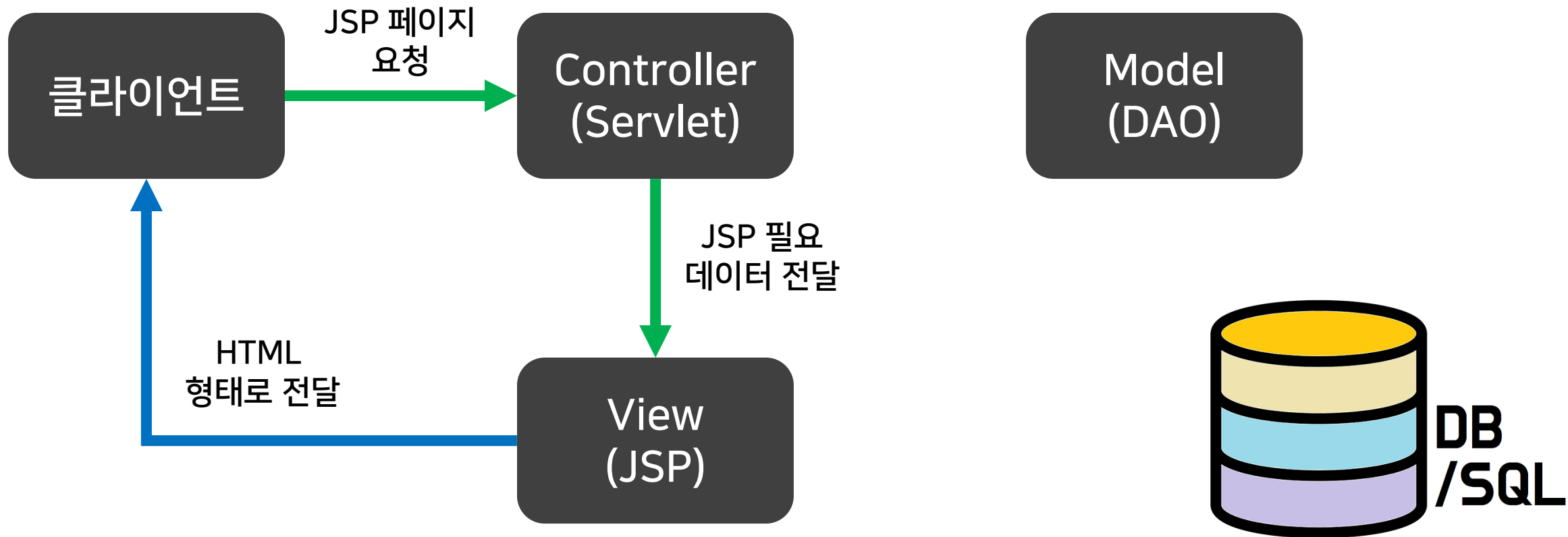
Web Browser



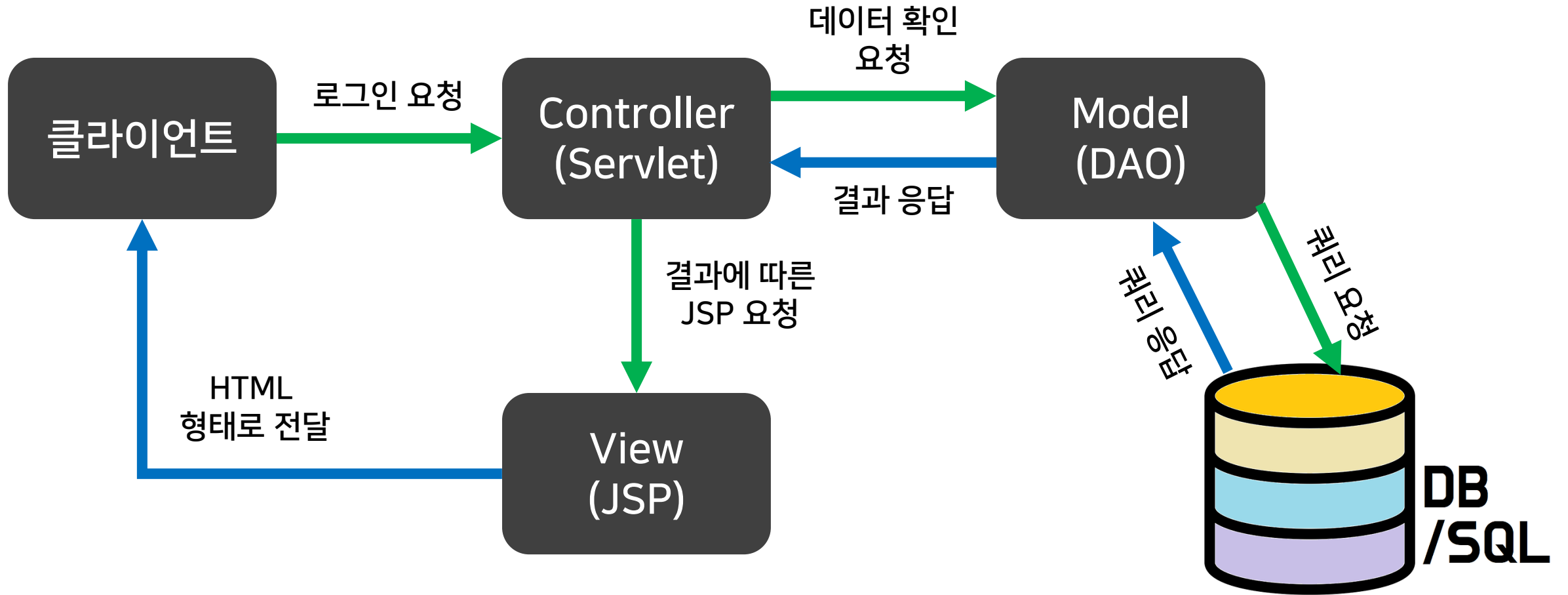
# Tomcat Server



## 일반 JSP 페이지만 요청한 경우!



## 데이터 관련 요청이 같이 들어간 경우





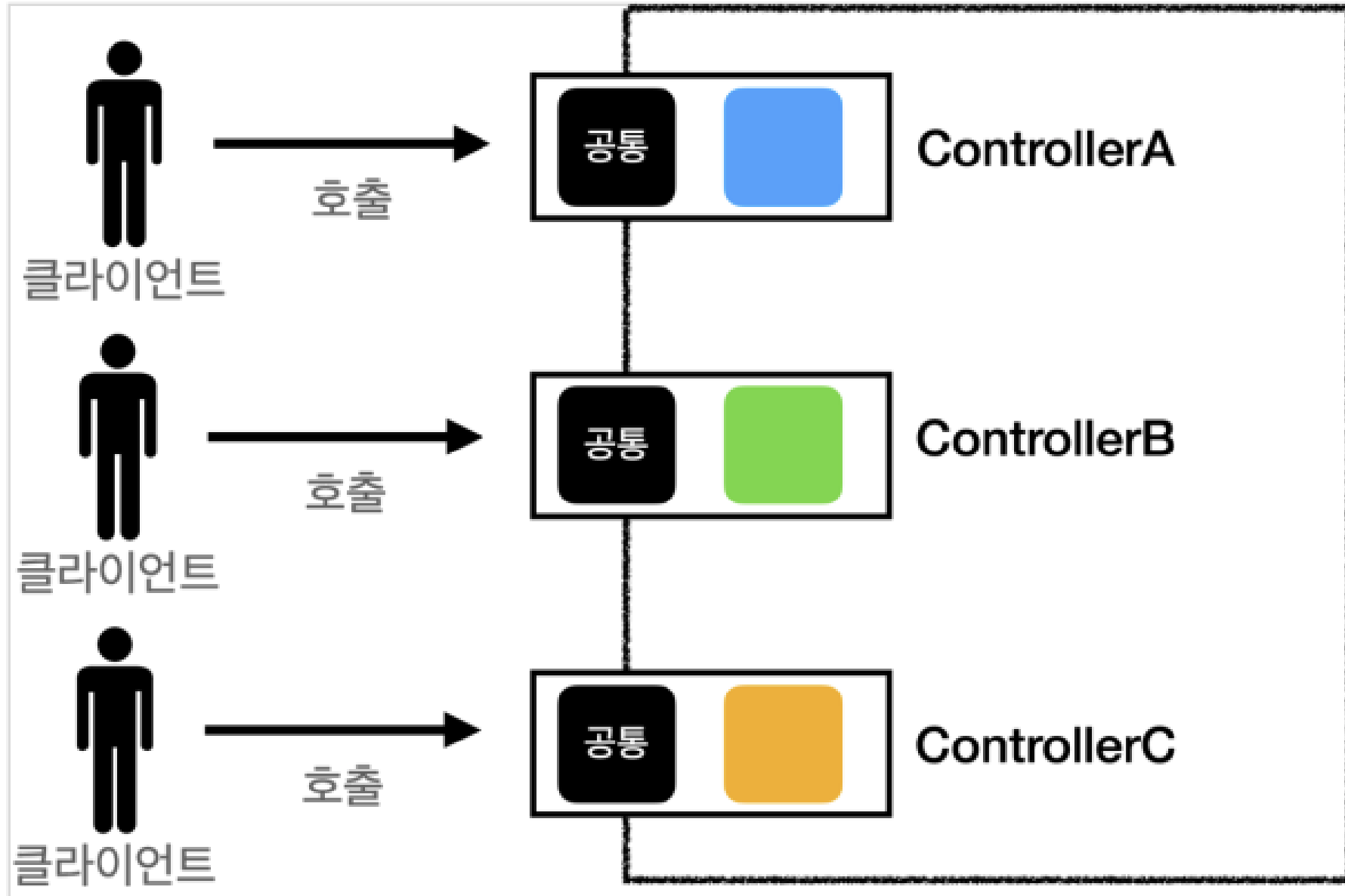
# FrontController

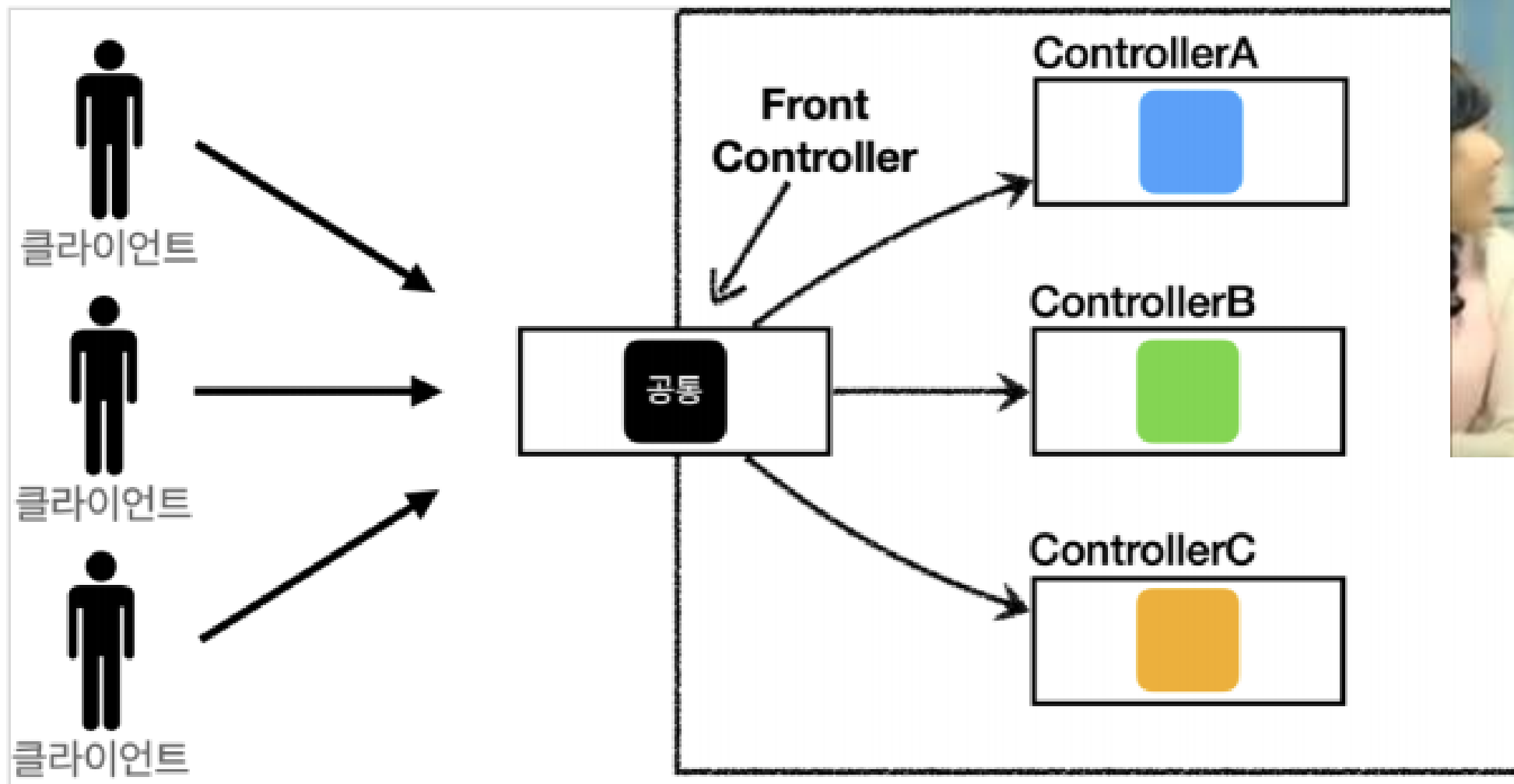


SNL  
크루소

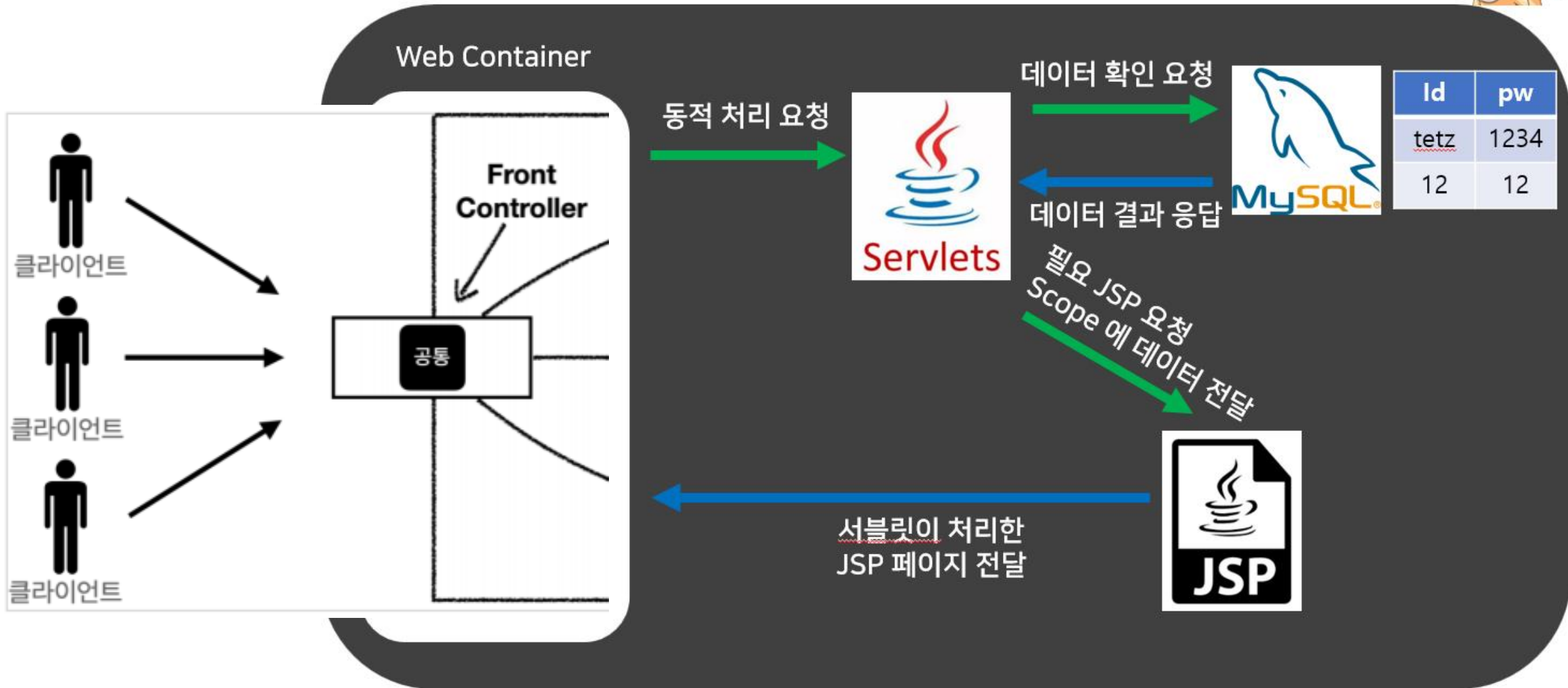
tvN







# Tomcat Server

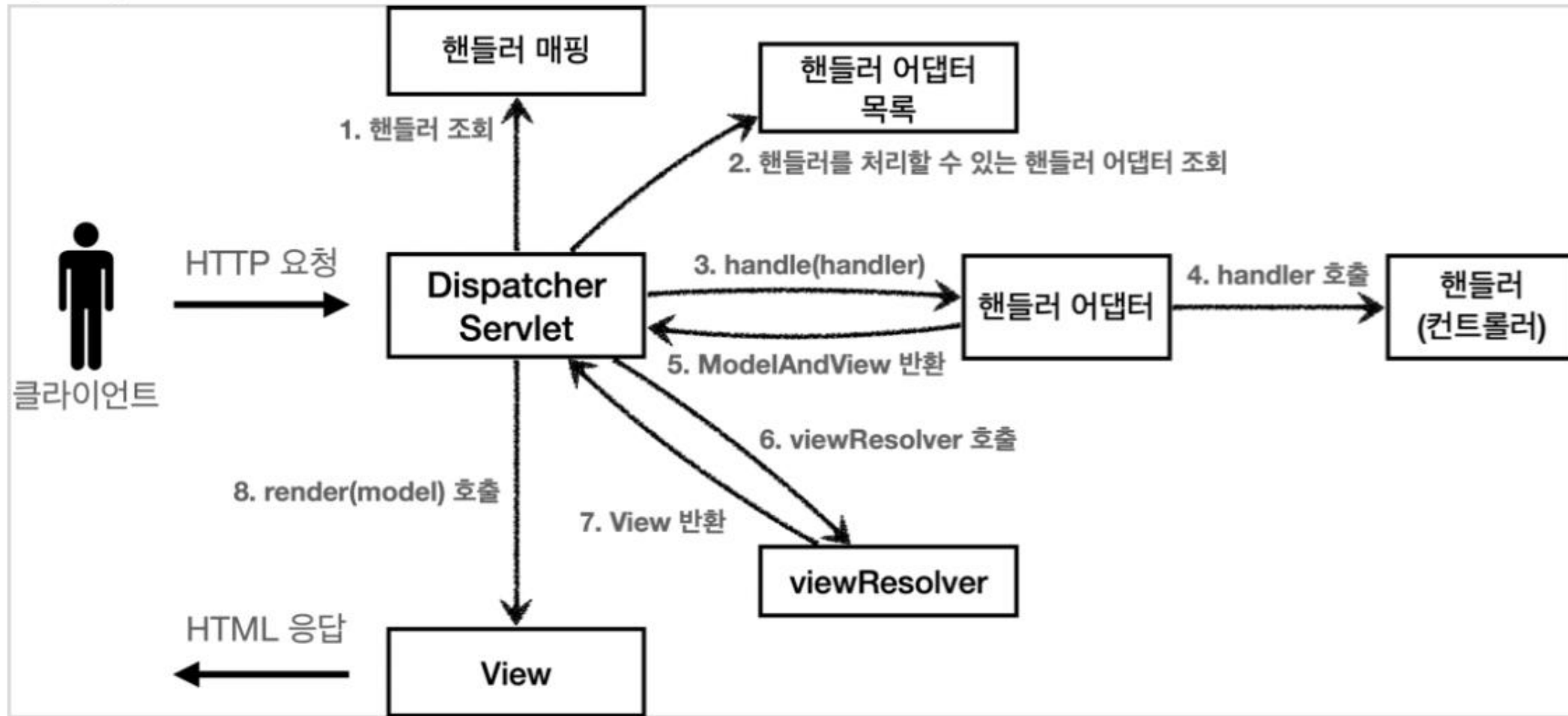




# FrontController

(= Dispatcher Servlet)

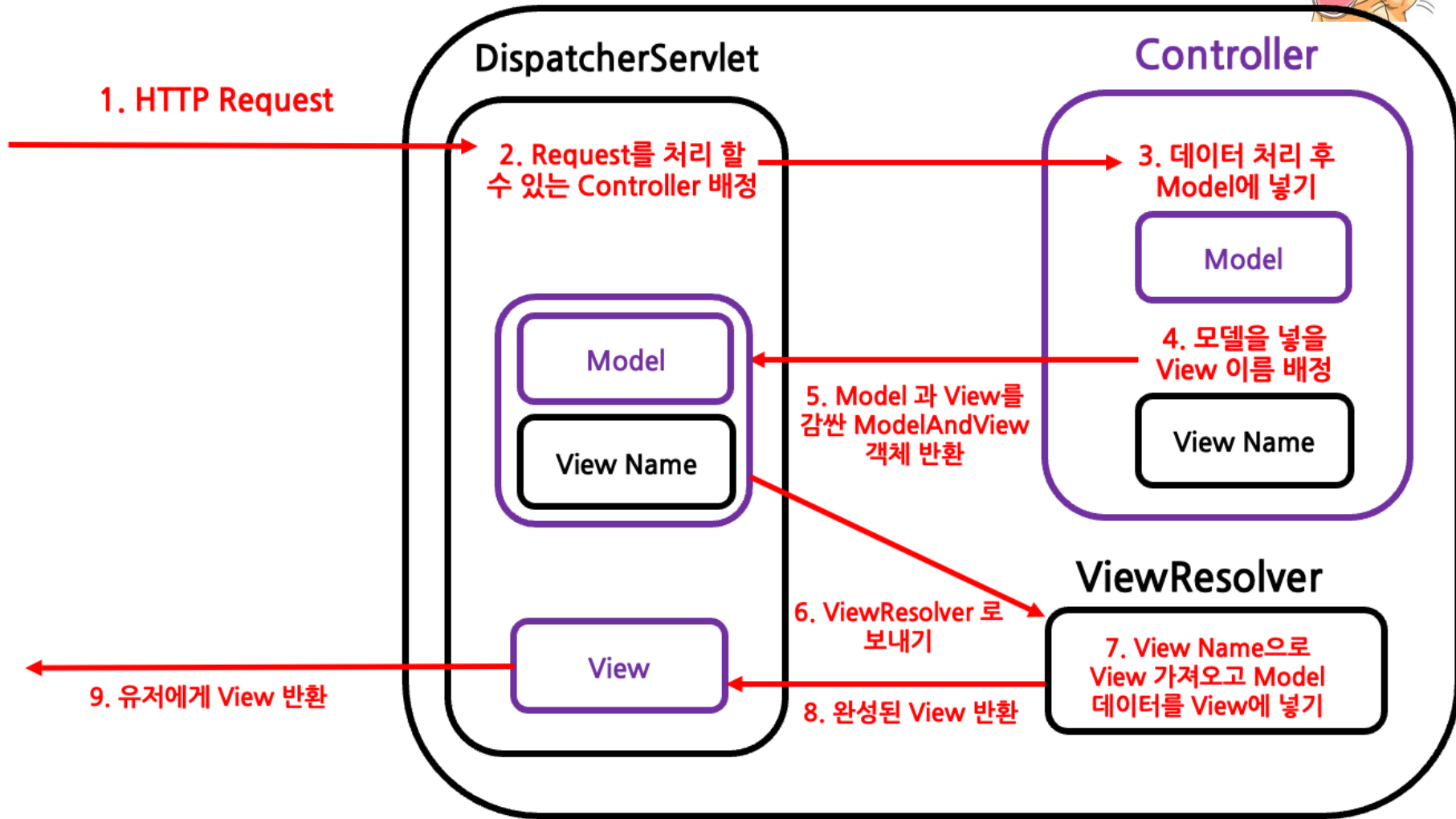
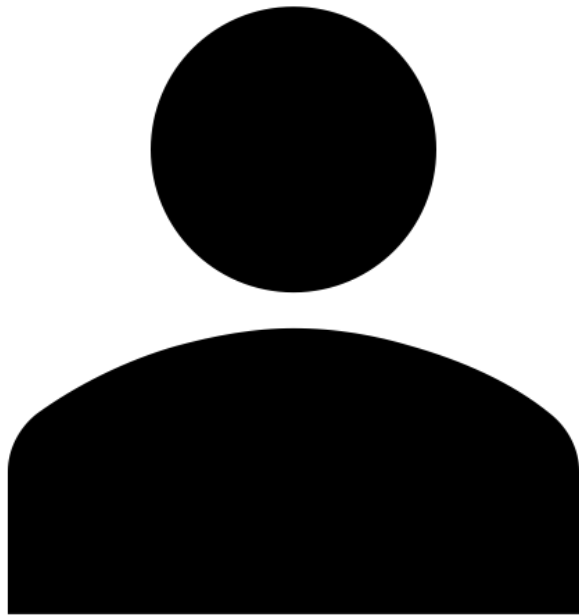
# SpringMVC 구조



# Server



유저





# Controller 와

# 의존성 주입





다형성.....

혹시 기억 하나요!?

다형성을 구성하는  
2가지 핵심 요소가 뭐였죠?



# 다형성의 핵심

## 2가지!



# 1. 다형적 참조

부모는 자식을 담을 수 있다!



Animal

Donkey

Dog

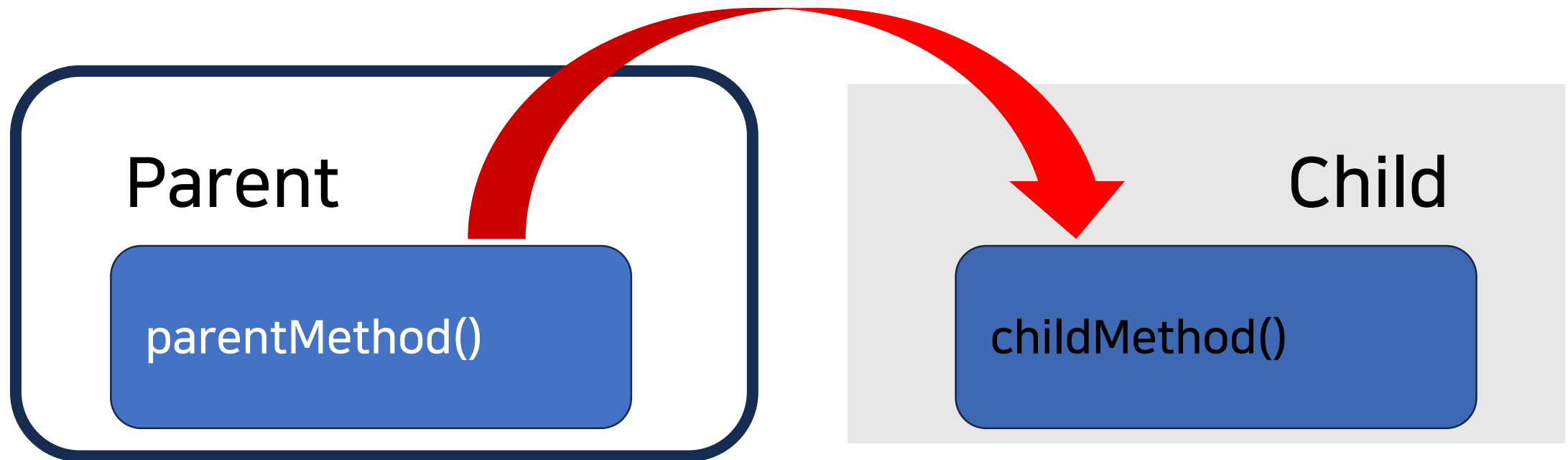
Cat

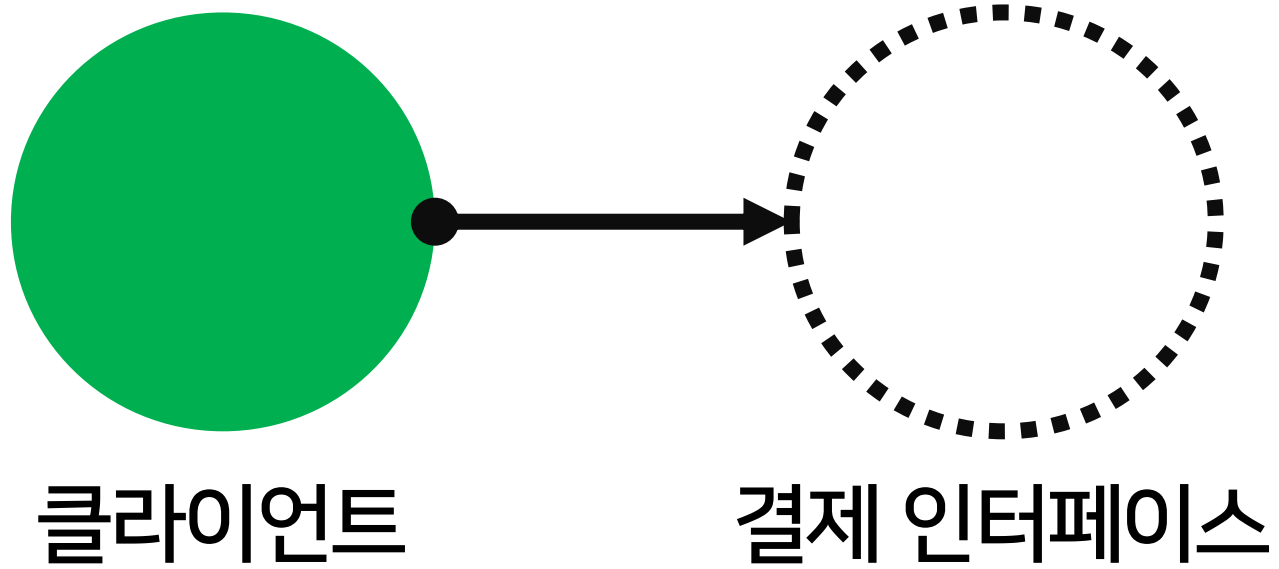
Chicken



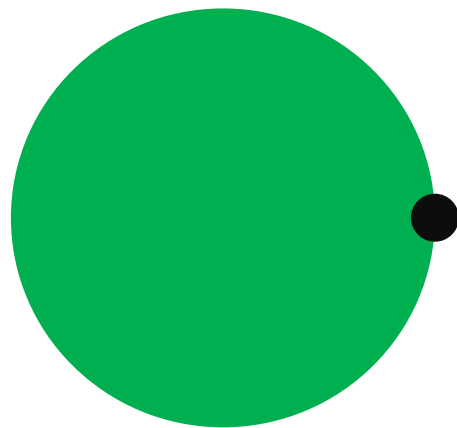
## 2. 메서드 오버라이딩

오버라이드 된 메서드가 우선권을 가진다

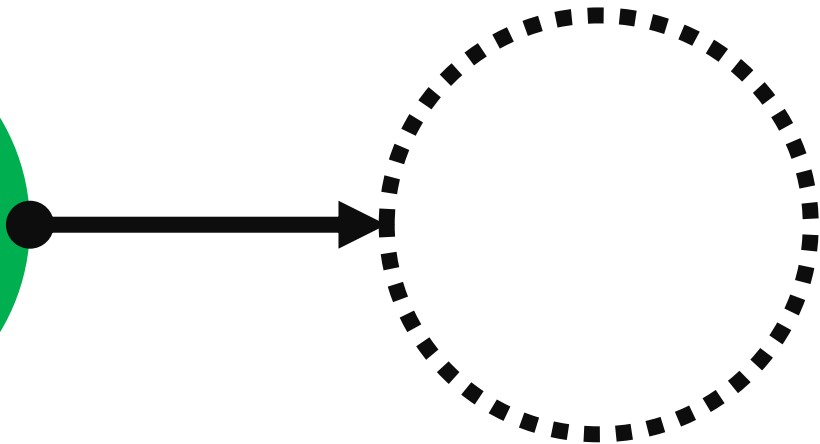




결제하기()  
수수료정산()  
결과화면표시()



클라이언트



결제 인터페이스



카드 결제  
인스턴스

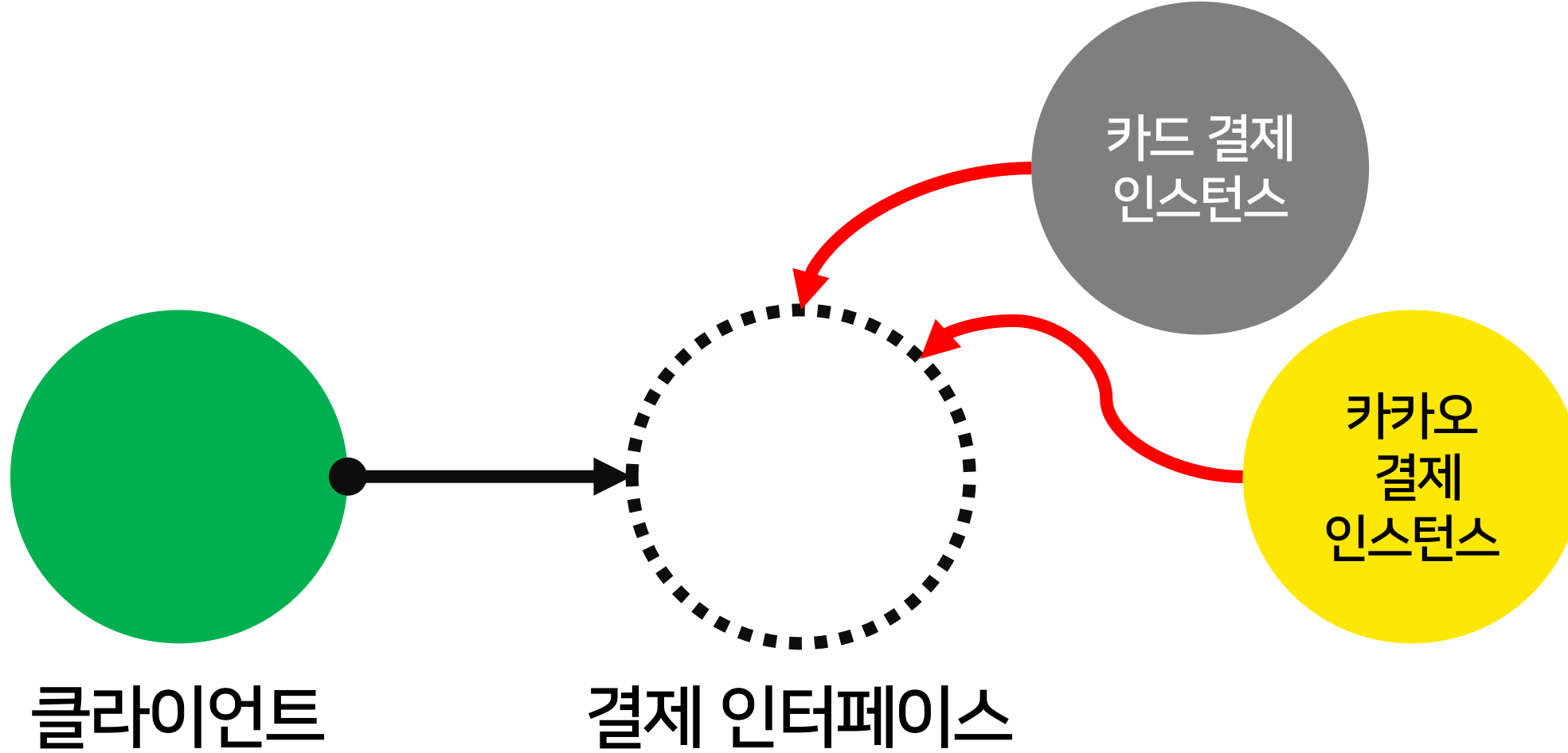


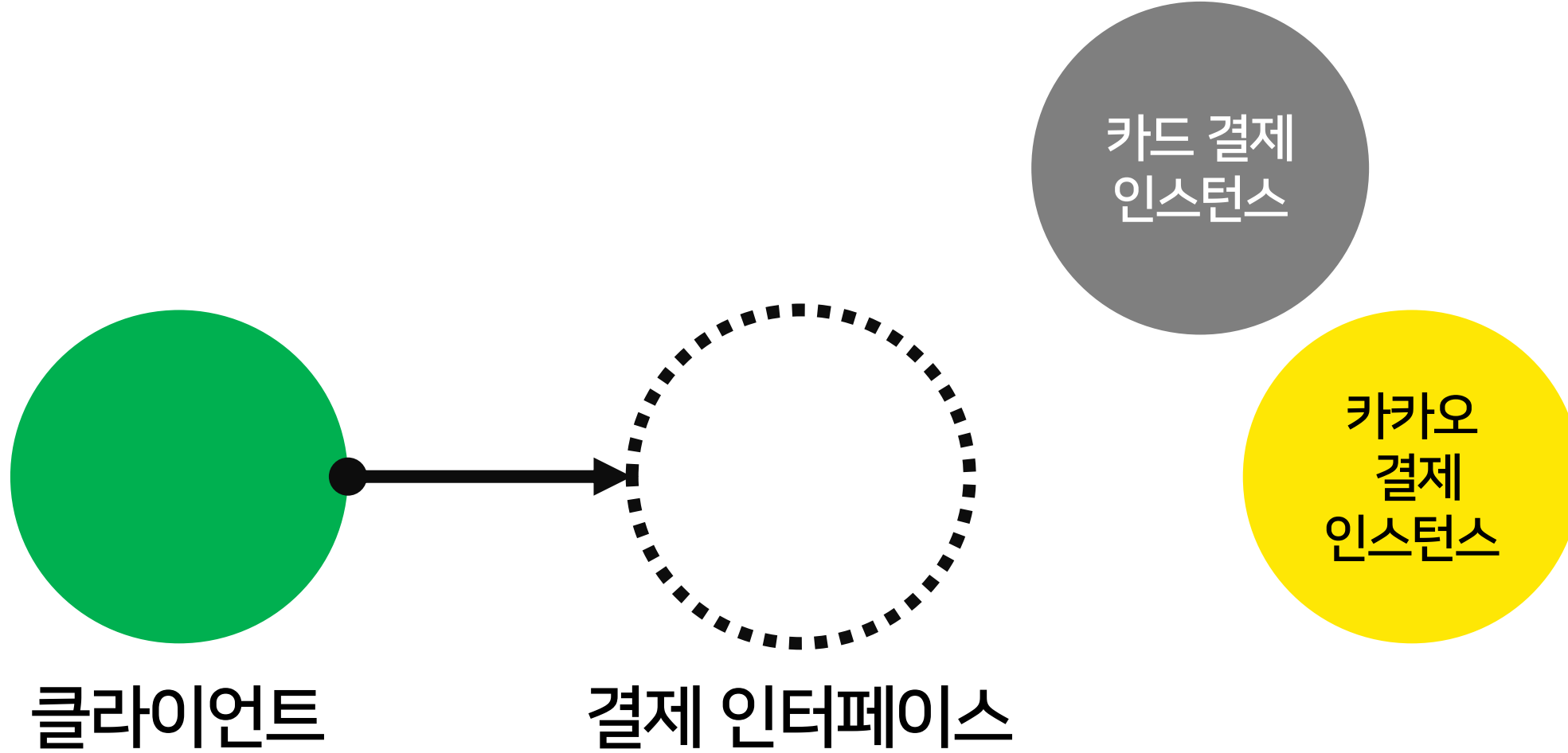
카카오  
결제  
인스턴스

결제하기()  
수수료정산()  
결과화면표시()



결제하기()  
수수료정산()  
결과화면표시()









나는 구현체다

특정 인터페이스를 상속 받은 친구는 수행

능히 할 수 있다

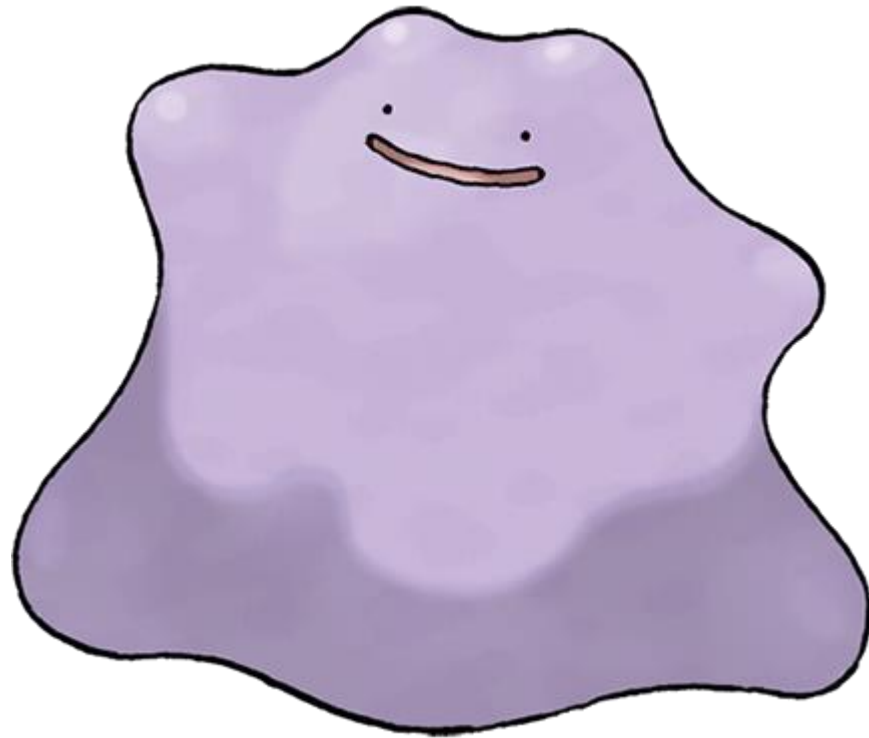


열어라!

무언가를 여는  
구현체



```
public interface Open {  
    void open();  
}
```



```
public class DoorOpen implements Door {
    @Override public void open() {
        System.out.println("문을 연다");
    }
}
```







열어라!





```
public class CarDoorOpen implements Open { no usages new *  
    @Override no usages new *  
    public void open() { System.out.println("차 문을 엽니다!"); }  
}
```







열어라!









열어라!

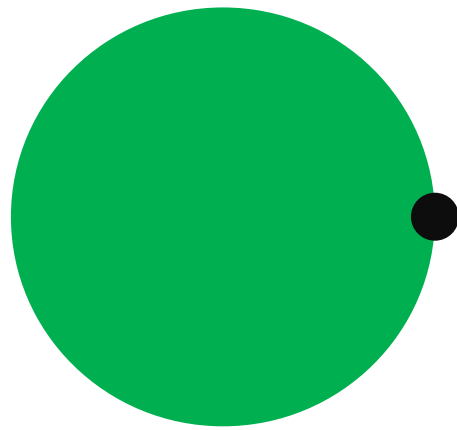
차 문을 여는  
서비스



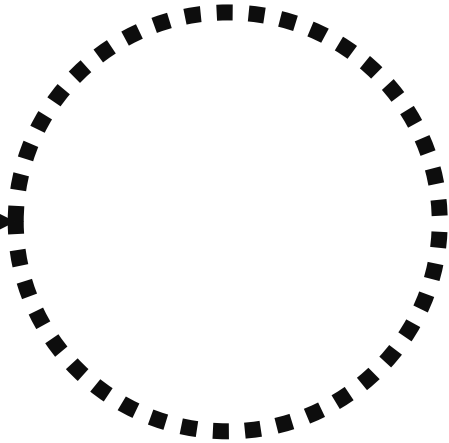


이걸 스프링에

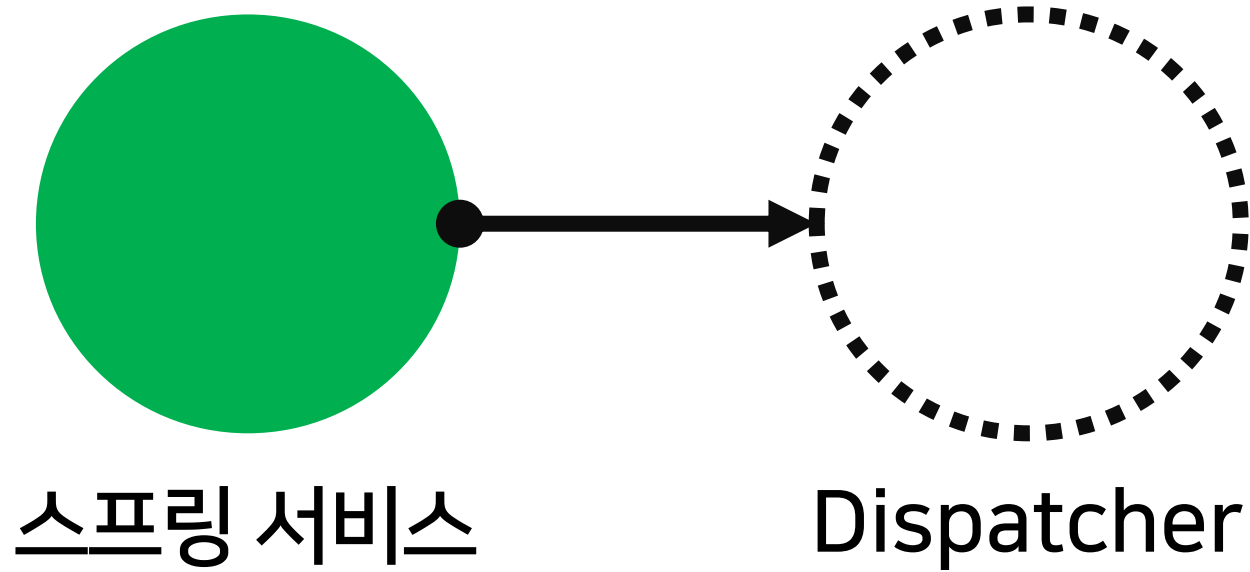
적용하면!?



스프링 서비스



Dispatcher

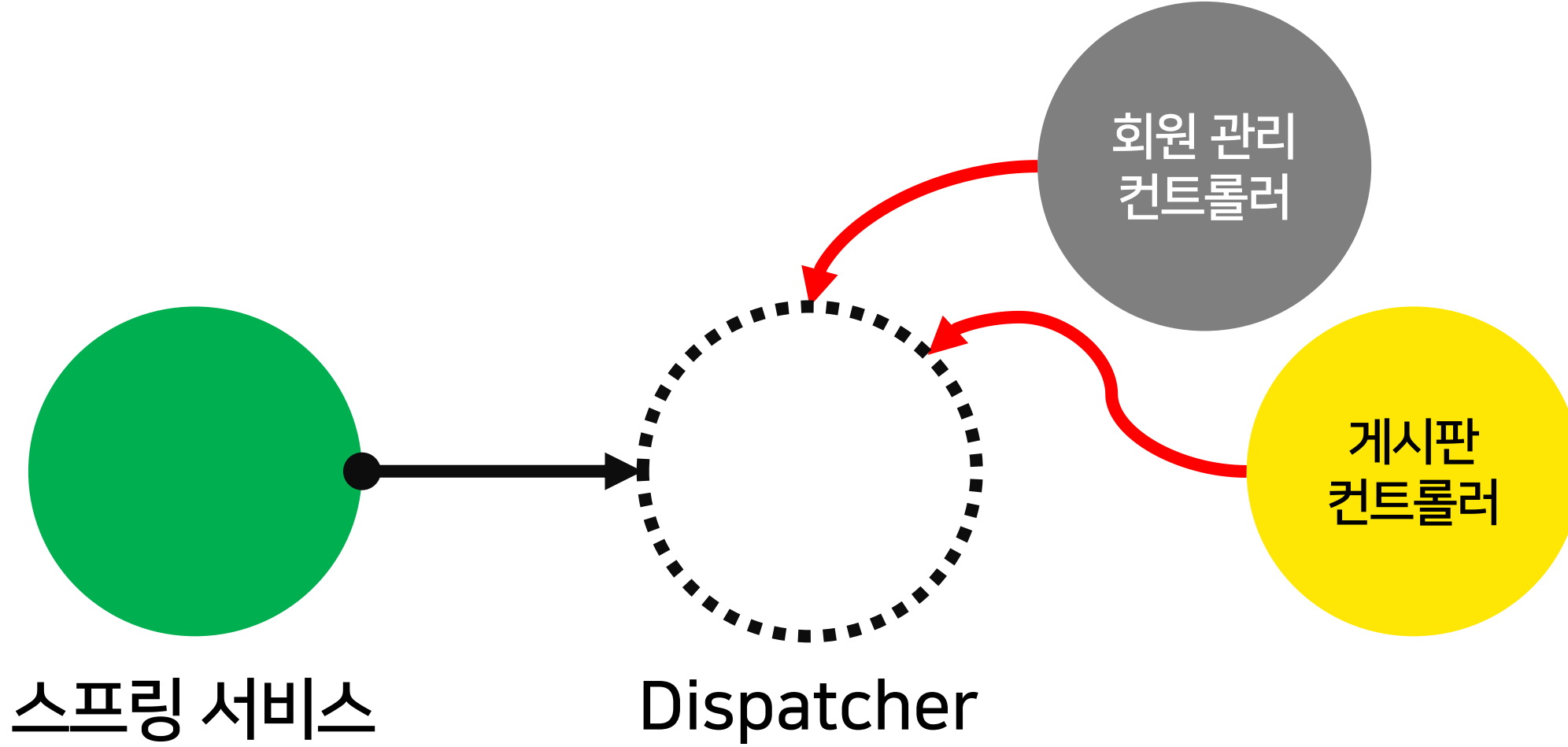


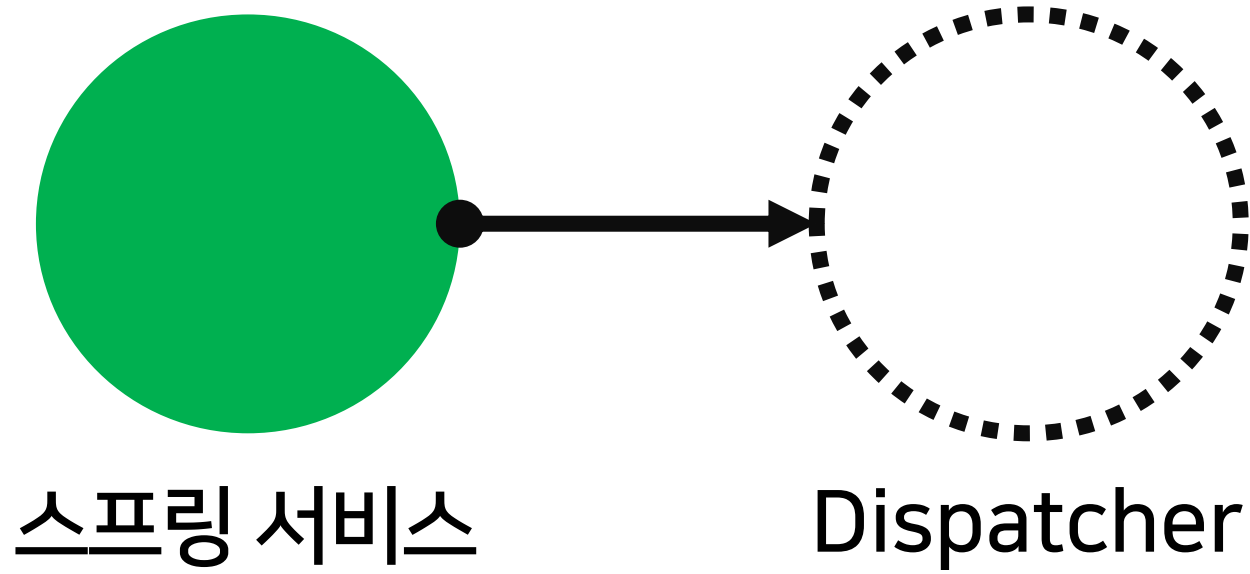
회원 관리  
컨트롤러

가입하기()  
정보수정()  
탈퇴하기()

게시판  
컨트롤러

글 쓰기()  
글 수정()  
글 삭제()









자, 여기까지 설명을 들어보니  
어떤 생각이 드시나요!?

그런데 말입니다





와우! 스프링이 귀찮은건 다 해주는 구나!!  
우린 이제 컨트롤러만 잘 만들면 되겠군!





와나...

난 이제 컨트롤러 생성 기계가 되겠구나...





결국 여러분들이 한동안 하실 일의 대부분은  
DB 에 맞는 DTO 객체 만들고  
해당 객체를 컨트롤하는 컨트롤러를 만드는 일입니다!  
;)





# 스프링에

# DTO 를 적용!



편안한 메뉴 이동 및 테스트를 위해서  
header 메뉴를 작업!

webapp  
WEB-INF  
views

JSP header.jsp

JSP index.jsp



```
<header>
```

```
<a href="/">HOME</a>
```

```
<a href="/member/form">REGISTER</a>
```

```
<a href="/member/form2">REGISTER2</a>
```

```
<a href="/member/show">SHOW</a>
```

```
<a href="/member/show2">SHOW2</a>
```

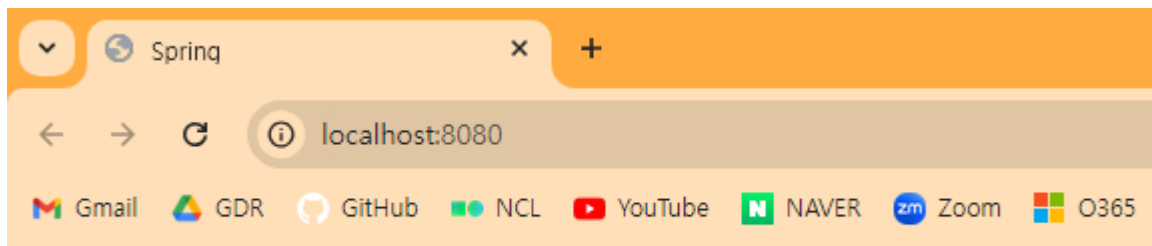
```
</header>
```

이제 member 는 회원 관리  
CONTEXT 가 되며, 뒤의 주소가  
실제적 서비스 요청 주소가 됩니다!



```
<%@ page contentType="text/html; charset=UTF-8"%>
<html>
<head>
    <meta charset="UTF-8">
    <title>Spring</title>
</head>
<body>
    <%@ include file="header.jsp"%>
    <h1>Hello, Spring World!</h1>
</body>
</html>
```

index.jsp 에 HEADER 추가!



[HOME](#) [REGISTER](#) [REGISTER2](#) [SHOW](#) [SHOW2](#)

# Hello, Spring World!





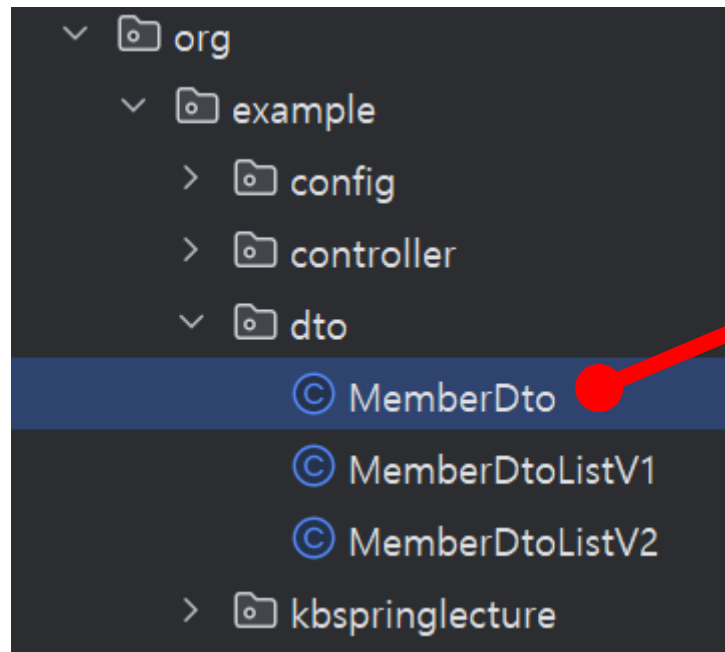


# 회원 관리 DTO

## 만들기



회원 관리를 위한  
DTO 클래스를 생성



```
public class MemberDto { 6 usages kdtTetz
    private String id; 3 usages
    private String name; 3 usages
```

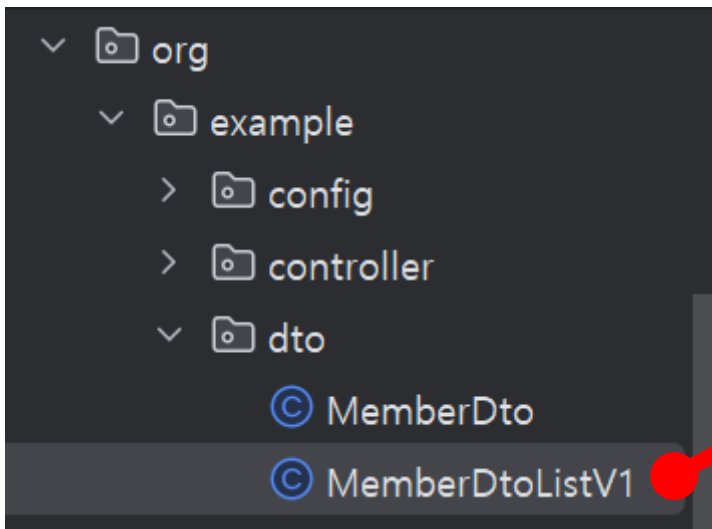
간단한 작업을 할 것이므로  
id 와 name 만 사용!

```
public MemberDto(String id, String name) { 2 usages
    this.id = id;
    this.name = name;
}
```

생성자, Getter / Setter 작업

```
public String getId() { return id; }
```

```
public void setId(String id) { this.id = id; }
```



실제 회원 정보를 담고 있는  
List 클래스를 생성!

여기서 싱글톤을 적용!!





그란데 말입니다...

싱글톤 패턴이 뭐죠!?

오랜만에 쿠폰 겁니다!! 😊

그란데 말입니다

MemberDtoList  
인스턴스 1



MemberDtoList  
인스턴스 2



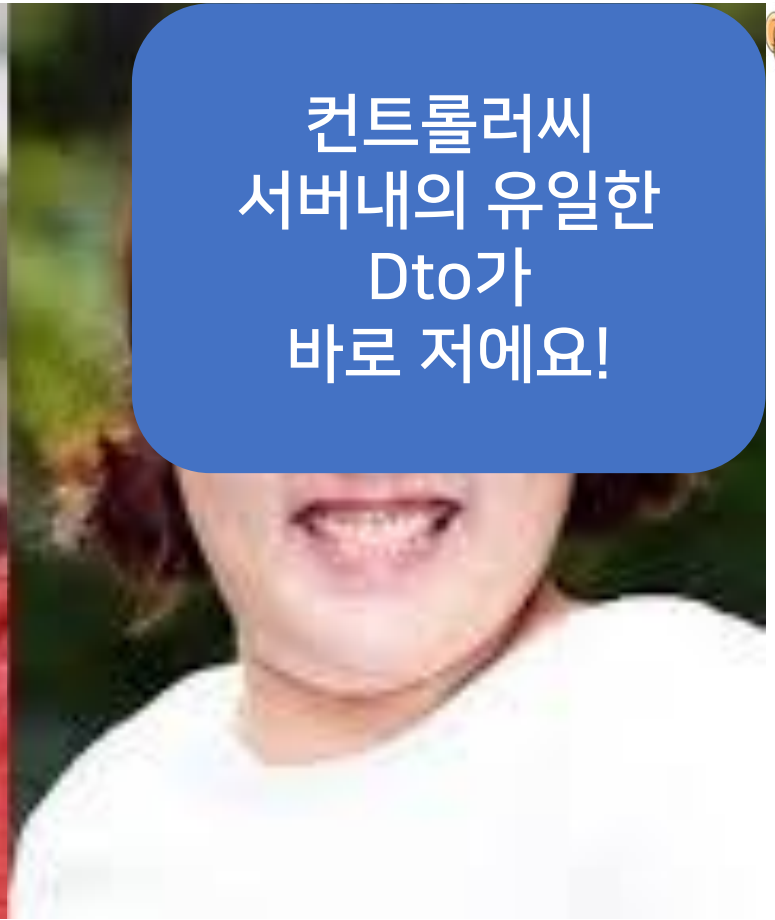
MemberDtoList  
인스턴스 3





Controller





컨트롤러씨  
서버내의 유일한  
Dto가  
바로 저예요!





특정 DTO 클래스의 인스턴스는  
하나만 만들고 이걸 공유하면 되겠구나!!

그리고 하나만 만들 수 있도록 클래스를  
구현하면 되겠구나!!

→ 싱글톤 패턴!





# 회원 리스트에 싱글톤 적용하기

```
public class MemberDtoListV1 { 16 usages kdtTetz
    private static MemberDtoListV1 instance; 2 us
    private List<MemberDto> memberDtoList; 3 usag

    private MemberDtoListV1() { 1 usage kdtTetz
        this.memberDtoList = new ArrayList<>(); // List 초기화
        // 테스트 데이터 추가
        this.addList(id: "tetz", name: "이효석");
        this.addList(id: "siwan", name: "김시완");
    }
```

단 하나만 존재하는 인스턴스 주소를  
가지게 될 instance 멤버

→ 어디서든 공유가 가능해야하므로  
static 으로 선언

```
public class MemberDtoListV1 { 16 usages  👤 kdtTetz
    private static MemberDtoListV1 instance; 3 usages
    private List<MemberDto> memberDtoList; 5 usages
```

실제 회원 정보가 저장 될,  
리스트 멤버

```
private MemberDtoListV1() { 1 usage  👤 kdtTetz
    this.memberDtoList = new ArrayList<>(); // List 초기화
    // 테스트 데이터 추가
    this.addList(id: "tetz", name: "이호석");
    this.addList(id: "siwan", name: "김시완");
}
```

클래스 생성 시, 배열을 생성하여  
멤버 변수에 저장

+ 테스트 데이터 추가!



// 싱글톤 인스턴스 반환 메소드

```
public static synchronized MemberDtoListV1 getInstance() {  
    if (instance == null) {  
        instance = new MemberDtoListV1();  
    }  
    return instance;  
}
```

```
public void addList(String id, String name) {  
    memberDtoList.add(new MemberDto(id, name));  
}
```

```
public List<MemberDto> getList() {  
    return memberDtoList;  
}
```

이제 회원 리스트가 필요하면  
인스턴스를 만들어서 쓰는 것이 아니라  
이미 만들어진 인스턴스의 주소를  
받아서 공용 인스턴스를 씁니다!



// 싱글톤 인스턴스 반환 메소드

```
public static synchronized MemberDtoListV1 getInstance() {  
    if (instance == null) {  
        instance = new MemberDtoListV1();  
    }  
    return instance;  
}
```

회원 추가 메서드

```
public void addList(String id, String name) {  
    memberDtoList.add(new MemberDto(id, name));  
}
```

회원 목록 데이터가 필요할 때  
데이터를 돌려주는 getList 메서드

```
public List<MemberDto> getList() {  
    return memberDtoList;  
}
```





# 회원 목록 보기

# 컨트롤러 작성



회원 목록을 보여주는 기능 작업을 위해  
MemberShowControllerV1 작성

controller

- HomeController
- MemberFormControllerV1
- MemberFormControllerV2
- MemberSaveControllerV1
- MemberSaveControllerV2
- MemberShowControllerV1
- MemberShowControllerV2

(서신은 본뚜 코아)





어노테이션으로 해당 클래스가  
컨트롤러이며 log 기능을 쓰겠다고 선언

```
@Controller  🌐 kdtTetz *  
@Slf4j  
public class MemberShowControllerV1 {  
    private MemberDtoListV1 memberList = MemberDtoListV1.getInstance(); 1 use  
  
    @GetMapping(🌐 "/member/show")  🌐 kdtTetz *  
    public String process(HttpServletRequest request, HttpServletResponse re  
        log.info("=====> 회원 조회 페이지 호출, /member/show");  
  
        request.setAttribute(s: "memberList", memberList.getList());  
        return "member-show";  
    }  
}
```

싱글톤 패턴이 적용된 회원 리스트를 쓸 것이므로  
클래스에 저장된 static 멤버 변수를 가져오기

```
@Controller
```

```
@Slf4j
```

```
public class MemberShowControllerV1 {  
    private MemberDtoListV1 memberList = MemberDtoListV1.getInstance();
```

```
@GetMapping("/member/show")
```

```
public String process(HttpServletRequest request, HttpServletResponse re  
    log.info("=====> 회원 조회 페이지 호출, /member/show");
```

```
    request.setAttribute("memberList", memberList.getList());
```

```
    return "member-show";
```

```
}
```

```
}
```

회원 관리를 담당하는 /member Context 의  
/show 주소에 매핑  
GetMapping 을 사용하여 GET 방식 요청에 대응

```
@Controller
@Slf4j
public class MemberShowControllerV1 {
    private MemberDtoListV1 memberList = MemberDtoListV1.getInstance();

    @GetMapping("/member/show")
    public String process(HttpServletRequest request, HttpServletResponse response) {
        log.info("=====> 회원 조회 페이지 호출, /member/show");

        request.setAttribute("memberList", memberList.getList());
        return "member-show";
    }
}
```



```
@Controller
@Slf4j
public class MemberShowControllerV1 {
    private MemberDtoListV1 memberList = MemberDtoListV1.getInstance();

    @GetMapping("/member/show")
    public String process(HttpServletRequest request) {
        log.info("=====");

        request.setAttribute("memberList", memberList.getList());
        return "member-show";
    }
}
```

회원 리스트 데이터를 getList 로 받아서  
request 스코프에 담아서 전달

목록을 보여줄 jsp 페이지를  
member-show.jsp 로 지정



# 회원 목록 보기

# jsp 페이지 작성





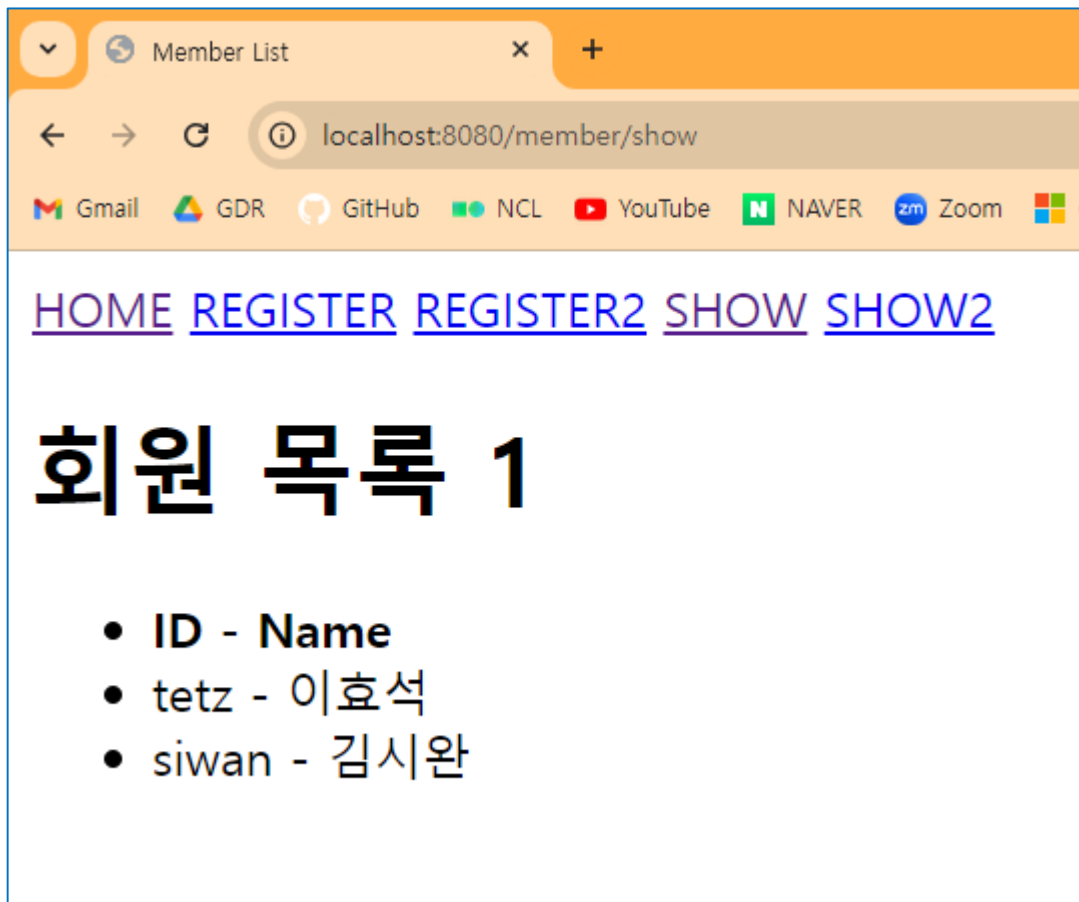
```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html>
<head>
    <title>Member List</title>
</head>
<body>
<%@ include file="header.jsp"%>
<h1>회원 목록 1</h1>
<ul>
    <li><b>ID - Name</b></li>
    <c:forEach var="member" items="${memberList}">
        <li>${member.id} - ${member.name}</li>
    </c:forEach>
</ul>
</body>
</html>
```

JSTL 및 페이지 기본 인코딩 설정

JSTL 과 EL 문법을 사용하여  
회원 목록을 출력!



자~ 이제 시작이야(내꿈을)





# 회원 추가

# jsp 페이지 작성



```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html>
<head>
    <title>Member Register</title>
</head>
<body>
<%@ include file="header.jsp"%>
<h1>회원 추가 1</h1>
<form method="get" action="/member/form/save">
    <label for="id">아이디 :</label>
    <input type="text" id="id" name="id" required>
    <br>
    <label for="name">이름 :</label>
    <input type="password" id="name" name="name" required>
    <br>
    <button type="submit">회원 추가</button>
</form>
</body>
</html>
```

ID와 이름을 입력하고 회원 추가 버튼을 누르면  
GET 방식으로 /member/form/show 주소로  
요청을 보내는 member-form.jsp 페이지를  
만들어 봅시다!

# 실습, MemberFormControllerV1 만들기



- /member/form 에 대응하는 MemberFormControllerV1 를 만들어 주세요
- 해당 컨트롤러는 /member/form 요청이 들어왔을 경우, 방금 작성한 /WEB-INF/views 폴더의 member-form1.jsp 파일을 서빙 합니다
- 헤더의 REGISTER 를 클릭하면 아래와 같은 페이지가 나오면 됩니다!
- log 객체를 이용해서, 요청 주소와 페이지의 이름을 아래와 같이 서버 로그에 출력해주시면 됩니다!

# 실습, MemberShowControllerV2 만들기



Member Register

localhost:8080/member/form

Gmail GDR GitHub NCL YouTube NAVER Zoom

[HOME](#) [REGISTER](#) [REGISTER2](#) [SHOW](#) [SHOW2](#)

## 회원 추가 1

아이디 :

이름 :

```
INFO : org.example.controller.MemberFormControllerV1 - =====> 회원 추가 페이지 호출,  
/member/register
```





# 회원 추가

# 컨트롤러 작성



Member Register

localhost:8080/member/form

Gmail GDR GitHub NCL YouTube NAVER Zoom O36

[HOME](#) [REGISTER](#) [REGISTER2](#) [SHOW](#) [SHOW2](#)

## 회원 추가 1

아이디 :

이름 :

회원 추가 버튼을 클릭하면  
GET 방식으로 요청이 날아갑니다!



그란데 말입니다

Member List

x

+



localhost:8080/member/form/save?id=12&name=12

요러한 요청은 어떻게 받아줘야 할까요!?



```
String id = request.getParameter(s: "id");  
String name = request.getParameter(s: "name");
```



서블릿 기준이면 request 객체에서  
getParameter 를 사용해서 받았습니다!



그란데 말입니다

페이지를 호출하는 요청과  
저렇게 특정 작업을 하는  
요청을 어떻게 구분할 수 있을까요?

만약 페이지가 많다면?  
서비스의 수가 엄청나다면!?

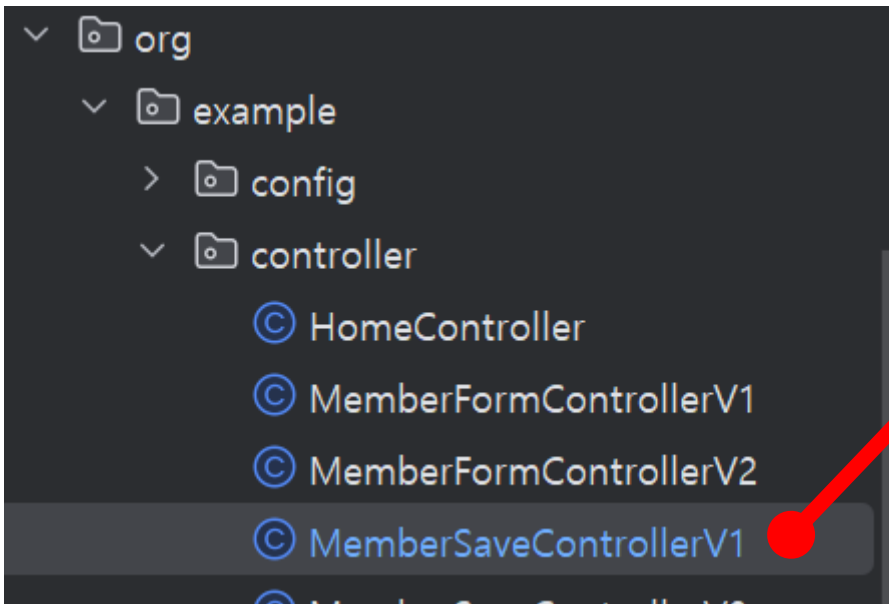
주소를 따로따로 분리하기가 매우 힘들어 집니다  
그리고 메서드 별로 매핑하기도 힘들어 집니다!

@RequestMapping



Request 요청만 따로 받는 어노테이션이 등장!  
(요게 스프링이 핫 해지는데 기여를 했습니다!)

이 몸, 등장



회원 저장 요청을 받아줄  
MemberSaveControllerV1 을 생성

회원 추가를 해야하므로  
회원 데이터를 관리하는 MemberDtoList 의  
인스턴스 가져오기

```
@Controller
@Slf4j
public class MemberSaveControllerV1 {
    private MemberDtoListV1 memberList = MemberDtoListV1.getInstance(); 2 usages

    @RequestMapping(value = "/form/save", method = RequestMethod.GET)
    public String process(HttpServletRequest request, HttpServletResponse response) {
        log.info("=====> 회원 추가 Request 호출, /member/form/save");

        String id = request.getParameter("id");
        String name = request.getParameter("name");

        memberList.addList(id, name);

        request.setAttribute("memberList", memberList.getList());
        return "member-show";
    }
}
```



GET 방식, /from/save 로 전달되는  
리퀘스트를 가져오도록  
@RequestMapping 을 사용하여 매핑

```
@Controller
@Slf4j
public class MemberSaveControllerV1 {
    private MemberDtoListV1 memberList = MemberDtoListV1.getInstance();

    @RequestMapping(value = "/form/save", method = RequestMethod.GET)
    public String process(HttpServletRequest request, HttpServletResponse response) {
        log.info("=====> 회원 추가 Request 호출, /member/form/save");

        String id = request.getParameter("id");
        String name = request.getParameter("name");

        memberList.addList(id, name);


        request.setAttribute("memberList", memberList.getList());
        return "member-show";
    }
}
```

@Controller  kdtTetz \*

@Slf4j

```
public class MemberSaveControllerV1 {
```

```
    private MemberDtoListV1 memberList = MemberDtoListV1.getInstance(); 2 usages
```

```
    @RequestMapping(value =  "/form/save", method = RequestMethod.GET) kdtTetz
```

```
    public String process(HttpServletRequest request, HttpServletResponse response) {
```

```
        log.info("=====> 회원 추가 Request 호출, /member/form/save");
```

```
        String id = request.getParameter(s: "id");
```

```
        String name = request.getParameter(s: "name");
```

```
        memberList.addList(id, name);
```

```
        request.setAttribute(s: "memberList", memberList.getList());
```

```
        return "member-show";
```

```
    }
```

```
}
```




파라미터로 부터 데이터를 받아서  
변수에 담기

@Controller  kdtTetz \*

@Slf4j

```
public class MemberSaveControllerV1 {
```

```
    private MemberDtoListV1 memberList = MemberDtoListV1.getInstance(); 2 usages
```

```
    @RequestMapping(value =  "/form/save", method = RequestMethod.GET) kdtTetz
```

```
    public String process(HttpServletRequest request, HttpServletResponse response) {
```

```
        log.info("=====> 회원 추가 Request 호출, /member/form/save");
```

```
        String id = request.getParameter(s: "id");
```

```
        String name = request.getParameter(s: "name");
```

```
        memberList.addList(id, name);
```

```
        request.setAttribute(s: "memberList", memberList.getList());
```

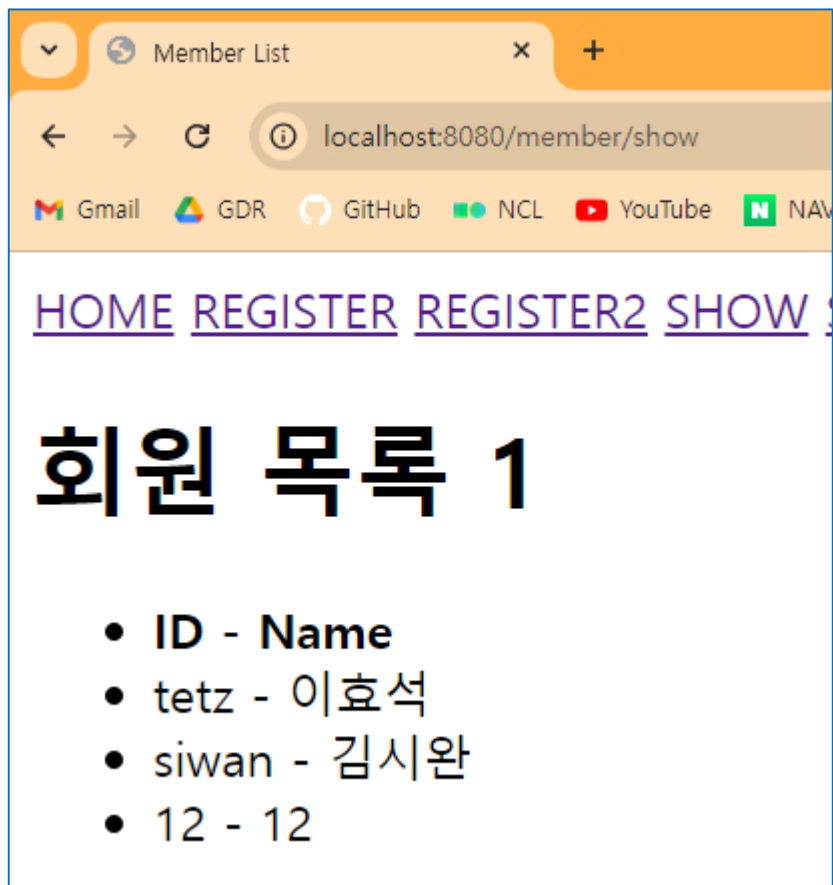
```
        return "member-show";
```

```
    }
```

```
}
```



멤버 리스트에 새로운 회원을 추가하고  
Request 스코프에 담아서  
member-show.jsp 페이지 호출



# 실습, Todo 기능을 구현해 봅시다!



- Todo 목록 보기와 추가 기능을 구현해 봅시다!
- 아래와 같이 Header 에 2가지 링크를 추가하고, 각각의 링크를 클릭하면 Todo 목록 보기와 Todo 추가 페이지가 보이면 됩니다
- Todo 추가 페이지에서 할 일을 입력하고 할 일 추가 버튼을 클릭하면 Todo 목록 페이지로 이동이 되며 추가 된 할일 목록까지 출력이 됩니다!
- 위의 기능을 위한 `TodoDtoListV1` / `TodoShowControllerV1` / `TodoFormControllerV1` / `TodoSaveControllerV1` 을 작성해 주세요!

# 실습, Todo 기능을 구현해 봅시다!



- TodoDto 코드입니다!

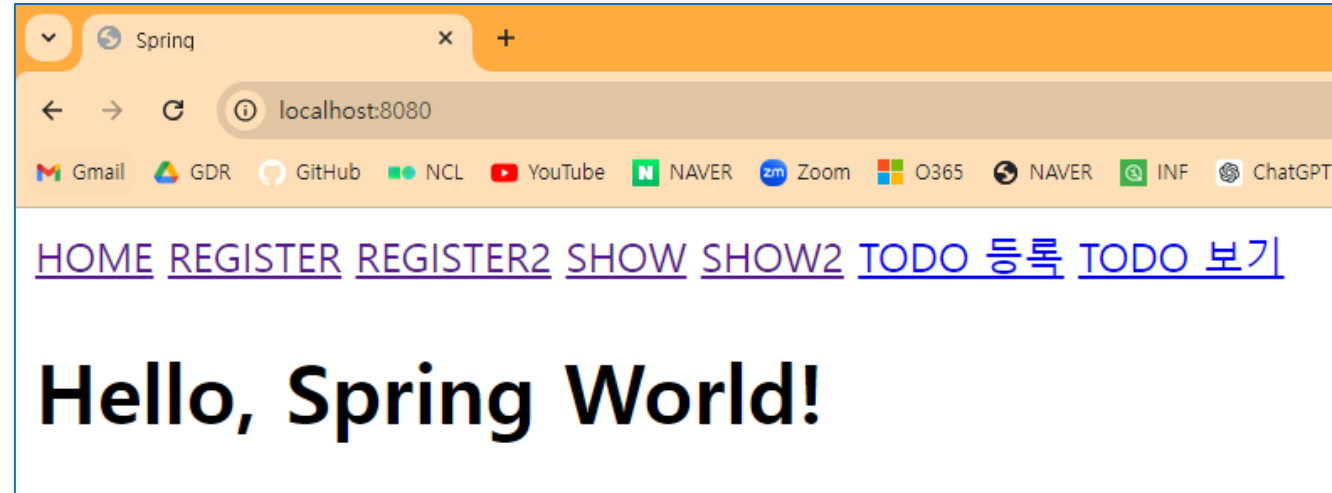
```
public class TodoDto { 3 usages  new *  
    private String todo; 3 usages  
  
    public String getTodo() { no usages  
        return todo;  
    }  
  
    public void setTodo(String todo) {  
        this.todo = todo;  
    }  
  
    public TodoDto(String todo) { 1 usage  
        this.todo = todo;  
    }  
}
```

# 실습, Todo 기능을 구현해 봅시다!



- 새로운 Header 코드 및 실제 화면 입니다!

```
<header>
  <a href="/">HOME</a>
  <a href="/member/form">REGISTER</a>
  <a href="/member/form2">REGISTER2</a>
  <a href="/member/show">SHOW</a>
  <a href="/member/show2">SHOW2</a>
  <a href="/todo/form">TODO 등록</a>
  <a href="/todo/show">TODO 보기</a>
</header>
```

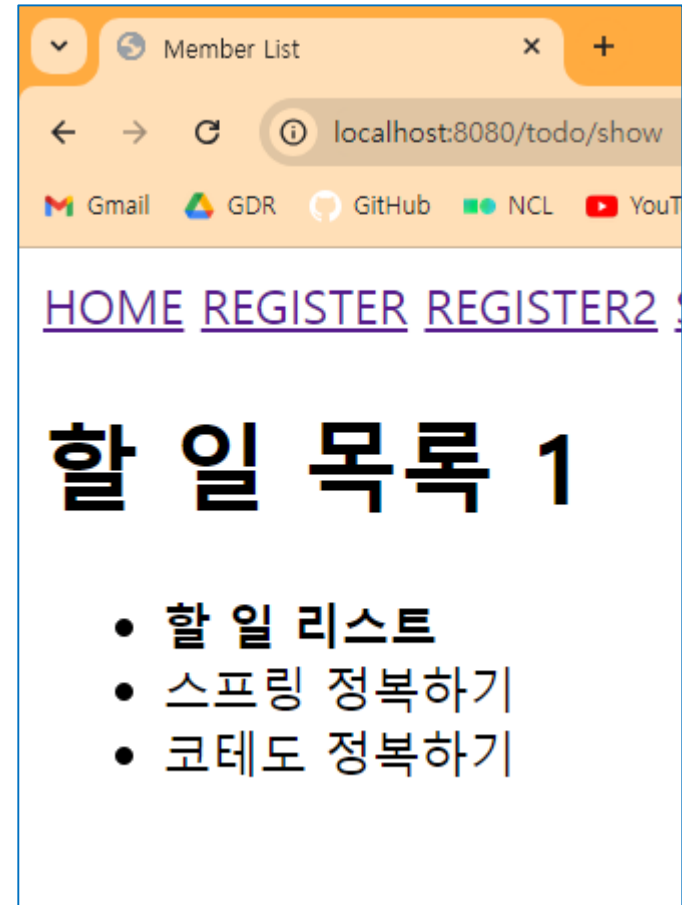


# 실습, Todo 기능을 구현해 봅시다!



- Todo 보기 페이지(todo-show.jsp) 코드 및 실제 화면 입니다

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html>
<head>
    <title>Member List</title>
</head>
<body>
<%@ include file="header.jsp"%>
<h1>할 일 목록 1</h1>
<ul>
    <li><b>할 일 리스트</b></li>
    <c:forEach var="todo" items="${todoList}">
        <li>${todo.todo}</li>
    </c:forEach>
</ul>
</body>
</html>
```





# 실습, Todo 기능을 구현해 봅시다!



- Todo 등록 페이지(todo-form.jsp) 코드 및 실제 화면 입니다

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html>
<head>
    <title>Member Register</title>
</head>
<body>
<%@ include file="header.jsp"%>
<h1>할 일 추가 1</h1>
<form method="get" action="/todo/form/save">
    <label for="todo">할 일 :</label>
    <input type="password" id="todo" name="todo" required>
    <br>
    <button type="submit">할 일 추가</button>
</form>
</body>
</html>
```

Member Register

localhost:8080/todo/form

Gmail GDR GitHub NCL YouTube

[HOME](#) [REGISTER](#) [REGISTER2](#) [SHC](#)

## 할 일 추가 1

할 일 :



# 코드 빈칸

# 채우기 및 제공 코드

# TodoDtoListV1 는 코드 전체를 드립니다! 😊



```
public class TodoDtoListV1 { 9 usages  new *
    private static TodoDtoListV1 instance; 3 usages
    private List<TodoDto> todoDtoList; 3 usages

    private TodoDtoListV1() { 1 usage  new *
        this.todoDtoList = new ArrayList<>(); // List 초기화
        // 테스트 데이터 추가
        this.addList(todo: "스프링 정복하기");
        this.addList(todo: "코테도 정복하기");
    }

    // 싱글톤 인스턴스 반환 메소드
    public static synchronized TodoDtoListV1 getInstance() {
        if (instance == null) {
            instance = new TodoDtoListV1();
        }
        return instance;
    }
}
```

```
public void addList(String todo) { 3 usages  new *
    todoDtoList.add(new TodoDto(todo));
}

public List<TodoDto> getList() { return todoDtoList; }
```

# TodoShowControllerV1



```
@Controller  new *
@Slf4j
public class TodoShowControllerV1 {
    private TodoDtoListV1 todoList = TodoDtoListV1.getInstance(); 1 usage

    @GetMapping("/todo/show")  new *
    public String process(HttpServletRequest request, HttpServletResponse resp

    }
}
```

요기를 완성하세요!

# TodoFormControllerV1



```
import ...

@Controller new *
@Slf4j
public class TodoFormControllerV1 {
    @GetMapping("/todo/form") new *
    public String home(HttpServletRequest request, HttpServletResponse response) {
    }
}
```

요기를 완성하세요!!

# TodoSaveControllerV1



```
@Controller new *
@Slf4j
public class TodoSaveControllerV1 {
    private TodoDtoListV1 todoDtoListV1 = TodoDtoListV1.getInstance(); 2 usages

    @RequestMapping(value = "/todo/form/save", method = RequestMethod.GET)
    public String process(HttpServletRequest request, HttpServletResponse response) {

    }
}
```

요기를 완성하세요!!