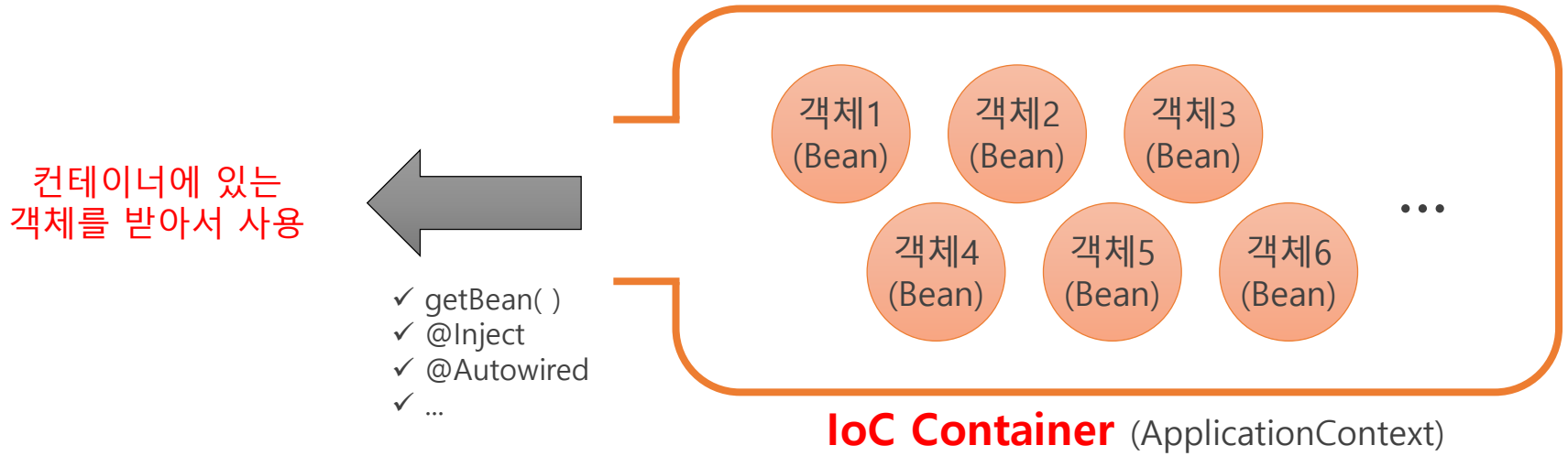
IoC

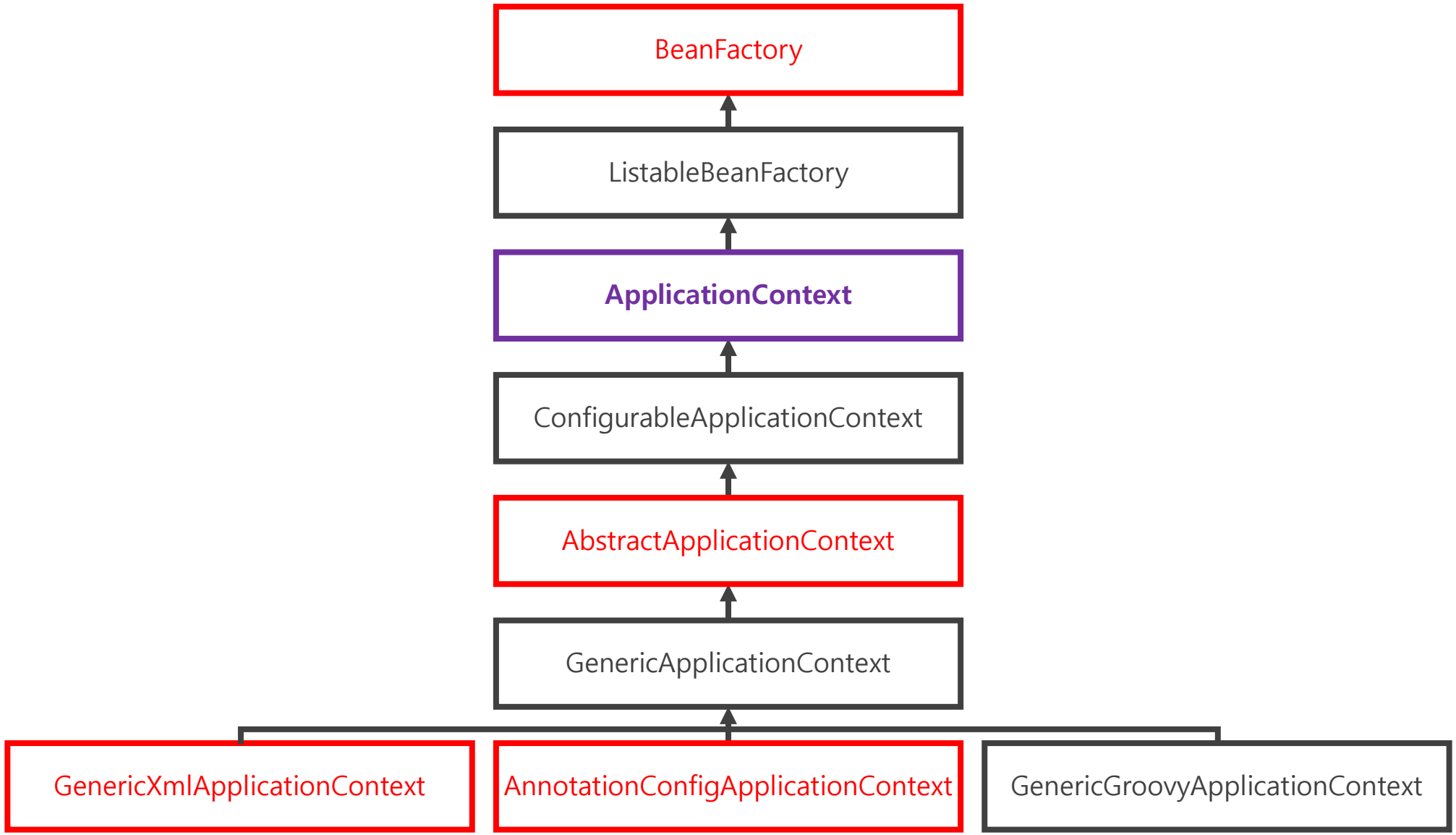
IoC

- IoC
  - ✓ IoC, Inversion of Control
  - ✓ 제어의 역전
  - ✓ 개발자가 프로그램을 제어하지 않고, Framework가 프로그램을 제어하는 것을 의미함
  - ✓ 객체 생성, 의존관계 설정(Dependency), 생명주기(Lifecycle) 등을 Framework가 직접 관리하는 것을 말함

- IoC 컨테이너
  - ✓ 컨테이너(Container) : 객체의 생명주기를 관리하고 생성된 인스턴스를 관리함
  - ✓ Spring Framework에서 객체를 생성과 소멸을 담당하고 의존성을 관리하는 컨테이너를 IoC 컨테이너라고 함
  - ✓ IoC 컨테이너 = 스프링 컨테이너



IoC 컨테이너 구조



IoC 컨테이너 종류

- IoC 컨테이너 주요 종류

컨테이너	의미
BeanFactory	• 스프링 빈 설정 파일(Spring Bean Configuration File)에 등록된 bean을 생성하고 관리하는 가장 기본적인 컨테이너 • 클라이언트 요청에 의해서 bean을 생성함
ApplicationContext	• 트랜잭션 관리, 메시지 기반 다국어 처리 등 추가 기능을 제공 • bean으로 등록된 클래스들을 객체 생성 즉시 로딩시키는 방식으로 동작
GenericXmlApplicationContext	• 파일 시스템 또는 클래스 경로에 있는 XML 설정 파일을 로딩하여 <bean> 태그로 등록된 bean을 생성하는 컨테이너
AnnotationConfigApplicationContext	• 자바 애너테이션(Java Annotation)에 의해서 bean으로 등록된 bean을 생성하는 컨테이너 • @Configuration, @Bean 애너테이션 등이 필요함

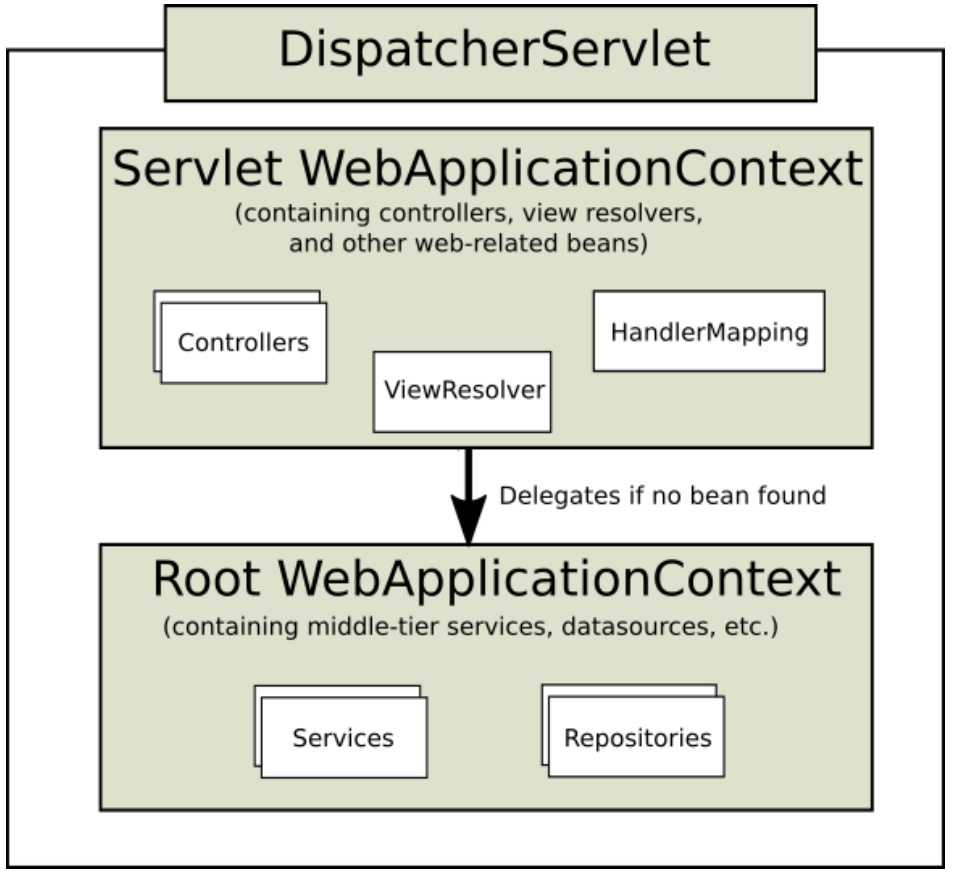
XML 파일을 이용한 Bean 생성

- root-context.xml
  - ✓ View와 관련이 없는 객체를 정의
  - ✓ Service, Repository(DAO), DB 등과 관련된 Bean을 등록함
  - ✓ <bean> 태그를 이용해서 Bean을 등록

- servlet-context.xml
  - ✓ 요청에 관련된 객체를 정의
  - ✓ Controller, ViewResolver, Interceptor 등과 관련된 설정을 처리함
  - ✓ <beans:bean> 태그를 이용해서 Bean을 등록함

Bean???

Spring Container에 의해서 관리되는
POJO(Plain Old Java Object)를
의미한다. 쉽게 말하면
자바 객체가 곧 Bean이다.



(출처 : https://docs.spring.io/)

<property> 태그

- <property> 태그
  - ✓ Setter를 이용해서 값을 전달하고 저장함
  - ✓ 작성방법-1
    <property name="필드">
    <value>값</value>
    </property>
  - ✓ 작성방법-2
    <property name="필드" value="값" />

```xml
<bean id="address" class="com.group.project.Address">
    <property name="zipCode" value="12345" />
    <property name="jibunAddr" value="서울시 강남구 논현동" />
    <property name="roadAddr" value="서울시 강남구 강남대로" />
</bean>
```

injection

```java
public class Address {
    private String zipCode;
    private String jibunAddr;
    private String roadAddr;
    public String getZipCode() {
        return zipCode;
    }
    public void setZipCode(String zipCode) {
        this.zipCode = zipCode;
    }
    public String getJibunAddr() {
        return jibunAddr;
    }
    public void setJibunAddr(String jibunAddr) {
        this.jibunAddr = jibunAddr;
    }
    public String getRoadAddr() {
        return roadAddr;
    }
    public void setRoadAddr(String roadAddr) {
        this.roadAddr = roadAddr;
    }
}
```

<property> 태그는
Setter를 이용해서 value를 전달한다.

spring

<property> 태그

✓ 주의!

참조 타입의 값을 저장할 때는 value가 아닌 ref 속성을 사용해야 함

```xml
<bean id="contact" class="com.group.project.Contact">
    <property name="tel" value="010-1111-1111" />
    <property name="addr" ref="address" />
</bean>

<bean id="address" class="com.group.project.Address">
    <property name="zipCode" value="12345" />
    <property name="jibunAddr" value="서울시 강남구 논현동" />
    <property name="roadAddr" value="서울시 강남구 강남대로" />
</bean>
```

injection

```java
public class Contact {
    private String tel;
    private Address addr;
    public String getTel() {
        return tel;
    }
    public void setTel(String tel) {
        this.tel = tel;
    }
    public Address getAddr() {
        return addr;
    }
    public void setAddr(Address addr) {
        this.addr = addr;
    }
}
```

특정 Bean을 값으로 전달할 때는
ref 속성을 사용한다.

<constructor-arg> 태그

- <constructor-arg> 태그
  - ✓ Constructor를 이용해서 값을 전달하고 저장함
  - ✓ 작성방법-1
    ```
    <constructor-arg>
        <value>값</value>
    </constructor-arg>
    ```
  - ✓ 작성방법-2
    ```
    <constructor-arg value="값" />
    ```

<constructor-arg> 태그는
Constructor를 이용해서 value를 전달한다.

```xml
<bean id="address" class="com.group.project.Address">
    <constructor-arg value="12345" />
    <constructor-arg value="서울시 강남구 논현동" />
    <constructor-arg value="서울시 강남구 강남대로" />
</bean>
```

injection

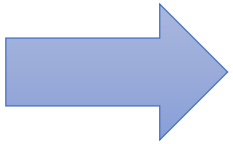
```java
public class Address {
    private String zipCode;
    private String jibunAddr;
    private String roadAddr;
    public Address(String zipCode, String jibunAddr, String roadAddr) {
        this.zipCode = zipCode;
        this.jibunAddr = jibunAddr;
        this.roadAddr = roadAddr;
    }
}
```

생성자의 매개변수와
순서가 반드시 일치해야 한다.

XML to Annotation

- XML을 이용한 Bean 생성의 한계
  - ✓ 프로젝트의 규모가 커지면서 XML의 관리가 어려워짐
  - ✓ XML에서는 자바 데이터를 사용하기가 불편함
  - ✓ 애노테이션을 이용하는 새로운 빈 생성 방식이 개발되었음
  - ✓ 기존 XML을 이용하는 방식은 애노테이션을 이용한 방식으로 대체되는 중임
  - ✓ Spring Boot 프로젝트는 빈 생성을 위한 root-context.xml이나 servlet-context.xml 파일을 기본적으로 지원하지 않고 있음

```
<xml>
    <bean>
    <beans:bean>
</xml>
```



```
@Annotation
    @Configuration
    @Bean
    @Component
```

Annotation을 이용한 Bean 생성

- @Configuration
  - ✓ 클래스 레벨 애노테이션
  - ✓ @Configuration이 명시된 클래스는 Bean으로 등록됨
  - ✓ @Bean을 등록하기 위해서는 반드시 @Configuration을 등록

- @Bean
  - ✓ 메소드 레벨 애노테이션
  - ✓ @Bean이 명시된 메소드가 반환하는 값이 Bean으로 등록됨
  - ✓ 개발자가 직접 Bean을 만들어서 반환하는 방식
  - ✓ 스프링은 메소드 이름을 Bean의 이름으로 등록함
  - ✓ @Bean을 사용하는 메소드가 포함된 클래스는 반드시 @Configuration을 사용해야 함

- @Component
  - ✓ 클래스 레벨 애노테이션
  - ✓ @Component가 명시된 클래스는 자동으로 Bean으로 등록됨
  - ✓ @Component가 명시된 클래스를 찾기 위해서는 반드시 미리 컴포넌트를 찾을 위치를 등록해야 하는데, 이를 Component Scan이라고 함
  - ✓ Spring Legacy Project는 servlet-context.xml에 Component Scan이 등록되어 있어 별도의 설정이 필요 없고, Spring Boot Project는 @SpringBootApplication에 Component Scan이 포함되어 있어 별도의 설정이 필요 없음
  - ✓ 스프링에서 가장 권장하는 Bean 생성 방식임

@Component

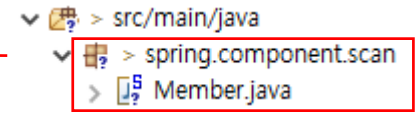
- @Component
  - ✓ @Component가 명시된 클래스는 스프링에 의해서 자동으로 Bean이 등록됨
  - ✓ 스프링은 Component Scan이 등록되어 있어야만 @Component가 명시된 클래스를 찾을 수 있음

- Component Scan
  - ➢ XML에서 Component Scan 등록

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

  - ➢ Java에서 Component Scan 등록

```
@ComponentScan(basePackages = {"spring.component.scan"})
```



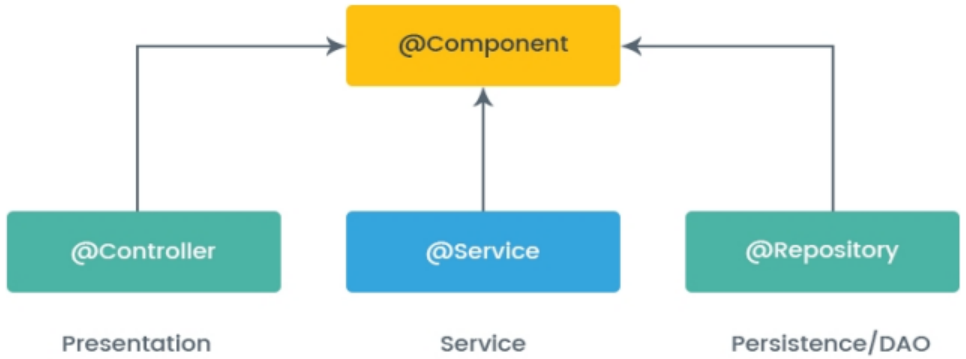
  - ➢ 자동 등록 대상

```
@Component
public class Member {

    private String id;
    private String pw;
```

@Component 계층 구조

- @Component의 계층 구조

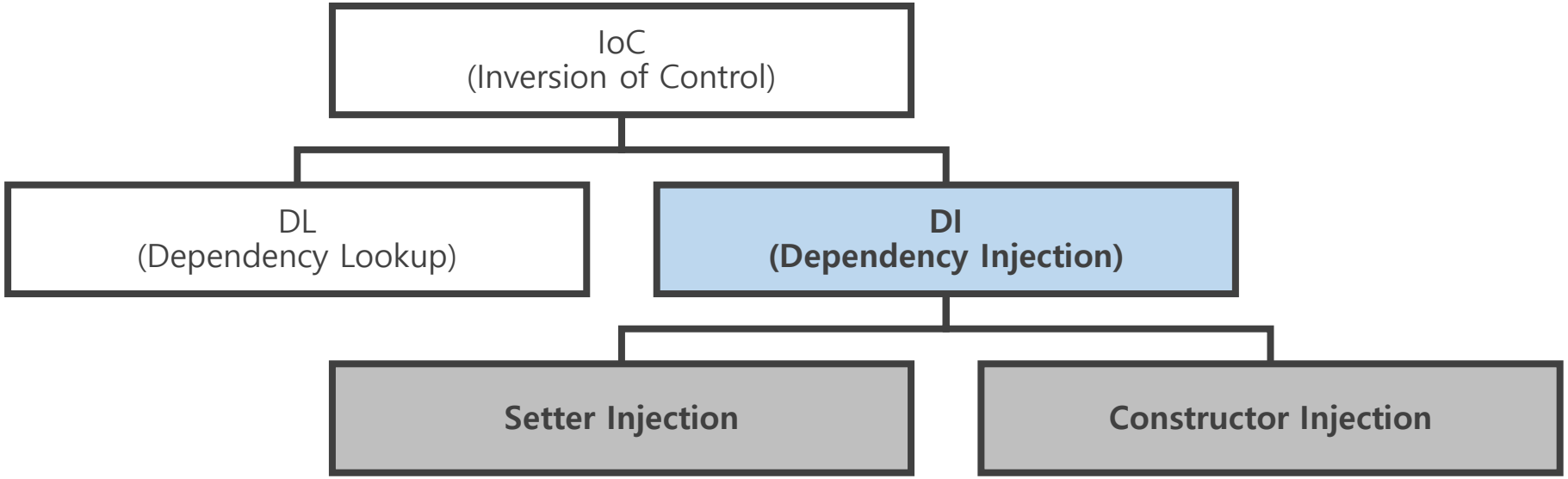


- 주요 애노테이션

애너테이션	의미
@Component	• 스테레오 타입의 최상의 객체
@Controller	• 요청과 응답을 처리하는 Controller 클래스에서 사용 • Spring MVC 아키텍처에서 자동으로 Controller로 인식됨
@Service	• 비즈니스 로직을 처리하는 Service 클래스에서 사용 • Service Interface를 구현하는 ServiceImpl 클래스에서 사용
@Repository	• 데이터베이스 접근 객체(DAO)에서 사용 • 데이터베이스 처리 과정에서 발생하는 예외를 변환해주는 기능을 포함함

IoC 구현 방식

- IoC 구현 방식



- Dependency Lookup
  - ✓ 컨테이너가 애플리케이션 운용에 필요한 객체를 생성하면 클라이언트는 컨테이너가 생성한 객체를 검색(Lookup)해서 사용하는 방식
  - ✓ 실제 애플리케이션 개발에서 사용하지 않음

- Dependency Injection
  - ✓ 객체 사이의 의존 관계를 컨테이너가 직접 설정하는 방식
  - ✓ 스프링 빈 설정 파일에 등록된 정보를 바탕으로 컨테이너가 객체를 처리하는 방식으로 Spring Framework에서 주로 사용하는 방식

DI

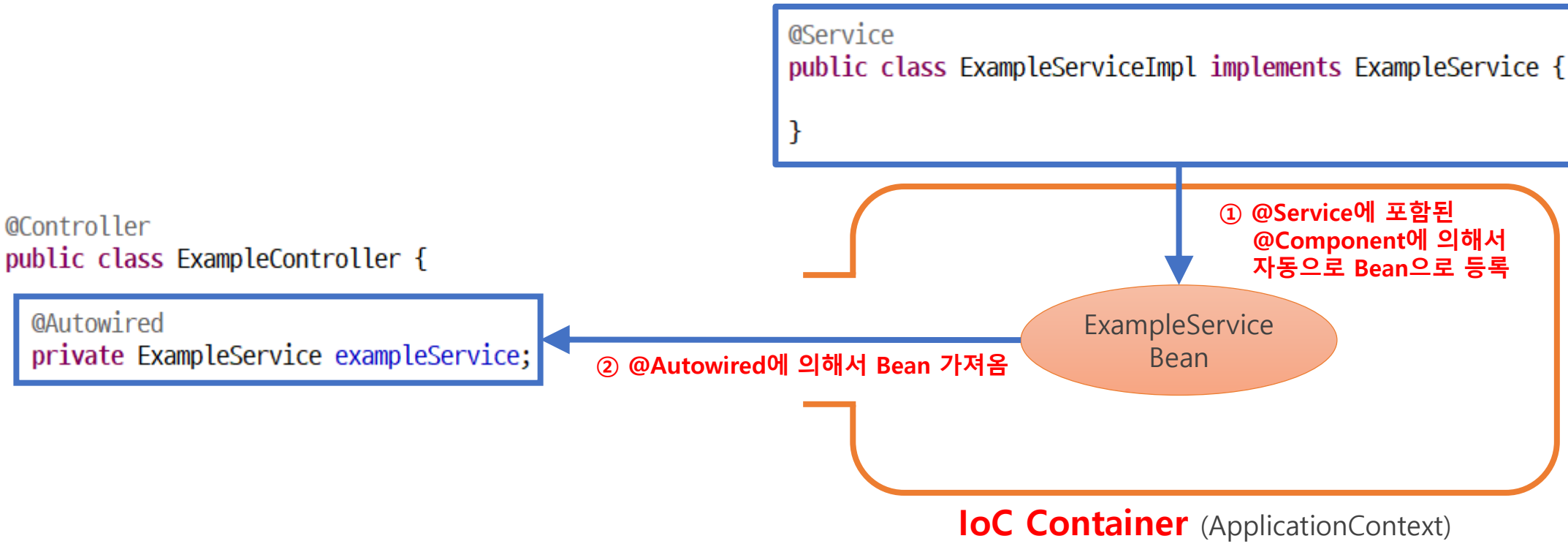
- DI
  - ✓ DI, Dependency Injection
  - ✓ 컨테이너에 등록된 Bean을 가져오는 방식
  - ✓ Field Injection, Setter Injection, Constructor Injection 방식이 있음
  - ✓ @Inject, @Autowired 등의 애노테이션을 이용해서 처리할 수 있음

- DI 애노테이션

애너테이션	의미
@Autowired	• Bean의 타입(class 속성)이 일치하면 가져옴 동일한 타입이 여러 개 있는 경우 Bean의 이름이 일치하면 가져옴 • 생성자에서는 @Autowired 생략 가능 • org.springframework.beans.factory.annotation.Autowired
@Qualifier	• Bean의 이름(id 속성)이 일치하면 가져옴 • 동일한 타입의 Bean이 여러 개 있는 경우 @Qualifier를 추가하여 Bean을 구분할 수 있음 • org.springframework.beans.factory.annotation.Qualifier
@Inject	• Bean의 타입(class 속성)이 일치하면 가져옴 동일한 타입이 여러 개 있는 경우 Bean의 이름이 일치하면 가져옴 • javax.inject.Inject

Field Injection

- Field Injection
  - ✓ 필드 주입
  - ✓ @Autowired가 명시된 필드로 Bean을 가져옴
  - ✓ 모든 필드마다 @Autowired를 명시해야 함
  - ✓ IoC Container에 생성된 Bean이 없어도 코드 작성 시 NULL 여부를 체크하지 않아서 위험할 수 있음
  - ✓ 코드 작성 시 순환 참조 여부를 미리 확인할 수 없음



Setter Injection

- Setter Injection
  - ✓ 세터 주입
  - ✓ @Autowired가 명시된 세터의 매개변수로 Bean을 가져옴
  - ✓ 세터에 @Autowired를 한 번만 명시하면 모든 매개변수로 Bean을 가져오기 때문에 편리함
  - ✓ Field Injection처럼 IoC Container에 생성된 Bean이 없어도 코드 작성 시 NULL 여부를 체크하지 않아서 위험할 수 있음
  - ✓ 코드 작성 시 순환 참조 여부를 미리 확인할 수 없음

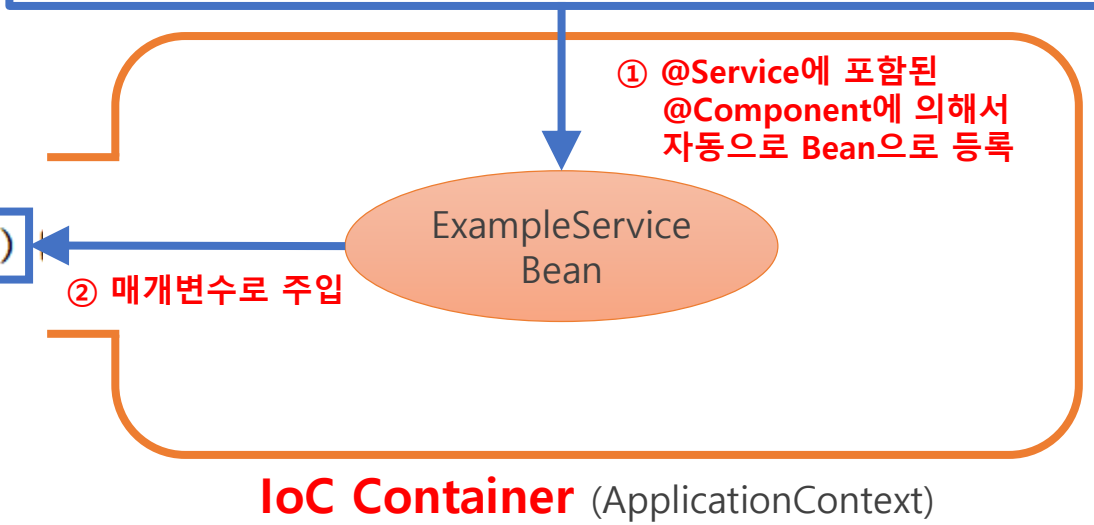
```java
@Controller
public class ExampleController {

  private ExampleService exampleService;

  @Autowired
  public void setExampleService(ExampleService exampleService) {
    this.exampleService = exampleService;
  }
```

```java
@Service
public class ExampleServiceImpl implements ExampleService {

}
```



Constructor Injection

- Constructor Injection
  - ✓ 생성자 주입
  - ✓ 생성자(Generate constructor using field)를 만들면 생성자의 매개변수로 Bean을 가져옴
  - ✓ @Autowired를 명시할 필요가 없음

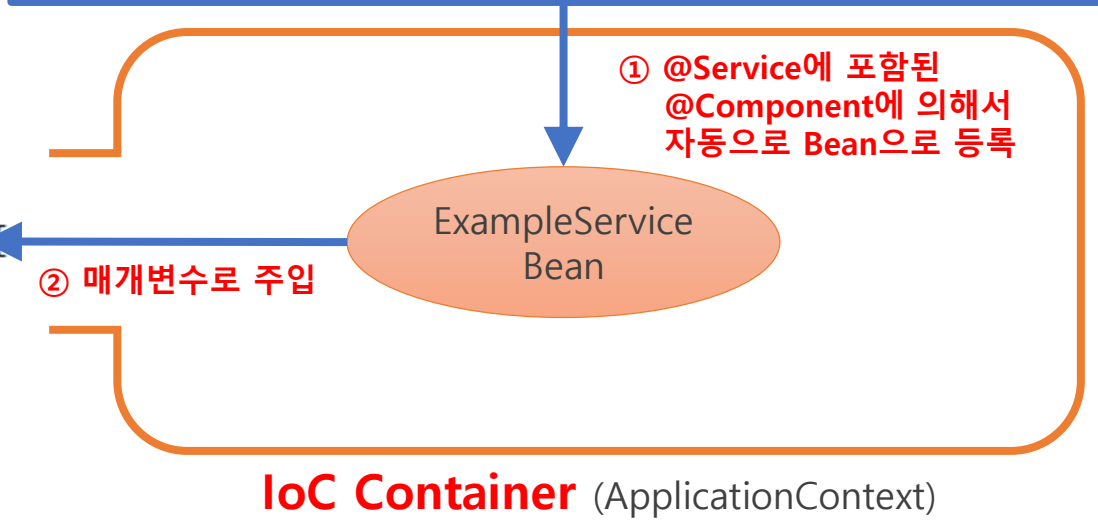
```java
@Controller
public class ExampleController {

  private ExampleService exampleService;

  public ExampleController(ExampleService exampleService) {
    this.exampleService = exampleService;
  }
```

```java
@Service
public class ExampleServiceImpl implements ExampleService {

}
```



Constructor Injection 사용 권장

- Constructor Injection 사용 권장
  - ✓ Constructor Injection에 잘못된 부분이 있는 경우 컴파일 오류가 발생함
  - ✓ 3가지 DI 방식 중 가장 안전한 방식

- Controller의 Constructor Injection 예시