

# 웹 개발 스터디

2강: JAVA, Spring<sup>[1]</sup> 그리고 SOLID원칙

[1] <https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#spring-core>





# JAVA: basic

- 객체지향 프로그래밍 언어
- 바이트코드(.class) 형태로 컴파일 되어, JVM(Java Virtual Machine) 환경 위에서 동작
  - 어떤 플랫폼(하드웨어, 운영체제)인지 관련없이, JVM만 있다면 동작한다.
- 높은 이식성(Portability)
  - 웹서버, 모바일 앱 개발 등에 널리 사용된다.



# JAVA: feature

## > 완전한 객체지향

- 클래스, 상속, 다형성, 캡슐화, SOLID등의 패러다임

## > 포인터가 없다

- 내부적으로 당연히 메모리를 가리키는 포인터가 있긴 하지만, 사용자 코드의 접근은 제한되어 있다. 대신 메모리 관리는 GC(Garbage Collector)가 수행한다.



# JAVA: example

- 모든것이 클래스...
  - main함수 까지도 class안에 정의한다

```
Main.java ×  
1  ▶ public class Main {  
2  ▶  └─ public static void main(String[] args) {  
3      └─ System.out.println("Hello World!");  
4      └─ }  
5  }
```



# JAVA: develop

## ➤ IDE

- Eclipse, IntelliJ IDEA가 대표적인 개발환경 또는 VS Code환경
- Spring framework또한 위의 도구를 통해 빌드할 수 있음

## ➤ Maven, Gradle을 통한 의존성관리 및 빌드

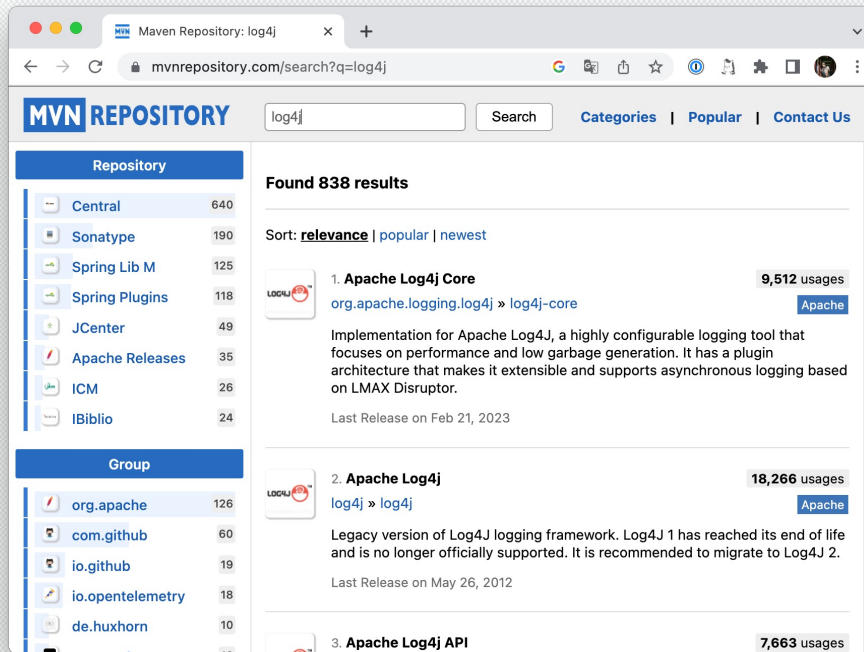
```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache
4         <modelVersion>4.0.0</modelVersion>
5         <parent>
6         <groupId>org.springframework.boot</groupId>
7         <artifactId>spring-boot-starter-parent</artifactId>
8         <version>2.2.6.RELEASE</version>
9         <relativePath />
10        </parent>
11        <groupId>com.kw.ac.kr</groupId>
12        <artifactId>server</artifactId>
13        <version>RELEASE-1.0.14</version>
14        <packaging>war</packaging>
15        <name>COM'S Homepage</name>
16        <description>COM's Web Server</description>
17        <properties>
18        <java.version>11</java.version>
19        </properties>
20
21        <dependencies>
22        <!-- Spring boot -->
23        <dependency>
24        <groupId>org.springframework.boot</groupId>
25        <artifactId>spring-boot-starter-tomcat</artifactId>
26        <scope>provided</scope>
27        </dependency>
```

# JAVA: develop

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache
4         <modelVersion>4.0.0</modelVersion>
5         <parent>
6             <groupId>org.springframework.boot</groupId>
7             <artifactId>spring-boot-starter-parent</artifactId>
8             <version>2.2.6.RELEASE</version>
9             <relativePath />
10        </parent>
11        <groupId>coms.kw.ac.kr</groupId>
12        <artifactId>server</artifactId>
13        <version>RELEASE-1.0.14</version>
14        <packaging>war</packaging>
15        <name>COM'S Homepage</name>
16        <description>COM's Web Server</description>
17        <properties>
18            <java.version>11</java.version>
19        </properties>
20
21        <dependencies>
22            <!-- Spring boot -->
23            <dependency>
24                <groupId>org.springframework.boot</groupId>
25                <artifactId>spring-boot-starter-tomcat</artifactId>
26                <scope>provided</scope>
27            </dependency>
```

# JAVA: develop

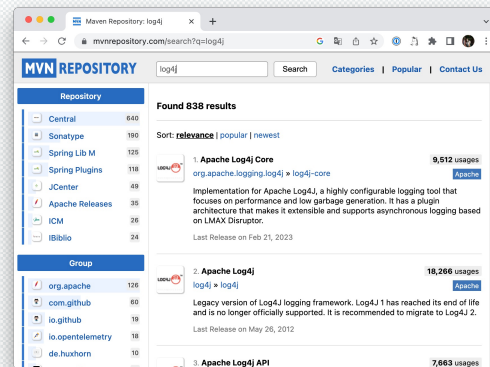
➤ 저 수많은 dependencies는 어디서 오는가



# JAVA: develop

## ➤ 저 수많은 dependencies는 어디서 오는가

- 원격 저장소에서 의존성을 받아온다
- 로컬 빌드 툴(Maven, Gradle)이 의존성과 사용자 코드를 잘 살펴보고 프로젝트를 빌드한다





# Spring Framework: basic

- 자바 어플리케이션 개발을 위한 통합 프레임워크
  - 주로 웹 백엔드 개발에 널리 사용된다
- 여러 Spring Module로 구성
  - Spring MVC
  - Spring Security
- 객체지향 개발을 위한 여러 기능을 제공
  - IOC(Inversion Of Control)
  - DI(Dependency Injection)
  - AOP(Aspect Oriented Programming)

# Spring Framework: IOC

- 일반적인 프로그래밍에서의 제어 흐름은 사용자 코드에 의해 결정
  - 사용자 코드가 입력을 받음
  - 사용자 코드가 입력에 대해 조건분기를 수행
  - 사용자 코드가 조건에 따라 프로그램을 수행
- Spring Framework는 IOC통해
  - 전체적인 제어 흐름을 결정
  - 사용자 코드가 처리해야할 순간에 사용자 코드를 호출하여 처리
  - 이를 통해 코드 결합도를 낮출 수 있음
- ????

# Spring Framework: IOC

## ➤ 코드로 이해 해 봅시다

- SpringBoot의 메인함수
- 그냥 자기자신(Application.class)를 어떤 인자와 함께 실행함
- 웹페이지를 띄우는 코드는 어디 있지? 적어도 이름은 보여야할 것 아닌가?

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

# Spring Framework: IOC

## > 의문점

- 웹페이지를 띄우는 코드는 어디 있지? 적어도 이름은 보여야할 것 아닌가?
- 그냥 자기자신(Application.class)를 어떤 인자와 함께 실행함

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

# Spring Framework: IOC

## > 의문점

- @SpringBootApplication이라는 Annotation
- Spring Framework는 해당 Annotation이 있는 클래스를 발견하면, 웹서버에 필요한 기능을 제공한다.

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

# Spring Framework: IOC

## > @GetMapping

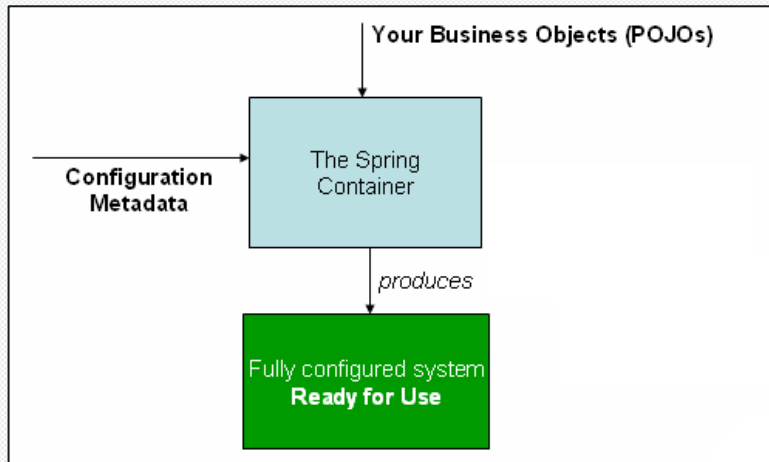
- HTTP GET메서드로 들어오는 요청에 대한 핸들러를 명시하는 어노테이션
- 요청을 받고, 미들웨어등 처리는 스프링 MVC가 수행한다
- 개발자는 핸들러 비즈니스 로직에 집중

```
@RestController
public class GreetingController {
    ... @Autowired
    ... private GreetingService greetingService;


    ... @GetMapping("/greet")
    ... public String greet(@RequestParam String name) {
    ...     ... return greetingService.greet(name);
    ... }
}
```

# Spring Framework: IOC

- 그래서 어떻게 IOC가 가능한가
  - 개발자는 Annotation을 적절히 삽입하여 비즈니스 로직 작성(POJO)
  - 프레임워크는 Configuration(Annotation, XML...)과 POJO코드를 합성하여 Bean이라는 객체의 형태로 IOC Container에 등록한다



# Spring Framework: DI

A yellow thinking emoji with a hand on its chin, positioned over the code. An orange oval highlights the `@Autowired` annotation.

```
@RestController
public class GreetingController {
    @Autowired
    private GreetingService greetingService;

    @GetMapping("/greet")
    public String greet(@RequestParam String name) {
        return greetingService.greet(name);
    }
}
```



# Spring Framework: DI

## > 어떤 객체A가 객체B의 인스턴스를 소유하는 경우를 생각해보자

- B의 구현이 달라져야 하는 경우, 메서드 구현이 바뀌는 경우 어찌지
- 그럴 때 쓰라고 Java에는 Interface라는 타입이 있습니다.



## > Interface를 사용한 예제

```
@Service
public class AttachmentServiceImpl implements AttachmentService {

    private final AttachmentDAO attachmentDAO;
    private final FileIOManager fileIOManager;

    @Autowired
    public AttachmentServiceImpl(AttachmentDAO attachmentDAO, String LOCAL_STORAGE_ROOT) {
        this.attachmentDAO = attachmentDAO;
        this.fileIOManager = new FileIOManager(LOCAL_STORAGE_ROOT);
    }

    //...
}
```

# Spring Framework: DI

## ➤ Interface를 사용한 예제

```
@Service
public class AttachmentServiceImpl implements AttachmentService {

    private final AttachmentDAO attachmentDAO;
    private final FileManager fileIOManager;

    @Autowired
    public AttachmentServiceImpl(AttachmentDAO attachmentDAO, String LOCAL_STORAGE_ROOT) {
        this.attachmentDAO = attachmentDAO;
        this.fileIOManager = new FileManager(LOCAL_STORAGE_ROOT);
    }

    //...
}
```

```
import coms.kw.ac.kr.server.vo.article.AttachmentVO;

public interface AttachmentDAO {

    public void insertAttachmentInfo(AttachmentVO attachment);
}
```

# Spring Framework: DI

## ➤ Interface를 사용한 예제

- 동일한 인터페이스를 사용하므로, 구현체가 달라짐에 따라 생기는 의존관계는 어느정도 해결
- 하지만, 여전히 객체B의 생성과 생명주기 관리는 객체A가 수행해야 함
- 어떻게 해결해야 하는가?

## ➤ Dependency Injection

- 그럼 생성과 생명주기 관리를 대신 해주는 누군가가 있으면 되잖아?
- 프레임워크(Spring IOC Container)가 주입될 후보(Bean)을 미리 1개 생성해두고 필요에 따라 주입(Inject)한다
- Injection이 필요한 부분에 @Autowired 어노테이션을 명시한다

# Spring Framework: DI

## > Dependency Injection

- 그림 생성과 생명주기 관리를 대신 해주는 누군가가 있으면 되잖아?
- 프레임워크(Spring IOC Container)가 주입될 후보(Bean)을 미리 1개 생성해두고 필요에 따라 주입(Inject)한다
- Injection이 필요한 부분에 @Autowired 어노테이션을 명시한다

```
@RestController
public class GreetingController {
    ... @Autowired
    ... private GreetingService greetingService;

    ... @GetMapping("/greet")
    ... public String greet(@RequestParam String name) {
    ...     ... return greetingService.greet(name);
    ... }
}
```

# Spring Framework: DI

## ➤ Dependency Injection

- 그럼 생성과 생명주기 관리를 대신 해주는 누군가가 있으면 되잖아?
- 프레임워크(Spring IOC Container)가 주입될 후보(Bean)을 미리 1개 생성해두고 필요에 따라 주입(Inject)한다
- Injection이 필요한 부분에 @Autowired 어노테이션을 명시한다

```
@Service
public class AttachmentServiceImpl implements AttachmentService {

    private final AttachmentDAO attachmentDAO;
    private final FileIOManager fileIOManager;

    @Autowired
    public AttachmentServiceImpl(AttachmentDAO attachmentDAO, String LOCAL_STORAGE_ROOT) {
        this.attachmentDAO = attachmentDAO;
        this.fileIOManager = new FileIOManager(LOCAL_STORAGE_ROOT);
    }

    //...
}
```