



IoC

IoC

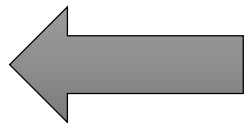
- IoC

- ✓ 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

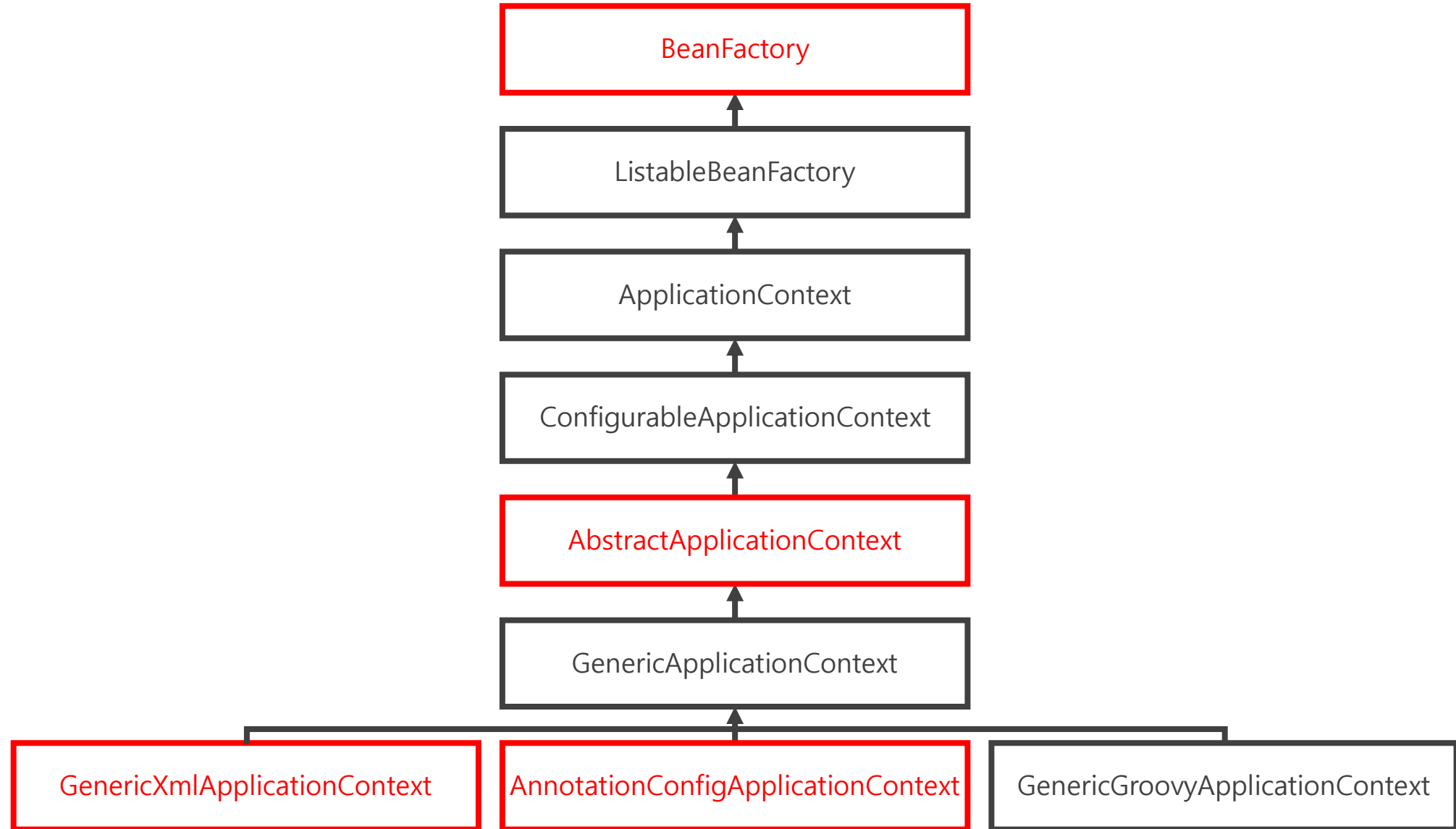
- IoC 컨테이너

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

getBean()을 통해서
생성되어 있는 객체를
받아서 사용



IoC 컨테이너 구조



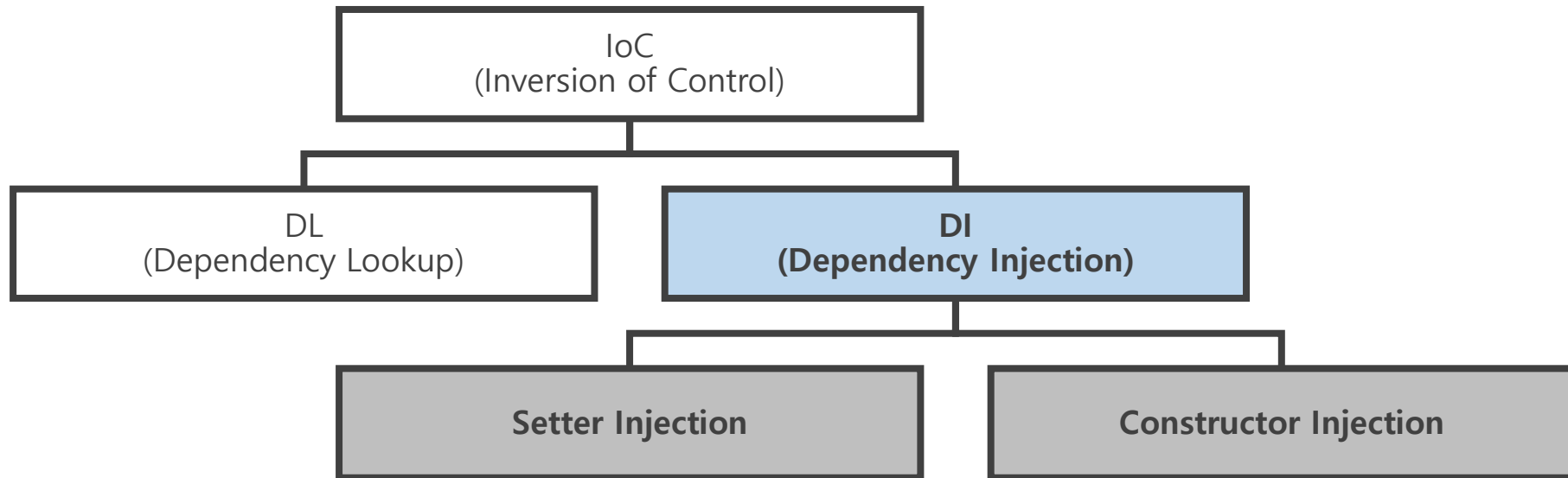
IoC 컨테이너 종류

- IoC 컨테이너 주요 종류

컨테이너	의미
BeanFactory	<ul style="list-style-type: none">• 스프링 빈 설정 파일(Spring Bean Configuration File)에 등록된 bean을 생성하고 관리하는 가장 기본적인 컨테이너• 클라이언트 요청에 의해서 bean을 생성함
ApplicationContext	<ul style="list-style-type: none">• 트랜잭션 관리, 메시지 기반 다국어 처리 등 추가 기능을 제공• bean으로 등록된 클래스들을 객체 생성 즉시 로딩시키는 방식으로 동작
GenericXmlApplicationContext	<ul style="list-style-type: none">• 파일 시스템 또는 클래스 경로에 있는 XML 설정 파일을 로딩하여 <bean> 태그로 등록된 bean을 생성하는 컨테이너
AnnotationConfigApplicationContext	<ul style="list-style-type: none">• 자바 애너테이션(Java Annotation)에 의해서 bean으로 등록된 bean을 생성하는 컨테이너• @Configuration, @Bean 애너테이션 등이 필요함

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

Setter Injection

- Setter Injection

- ✓ Setter를 이용한 의존성 주입
- ✓ <property> 태그를 활용
- ✓ name 속성에 Bean(객체)의 필드명을 작성
- ✓ String을 포함한 기본 타입은 value 속성으로 주입
- ✓ String을 제외한 참조 타입은 ref 속성으로 주입

기본 타입은 value 속성

```
<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>
```

appCtx.xml

injection

```
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;
    }
}
```

Setter Injection

- Setter Injection
 - ✓ Setter를 이용한 의존성 주입
 - ✓ <property> 태그를 활용
 - ✓ name 속성에 Bean(객체)의 필드명을 작성
 - ✓ String을 포함한 기본 타입은 value 속성으로 주입
 - ✓ String을 제외한 참조 타입은 ref 속성으로 주입

참조 타입은 ref 속성

```
<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>
```

appCtx.xml

injection

```
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;
    }
}
```

Constructor Injection

- Constructor Injection
 - ✓ Constructor(생성자)를 이용한 의존성 주입
 - ✓ <constructor-arg> 태그를 활용
 - ✓ String을 포함한 기본 타입은 value 속성으로 주입
 - ✓ String을 제외한 참조 타입은 ref 속성으로 주입

생성자의 매개변수와
순서가 일치해야 함

```
<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>
```

appCtx.xml

injection

```
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;  
    }  
}
```


Annotation 기반 DI

- XML 기반의 DI의 한계
 - ✓ 스프링 빈 설정 파일(Spring Bean Configuration File)에 생성해야 할 Bean을 모두 등록시켜 놓는 방식
 - ✓ 개발자가 의존 관계를 파악하여 Setter Injection 또는 Constructor Injection 방식으로 주입해야 함
 - ✓ 규모가 큰 프로젝트인 경우 생성해야 할 Bean의 개수가 수백개에 이를 수 있는데 이런 경우 XML을 이용해서 Bean 사이의 의존 관계를 파악하여 주입하는 것이 현실적으로 매우 어려움
 - ✓ 개발자가 의존 관계를 일일이 설정하지 않아도 자동으로 설정되는 각종 애너테이션이 개발되어 있음
- 주요 Annotation 종류

애너테이션	의미
@Autowired	<ul style="list-style-type: none">• Bean의 타입(class 속성)이 일치하는 객체를 할당함• 동일한 타입이 여러 개 있는 경우 Bean의 이름이 일치하는 객체를 할당함• 필드, 생성자, setter에서 사용 가능• 생성자에서는 @Autowired 생략 가능• org.springframework.beans.factory.annotation.Autowired
@Qualifier	<ul style="list-style-type: none">• Bean의 이름(id 속성)이 일치하는 객체를 할당함• 동일한 타입의 Bean이 여러 개 있는 경우 @Autowired를 이용해서 할당할 수 없으므로 @Qualifier를 추가하여 Bean을 구분할 수 있음• org.springframework.beans.factory.annotation.Qualifier
@Inject	<ul style="list-style-type: none">• Bean의 타입(class 속성)이 일치하는 객체를 할당함• javax.inject.Inject

Component Scan

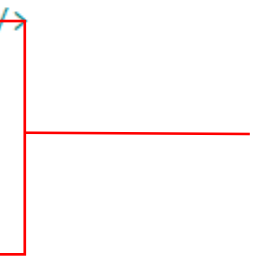
- Component Scan
 - ✓ 특정 패키지에 위치한 클래스를 Bean으로 자동 등록하는 기능
 - ✓ @Component가 추가된 클래스를 Bean으로 자동 등록함

➤ XML에서 Component Scan 등록

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

➤ Java에서 Component Scan 등록

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



```
src/main/java  
└─ spring.component.scan  
    └─ Member.java
```

➤ 자동 등록 대상

```
@Component  
public class Member {  
  
    private String id;  
    private String pw;  
}
```

@Component 상속 관계

- @Component을 상속 받는 Annotation
 - @Component에 의해서 생성된 Bean이 어떤 역할을 수행하는지 알기 쉽게 정리해 놓은 Annotation
- 주요 Annotation 종류

애너테이션	의미
@Controller	<ul style="list-style-type: none">• 요청과 응답을 처리하는 Controller 클래스에서 사용• Spring MVC 아키텍처에서 자동으로 Controller로 인식됨
@Service	<ul style="list-style-type: none">• 비즈니스 로직을 처리하는 Service 클래스에서 사용• Service Interface를 구현하는 ServiceImpl 클래스에서 사용
@Repository	<ul style="list-style-type: none">• 데이터베이스 접근 객체(DAO)에서 사용• 데이터베이스 처리 과정에서 발생하는 예외를 변환해주는 기능을 포함함