Spring DI

Bok, Jong Soon
javaexpert@nate.com
https://github.com/swacademy/Spring5

IoC(Inversion of Control)

- 객체의 생성, 생명주기의 관리까지 모든 객체에 대한 제어권이 바뀌었다는 것을 의미
- Component 의존관계 결정(Component Dependency Resolution), 설정(Configuration) 및 Lifecycle를 해결하기 위한 Design Pattern
- 의존(Dependency)이란 변경에 의해 영향을 받는 관계라는 의미.
- 한개의 Class의 내부 Code가 변경되었을 때 이와 관련된 다른 Class도 함께 변경해야 한다면 이를 변경에 따른 영향이 전파되는 관계로서 **의존**한다고 표현한다.
- 의존하는 대상이 있으면, 그 대상을 구하는 방법 필요.
- 가장 쉬운 방법은 의존 대상 객체를 직접 생성하는 것이다.
- 그래서 의존받는 Class를 생성하면 그 Class가 의존하고 있는 Class도 동시에 생성이 된다.
- 이렇게 Class 내부에서 직접 의존 객체를 생성하는 것은 쉽지만, 유지 보수 관점에서 보면 문제점이 유발될 수 있다.

IoC(Inversion of Control) Container

- Spring Framework도 객체에 대한 생성 및 생명주기를 관리할 수 있는 기능을 제공하고 있음.
- IoC Container 기능 제공.
- IoC Container는 객체의 생성을 책임지고, 의존성을 관리.
- POJO의 생성, 초기화, Service, 소멸에 대한 권한을 가진다.
- 개발자들이 직접 POJO를 생성할 수 있지만 Container에게 맡긴다.

IoC(Inversion of Control) 분류

- DI : Dependency Injection
  - Spring, PiconContainer
  - Setter Injection, Constructor Injection, Method Injection
  - 각 Class간의 의존관계를 빈 설정(Bean Definition) 정보를 바탕으로 Container가 자동으로 연결해주는 것
- DL : Dependency Lookup
  - EJB, Spring
  - 의존성 검색 : 저장소에 저장되어 있는 Bean에 접근하기 위해 Container가 제공하는 API를 이용하여 Bean을 Lookup 하는 것
- DL 사용시 Container 종속성이 증가하여, 주로 DI를 사용함.

Lab. Non-DI Spring Project

Lab. DI Demo in Spring

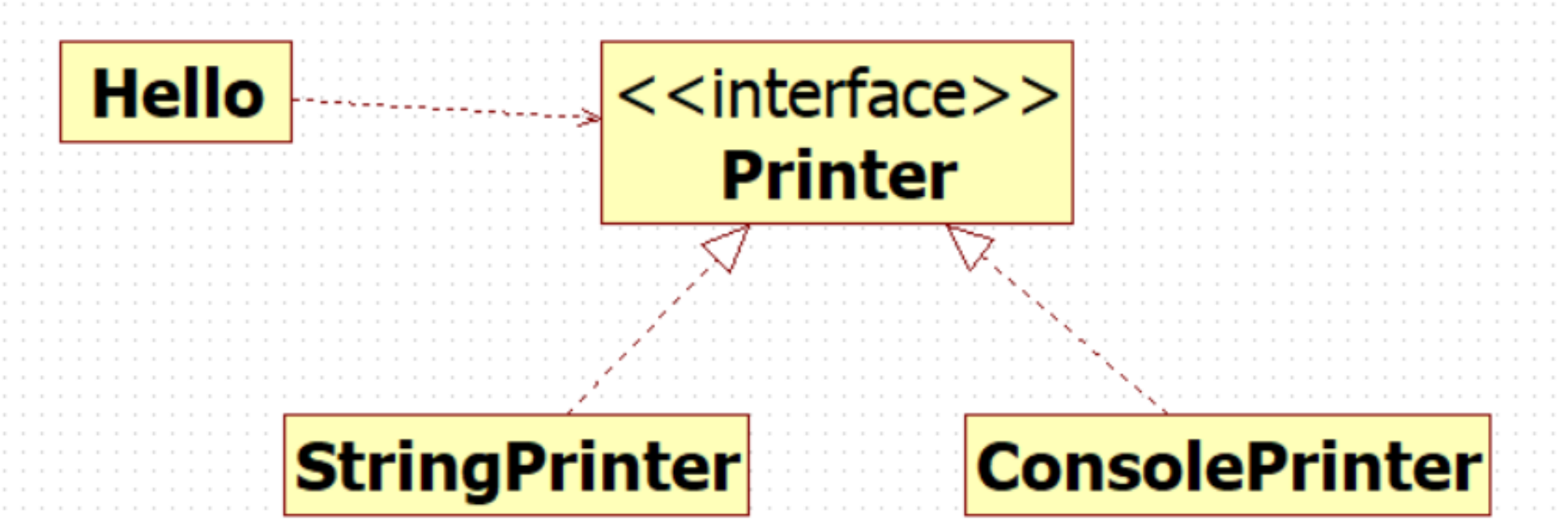
DI(Dependency Injection) 개념

- 각 Class간의 의존관계를 빈 설정(Bean Definition) 정보를 바탕으로 Container가 자동으로 연결해 주는 것을 말함.
- 개발자들은 단지 Bean 설정 File에서 의존관계가 필요하다는 정보를 추가하면 된다.
- 객체 Reference를 Container로부터 주입 받아서, 실행시에 동적으로 의존관계가 생성된다.
- Container가 흐름의 주체가 되어 Application Code에 의존관계를 주입해주는 것이다.
- 장점
  - Code 단순.
  - Component 간의 결합도가 제거됨.

DI(Dependency Injection) 유형

- Setter Injection
  - Setter method를 이용한 의존성 삽입
  - 의존성을 입력 받는 Setter Method를 만들고, 이를 통해 의존성을 주입한다.
- Constructor Injection
  - 생성자를 이용한 의존성 삽입
  - 필요한 의존성을 포함하는 Class의 생성자를 만들고 이를 통해 의존성을 주입한다.
- Method Injection
  - 일반 Method를 이용한 의존성 삽입
  - 의존성을 입력받는 일반 method를 만들고 이를 통해 의존성을 주입한다.

DI(Dependency Injection)를 이용한 Class 호출방식



- Hello Class가 직접 StringPrinter나 ConsolePrinter를 찾아서 사용하는 것이 아니라 설정 File(Spring Bean Configuration File)에 설정하면 Container가 연결해준다.

Setter Injection

- beans.xml

```
<bean id="hello" class="bean.Hello">
<!--bean은 Spring이 관리해주는 객체라는 뜻  -->
     <property name="name" value="Spring" />
     <property name="printer" ref="printer" />
</bean>
<bean id="printer" class="bean.StringPrinter" />
<bean id="consolePrinter" class="bean.ConsolePrinter" />
```

Setter Injection (Cont.)

- Hello.java

```
package bean;
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;
	}
}
```

Constructor Injection

- beans.xml

```
<bean id="hello" class="bean.Hello">
<!--bean은 Spring이 관리해주는 객체라는 뜻  -->
     <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" />
```

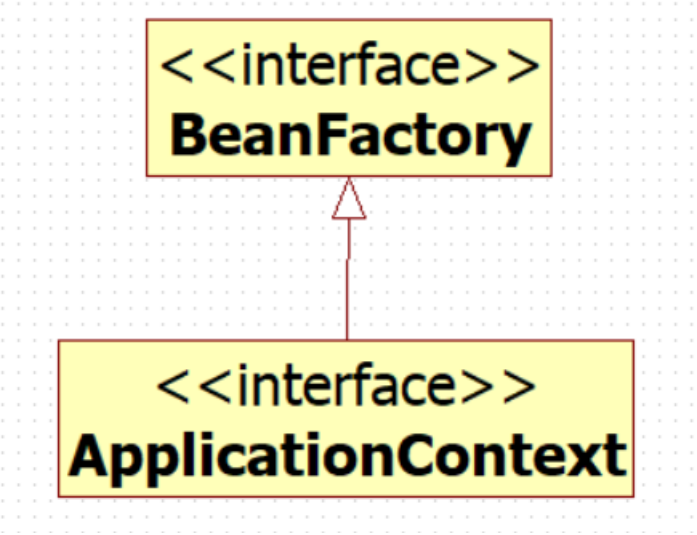
Constructor Injection (Cont.)

- Hello.java

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

Spring DI Container

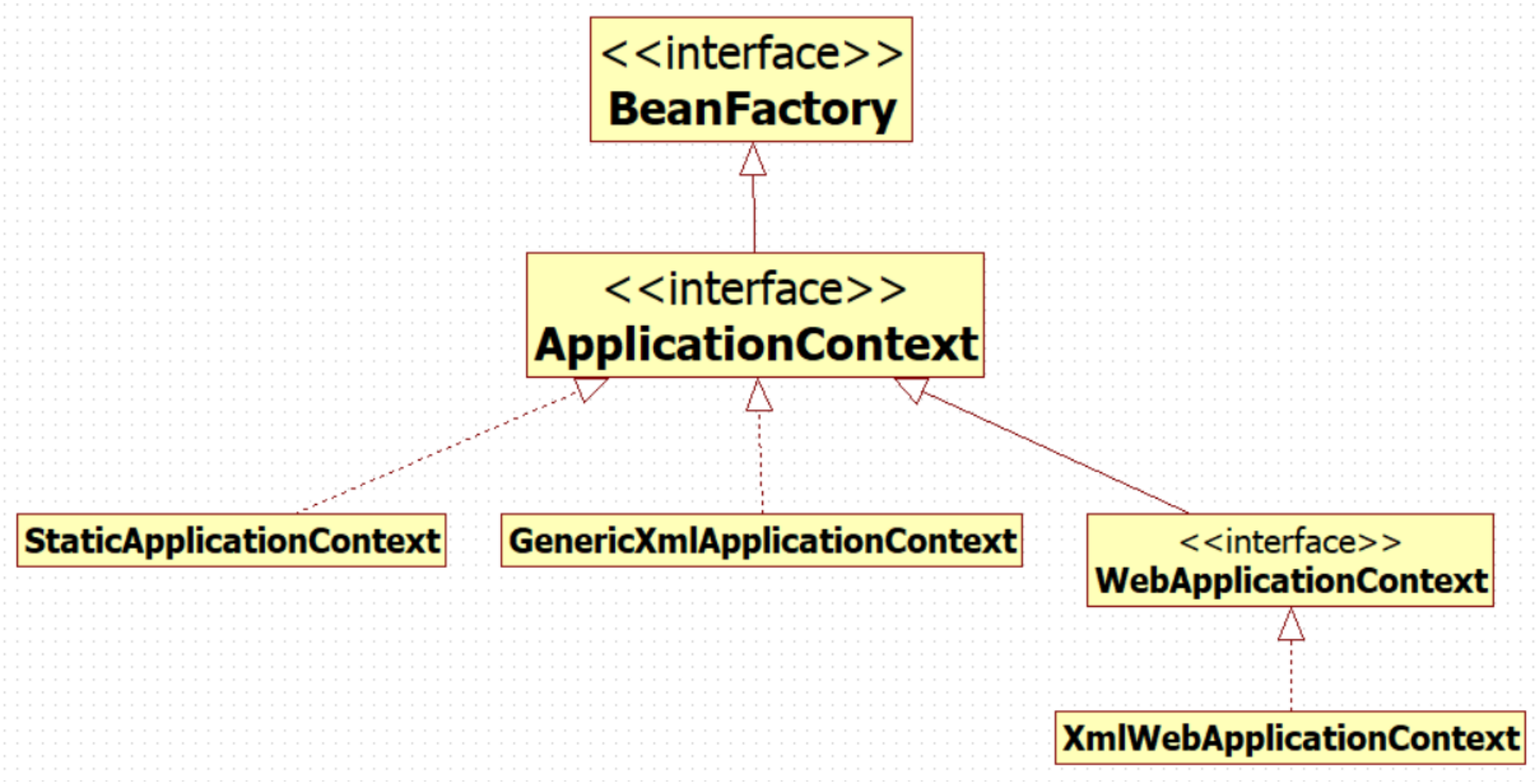
- Spring DI Container가 관리하는 객체를 빈(Bean)이라고 하고, 이 Bean들을 관리한다는 의미로 Container를 빈 팩토리(BeanFactory)라고 부른다.
- 객체의 생성과 객체 사이의 런타임(run-time) 관계를 DI 관점에서 볼 때는 Container를 **`BeanFactory`**라고 한다.
- BeanFactory에 여러 가지 Container 기능을 추가하여 어플리케이션 컨텍스트(**`ApplicationContext`**)라고 부른다.



Spring DI Container (Cont.)

- **`BeanFactory`**
  - Bean을 등록, 생성, 조회, 반환 관리함
  - 보통은 **`BeanFactory`**를 바로 사용하지 않고, 이를 확장한 **`ApplicationContext`**를 사용함
  - **`getBean()`** method가 정의되어 있음.
- **`ApplicationContext`**
  - Bean을 등록, 생성, 조회, 반환 관리하는 기능은 **`BeanFactory`**와 같음.
  - Spring의 각종 부가 Service를 추가로 제공함.
  - Spring이 제공하는 **`ApplicationContext`** 구현 class가 여러가지 종류가 있음.

Spring DI Container (Cont.)



Spring DI Terminology

- Bean
  - Spring이 IoC 방식으로 관리하는 객체라는 뜻
  - Spring이 직접 생성과 제어를 담당하는 객체를 Bean이라고 부른다.
- **`BeanFactory`**
  - Spring의 IoC를 담당하는 핵심 Container
  - Bean을 등록, 생성, 조회, 반환하는 기능을 담당.
  - **`BeanFactory`**를 바로 사용하지 않고 이를 확장한 **`ApplicationContext`**를 주로 이용

Spring DI Terminology (Cont.)

- **ApplicationContext**
  - **BeanFactory**를 확장한 Ioc Container
  - Bean을 등록하고 관리하는 기능은 **BeanFactory**와 동일하지만 Spring이 제공하는 각종 부가 service를 추가로 제공
  - Spring에서는 **ApplicationContext**를 **BeanFactory**보다 더 많이 사용
- Configuration metadata
  - **ApplicationContext** 또는 **BeanFactory**가 IoC를 적용하기 위해 사용하는 meta정보
  - 설정 meta정보는 IoC Container에 의해 관리되는 Bean 객체를 생성하고 구성할 때 사용됨.

jUnit의 개요와 특징

- TDD의 창시자인 Kent Beck과 Design Pattern 책의 저자인 Erich Gamma가 작성
- 단정(Assert) method로 Test Case의 수행 결과를 판별
  - **`assertEquals`**(예상 값, 실제 값)
- jUnit4부터는 test를 지원하는 annotation 제공, **`@Test`**, **`@Before`**, **`@After`**
- 각 **`@Test`** method가 호출할 때마다 새로운 instance를 생성하여 독립적인 test가 이루어지도록 한다.

jUnit Library 설치

- http://mvnrepository.com
- jUnit으로 검색
- jUnit 4.12 version을 pom.xml에 추가

```
<dependency>
	<groupId>junit</groupId>
	<artifactId>junit</artifactId>
	<version>4.12</version>
	<scope>test</scope>
</dependency>
```

- pom.xml > right-click > Run As > Maven Install

jUnit에서 test를 지원하는 Annotation

- **`@Test`**
  - 이것이 선언된 method는 test를 수행하는 method가 된다.
  - jUnit은 각각의 test가 서로 영향을 주지 않고 독립적으로 실행됨을 원칙으로 함으로 **`@Test`** 마다 객체를 생성한다.
- **`@Ignore`**
  - 이것이 선언된 method는 test를 실행하지 않게 한다.
- **`@Before`**
  - 이것이 선언된 method는 **`@Test`**가 실행되기 전에 반드시 실행된다.
  - **`@Test`** method에서 공통으로 사용하는 code를 **`@Before`** method에 선언하여 사용하면 된다.

jUnit에서 test를 지원하는 Annotation (Cont.)

- **`@After`**
  - 이것이 선언된 method는 **`@Test`** method가 실행된 후 실행된다.
- **`@BeforeClass`**
  - 이 annotation은 **`@Test`** method보다 먼저 한번만 수행되어야 할 경우에 사용하면 된다.
- **`@AfterClass`**
  - 이 annotation은 **`@Test`** method보다 나중에 한번만 수행되어야 할 경우에 사용하면 된다.

test 결과를 확인하는 단정(Assert) method 종류

- **`assertEquals(a, b)`**
  - 객체 a와 b가 일치함을 확인
- **`assertArrayEquals(a, b)`**
  - 배열 a, b가 일치함을 확인
- **`assertSame(a, b)`**
  - 객체 a, b가 같은 객체임을 확인
  - **`assertEquals()`** method는 값이 같은지를 확인하는 것이고, **`assertSame()`** method는 두 객체의 reference가 같은지를 확인한다.(==연산자)
- **`assertTrue(a)`**
  - 조건 a가 참인가를 확인
- **`assertNotNull(a)`**
  - 객체 a가 null이 아님을 확인한다.
- 이외에도 다양한 assert method가 존재함
- http://junit.sourceforge.net/javadoc/org/junit/Assert.html

Lab. jUnit을 사용한 DI Test Class 작성하기

Spring TestContext Framework

- **`@RunWith(SpringJUnit4ClassRunner.class)`**
  - jUnit Framework의 test 실행방법을 확장할 때 사용하는 Annotation
  - **`SpringJUnit4ClassRunner`**라는 class를 지정해주면 jUnit이 Test를 진행하는 중에 **`ApplicationContext`**를 만들고 관리하는 작업을 진행해 준다.
  - Annotation은 각각의 test 별로 객체가 생성되더라도 Singleton의 **`ApplicationContext`**를 보장한다.
- **`@ContextConfiguration`**
  - Spring bean 설정 file의 위치를 지정할 때 사용되는 Annotation
- **`@Autowired`**
  - Spring DI에서 사용되는 특별한 Annotation
  - 해당 변수에 자동으로 빈(Bean)을 매핑해준다.
  - Spring bean 설정 file을 읽기 위해 굳이 **`GenericXmlApplicationContext`**를 사용할 필요가 없다.

Lab. Spring TextContext Framework

Dependency Injection(의존주입) 방법의 종류

- XML file을 이용한 DI 설정 방법
  - setter 이용하기
  - 생성자 이용하기
- Java Annotation 이용한 DI 설정 방법
- Java Annotation과 XML 을 이용한 DI 설정 방법
  - XML file에 Java file을 포함시켜 사용하는 방법
  - Java file에 XML file을 포함시켜 사용하는 방법

Setter를 이용한 의존주입하기 – Setter Injection

- Setter method를 통해 의존 관계가 있는 bean을 주입하려면 **<property>** 태그를 사용할 수 있다.
- **ref** 속성은 사용하면 bean이름을 이용해서 주입할 bean을 찾는다.
- **value** 속성은 단순 값 또는 bean이 아닌 객체를 주입할 때 사용한다.
- 단순 값(문자열이나 숫자)의 주입
  - Setter method를 통해 bean의 레퍼런스가 아니라 단순 값을 주입하려고 할 때는 **<property>** 태그의 **value**속성을 사용한다.

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

```
<bean id="hello" class="com.example.Hello">
    <property name="name" value="Spring" />
```

Setter를 이용한 의존주입하기 – Setter Injection (Cont.)

- Collection 타입의 값 주입
  - Spring은 List, Set, Map, Properties와 같은 Collection 타입을 XML로 작성해서 property에 주입하는 방법을 제공한다.
  - List 타입 : **<list>**와 **<value>** 태그를 이용
  - Set 타입 : **<set>**과 **<value>** 태그를 이용
  - Map 타입 : **<map>**과 **<entry>** 태그를 이용
  - Properties 타입 : **<props>**와 **<prop>**를 이용

Setter를 이용한 의존주입하기 – Setter Injection (Cont.)

- Collection 타입의 값 주입
  - List 타입 : **<list>**와 **<value>** 태그를 이용
  - Set 타입 : **<set>**과 **<value>** 태그를 이용

```
public class Hello{
    List<String> names;
    public void setNames(List<String> list){
        this.names = list;
    }
}
```

Setter를 이용한 의존주입하기 – Setter Injection (Cont.)

- Collection 타입의 값 주입
  - List 타입 : **<list>**와 **<value>** 태그를 이용
  - Set 타입 : **<set>**과 **<value>** 태그를 이용

```
<bean id="hello" class="com.example">
    <property name="names">
        <list>
            <value>Spring</value>
            <value>IoC</value>
            <value>DI</value>
        </list>
    </property>
</bean>
```

Setter를 이용한 의존주입하기 – Setter Injection (Cont.)

- Collection 타입의 값 주입
  - List 타입 : **<list>**와 **<value>** 태그를 이용
  - Set 타입 : **<set>**과 **<value>** 태그를 이용

```
<bean id="hello" class="com.example">
    <property name="foods">
        <set>
            <value>Chicken</value>
            <value>Pizza</value>
            <value>Bread</value>
        </set>
    </property>
</bean>
```

Setter를 이용한 의존주입하기 – Setter Injection (Cont.)

- Collection 타입의 값 주입
  - Map 타입 : **<map>**과 **<entry>** 태그를 이용

```
public class Hello{
    Map<String, Integer> ages;

    public void setAges(Map<String, Integer> ages){
        this.ages = ages;
    }
}
```

Setter를 이용한 의존주입하기 – Setter Injection (Cont.)

- Collection 타입의 값 주입
  - Map 타입 : **<map>**과 **<entry>** 태그를 이용

```
<bean id="hello" class="com.example.Hello">
    <property name="ages">
        <map>
            <entry key="나훈아" value="30" />
            <entry key="이미자" value="50" />
            <entry>
                <key>
                    <value>설운도</value>
                </key>
                <value>60</value>
            </entry>
        </map>
    </property>
</bean>
```

Setter를 이용한 의존주입하기 – Setter Injection (Cont.)

- Collection 타입의 값 주입
  - Properties 타입 : <props>와 <prop>를 이용

```
<bean id="hello" class="com.example.Hello">
    <property name="ages">
        <props>
            <prop key="나훈아">서울시 강남구 역삼동</prop>
            <prop key="이미자">경기도 수원시 장안구</prop>
        </props>
    </property>
</bean>
```

Setter를 이용한 의존주입하기 – Setter Injection (Cont.)

- Collection 타입의 값 주입
  - null값 추가

```
<set>
    <value>Element 1</value>
    <value>Element 2</value>
    <null />
</set>

<map>
    <entry>
        <key>
            <null />
        </key>
        <null />
    </entry>
</map>
```

Setter를 이용한 의존주입하기 – Setter Injection (Cont.)

- 배열의 값 지정

```
<property name="">
    <array>
        <value>1</value>
        <value>2</value>
    </array>
</property>
```

Setter를 이용한 의존주입하기 – Setter Injection (Cont.)

- 실제 Application 개발 Scenario에서 사용되는 Spring bean의 속성과 생성자 인자 형식은 String 형식, 다른 bean의 참조, 여러 표준 형식 (**`java.util.Date`**, **`java.util.Map`** 등) 또는 사용자 지정 형식(예, **`Address`**)까지 매우 다양하다.
- **`java.util.Date`**, **`java.util.Currency`**, 기본 형식 등의 bean 속성과 생성자 인자를 간편하게 전달하기 위해 Spring 에서는 기본적으로 **`PropertyEditor`**를 제공하고 있다.