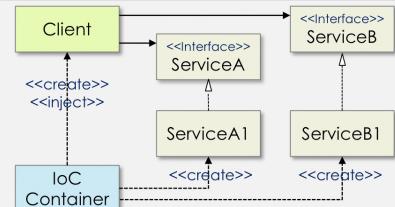


### 3. Dependency Injection

- 디자인 패턴으로 일상에서 흔히 사용되는 기능 구현하기 위해 여러 가지 형태로 활용되고 있다.  
 → DB 접근  
 → GUI "  
 → 서비스 "  
 → 디자인 패턴 활용.
- 여러 가지 형태로 활용되며 이를 위해 DI(Dependency Injection) 디자인 패턴을 활용함.

#### DI

- 의존관계 주입
- 외부 객체 간의 의존관계를 만드는 것
- 스프링은 주입해 사용할 객체들 간의 의존관계를 확보함.
- 객체 간의 결합도를 낮춤.



Spring Core = IoC Container = DI Container = DI&AOP Container

#### IoC (Inversion of Control)

- 예전에는, IDE로 제작하는 프로그램이 예전과 같은 의미

↳ 컨트롤의 관리자는 인터페이스와 같은 관계성을 제공하는 대신 IoC 컨테이너가 대신 관리해야 하는 책임을 확장하고 컨트롤

IoC 컨테이너 → 스프링의 제공.

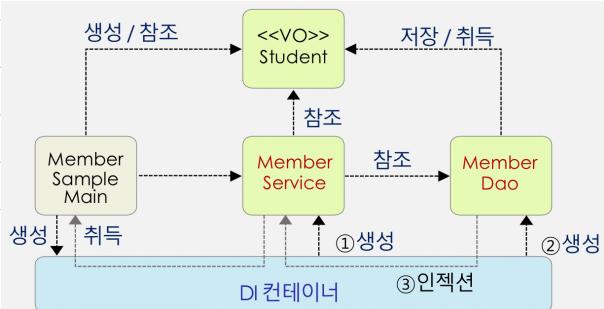
- 인터페이스 성질과 혼동 및 의존 관계 구성을 처리함.

이전 주제의 예제처럼 예제 → new 연산자 사용

→ main() Service 를 new로 생성 → Service가 Dao를 new로 생성 → 각자의 인터페이스 이용

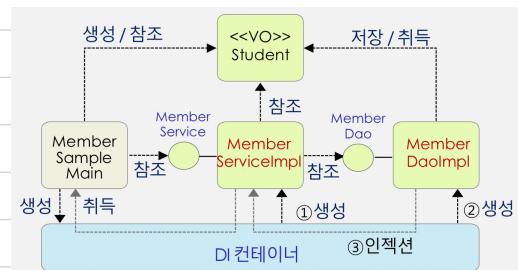
DI 컨테이너 활용 예제 → new 대체하기

- Service 인터페이스, Dao 인터페이스 DI 컨테이너에  
생성
- Dao 인터페이스를 Service에 의존(의존 관계 주입)



DI 컨테이너를 활용한 예 → QTP에 대한 강의화

- Service와 Dao를 인터페이스로 하고, 구현 클래스는 인터페이스 이름에 Impl을 더붙임.



### ~~스프링 빈(Spring Bean)~~

- 스프링 컨테이너가 관리하는 객체

### IoC (Inversion of Control) 컨테이너

- 스프링 빈의 생성, 관리, 조립, 생명주기를 관리하는 스프링 초기화후의 핵심
- 외부에게 주입하여 영구하는 커스텀화된 설정들을 관리한다.

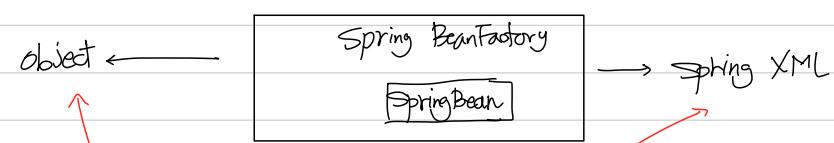
스프링 컨테이너 종류

↳ BeanFactory (org.springframework.beans.factory.BeanFactory)

↳ ApplicationContext (org.springframework.context.ApplicationContext)

### BeanFactory

- 디렉터리 별의 파일에 의해 동의 DI의 기본 기능을 제공.
- 파일이 많지 X, 경량 컨테이너로 작업할 때 흔들
- XML 파일로부터 설정 정보를 학습하는 가장 대중적 사용하는 클래스  
→ org.springframework.beans.factory.xml.XmlBeanFactory



```
Resource xmlFile = new ClassPathResource("applicationContext.xml");
MyBean myBean = (MyBean) context.getBean("myBean");
```

## ApplicationContext

- ~~Object의 데스 페인트.~~
- BeanFactory 대체로는 특별한 특의 인터페이스로 확장된 기능 제공  
→ Internationalization, AOP, Transaction Manager
- XML THROGHTI B2B 기능은 Spring을 가장 많이 사용하는 툴이.  
→ org.springframework.context.support.ClassPathXmlApplicationContext

~~부록~~

## Web ApplicationContext

- 원래 애플리케이션 ApplicationContext
- XML THROGHTI B2B 기능은 가장 많이 사용되는 툴.  
→ org.springframework.web.context.support.XmlWebApplicationContext

- 2가지 종류의 WAC

### 1) ContextLoaderListener에 의해 생성되는 WAC

- Persistence(DAO), Service API 구현체를 통한.
- 웹 애플리케이션에서 WAC 구조체 생성
- root-context.xml 파일에��

### 2) DispatcherServlet에 의해 생성되는 WAC

- 컨테이너에 의해 생성되는 툴
- 해당 컨테이너가 WAC 구조체 생성
- servlet-context.xml 파일에配

## 웹 애플리케이션 객체

- web.xml의 ContextLoaderListener, DispatcherServlet를 HTTP의 ApplicationContext 상속.

## DI(의존 관계 주입)

- XML, Annotation, JAVA 기반 설정을 통하여 객체간의 의존 관계를 처리
- DI 방식 방법

XML 기반 설정 → XML 파일 내용은 <Bean> 요소 개수의 설정

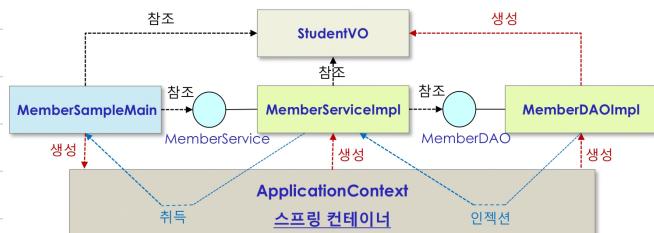
Annotation 기반 설정 → ① Component 어노테이션이 부여된 클래스를 DI 컨테이너가 Bean으로 등록하는 방법

Java 기반 설정 → 자바 클래스에 ② Configuration 어노테이션, 데코레이터 ③ Bean 어노테이션 사용하여 Bean을 등록하는 방법

di-xml 예제 프로젝트 DI 맵핑.

- DI를 이용한 의존 관계 기반의 구조화된 디자인

→ 변경 or 확장하는데 해당 코드를 이용하는 다른 클래스에 영향을 주도록.



## DI 의존성 주입 방식

- 생성자 기반 의존성 주입 (Constructor based dependency injection)

→ 생성자의 인수를 사용한 의존성 주입

→ 초기화 파일 XML에 <construct-arg> 태그를 사용하여 주입한 구조화된 방식.

- 설정자 기반 의존성 주입 (Setter based dependency injection)

→ 대체로 인수를 통해 의존성 주입

→ 초기화 파일 XML에 <property> 요소의 name 속성에 주입할 컨테이너 이름 지정

& constructor-arg ref= "memberDAO" /> → 클래스 생성자 초기화

< property name = "memberDAO" ref = "memberDAO" > → setMemberDAO()

## bean 属性 목록

속성	의미
<code>id</code>	오브젝트를 유일하게 하는 ID
<code>name</code>	오브젝트명을 정의 오브젝트에 여러 이름을 설정하고 싶을 때나 ID에는 설정할 수 없는 이름을 지정할 때 이용
<code>class</code>	id의 실제 패키지명과 클래스명으로 구성
<code>scope</code>	오브젝트의 스코프를 지정( <code>singleton, prototype, request, session, application</code> )
<code>parent</code>	설정을 물려받을 오브젝트명을 지정
<code>abstract</code>	<code>true</code> 인스턴스를 만들지 않고 공통 설정을 정의해두고 싶을 때 지정 <code>false</code> 속성 생략 시 기본값. 인스턴스를 만들고 싶을 때 지정
<code>singleton</code>	<code>true</code> 속성 생략 시 기본값. <code>getBean</code> 메서드로 얻는 인스턴스는 싱글턴 <code>false</code> <code>getBean</code> 메서드로 얻는 인스턴스는 매번 인스턴스화된 것
<code>lazy-init</code>	<code>true</code> 인스턴스 생성을 지연시킴 <code>false</code> 속성 생략 시 기본값. BeanFactory 시작 시 인스턴스를 생성

속성	의미
<code>depend-on</code>	의존 관계의 대상이 되는 오브젝트가 있는지 검사
<code>init-method</code>	메서드 명을 기술함으로써 인스턴스 변수의 설정 후에 호출됨 여기서 지정하는 메서드에는 인수가 없고 반환 값이 void 형이 됨
<code>destroy-method</code>	메서드 명을 기술함으로써 시스템 종료시 호출됨 여기서 지정하는 메서드에는 인수가 없고 반환 값이 void 형이 됨. 또한 메서드를 가진 오브젝트는 상클론임
<code>parent</code>	설정을 물려받을 오브젝트 명을 지정
<code>autowire</code>	<code>no</code> 속성 생략 시 기본값. property 태그에 ref 태그로 지정된 오브젝트가 인스턴스 변수로 설정됨 <code>byName</code> 프로퍼티 명과 일치하는 오브젝트 명의 Bean이 자동으로 인젝션됨 <code>byType</code> 프로퍼티 형과 일치하는 Bean이 인젝션됨 <code>constructor</code> 생성자를 이용해 인젝션

## Annotation

- DI(Dependency Injection)를 XML 등의 방식에 보장하는 것과 달리 Annotation은 객체의 주변에 디렉티브로 표기
- 클래스, 메서드, 필드의 주변에 표기하여 초기화 및 설정 가능 적용됨을 보여줌
- 앱리케이션 커스텀을 XML 방식 허용 → 애플리케이션 적용 가능

### 주요 Annotation

① `@Autowired` → 컨테이너가 빈의 빈의 의존성을 자동으로 연결하도록 하는 주석  
 ② `@Inject`  
 ③ `@Resource`   
`<context:annotation-config />` 또는 `<context:component-scan />`에 대해서만 사용 가능

① `@Component` → 컨테이너가 초기성을 위한 인터페이스(빈)를 설정하는 주석.

클래스 또는 객체에 `@Component` 를 붙이면 컨테이너가 알아서 관리하고

② `@Autowired` 를 인터페이스(빈)에 주입해줌

`<context: component-scan base-package = "패키지 이름" />` 혹은  
 ② `Component`이 붙은 클래스를 자동으로 빈으로 등록하는 기능

### 빈 정의 파일 주요 대상 - ApplicationContext.xml에 기술

bean spring-bean.xsd /schema/beans Bean(컴포넌트) 설정

context spring-context.xsd /schema/context Bean(컴포넌트) 설정과 어노테이션 설정

~~스프링 구조도~~

스프링 구조도

< context : annotation - config />

- ① Autowired, ② Resource, ③ Required 어노테이션에 따른

- XML 파일에 이어 등록된 Bean의 어노테이션 기능 제공 가능 조건 추가 - 추가로 등록

- 적용 대상 범위에 따라 선택 가능

→ <mvc : annotation - driven />

→ < context : component - scan />

④ Component 어노테이션 X

→ 프로그래밍 XML 설정이 되어

아직은 가능하에 ① Autowired로

구현하는 것.

< context : component - scan base - package = "파ackage" />

- ① Component, ② Service, ③ Repository, ④ Controller 등의 주제로 이용 가능

- 등록 대상이 되는 클래스 경상에서 Bean 자동으로 찾아서 DI 컨테이너에 등록

⑤ Component 등록 어노테이션

⑥ Controller → 프레젠테이션 층 스프링 MVC용 어노테이션

⑦ Service → 서비스 층 Service용 어노테이션, ⑧ Componentlet 등록

⑨ Repository → 데이터베이스 층 DAO용 어노테이션

DAO 관련 예외를 모두 DataAccessException으로 처리

⑩ Configuration → Bean 정의를 자바 프로그램이나 YAML, Java Config로 어노테이션.

```
package org.kpu.di.service;
import org.kpu.di.domain.MemberVO;
import org.kpu.di.persistence.MemberDAO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class MemberServiceimpl implements MemberService {

    @Autowired
    private MemberDAO memberDAO;

    public StudentVO readMember(String id) throws Exception {
        return memberDAO.read(id);
    }

    public void addMember(StudentVO student) throws Exception {
        memberDAO.add(student);
    }
}
```

## Java로 DI 설정

- XML 설정 대로 자바 코드 빌드를.  
→ 단위 테스트에서 사용하여 DI하고자 할 경우, 코드가지고 있지 않아 어노테이션 이용 방법 불가 → XML 설정
- 서비스 인터페이스 대로 설정 방식 선택  
→ XML 기반 + 어노테이션, 자바기반 + 어노테이션 기반
- Java로 DI 설정 코드 작성 → Type Safe, Refactoring에 DI후 가능  
→ 프로퍼티 명 or 클래스 명 틀 경우 컴파일 에러
- 구글테이블 네임 클래스  
→ AnnotationConfigApplicationContext

## 주요 Annotation. (스프링 3.0 ~)

- XML 설정 없이 자바 코드 이용해서 빈 객체 생성과 빈 객체간 의존관계 설정
  - 단위 테스트 대로 빠른 테스트 가능
- ① Configuration → 빈 설정 여러 정보를 담고 있는 클래스로  
    해당 클래스 스프링 컨테이너로 사용됨
- ② Bean → 클래스 내 멤버에 설정하여 새로운 빈 객체 생성하는 대 사용  
    name 속성 사용하여 새로운 빈 이름 적용 가능

## 자바 설정과 XML 비교

- ① Bean과��의 이름을 이용해서 그赖以生存하기 사용하는 빈 객체 생성.
  - 자바 설정에서는 빈 객체 직접 생성.
- ② Bean DI는 ~~수동으로~~ 객체로 추적.
- DI하는 이름은 빈의 브랜치로 사용.
  - 자바 설정에서는 직접적으로 객체를 관리해야 함.

## 빈 객체 스크립트 일어남

- 빈이 생성될 수 있는 범위를 설정하기 위해서는 scope 속성 사용

singleton → 기본 설정, 컨테이너는 한 개의 빈 객체만 생성

prototype → 빈 요청한 때마다 빈 객체 생성

request → 각 요청으로 한 개의 빈 객체 생성 ( WebApplicationContext 에서만 적용 )

session → 각 세션으로 한 개의 빈 객체 생성 ( " )

application → 서버가 컨테이너 생성될 때 빈 객체 생성 ( " )

## 빈 초기화 타이프 목록 일어남

- 초기화 → 이용 → 종료 3단계로 진행
- 빈 생성 후 초기화 작업과 빈 종료 전 초기화 작업을 구현할 수 있는 방법 제공

빈 초기화 → 빈 사용 → 빈 종료  
(Initialization)                          (use)                          (Destruction)

(Initialization) 빈 생성 후 초기화	XML 기반	<bean> 요소의 init-method 속성에 메서드 지정
	애너테이션 기반	@PostConstruct 애너테이션 붙은 메서드
	자바 기반	@Bean의 initMethod 속성에 지정된 메서드
	인터페이스 구현	InitializeBean 인터페이스의 afterPropertiesSet 메서드
(Destruction) 빈 종료 전 처리	XML 기반	<bean> 요소의 destroy-method 속성에 메서드 지정
	애너테이션 기반	@PreDestroy 애너테이션 붙은 메서드
	자바 기반	@Bean의 destroyMethod 속성에 지정된 메서드
	인터페이스 구현	DisposableBean 인터페이스의 destroy 메서드