



# DAY - 01

Spring IoC

# Framework 개요

- Framework의 의미

### 영어사전

**framework** 미국식 [ˈfreɪmwɜːrk]  영국식 [ˈfreɪmwɜːk]  ★ [다른 뜻\(1건\)](#) [예문보기](#)

#### 학습정보

1. (건물 등의) 뼈대 2. (판단·결정 등을 위한) 틀 3. 체제, 체계

- Framework는 어플리케이션을 개발할 때, 아키텍처에 해당하는 골격 코드를 제공한다.
- Solution이 완제품이라면 Framework는 반제품에 해당한다.

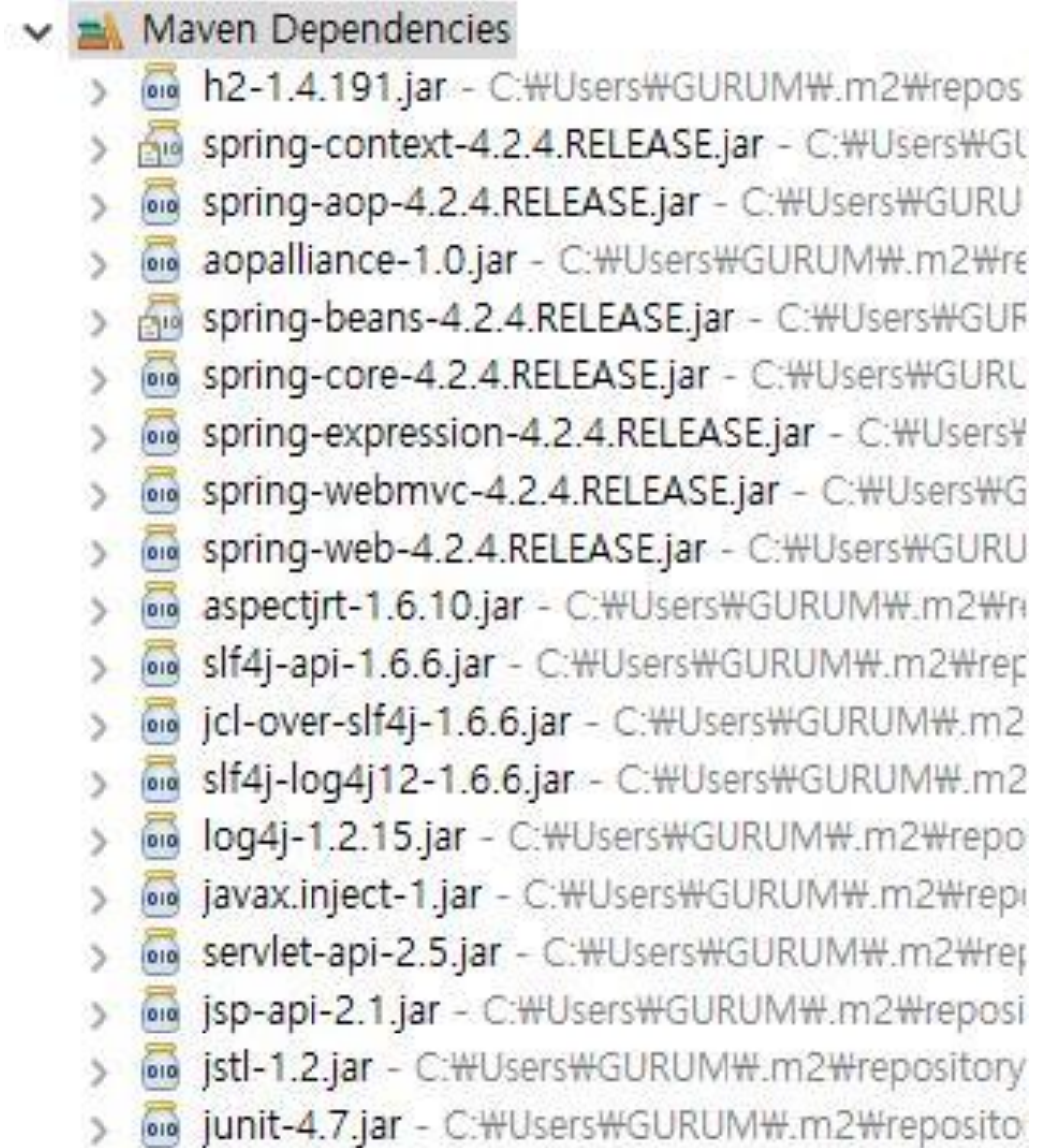
- Framework의 장점
  - 빠른 구현 시간
  - 관리 용이성 증가
  - 개발자의 역량 획일화
  - 검증된 아키텍처의 재사용과 일관성 유지

## • Java 기반의 Framework

처리 영역	프레임워크	설명
Presentation	Struts	UI Layer에 중점을 두고 개발된 MVC(Model View Controller) 프레임워크임.
	Spring (MVC)	Struts와 동일하게 MVC 아키텍처를 제공하는 UI Layer 프레임워크임. Struts처럼 독립된 프레임워크는 아니고 Spring 프레임워크에 포함되어 있음.
Business	Spring (IoC, AOP)	Spring의 IoC와 AOP 모듈을 이용하여 Spring 컨테이너에서 동작하는 엔터프라이즈 비즈니스 컴포넌트를 개발할 수 있음.
Persistence	Hibernate or JPA	Hibernate는 완벽한 ORM(Object Relation Mapping) 프레임워크임. ORM 프레임워크는 SQL을 프레임워크가 자체적으로 생성하여 DB 연동을 처리함. JPA는 Hibernate를 비롯한 모든 ORM의 자바 표준 API임.
	iBatis or Mybatis	iBatis 프레임워크는 개발자가 작성한 SQL 명령어와 자바 객체(VO 혹은 DTO)를 매핑해주는 기능을 제공함. Mybatis는 iBatis에서 파생된 프레임워크로서 기본 개념과 문법은 거의 유사함.

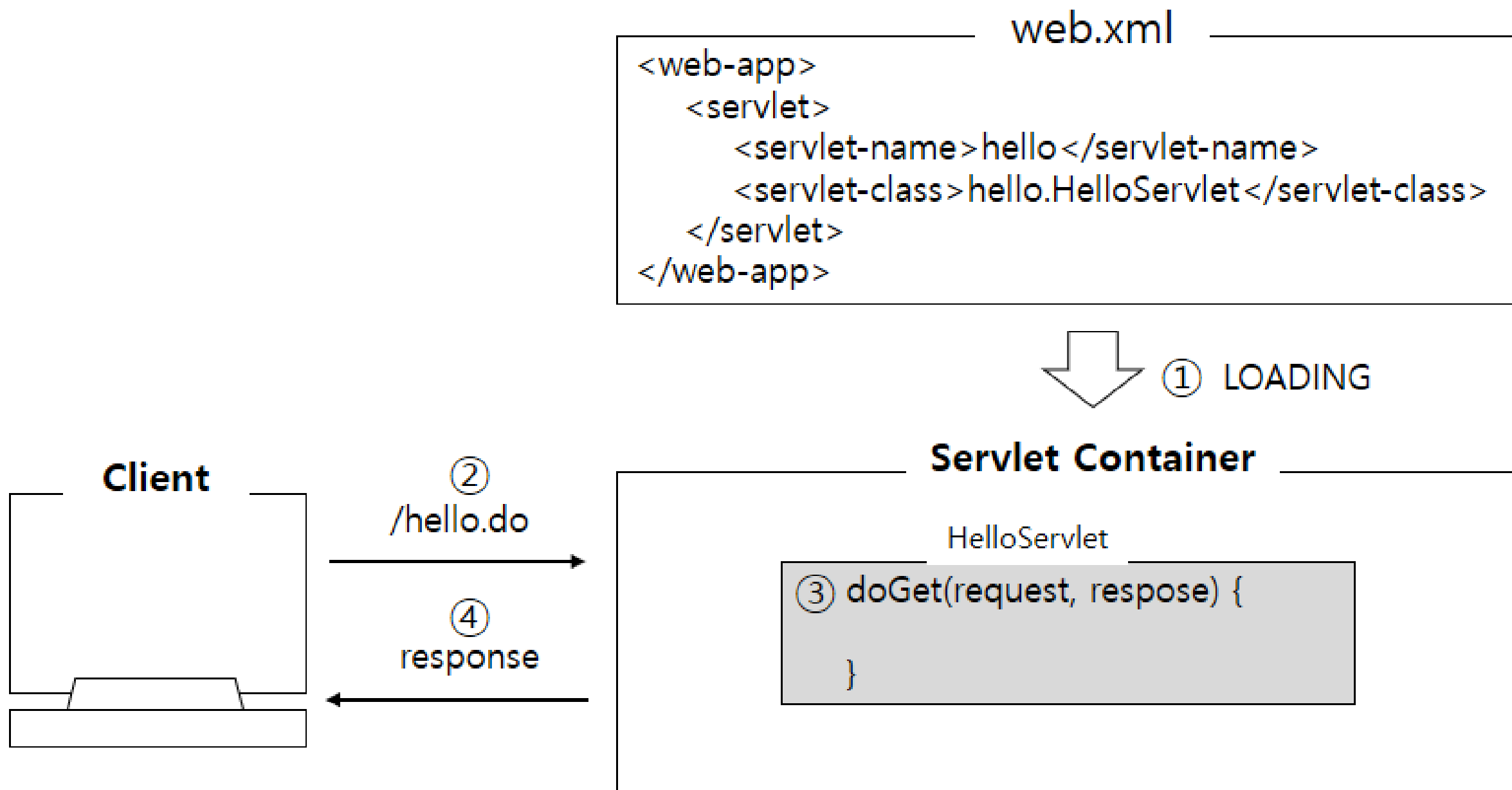
# • Spring Framework의 특징

- 경량(lightweight)
- IoC(Inversion of Control)
- AOP(Aspect Oriented Programming)
- 컨테이너(Container)
- Framework



# Spring Container

- Servlet 컨테이너의 동작





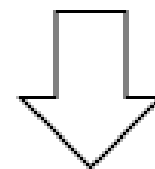
- Spring 컨테이너의 종류

구현 클래스	기능
<b>GenericXmlApplicationContext</b>	파일 시스템이나 클래스 경로에 있는 XML 설정 파일을 로딩하여 구동하는 컨테이너.
<b>XmlWebApplicationContext</b>	웹 기반의 스프링 애플리케이션을 개발할 때 사용하는 컨테이너.

- Spring 컨테이너의 동작

applicationContext.xml

```
<beans>  
  <bean id="tv" class="polymorphism.SamsungTV"/>  
</beans>
```



LOADING

**TVUser**

③ `getBean("tv")`

④ `return`

① **Container(ApplicationContext)**

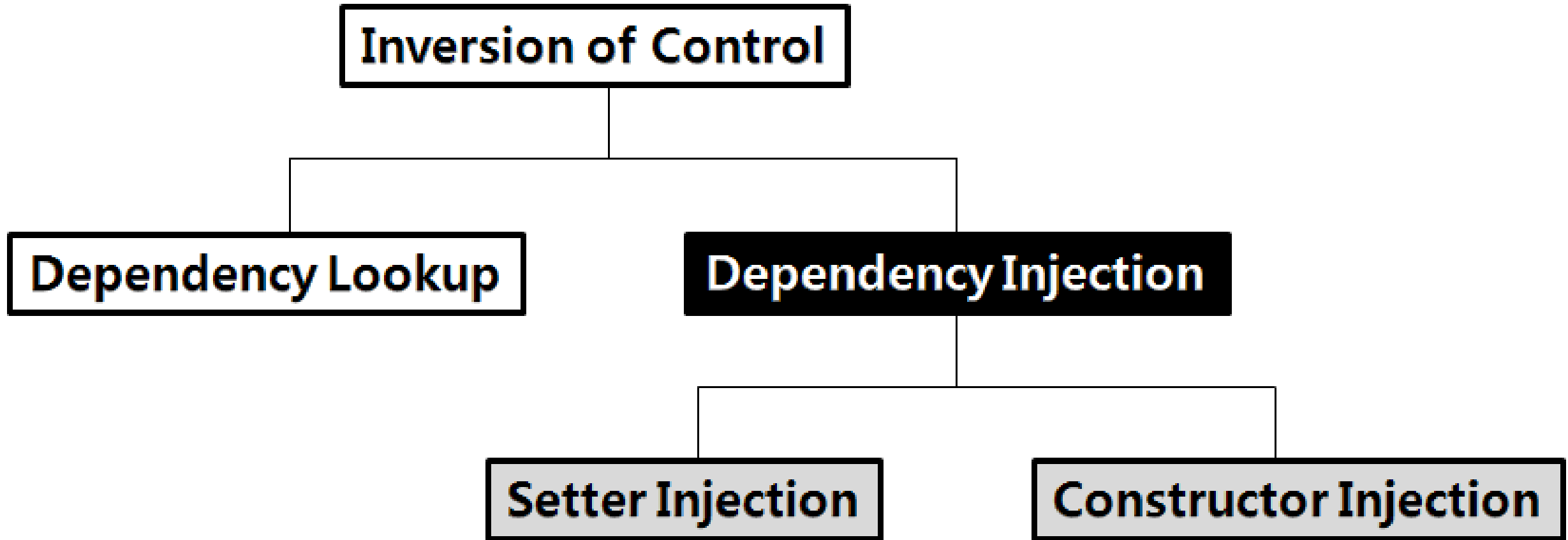
②

SamsungTV

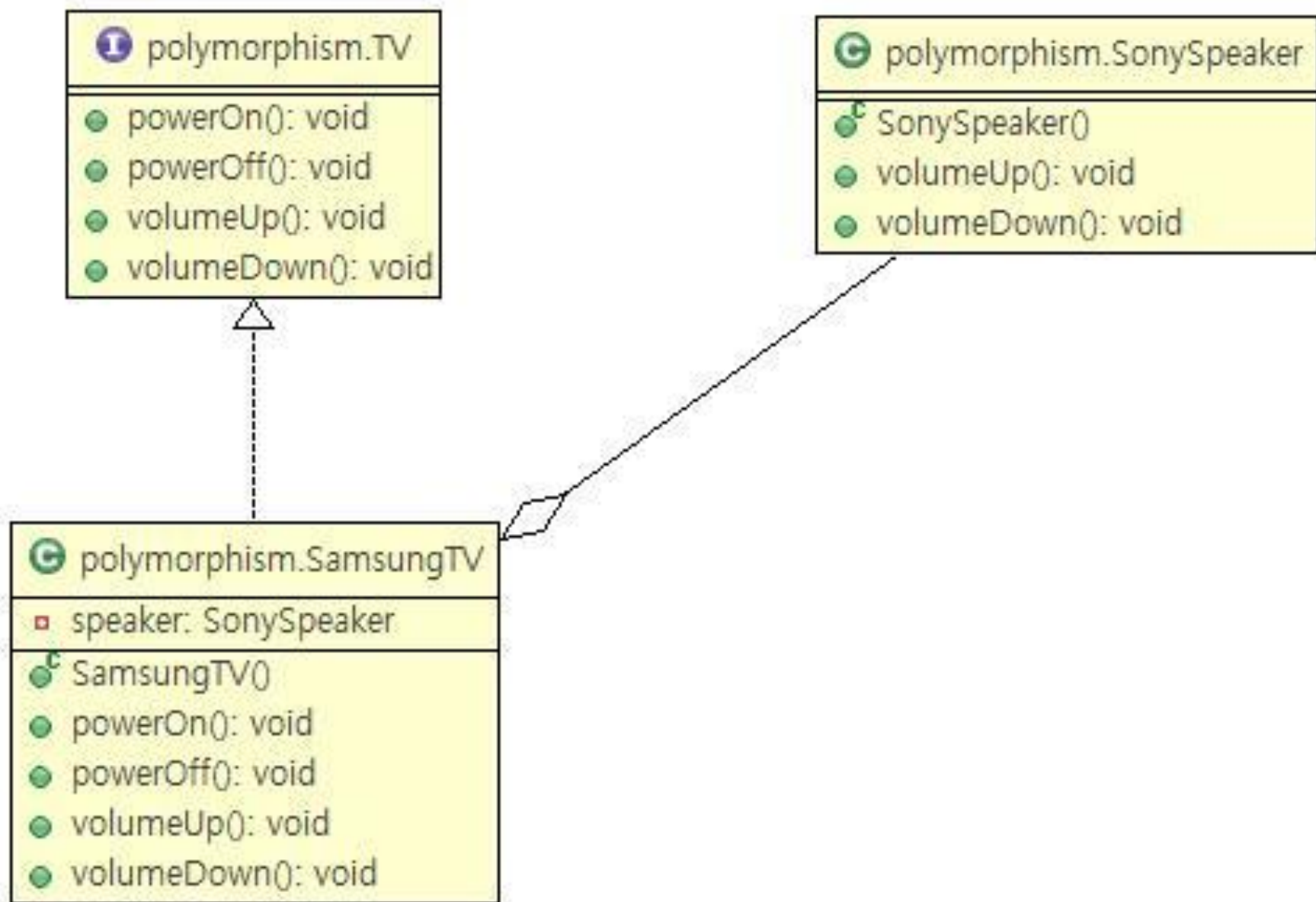


# Dependency Injection

- Dependency Injection



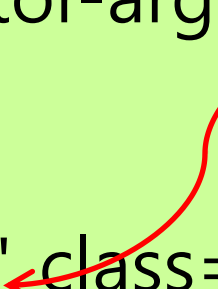
- 의존(Dependency) 관계



- Constructor Injection (1)

```
public SamsungTV(SonySpeaker speaker) {  
    System.out.println("===> SamsungTV 객체 생성");  
    this.speaker = speaker;  
}
```

```
<bean id="tv" class="polymorphism.SamsungTV">  
    <constructor-arg ref="sony"></constructor-arg>  
</bean>  
  
<bean id="sony" class="polymorphism.SonySpeaker"></bean>
```



- Constructor Injection (2)

```
public SamsungTV(SonySpeaker speaker, int price) {  
    System.out.println("===> SamsungTV 객체 생성");  
    this.speaker = speaker;  
    this.price = price;  
}
```

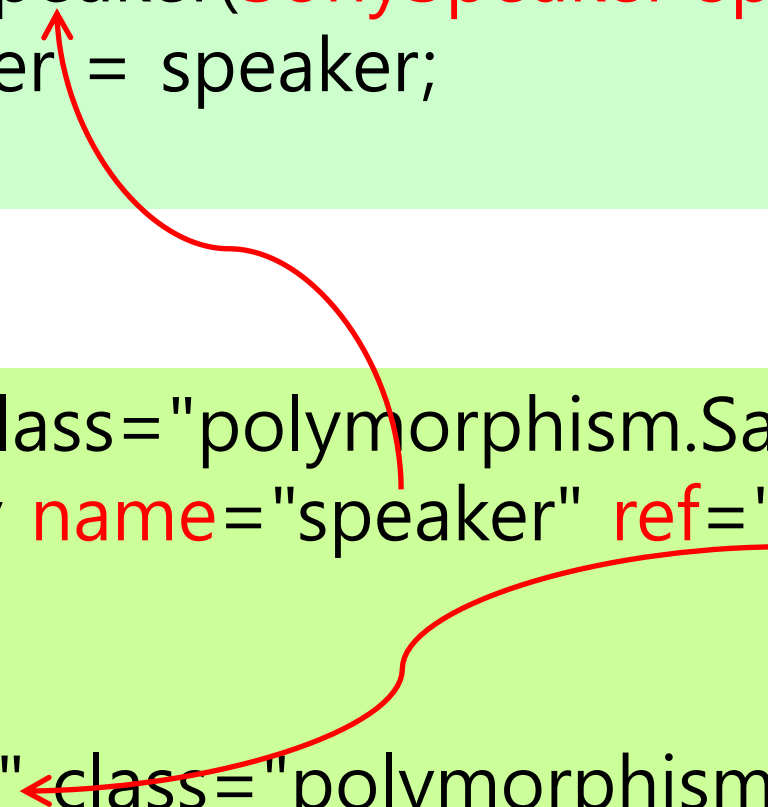
```
<bean id="tv" class="polymorphism.SamsungTV">  
    <constructor-arg ref="sony"></constructor-arg>  
    <constructor-arg value="2700000"></constructor-arg>  
</bean>
```

- Setter Injection (1)

```
public void setSpeaker(SonySpeaker speaker) {  
    this.speaker = speaker;  
}
```

```
<bean id="tv" class="polymorphism.SamsungTV">  
    <property name="speaker" ref="sony"> </property>  
</bean>
```

```
<bean id="sony" class="polymorphism.SonySpeaker"> </bean>
```

A red arrow originates from the 'ref="sony"' attribute in the first XML bean definition, curves downwards and to the left, and points to the 'class="polymorphism.SonySpeaker"' attribute in the second XML bean definition. Another red arrow originates from the 'ref="sony"' attribute, curves upwards and to the left, and points to the 'speaker' parameter in the Java setter method signature.



- Setter Injection (2)

```
public SamsungTV(SonySpeaker speaker, int price) {  
    System.out.println("=== > SamsungTV 객체 생성");  
    this.speaker = speaker;  
    this.price = price;  
}
```

```
<bean id="tv" class="polymorphism.SamsungTV">  
    <property name="speaker" ref="sony"> </property>  
    <property name="price" value="2700000"> </property>  
</bean>
```

- p 네임스페이스 사용

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="lg" class="polymorphism.SamsungTV" p:speaker-ref="sony"/>

  <bean id="sony" class="polymorphism.SonySpeaker"/>

</beans>
```

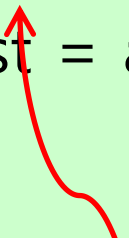
SamsungTV 클래스  
setSpeaker() 호출

- 컬렉션 주입(Collection Injection)

컬렉션 유형	엘리먼트
java.util.List, 배열	<list>
java.util.Set	<set>
java.util.Map	<map>
java.util.Properties	<props>

- java.util.List OR 배열

```
public class CollectionBean {  
    private List<String> addressList;  
  
    public void setAddressList(List<String> addressList) {  
        this.addressList = addressList;  
    }  
}
```

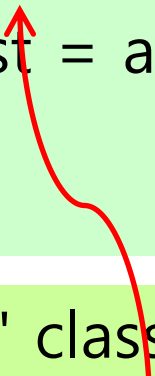


```
<bean id="collectionBean" class="com.springbook.ioc.injection.CollectionBean">  
    <property name="addressList">  
        <list>  
            <value>서울시 강남구 역삼동</value>  
            <value>서울시 성동구 성수동</value>  
        </list>  
    </property>  
</bean>
```

- java.util.Set

```
public class CollectionBean {  
    private Set<String> addressList;  
  
    public void setAddressList(Set<String> addressList){  
        this.addressList = addressList;  
    }  
}
```

```
<bean id="collectionBean" class="com.springbook.ioc.injection.CollectionBean">  
    <property name="addressList">  
        <set value-type="java.lang.String">  
            <value>서울시 강남구 역삼동</value>  
            <value>서울시 성동구 성수동</value>  
            <value>서울시 성동구 성수동</value>  
        </set>  
    </property>  
</bean>
```



- java.util.Map


```
public class CollectionBean {  
    private Map<String, Controller> addressList;  
  
    public void setAddressList(Map<String, Controller> mappings){  
        this.mappings = mappings;  
    }  
}
```

```
<bean id="collectionBean" class="com.springbook.ioc.injection.CollectionBean">  
    <property name="addressList">  
        <map>  
            <entry>  
                <key> <value> 고길동 </value> </key>  
                <value> 서울시 강남구 역삼동 </value>  
            </entry>  
        </map>  
    </property>  
</bean>
```

- java.util.Set

```
public class CollectionBean {  
    private Properties addressList;  
  
    public void setAddressList(Properties mappings){  
        this.mappings = mappings;  
    }  
}
```

```
<bean id="collectionBean" class="com.springbook.ioc.injection.CollectionBean">  
    <property name="addressList">  
        <props>  
            <prop key="고길동">서울시 강남구 역삼동</prop>  
            <prop key="마이콜">서울시 강서구 화곡동</prop>  
        </props>  
    </property>  
</bean>
```



Annotation 설정



- Component-Scan

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:p="http://www.springframework.org/schema/p"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.2.xsd">

  <context:component-scan base-package="com.springbook.biz"/>

</beans>
```

**com.multicampus.biz** 패키지로 시작하는 모든 클래스를 스캔한다.

- @Component 설정

```
<bean id="tv" class="polymorphism.LgTV"> </bean>
```



```
@Component("tv")
```

```
public class LgTV implements TV {  
    public LgTV() {  
        System.out.println("===> LgTV 객체 생성");  
    }  
}
```

- Dependency Injection 설정

어노테이션	설명
<b>@Autowired</b>	주로 변수 위에 설정하여 해당 타입의 객체를 찾아서 자동으로 할당 org.springframework.beans.factory.annotation.Autowired
<b>@Qualifier</b>	특정 객체의 이름을 이용하여 의존성 주입할 때 사용 org.springframework.beans.factory.annotation.Qualifier
<b>@Inject</b>	@Autowired와 동일한 기능을 제공 javax.annotation.Resource
<b>@Resource</b>	@Autowired와 @Qualifier의 기능을 결합한 어노테이션 javax.inject.Inject

- @Autowired

```
@Component("tv")
```

```
public class LgTV implements TV {
```

```
    @Autowired
```

```
    private Speaker speaker;
```

**Speaker 타입의 객체를  
메모리에서 찾아 할당한다.**

```
    public LgTV() {
```

```
        System.out.println("==> LgTV 객체 생성");
```

```
    }
```

```
    public void volumeUp() {
```

```
        speaker.volumeUp();
```

```
    }
```

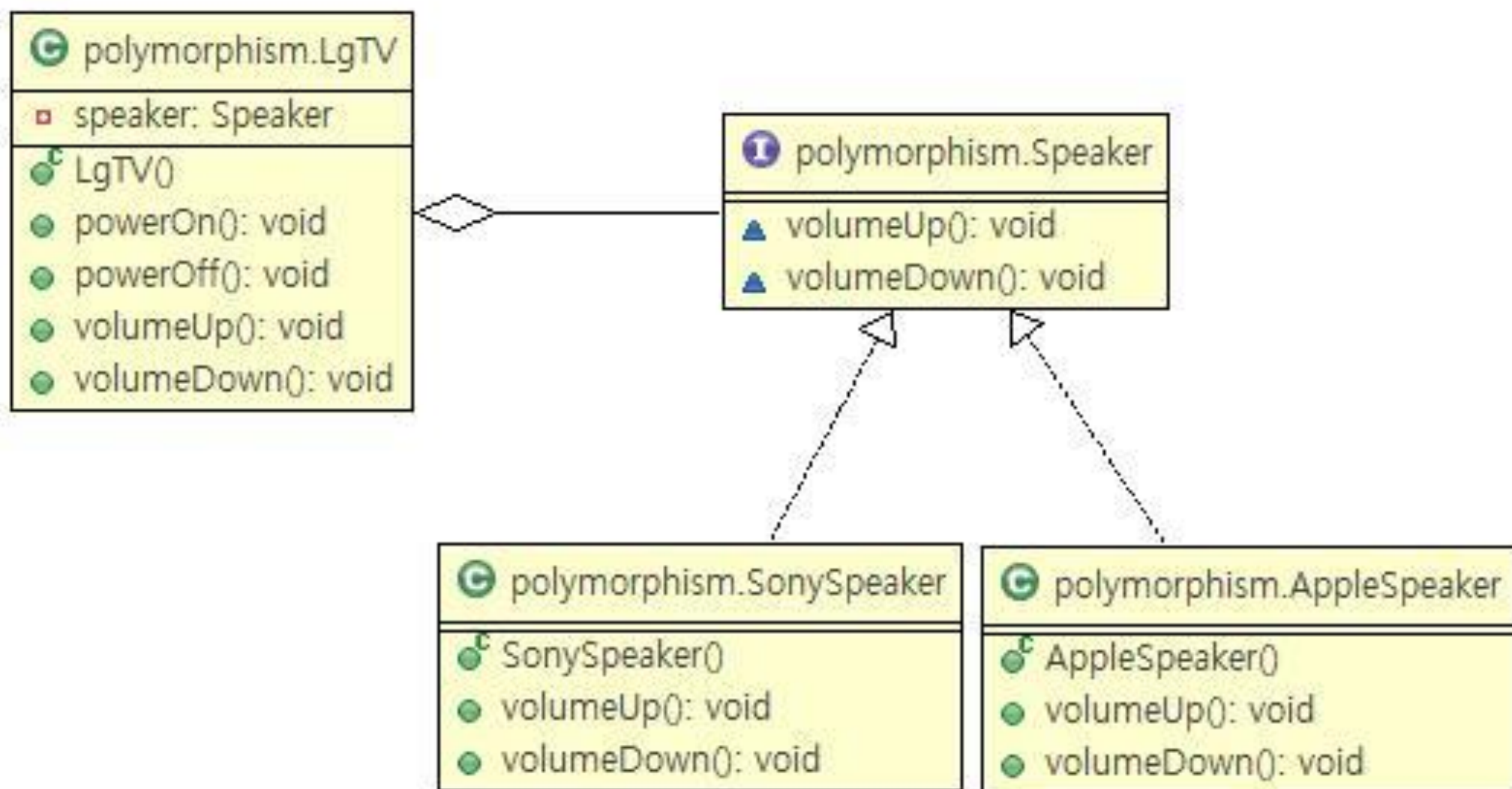
```
    public void volumeDown() {
```

```
        speaker.volumeDown();
```

```
    }
```

```
}
```

- @Qualifier



- @Qualifier

- 의존성 주입 대상 객체가 두 개 이상일 때 에러 발생.

```
@Component("tv")  
public class LgTV implements TV {
```

```
    @Autowired
```

```
    @Qualifier("apple")
```

```
    private Speaker speaker;
```

```
    public LgTV() {
```

```
        System.out.println("===> LgTV 객체 생성됨");
```

```
    }
```

```
    ~생략~
```

```
}
```

Speaker 타입의 객체 중  
아이디가 "apple"인 객체 할당

- @Resource

- @Resource는 객체의 이름을 이용하여 의존성 주입을 처리

```
@Component("tv")
public class LgTV implements TV {
    @Resource(name="apple")
    private Speaker speaker;

    public LgTV() {
        System.out.println("==> LgTV 객체 생성됨");
    }
    ~생략~
}
```

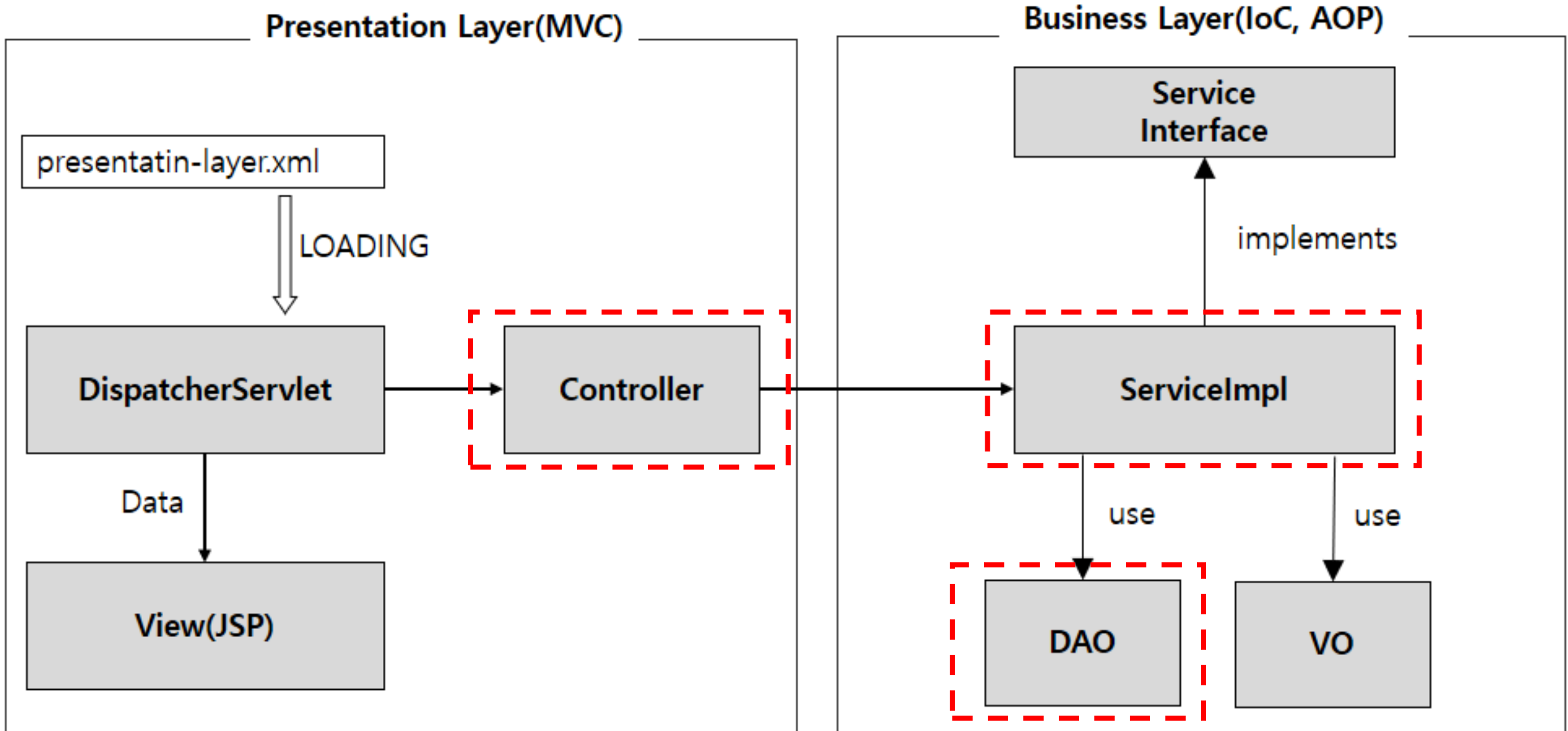
Speaker 타입의 객체 중  
아이디가 "apple"인 객체 할당

- Annotation VS. <bean> 등록

- 유지 보수 과정에서 자주 변경되는 객체는 <bean> 등록으로 처리한다.
- 유지 보수 과정에서 자주 변경되지 않는 객체는 Annotation으로 처리한다.
- 의존성 주입은 Annotation으로 처리한다.



- Layered Architecture

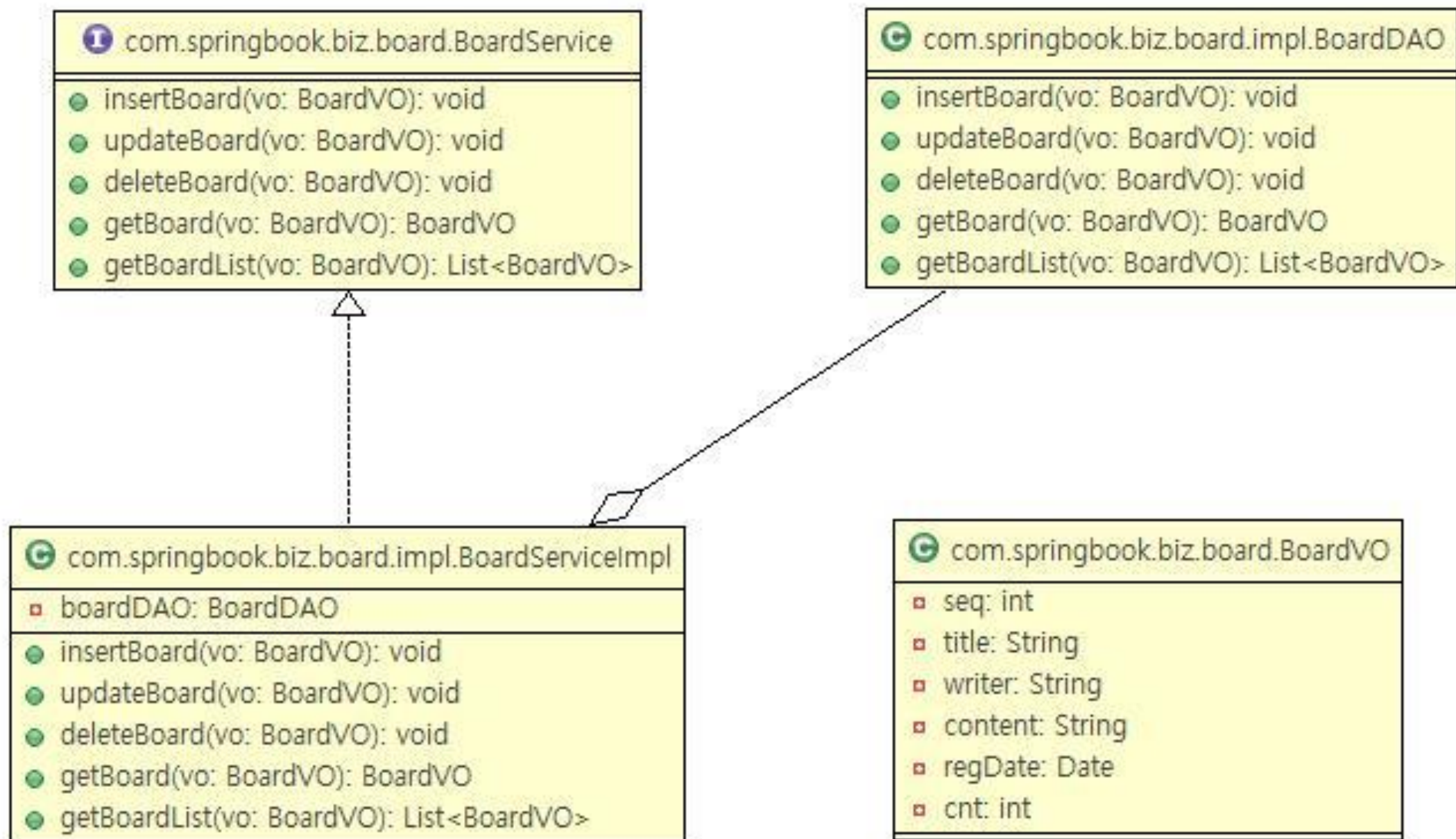


- Layer 별 Annotation

어노테이션	위치	의미
<b>@Service</b>	XXXServiceImpl	비즈니스 로직을 처리하는 Service 클래스
<b>@Repository</b>	XXXDAO	데이터베이스 연동을 처리하는 DAO 클래스
<b>@Controller</b>	XXXController	사용자 요청을 제어하는 Controller 클래스

Spring IoC 실습

## • BoardService 컴포넌트



- 실습 순서

- 1. BoardVO 클래스 작성
- 2. BoardDAO 클래스 작성
- 3. BoardService 인터페이스 작성
- 4. BoardServiceImpl 클래스 작성
- 5. Spring 설정 파일(applicationContext.xml) 작성
- 6. BoardServiceClient 작성 및 테스트