

Spring Framework

2강_DI(Dependency injection)

2-1. DI - IoC

2-2. 생성자, setter를 통한 DI

2-3. bean의 범위

3. 의존 객체 자동 주입

2-1: DI-IoC

Spring DI/IoC

JAVA 의 Class 상속 / Interface 를 이용한 추상화를 기반으로 하는 개발 방법.
Spring은 아래 DI/IoC 를 강력하게 지원하는 프레임워크.

IoC : Inversion of Control

프로그램을 제어하는 패턴 중 하나.

DI 는 IoC패턴의 구현방법 중 하나.

DI에 따라 프로그램의 흐름이 완전히 변경됨.

DI : Dependency Injection

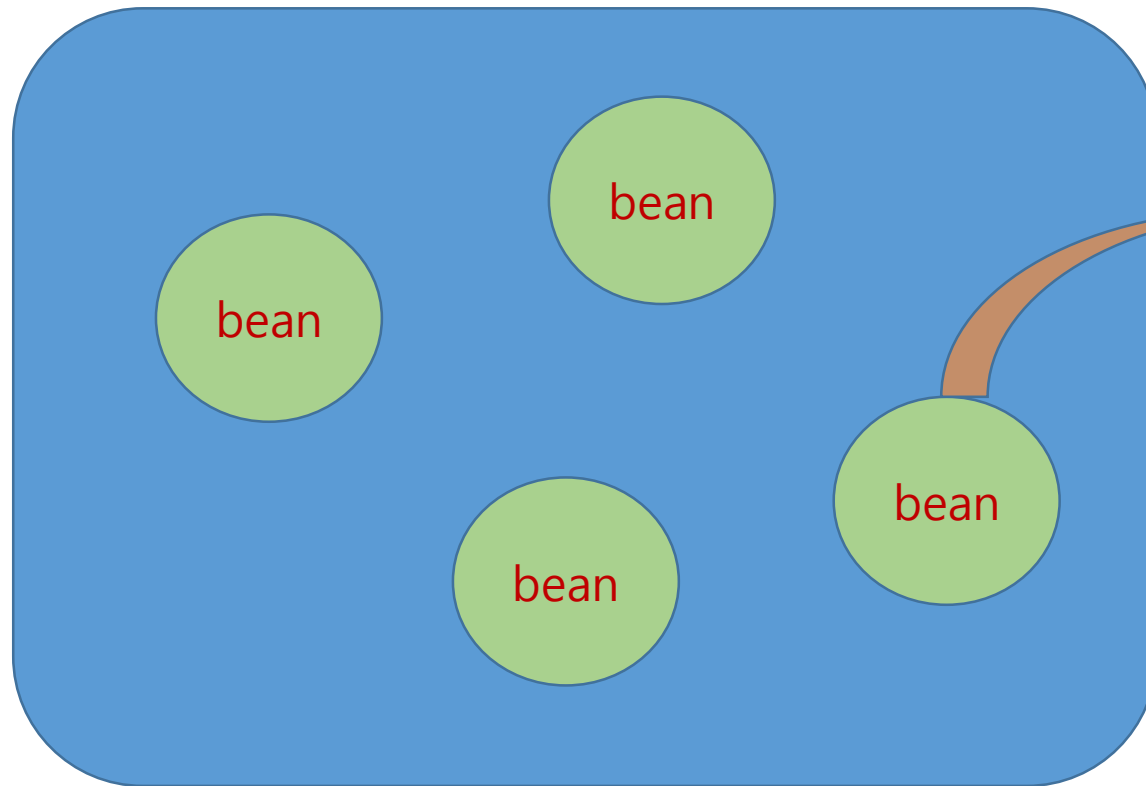
스프링 Container 에 만들어둔 각종 클래스(bean)들은 서로 의존적이다
A객체가 B객체 없이 동작이 불가능한 상황

스프링은 DI를 기준으로 많은 프레임워크모듈 들이 만들어짐.

Spring 은 DI Framework 혹은 IoC Framework 라고 부름.

1-2 :스프링 컨테이너(IoC)

스프링 IoC컨테이너



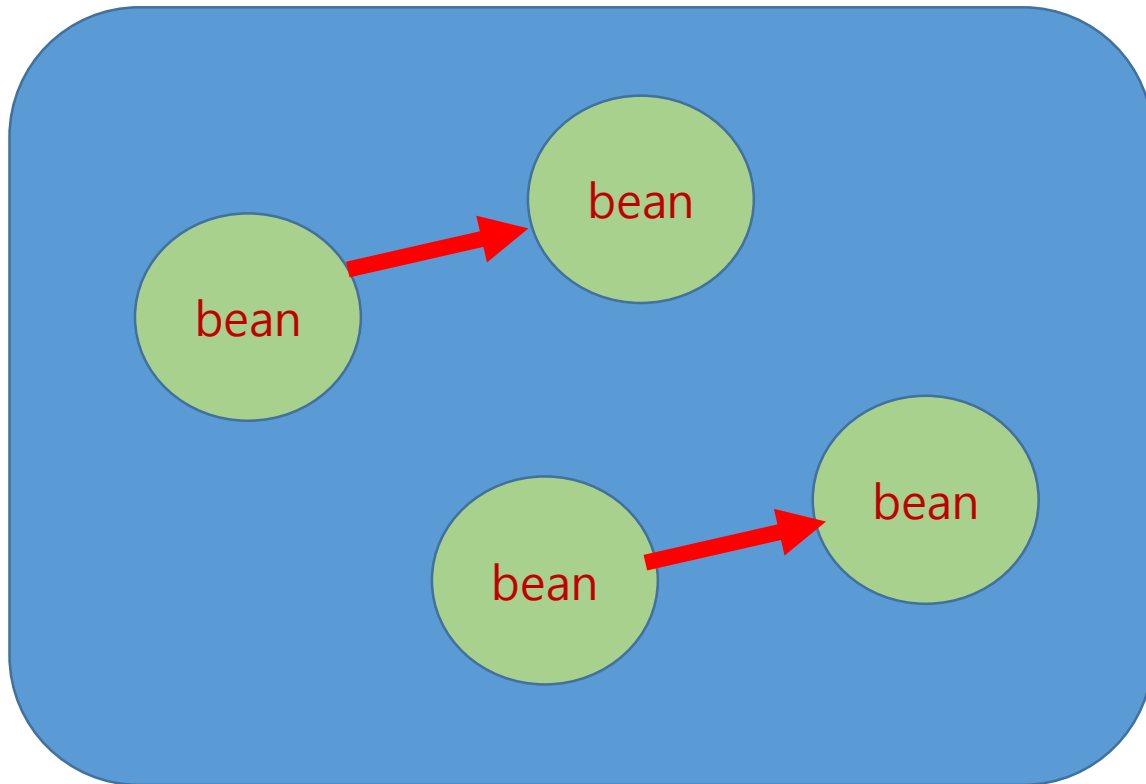
IoC : Inversion of Control

- 제어의 역전
- 객체를 필요할 때 생성해서 사용하던 방식을
- 미리 생성해 놓고 꺼내서 사용하는 형태로 변경

사용

2-1: DI란?

스프링 IoC컨테이너

**DI : Dependency Injection**

스프링 Container 에 만들어둔 각종 클래스(bean)들은 서로 **의존적**이다

ex) 의존적이다?

A클래스는 B클래스가 없으면 실행 할 수 없다
A는 B에 의존적이다

즉 **객체 안에 객체가 저장되는 형태**이다

2-1: DI란?

배터리에 의존해서 장난감을 만들었다. ➡ 배터리에 의존적이다.



배터리 일체형

배터리가 떨어지면
장난감을 새로 구입해야 한다.



배터리 분리형

배터리가 떨어지면
배터리만 교체하면 된다.



배터리 분리형

배터리가 떨어지면
배터리만 교체하면 된다.

DI- 프로그래밍에서 객체를 만들어서 외부에서 따로 주입하는 방법

2-1: DI란?

```
public class CarToy {  
    private Battery battery;  
  
    //생성자  
    public CarToy() {  
        this.battery = new Battery();  
    }  
}
```

배터리 일체형



배터리가 떨어지면
장난감을 새로 구입해야 한다.

```
public class CarToy {  
    private Battery battery;  
  
    //생성자  
    public CarToy() {  
        this.battery = new Battery();  
    }  
    //setter  
    public void setBattery(Battery battery) {  
        this.battery = battery;  
    }  
}
```

배터리 분리형



배터리가 떨어지면
배터리만 교체하면 된다.

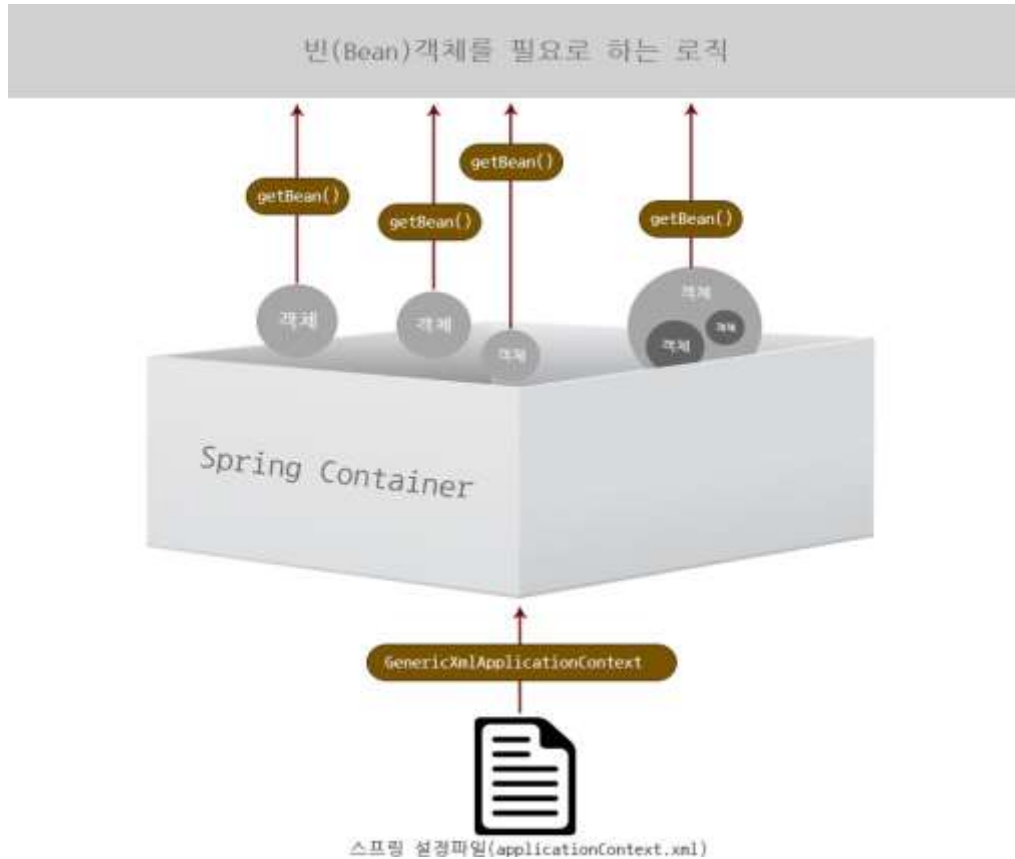
```
public class CarToy {  
    private Battery battery;  
  
    //생성자  
    public CarToy() {  
        this.battery = new Battery();  
    }  
    //setter  
    public void setBattery(Battery battery) {  
        this.battery = battery;  
    }  
    //getter  
    public Battery getBattery() {  
        return battery;  
    }  
}
```

배터리 탈부착형



배터리가 떨어지면
배터리만 교체하면 된다.

2-1: DI 설정 방법



스프링 컨테이너 생성 및 빈(Bean)객체 호출 과정

2가지 의존성 주입 방법

1. 생성자를 통한 의존성 주입

```
<constructor-arg ref="빈id"></constructor-arg>
```

1. setter를 통한 의존성 주입

```
<property name="변수명" value="값"/>
```

```
<property name="변수명" ref="객체"/>
```

XML 설정

2-1: DI 설정 방법

스프링 설정파일에 추가

```
<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">
```

`xmlns` - 속성값은 네임스페이스로 태그를 식별하기 이름

`xmlns:xsi` - XML정보를 가르키는 주소

`xsi:schemaLocation` - 두개의 값이 공백으로 구분 됨
첫번째는 사용할 네임스페이스
두번째는 참조할 네임스페이스 위치

2-2: ★생성자 를 통한 의존객체 주입

```
<?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">
```

```
<!-- day02 -->
```

```
<bean id="chef" class="day02.ex01.construct.Chef" />
```

```
<!-- 생성자 주입 -->
```

```
<bean id="hotel" class="day02.ex01.construct.Hotel">
    <constructor-arg ref="chef"></constructor-arg>
</bean>
```

```
public Hotel(Chef chef) {
    this.chef = chef;
}
```

코드해석

Hotel클래스를 hotel이름으로 빈생성

생성자 인자값으로 ref="chef" 로 생성된 빈 참조

2-2: ★setter 를 통한 의존객체 주입

```
public void setUrl(String url) {  
    this.url = url;  
}  
public void setUid(String uid) {  
    this.uid = uid;  
}  
public void setUpw(String upw) {  
    this.upw = upw;  
}
```

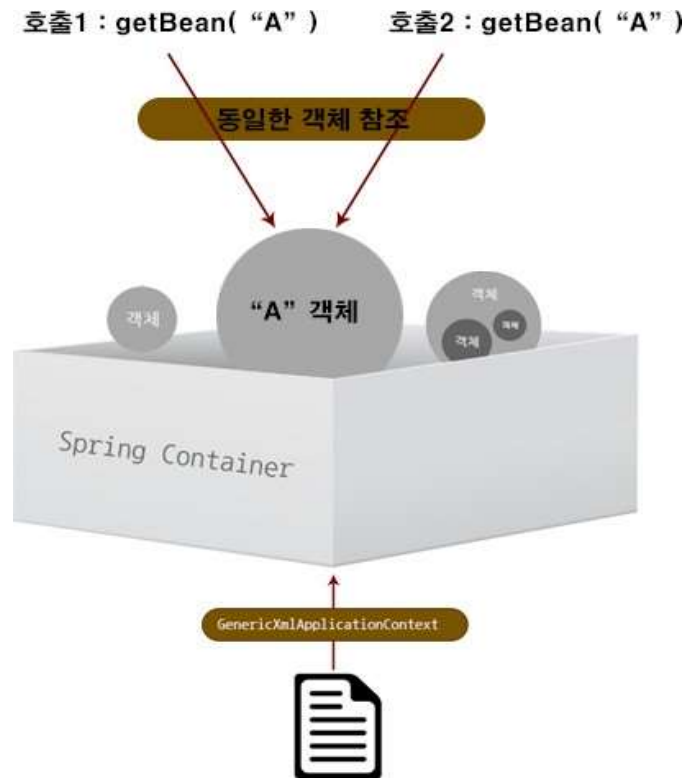


```
<bean id="DBdev" class="day02.ex02.setter.DatabaseDev">  
    <property name="url" value="jdbc:mysql://localhost:3306/test"/>  
    <property name="uid" value="jsp"/>  
    <property name="upw" value="jsp"/>  
  
</bean>
```

2-3 : 빈(Bean)의 범위

싱글톤(Singleton)

스프링 컨테이너에서 생성된 빈(Bean)객체의 경우 동일한 타입에 대해서는 기본적으로 한 개만 생성이 되며, `getBean()` 메소드로 호출될 때 동일한 객체가 반환 된다.



프로토타입(Prototype)

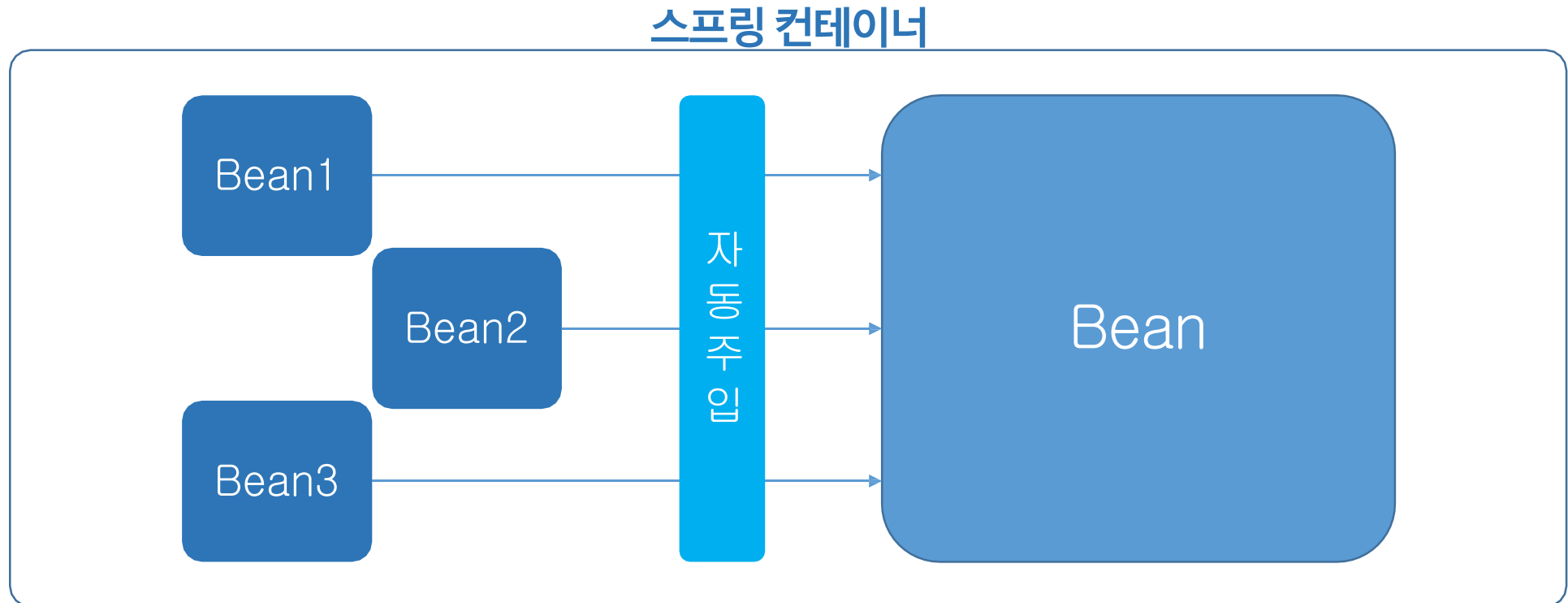
싱글톤 범위와 반대의 개념도 있는데 이를 프로토타입(Prototype) 범위라고 한다. 프로토타입의 경우 개발자는 별도로 설정을 해줘야 하는데, 스프링 설정 파일에서 빈 (Bean)객체를 정의할 때 `scope`속성을 명시해 주면 된다.

```
<bean id="good" class="day01.SpringTest"
scope="prototype"/>
```

3: 의존객체 자동 주입이란?

의존 객체 자동 주입이란?

스프링 설정 파일에서 의존 객체를 주입할 때 `<constructor-arg>` 또는 `<property>` 태그로 의존 대상 객체를 명시하지 않아도 스프링 컨테이너가 자동으로 필요한 의존 대상 객체를 찾아서 의존 대상 객체가 필요한 객체에 주입해 주는 기능이다. 구현 방법은 `@Autowired`와 `@Resource` 어노테이션을 이용해서 쉽게 구현할 수 있다.



2-1: DI 자동 주입 설정 방법

스프링 설정파일 에 추가

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
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-4.3.xsd">
```

`xmlns` - 속성값은 네임스페이스로 태그를 식별하기 이름

`xmlns:xsi` - XML정보를 가르키는 주소

`xsi:schemaLocation` - 두개의 값이 공백으로 구분 됨
첫번째는 사용할 네임스페이스
두번째는 참조할 네임스페이스 위치

3: 의존객체 자동 주입 태그

@Autowired (required = false) => ,

타입을 기준으로 의존성을 주입,
같은 타입 빈이 두 개 이상 있을 경우 변수이름으로 빈을 찾음
Spring 아노테이션

@Qualifier

빈의 이름으로 의존성 주입
@Autowired와 같이 사용
Spring 아노테이션

@Resource ✕ ->

name을 속성을 이용하여 빈의 이름을 직접 지정
JavaSE의 아노테이션 (JDK9에는 포함 안되 있음)

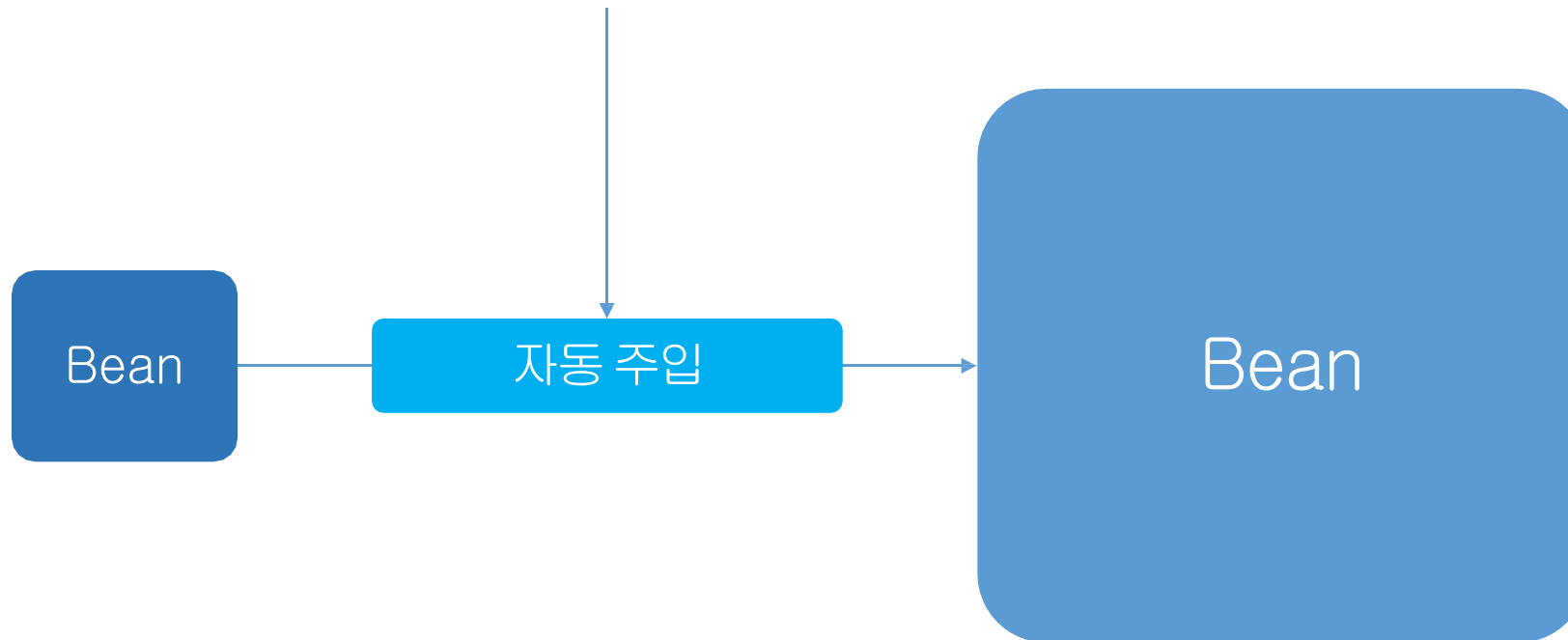
@Inject ✕

@Autowired 아노테이션을 사용하는 것과 같다
JavaSE의 아노테이션

3 :@Autowired

-속성값, 세터, 생성자 적용가능

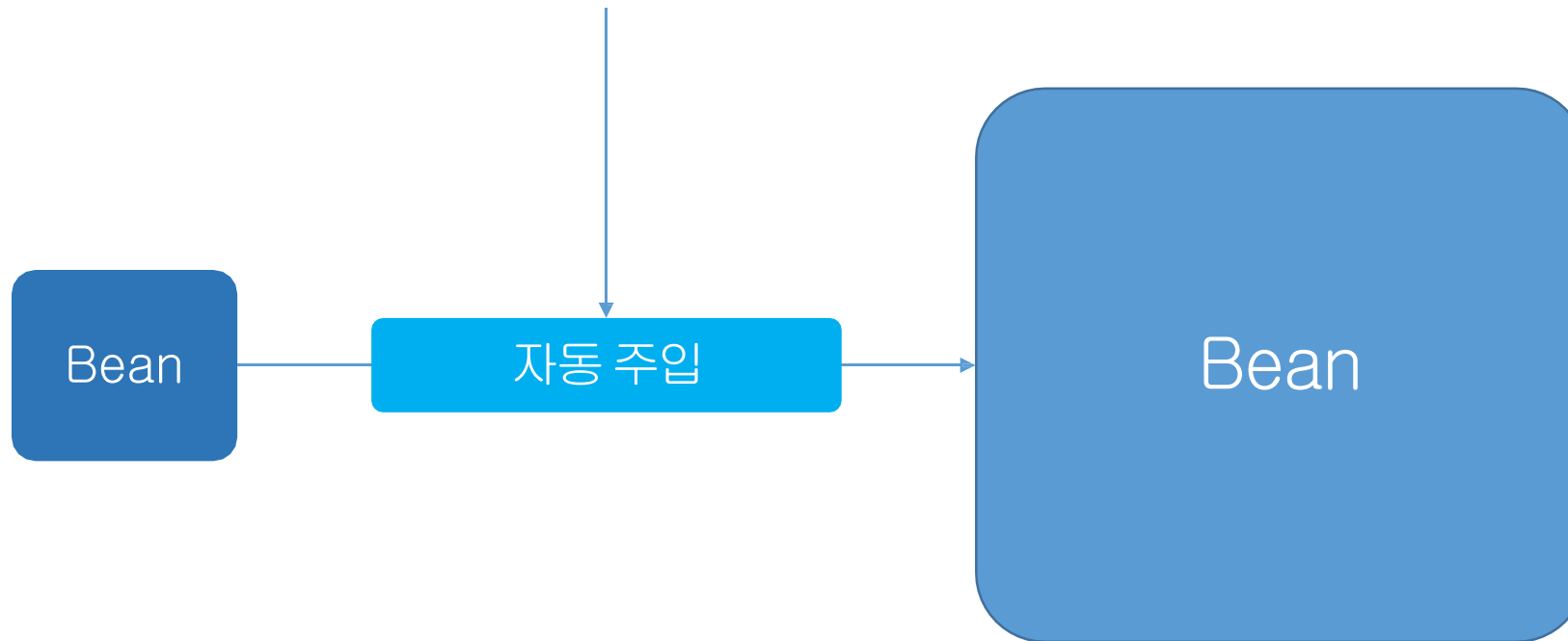
주입하려고 하는 객체의 타입이 일치하는 객체를 자동으로 주입한다.



3 :@Resource

-속성값, 세터 적용가능

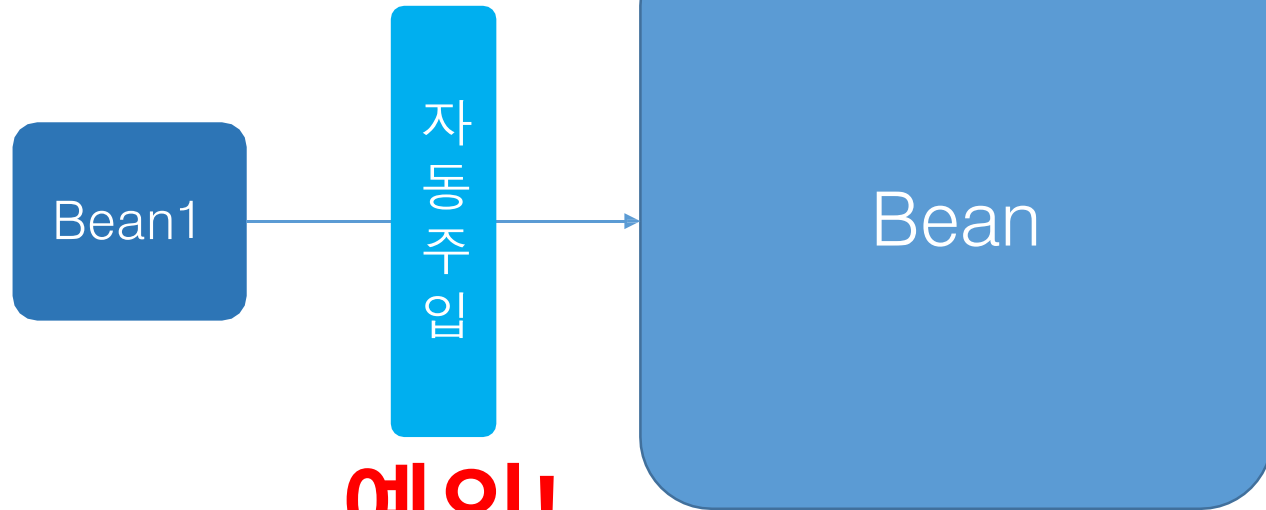
주입하려고 하는 객체의 이름이 일치하는 객체를 자동으로 주입한다.



3: :@Qualifier

- 모호한 bean의 강제 연결

스프링 컨테이너



예외!

```
<bean id="wordDao" class="com.word.dao.WordDao" >  
  <qualifier value="usedDao" />  
</bean>  
<bean id="wordDao2" class="com.word.dao.WordDao" />  
<bean id="wordDao3" class="com.word.dao.WordDao" />
```

```
@Autowired  
@Qualifier("usedDao")  
private WordDao wordDao;
```

동일한 객체가 2개 이상인 경우 스프링 컨테이너는
자동 주입 대상 객체를 판단하지 못해서 Exception을 발생시킨다.

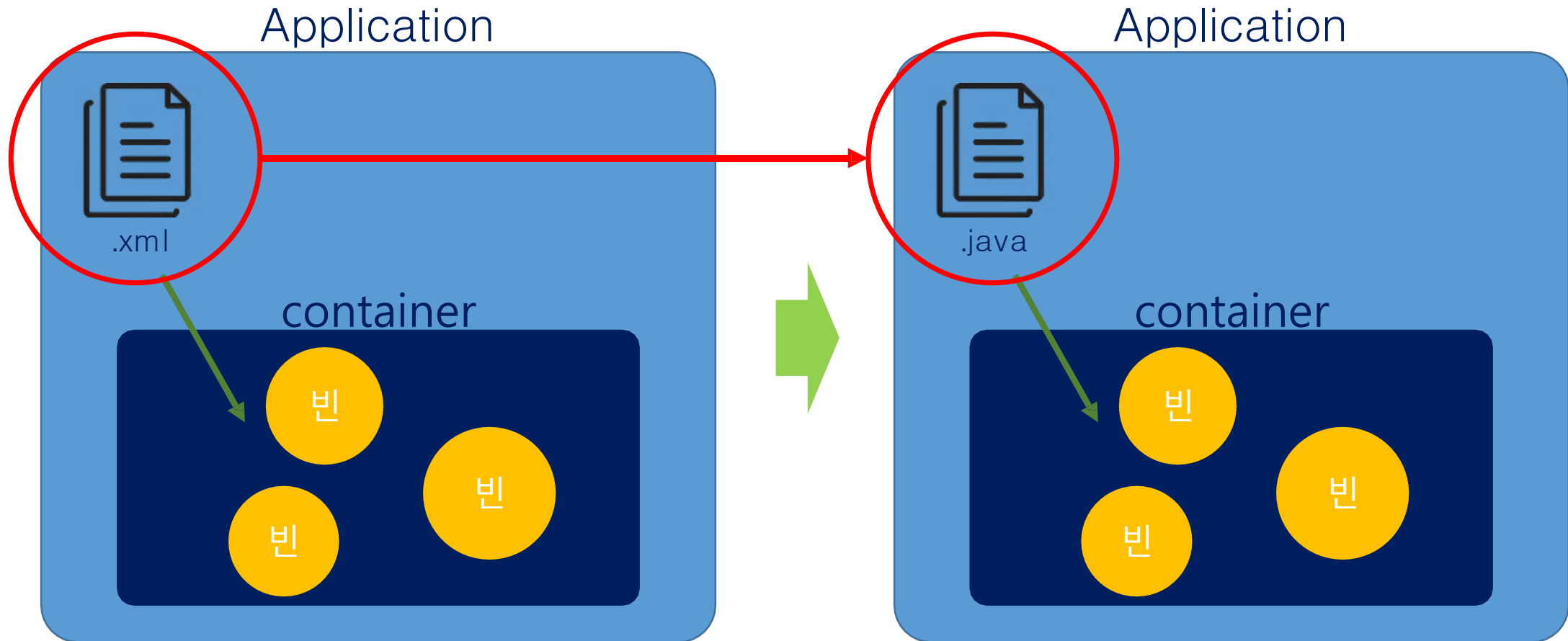
3. 빈 생성과 의존성 주입 비교

	XML 설정파일		Annotation
빈 생성	<ul style="list-style-type: none"> <code><bean id="빈이름" class="패키지명.클래스명" /></code> 		<ul style="list-style-type: none"> 설정파일에 컴포넌트 스캔 태그 추가 <code><context:component-scan base-package="패키지명"/></code> 자바클래스 위에 <code>@Controller</code>, <code>@Component</code>, <code>@Service</code>, <code>@Repository</code> 아노테이션 중에서 하나 선언 빈 이름은 클래스 이름에서 첫 문자만 소문자로 바뀐 이름으로 지정됨
의존성 주입	생성자	<ul style="list-style-type: none"> 자바클래스에 생성자 추가 	<ul style="list-style-type: none"> 자바 클래스 필드, 생성자, setter 메서드 위에 <code>@Autowired</code> 또는 <code>@Inject</code> 아노테이션 중 하나 선언(타입 기준으로 의존성 주입) 인터페이스를 구현한 클래스가 두 개 이상이면 <code>@Autowired</code> 아래에 <code>@Qualifier("빈이름")</code>을 추가하거나 <code>@Resource(name="빈이름")</code>으로 선언
		<ul style="list-style-type: none"> <code><constructor-arg name="변수명" ref="빈이름" /></code> 	
	setter	<ul style="list-style-type: none"> 자바클래스에 setter 메서드 추가 	
		<ul style="list-style-type: none"> <code><property name="변수명" ref="빈이름" /></code> 	

3 : XML파일을 Java파일로 변경하기

@Configuration – 스프링 컨테이너를 대신 생성하는 어노테이션

@Bean – 빈으로 등록하는 어노테이션



3 : XML파일을 Java파일로 변경하기 예제

```
@Configuration
public class JavaConfig {
    //<bean id="good" class="test01.SpringTest"/>
    @Bean
    public SpringTest good() {
        return new SpringTest();
    }
    //<bean id="chef" class="day02.ex01.construct.Chef" />
    @Bean
    public Chef chef() {
        return new Chef();
    }
    //<bean id="hotel" class="day02.ex01.construct.Hotel">
    @Bean
    public Hotel hotel() {
        //Hotel은 생성자로 Chef객체를 받기 때문에 매개값으로 chef()함수를 주입합니다.
        return new Hotel(chef());
    }
    @Bean
    public DatabaseDev DBdev() {
        //setter를 통해 값을 받고 있기 때문에 객체를 생성하고 세터 지정후 반환합니다.
        DatabaseDev dv = new DatabaseDev();
        dv.setUrl("jdbc:mysql://localhost:3306/test");
        dv.setUid("jsp");
        dv.setUpw("jsp");
        return dv;
    }
}
```

```
public static void main(String[] args) {

    AnnotationConfigApplicationContext ctx =
        new AnnotationConfigApplicationContext(JavaConfig.class);

    SpringTest t = ctx.getBean("good", SpringTest.class);
    t.method1();
    t.method2();

    Hotel h = ctx.getBean("hotel", Hotel.class);
    h.getChef().cook();

}
```