

2. DI / IoC

1. IoC란
2. DI(Dependency Injection)
3. 애노테이션 기반 IoC 설정
4. Java 기반 설정

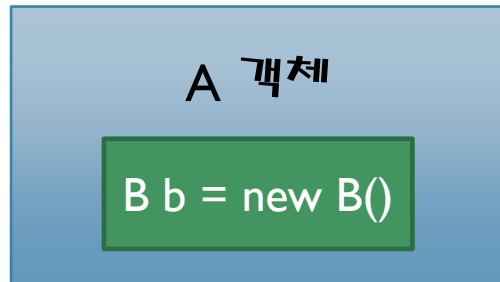
학습 목표

- DI 는 프로그램 부분적인 전환(교체)를 쉽게 만드는 것
 - 클래스의 소스를 변경하지 않고 구현클래스를 전환
 - 인터페이스 구현
- DI 애노테이션
 - @Controller, @Service, @Repository, @Component, @Bean

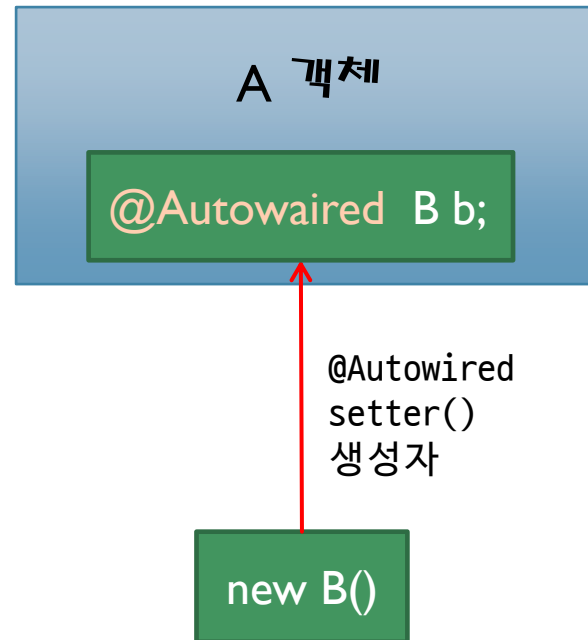
I.I IoC란

- Inversion of Control
- 제어의 역전이란 객체의 생성, 관리에 대한 제어권이 바뀜

필요한 객체는 직접 생성

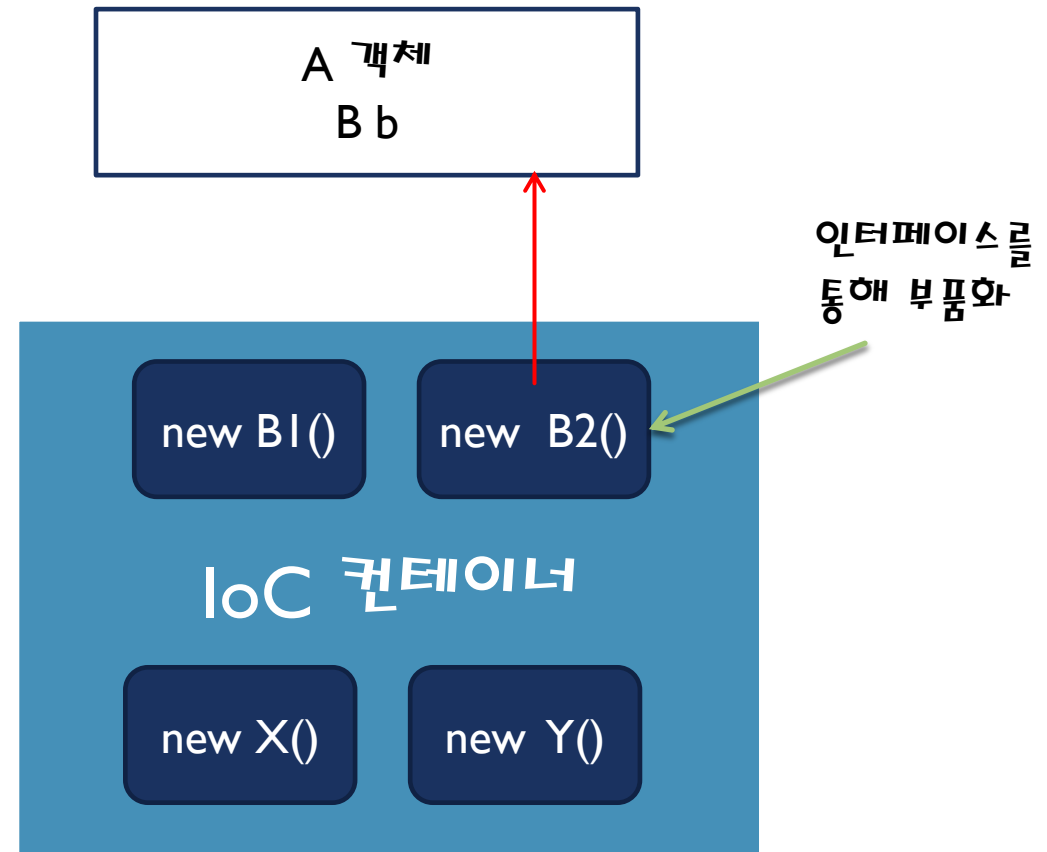


IoC : 필요한 객체를 주입받음



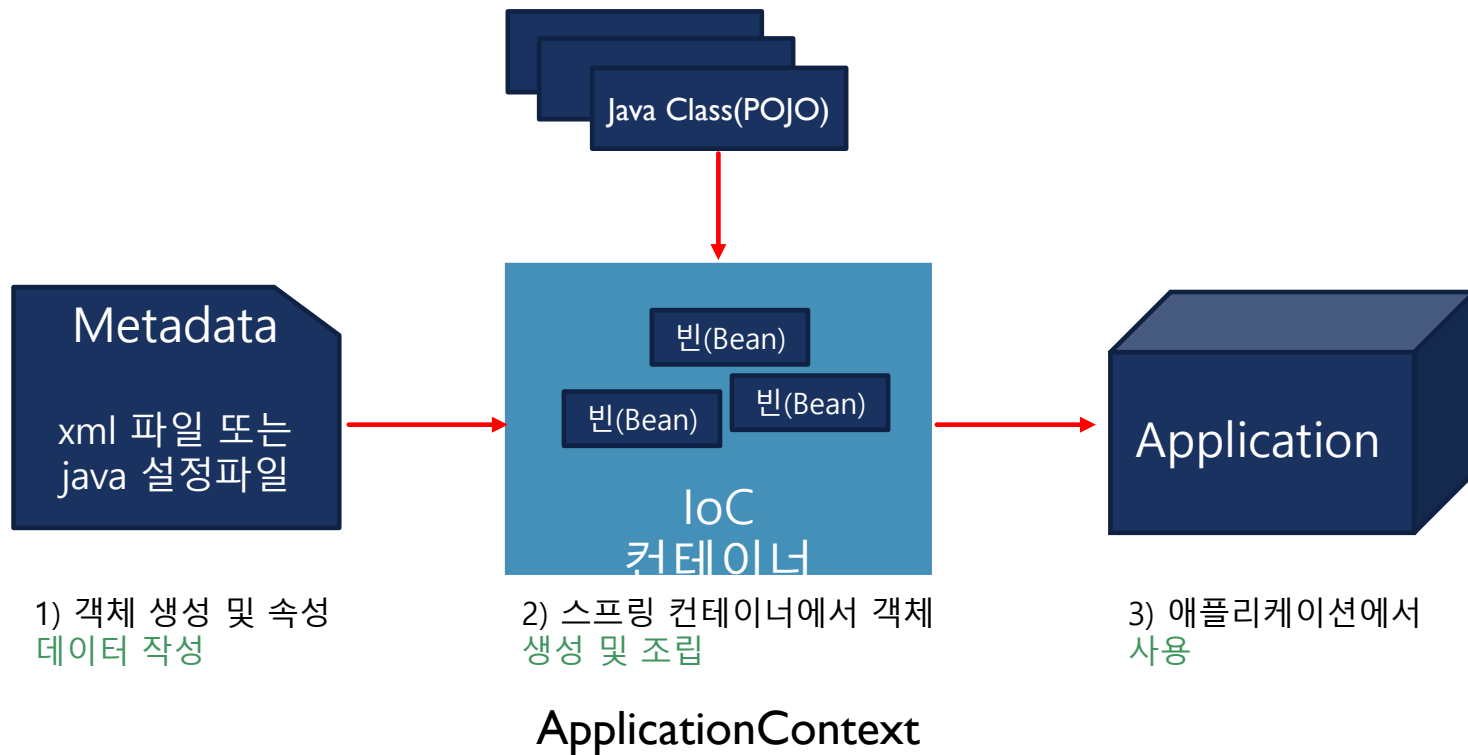
1.2 IoC 컨테이너

- 컨테이너에서 객체를 생성하고 조립(주입)해주는 역할
- 컨테이너가 관리하는 객체를 빈(Bean)이라고 함



1.3 스프링 IoC 컨테이너

- 스프링은 부품을 생성하고 조립하는 라이브러리 집합체 = IoC 컨테이너 = ApplicationContext



4.2 스프링 IoC 컨테이너 실습

- spring 설정파일
 - File -> New -> Spring Bean Configuration File
 - filename : applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="tv" class="co.spring.Tv"/>

</beans>
```

```
public class Tv {
    public void on() {
        System.out.println("on");
    }
}
```

4.2 스프링 IoC 컨테이너 실습

■ 스프링 IoC 컨테이너 실습

```
public class Test {  
    public static void main(String[] args) {  
        GenericXmlApplicationContext ctx =  
            new GenericXmlApplicationContext("classpath:applicationContext.xml");  
        Tv tv = ctx.getBean(Tv.class);  
        tv.on();  
    }  
}
```

```
Tv tv = new Tv();
```

I.4 ApplicationContext

▶ ApplicationContext API

<https://docs.spring.io/spring/docs/4.3.23.RELEASE/javadoc-api/>

The screenshot shows the Javadoc API for the `org.springframework.context` package. The left sidebar lists interfaces and classes. The main content area shows the 'All Known Implementing Classes' list. Annotations are added to the image to explain the role of the ApplicationContext:

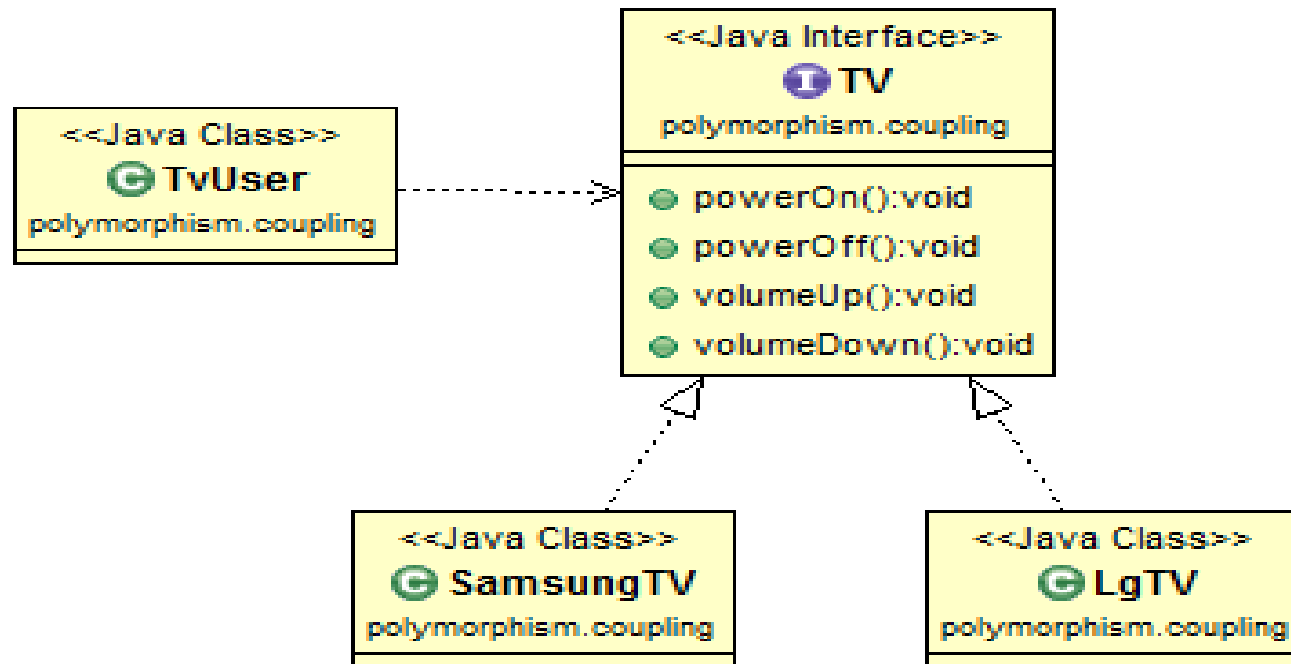
- 컨테이너 (Container):**
 - 즉시 로딩(pre Loading)
 - 빈 객체 관리 기능외 에도 트랜잭션 관리, 다국어 처리 등을 지원
- Bean Factory:**
 - 지연 로딩(Lazy Loading)

Additional annotations with arrows pointing to specific classes in the Javadoc list:

- 파일시스템이나 클래스 경로에 있는 XML 설정 파일을 로딩하여 구동하는 컨테이너** (Container that loads XML configuration files from the file system or classpath to run).
- 웹 기반의 스프링 애플리케이션을 개발할 때 사용. 직접 생성하지 않음** (Used for developing web-based Spring applications. Do not create directly).

1.3 결합도 낮추기

- 다형성 이용하기 - 인터페이스



결합도 낮추기

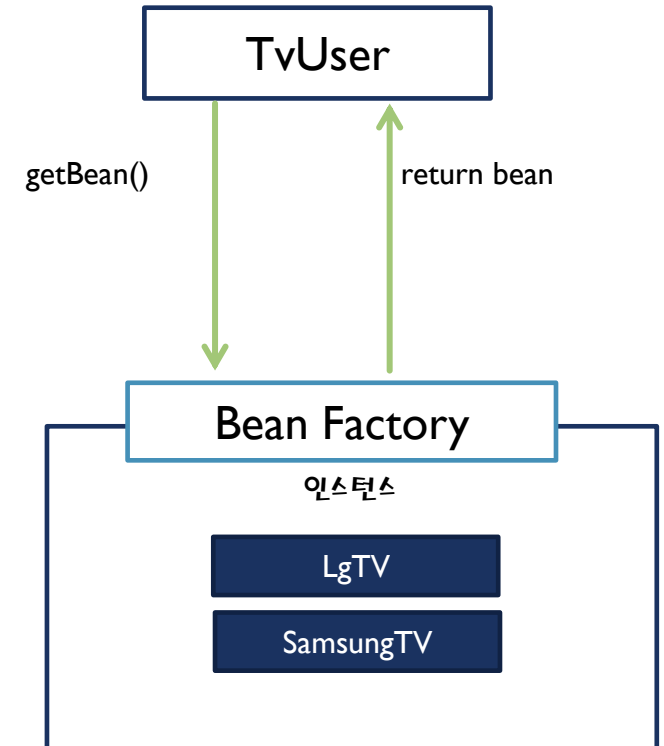
■ 디자인 패턴 이용 - Factory 패턴

클라이언트(TvUser)

```
public class TVUser {
    public static void main(String[] args) {
        TV tv = (TV)BeanFactory.getBean(args[0]);
        tv.powerOn();
        tv.volumeUp();
        tv.powerOff();
    }
}
```

Bean Factory

```
public class BeanFactory {
    public static Object getBean(String beanName) {
        if ( beanName.equals("samsung") ) {
            return new SamsungTV();
        } else if ( beanName.equals("lg") ) {
            return new LgTV();
        }
        return null;
    }
}
```



XML을 이용한 DI 설정

■ ApplicationContext

클라이언트(TvUser)

```
public class TVUser {
    public static void main(String[] args) {
        //spring 컨테이너 구동
        AbstractApplicationContext factory =
            new GenericXmlApplicationContext("applicationContext.xml");

        //객체 요청
        TV tv = (TV)factory.getBean("tv");

        tv.powerOn();
        tv.volumeUp();
        tv.powerOff();

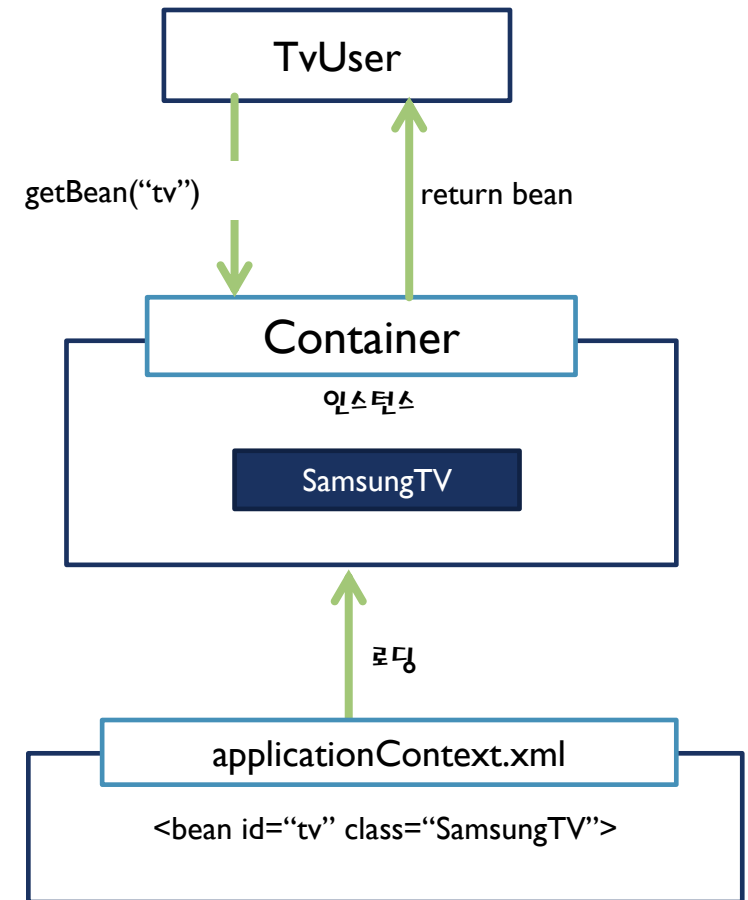
        //컨테이너 종료
        factory.close();
    }
}
```

스프링 설정파일(applicationContext.xml)

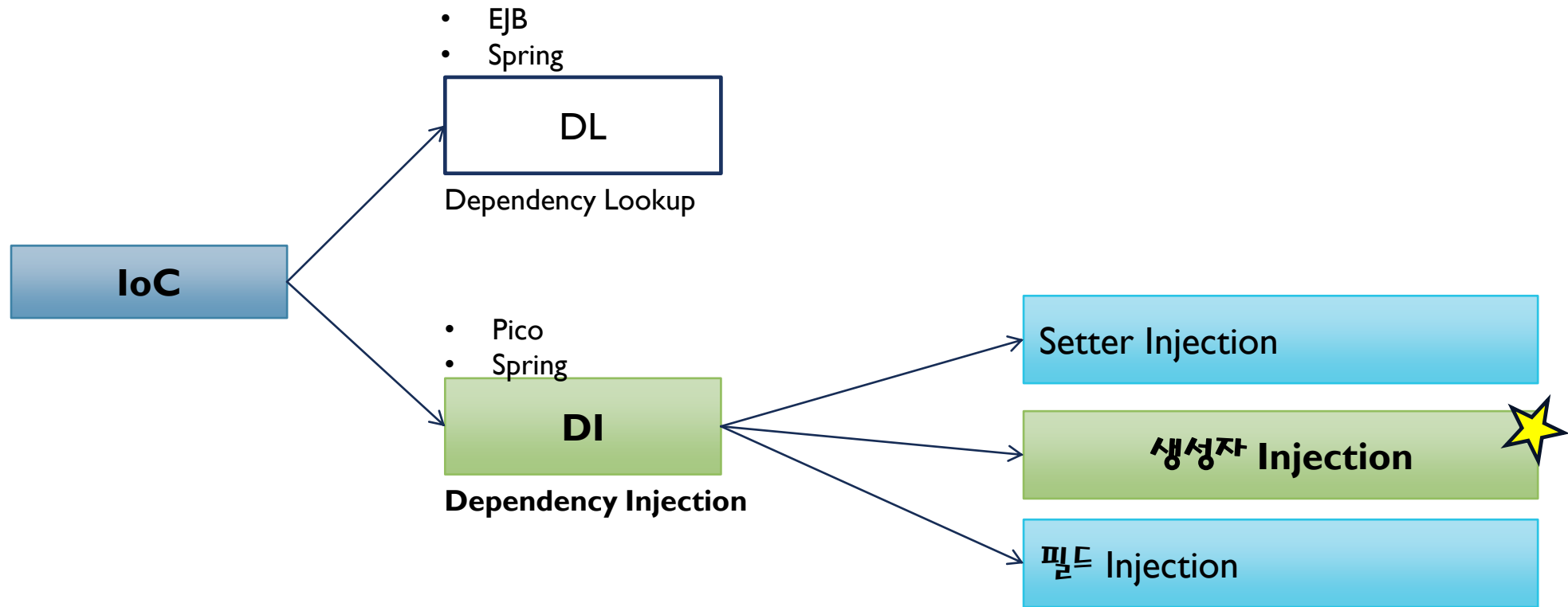
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"

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

</beans>
```



의존성 관리



★ 생성자 주입 방식을 권장

1. 의존관계 설정은 단 한번만 일어남. (setter 주입의 경우 항상 변경 가능성이 있음)
2. final로 선언된 필드에 의존성을 주입하는 코드가 누락되면 컴파일 에러

2.1 DI(Dependency Injection) : xml 기반

- 객체 사이의 의존 관계를 스프링 설정 파일에 등록된 정보를 바탕으로 컨테이너가 자동으로 처리
- 의존관계를 변경하고 싶을 때 프로그램 코드 변경 없이 스프링 설정 파일 수정만으로 변경사항 적용

■ 생성자 인젝션(Constructor Injection)

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

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

■ 세터 인젝션(Setter Injection)

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

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

2.2 DI(Dependency Injection)

- ApplicationContext에 빈 등록하기 (applicationContext.xml)

```
<bean id="speaker" class="polymorphism.SonySpeaker"
      init-method="initMethod"
      destroy-method="destroyMethod"
      lazy-init="true"
      scope="singleton"/>
```

<!-- 생성자 인젝션 방식 -->

```
<bean id="samsung" class="polymorphism.SamsungTV" >
  <constructor-arg ref="speaker" />
  <constructor-arg value="2000" />
</bean>
```

<!-- setter 인젝션 방식 -->

```
<bean id="samsung" class="polymorphism.SamsungTV">
  <property name="speaker" ref="speaker"></property>
  <property name="price" value="4000"></property>
</bean>
```

3.1 어노테이션 기반 설정

컴포넌트 스캔 설정

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

빈 등록

어노테이션	설명
@Component	클래스 선언부에 설정(빈 등록)
@Service	비즈니스 로직을 처리하는 service 클래스
@Repository	데이터베이스 연동을 처리하는 DAO 클래스
@Controller	사용자 요청을 처리하는 Controller 클래스

의존성 주입

어노테이션	설명
@Autowired	해당 타입의 객체를 찾아서 자동으로 할당
@Qualifier	특정 객체의 이름을 이용하여 의존성 주입
@inject	@Autowired와 동일한 기능 제공
@Resource	@Autowired 와 @Qualifier의 기능을 결합

3.1 어노테이션 기반 설정

컴포넌트 스캔 설정

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

빈 등록

```
@Component  
public class SonySpeaker implements Speaker {
```

의존성 주입

```
@Component  
public class SamsungTV implements TV {  
  
    @Autowired  
    private Speaker speaker;
```


3.1 어노테이션 기반 - 의존성 주입

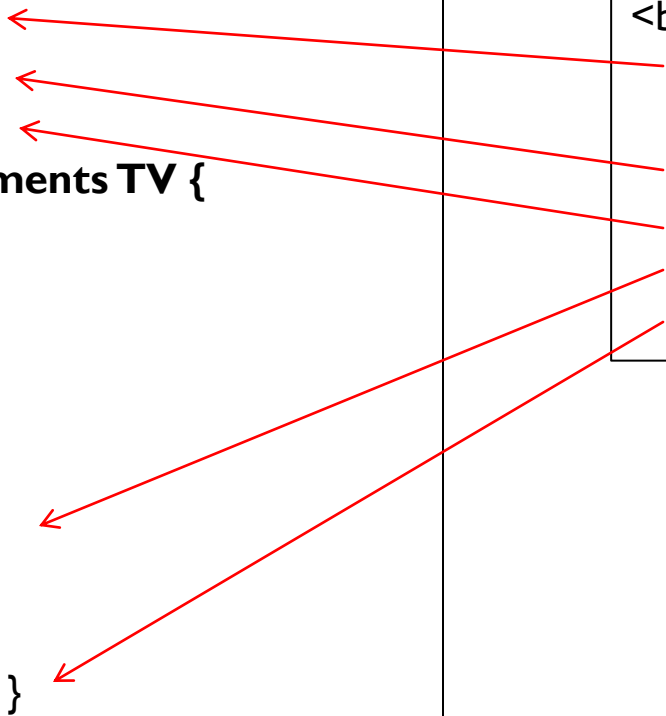
```
@Component("stv")
@Scope("singleton")
@Lazy("true")
public class SamsungTV implements TV {
```

```
@Autowired
private Speaker speaker;
private int price;
```

```
@PostConstruct
public void initMethod() { }
```

```
@PreDestroy
public void destroyMethod() { }
```

```
<bean
  id="speaker"
  class="polymorphism.SonySpeaker"
  scope="singleton"
  lazy-init="true"
  init-method="initMethod"
  destroy-method="destroyMethod"/>
```



3.2 의존성 주입

필드 주입

```
@Component
public class Restaurant {
    @Autowired private Chef chef;
```

생성자 주입

```
@Component
public class Restaurant {
    private Chef chef;
    public Restaurant(Chef chef) {
        this.chef = chef;
    }
```

setter 주입

```
@Component
public class Restaurant {
    private Chef chef;
    @Autowired
    public void setChef(Chef chef) {
        this.chef = chef;
    }
```

Lombok

```
@Data
@RequiredArgsConstructor
@Component
public class Restaurant {
    final private Chef chef;
}
```

```
@Data
@Component
public class Restaurant {
    @Setter(onMethod_ = {@Autowired} )
    private Chef chef;
}
```

3.2 의존성 주입 - 필드 주입 실습

Restaurant.java ×

```
1 package com.example.demo;
2
3 import org.springframework.stereotype.Component;
4 import lombok.Data;
5
6 @Component
7 @Data
8 public class Restaurant {
9
10 }
11
```

DITest.java ×

```
1 package com.example.demo;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
5 import org.junit.jupiter.api.Test;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.boot.test.context.SpringBootTest;
8
9 @SpringBootTest
10 public class DITest {
11
12     @Autowired
13     private Restaurant restaurant;
14
15     @Test
16     public void 의존성주입() {
17         assertThat(restaurant).isNotNull();
18     }
19 }
20
```

3.2 의존성 주입 - 생성자 주입 실습

```
Restaurant.java ×
1 package com.example.demo;
2
3 import org.springframework.stereotype.Component;
4
5 @Component
6 public class Restaurant {
7
8     private Chef chef;
9
10    public Restaurant(Chef chef) {
11        this.chef = chef;
12    }
13
14    public Chef getChef() { return chef; }
15
16    public void setChef(Chef chef) {
17        this.chef = chef;
18    }
19 }
20
```

```
Chef.java ×
1 package com.example.demo;
2
3 import org.springframework.stereotype.Component;
4
5 @Component
6 public class Chef {
7
8 }
```

```
DItest.java ×
1 package com.example.demo;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
5 import org.junit.jupiter.api.Test;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.boot.test.context.SpringBootTest;
8
9 @SpringBootTest
10 public class DItest {
11
12     @Autowired
13     private Restaurant restaurant;
14
15     @Test
16     public void 의존성주입() {
17         assertThat(restaurant.getChef()).isNotNull();
18     }
19 }
20
```

3.2 의존성 주입 - 생성자 주입 실습(Lombok)

```
Restaurant.java ×
1 package com.example.demo;
2
3 import org.springframework.stereotype.Component;
4 import lombok.Data;
5 import lombok.RequiredArgsConstructor;
6
7 @Component
8 @Data
9 @RequiredArgsConstructor
10 public class Restaurant {
11
12     final private Chef chef;
13 }
14
```

```
Chef.java ×
1 package com.example.demo;
2
3 import org.springframework.stereotype.Component;
4 import lombok.Data;
5
6 @Component
7 @Data
8 public class Chef {
9
10 }
```

```
DITest.java ×
1 package com.example.demo;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
5 import org.junit.jupiter.api.Test;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.boot.test.context.SpringBootTest;
8
9 @SpringBootTest
10 public class DITest {
11
12     @Autowired
13     private Restaurant restaurant;
14
15     @Test
16     public void 의존성주입() {
17
18         assertThat(restaurant.getChef()).isNotNull();
19     }
20 }
21
22
```

3.2 의존성 주입 - setter 주입 실습

```
Restaurant.java ×
1
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Component;
5
6 @Component
7 public class Restaurant {
8
9     private Chef chef;
10
11     public Chef getChef() {
12         return chef;
13     }
14
15     @Autowired
16     public void setChef(Chef chef) {
17         this.chef = chef;
18     }
19 }
```

```
Chef.java ×
1 package com.example.demo;
2
3 import org.springframework.stereotype.Component;
4
5 @Component
6 public class Chef {
7
8 }
```

```
DITest.java ×
1 package com.example.demo;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
5 import org.junit.jupiter.api.Test;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.boot.test.context.SpringBootTest;
8
9 @SpringBootTest
10 public class DITest {
11
12     @Autowired
13     private Restaurant restaurant;
14
15     @Test
16     public void 의존성주입() {
17         assertThat(restaurant.getChef()).isNotNull();
18     }
19 }
20 }
```

3.2 의존성 주입 - setter 주입 실험(Lombok)

```
Restaurant.java ×
1 package com.example.demo;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Component;
5
6 import lombok.Data;
7 import lombok.Setter;
8
9 @Data
10 @Component
11 public class Restaurant {
12
13     @Setter(onMethod_ = {@Autowired} )
14     private Chef chef;
15 }
16
```

```
Chef.java ×
1 package com.example.demo;
2
3 import org.springframework.stereotype.Component;
4
5 @Component
6 public class Chef {
7
8 }
```

```
DItest.java ×
1 package com.example.demo;
2
3 import static org.assertj.core.api.Assertions.assertThat;
4
5 import org.junit.jupiter.api.Test;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.boot.test.context.SpringBootTest;
8
9 @SpringBootTest
10 public class DItest {
11
12     @Autowired
13     private Restaurant restaurant;
14
15     @Test
16     public void 의존성주입() {
17         assertThat(restaurant.getChef()).isNotNull();
18     }
19 }
20
```

4. 자바기반의 DI 설정

```
@Configuration
public class JavaConfig {

    @Bean
    public Speaker spaker() {
        return new Speaker();
    }
}
```


5. configuration metadata

- Annotation-based configuration
 - spring 2.5부터
- XML-based configuration
 - 루트 엘리먼트 <beans/> 안에 <bean/> 엘리먼트를 이용하여 설정
- Java-based configuration
 - spring 3.0부터
 - @Configuration, @Bean