장 희 정

Spring IoC & DI

IoC(Inversion of Control)의 개념

- IoC(제어의 역전)이란, 객체의 생성, 생명주기의 관리까지 모든 객체에 대한 제어권이 바뀌었다는 것을 의미

- 컴포넌트의 의존관계 결정(Component dependency resolution), 설정(configuration) 및 생명주기(lifecycle)를 해결하기 위한 디자인 패턴(Design Pattern)

IoC 컨테이너

스프링 프레임워크도 객체에 대한 생성 및 생명주기를 관리 할수 있는 기능을 제공하고 있다. 즉, IoC 컨테이너 기능을 제공한다.

: IoC 컨테이너는 객체의 생성을 책임지고 의존성을 관리한다.

: POJO의 생성, 초기화, 서비스, 소멸에 대한권한을 가진다.

: 개발자들이 직접 POJO를 생성할 수 있지만 컨테이너에게 맡긴다.

DI(Dependency Injection)

- 각 클래스간의 의존관계를 빈 설정(Bean Definition) 정보를 바탕으로 컨테이너가 자동으로 연결해주는 것을 말함.

-개발자들은 단지 빈 설정 파일에서 의존관계가 필요하다는 정보를 추가하면 된다.

-객체 레퍼런스를 컨테이너로 부터 주입 받아서, 실행 시에 동적으로 의존관계가 생성된다.

-컨테어니가 흐름의 주체가 되어 애플리케이션 코드에 의존관계를 주입해주는 것이다.

* 장점 : 코드가 단순해지고 컴포넌트 간의 결합도가 느슨해진다.

DI 유형

- Setter Injection

  : 의존성을 입력 받는 setter 메소드를 만들고 이를 통해서 의존성을 주입

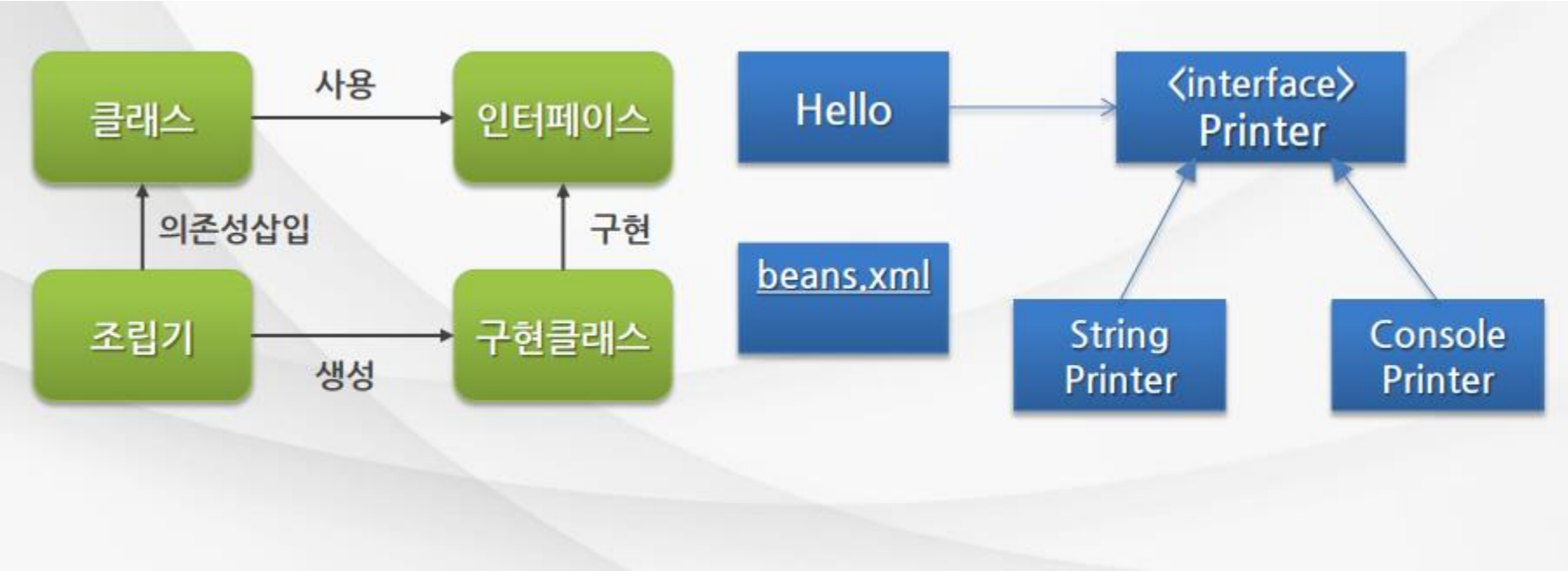
-Construction Injection

  : 필요한 의존성을 포함하는 클래스의 생성자를 만들고 이를 통해 의존성을 주입

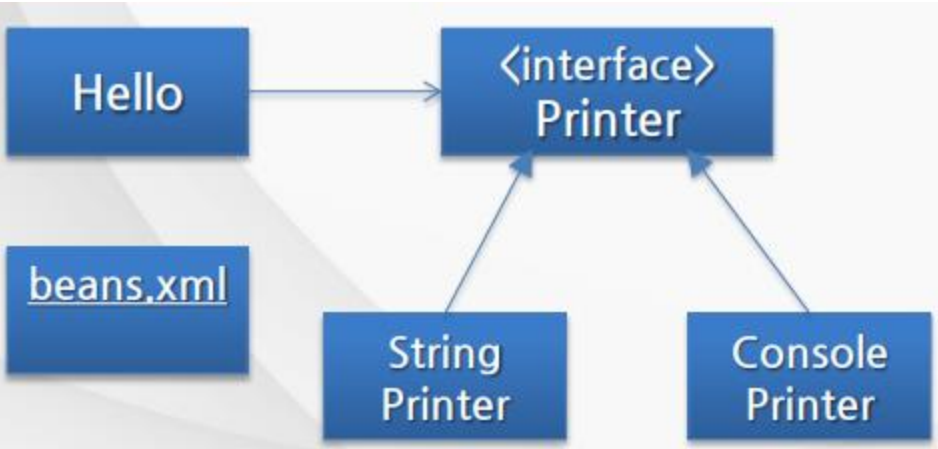
-Method Injection

  : 의존성을 입력 받는 일반 메소드를 만들고 이를 통해서 의존성을 주입

DI를 이용한 클래스 호출 방식



DI – Setter Injection



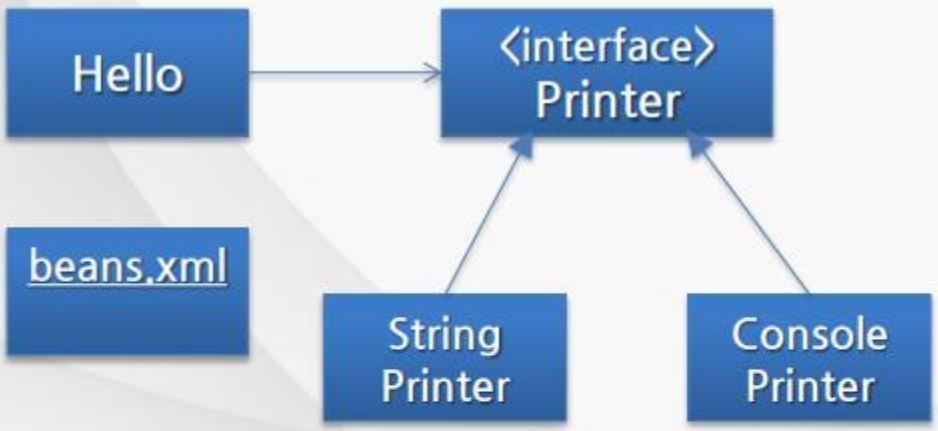
Hello.java

```java
package bean;

import java.util.List;

public class Hello {
    String name;
    Printer printer;

    public Hello() { }

    public void setName(String name) {
        this.name = name;
    }

    public void setPrinter(Printer printer) {
        this.printer = printer;
    }
```

beans.xml

```xml
<bean id="hello" class="bean.Hello">
    <property name="name" value="Spring" />
    <property name="printer" ref="printer" />
</bean>

<bean id="printer" class="bean.StringPrinter" />
<bean id="consolePrinter" class="bean.ConsolePrinter" />
```

DI – Constructor Injection



Hello.java

```java
package bean;

import java.util.List;

public class Hello {
    String name;
    Printer printer;

    public Hello() { }

    public Hello(String name, Printer printer) {
        this.name = name;
        this.printer = printer;
    }
```

beans.xml

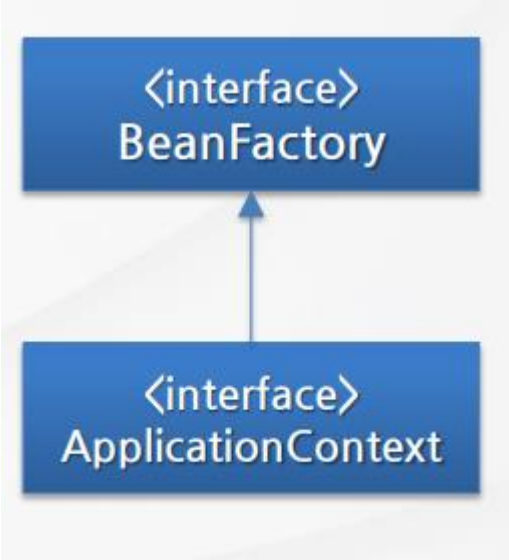
```xml
    <bean id="hello" class="bean.Hello">
        <constructor-arg index="0" value="Spring" />
        <constructor-arg index="1" ref="printer" />
    </bean>

    <bean id="printer" class="bean.StringPrinter" />
    <bean id="consolePrinter" class="bean.ConsolePrinter" />
```

Spring 컨테이너

- Spring 컨테이너가 관리하는 객체를 빈(Bean)이라고 하고, 이 빈(bean)들을 관리한다는 의미로 컨테이너를 빈 팩토리(BeanFactory)라고 부른다.

-객체의 생성과 객체 사이의 런타임 관계를 DI관점에서 볼때 컨테이너를 BeanFactory라고 한다.

-BeanFactory에 여러가지 컨테이너 기능을 확장하여 ApplicationContext라고 부른다.



BeanFactory와 ApplicationContext

BeanFactory	▪ Bean을 등록,생성, 조회, 반환 관리함 ▪ 보통은 BeanFactory를 바로 사용하지 않고, 이를 확장한 ApplicationContext를 사용함 ▪ getBean() 메서드가 정의되어 있음
Application Context	▪ Bean을 등록,생성, 조회, 반환 관리하는 기능은 BeanFactory와 같음 ▪ Spring의 각종 부가 서비스를 추가로 제공함 ▪ Spring이 제공하는 ApplicationContext 구현 클래스가 여러 가지 종류가 있음

Spring DI 용어 정리

빈 (Bean)	▪ 스프링이 IoC 방식으로 관리하는 객체라는 뜻 ▪ 스프링이 직접 생성과 제어를 담당하는 객체를 Bean이라고 부름
빈 팩토리 (BeanFactory)	▪ 스프링의 IoC를 담당하는 핵심 컨테이너를 가리킴 ▪ Bean을 등록, 생성, 조회, 반환하는 기능을 담당함 ▪ 이 BeanFactory를 바로 사용하지 않고 이를 확장한 ApplicationContext를 주로 이용함
애플리케이션 컨텍스트 (Application Context)	▪ BeanFactory를 확장한 IoC 컨테이너 ▪ Bean을 등록하고 관리하는 기능은 BeanFactory와 동일하지만 스프링이 제공하는 각종 부가 서비스를 추가로 제공함 ▪ 스프링에서는 ApplicationContext를 BeanFactory 보다 더 많이 사용함
설정 메타정보 (Configuration metadata)	▪ ApplicationContext 또는 BeanFactory가 IoC를 적용하기 위해 사용하는 메타정보를 말함 ▪ 설정 메타정보는 IoC컨테이너에 의해 관리되는 Bean 객체를 생성하고 구성할 때 사용됨

Bean등록 메타정보

모든 Bean을 명시적으로 xml에 등록하는 방법

-생성되는 모든 Bean을 xml에서 확인 할수 있다는 장점이 있으나 Bean의 개수가 많아지면 xml파일을 관리하기가 번거롭다.

-여러 개발자가 같은 설정파일을 공유해서 개발하다 보면 설정파일을 동시에 수정하다가 충돌이 일어나는 경우도 생긴다.

-DI에 필요한 적절한 setter메소드 나 또는 constructor가 코드 내에 반드시 존재해야 한다.

-개발 중에는 어노테이션 설정 방법을 사용했지만, 운영 중에는 관리의 편의성을 위해 Xml 설정으로 변경하는 전략이 필요 할 수도 있다.

Bean등록 메타정보

Xml과 빈 스캐닝의 혼용

-Bean으로 사용될 클래스에 특별한 어노테이션을 부여해주면 이런 클래스를 자동으로 찾아서 Bean으로 등록한다.

-특정 어노테이션이 붙은 클래스를 자동으로 찾아서 Bean으로 등록해주는 방식을 빈 스캐닝을 통한 자동 인식 Bean등록기능이라고 한다.

-어노테이션을 부여하고 자동 스캔으로 빈을 등록하면 xml문서 생성과 관리에 따른 수고를 덜어주고 개발 속도를 향상 시킬 수 있다.

-애플리케이션에 등록 될 Bean이 어떤 것들이 있고, Bean들 간의 의존관계가 어떻게 되는지를 한눈에 파악 할 수 없다는 단점이 있다.

Bean등록 Annotation

@Component	컴포넌트를 나타내는 일반적인 스테레오 타입으로 <bean> 태그와 동일한 역할을 함
@Repository	퍼시스턴스(persistence) 레이어, 영속성을 가지는 속성(파일, 데이터베이스)을 가진 클래스
@Service	서비스 레이어, 비즈니스 로직을 가진 클래스
@Controller	프리젠테이션 레이어, 웹 어플리케이션에서 웹 요청과 응답을 처리하는 클래스

@Repository, @Service, @Controller는 @Componet의 구체화된 형태이다.

Bean등록 Annotation

@Autowired, @Resource 어노테이션은 의존하는 객체를 자동으로 주입해 주는 어노테이션이다.

@Autowired

- 정밀한 의존관계 주입 (Dependency Injection)이 필요한 경우에 유용하다.
- @Autowired는 프로퍼티, setter 메서드, 생성자, 일반메서드에 적용 가능하다.
- 의존하는 객체를 주입할 때 주로 Type을 이용하게 된다.
- @Autowired는 <property>, <constructor-arg> 태그와 동일한 역할을 한다.

Bean등록 Annotation

@Autowired, @Resource 어노테이션은 **의존하는 객체를 자동으로 주입해 주는 어노테이션**이다.

@Autowired는 타입으로, @Resource는 이름으로 연결한다는 점이 다르다.

@Resource

- 어플리케이션에서 필요로 하는 자원을 자동 연결할 때 사용된다.
- @Resource는 프로퍼티, setter 메서드에 적용 가능하다.
- 의존하는 객체를 주입할 때 주로 **Name**을 이용하게 된다.

Bean등록 Annotation

@Value

- 단순한 값을 주입할 때 사용되는 어노테이션이다.
- @Value("Spring")은 <property .. value="Spring" /> 와 동일한 역할을 한다.

@Qualifier

- @Qualifier는 @Autowired 어노테이션과 같이 사용되어 진다.
- @Autowired는 타입으로 찾아서 주입하므로, 동일한 타입의 Bean객체가 여러 개 존재할 때 특정 Bean을 찾기 위해서는 @Qualifier를 같이 사용해야 한다.

Bean등록 Annotation

Component Scan 지원하는 태그

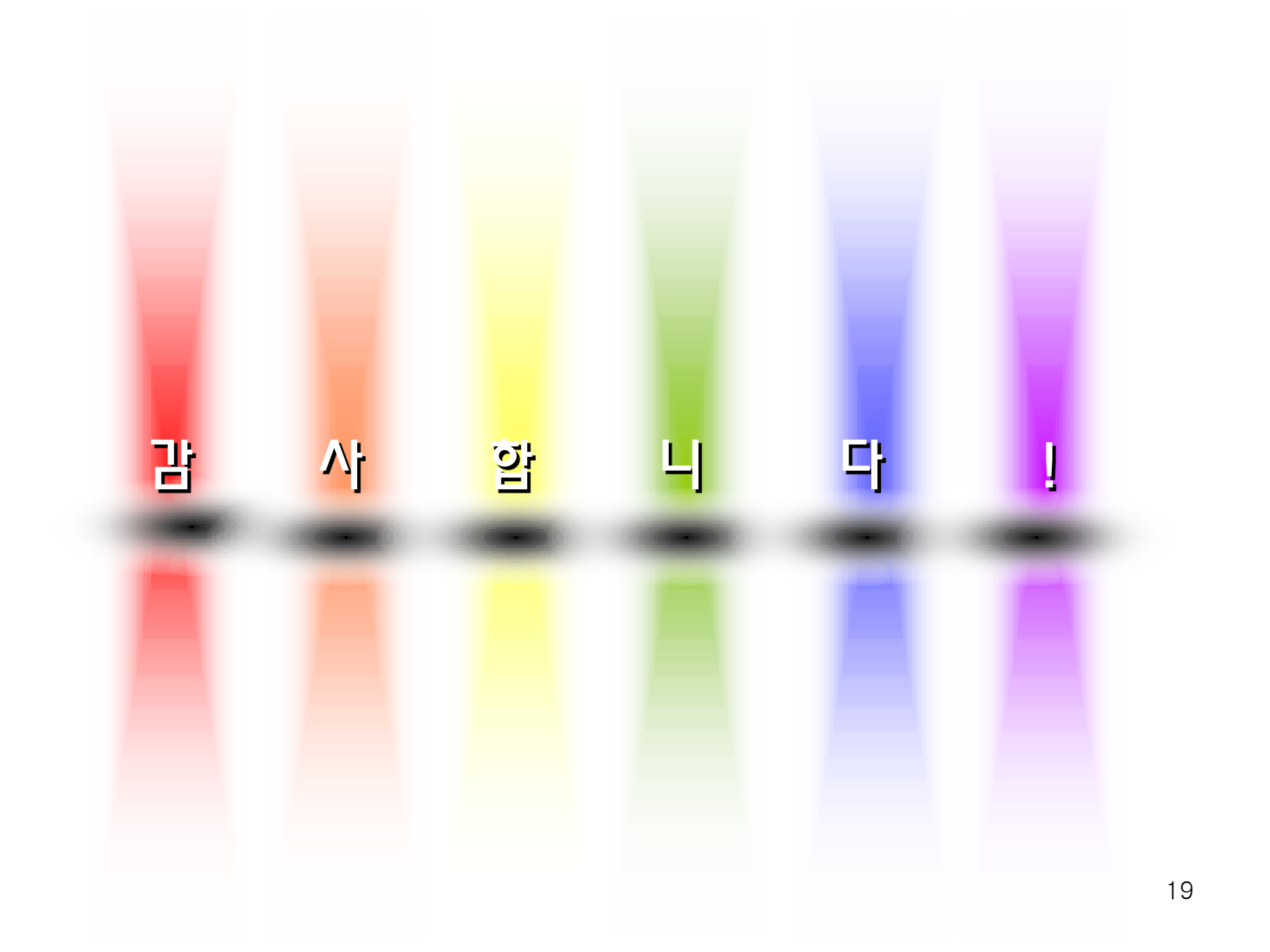
<context:component-scan>

: @Component를 통해 자동으로 Bean을 등록하고

@Autowired로 의존관계를 주입 받는 어노테이션을 클래스에서 선언하여 사용했을 경우에는 해당 클래스가 위치한 특정 패키지를 Scan하기 위한 설정을 xml에 해주어야 한다.

```
<context:component-scan base-package="myspring.di.annot" />
```

- <context:include-filter>태그와 <context:exclude-filter>태그를 같이 사용하면 자동 스캔 대상에 포함시킬 클래스와 포함시키지 않을 클래스를 구체적으로 명시할 수 있다.


감 사 합 니 다 !