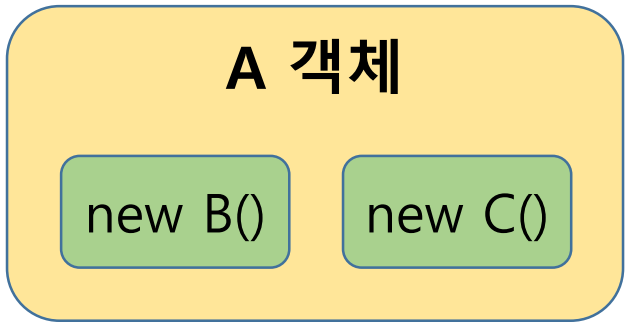
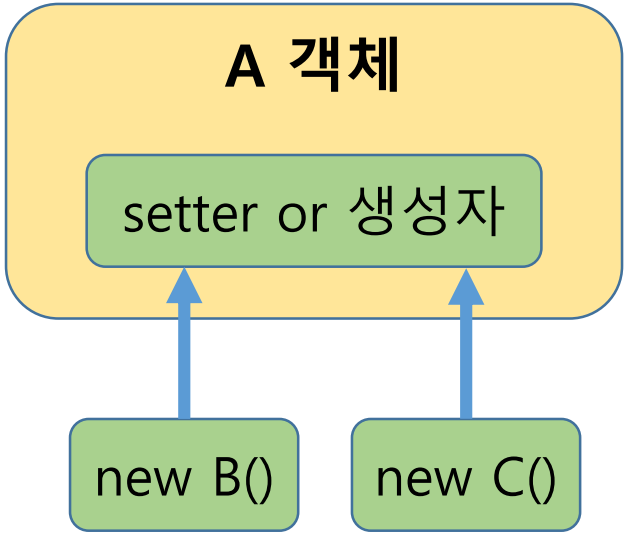
Spring DI

Spring DI(Dependency Injection)란 스프링이 제공하는 의존 관계 주입 기능으로, 객체를 직접 생성하는 게 아니라 외부에서 생성한 후 주입 시켜주는 방식이다.
DI(의존성 주입)를 통해서 모듈 간의 결합도가 낮아지고 유연성이 높아진다.

첫번째 방식



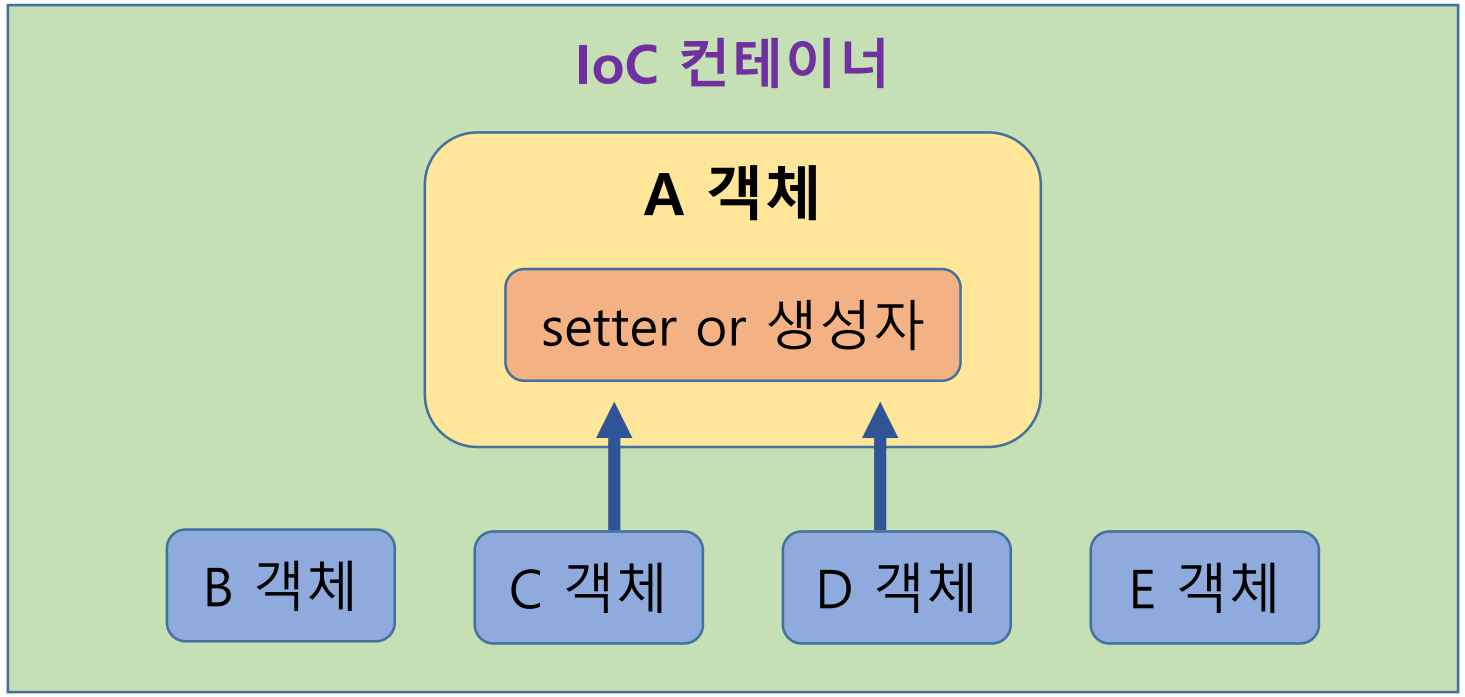
두번째 방식



첫번째 방식은 A객체가 B와 C객체를 new 연산자를 사용하여 직접 생성하는 방식이고, 두번째 방식은 외부에서 생성 된 객체를 setter 메소드 또는 생성자를 통하여 객체를 얻은 후 사용하는 방식이다.

DI(Dependency Injection)

Spring 의 DI는 두번째 방식을 택한다. A 객체에서 B, C등의 객체를 사용(의존)할 때 A 객체에서 직접 생성 하는 것이 아니라 외부(IoC컨테이너)에서 생성된 B, C객체를 주입(조립)시켜 setter or 생성자를 통해 사용하는 방식이다.



IoC (Inversion of Control)

IoC (Inversion of Control) 란 제어반전을 뜻하고 있다. 제어 반전이란, 객체의 생성, 생명주기의 관리까지 모든 객체에 대한 제어권이 바뀌었다는 것을 의미한다.

컨테이너란? 컨테이너는 보통 객체의 생명주기를 관리, 생성된 인스턴스들에게 추가적인 기능을 제공하도록 하는 것을 말한다.

IoC 컨테이너란? 스프링 프레임워크도 객체를 생성하고 관리하고 책임지고 의존성을 관리해주는 컨테이너가 있는데, 그것이 바로 IoC 컨테이너 이다.

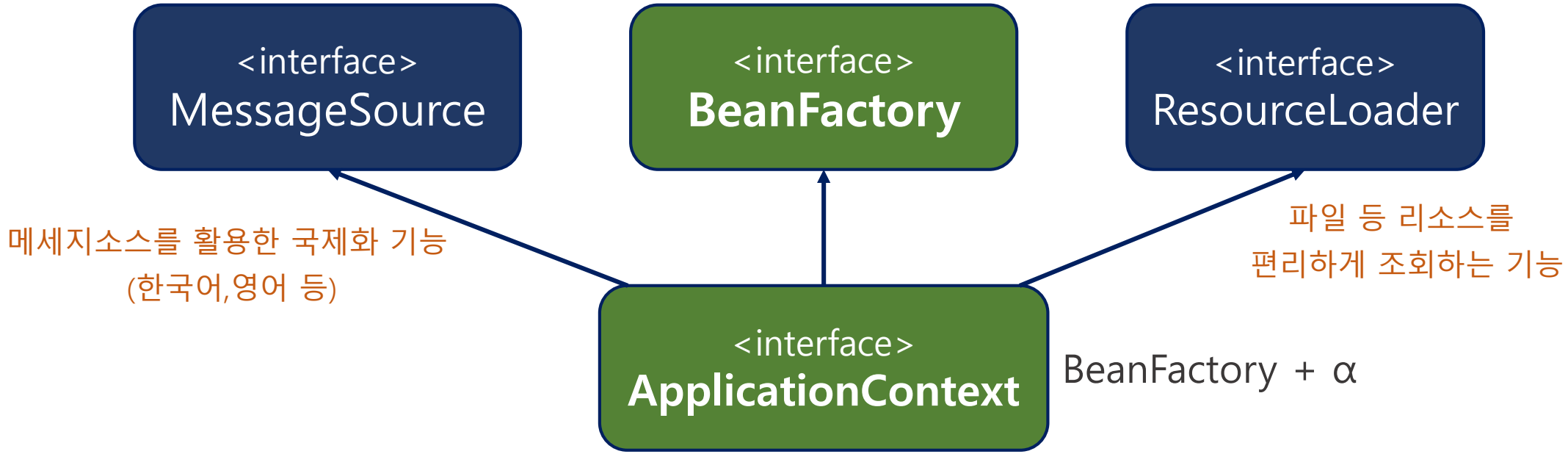
스프링 IoC 컨테이너

인스턴스 생성부터 소멸까지의 생명주기 관리를 IoC 컨테이너가 해준다.
객체관리 주체가 Ioc 컨테이너(Container)가 되기 때문에 개발자는 로직에 집중할 수 있는 장점이 있다.

◈ IoC 컨테이너는 객체의 생성을 책임지고, 의존성을 관리한다.
  의존성 주입은 IoC 컨테이너에 빈으로 등록이 되어야 의존성 주입을 할 수 있다.

◈ POJO의 생성, 초기화, 서비스, 소멸에 대한 권한을 가진다.
  보통 개발자들이 직접 POJO를 생성할 수 있지만 컨테이너에게 맡긴다.

스프링 컨테이너 종류



BeanFactory 는 스프링 컨테이너의 최상의 인터페이스로 스프링 빈을 관리하고 조회하는 역할을 담당한다. ApplicationContext 는 BeanFactory 를 상속받아 BeanFactoy의 모든 기능과 추가적인 부가기능을 제공한다.

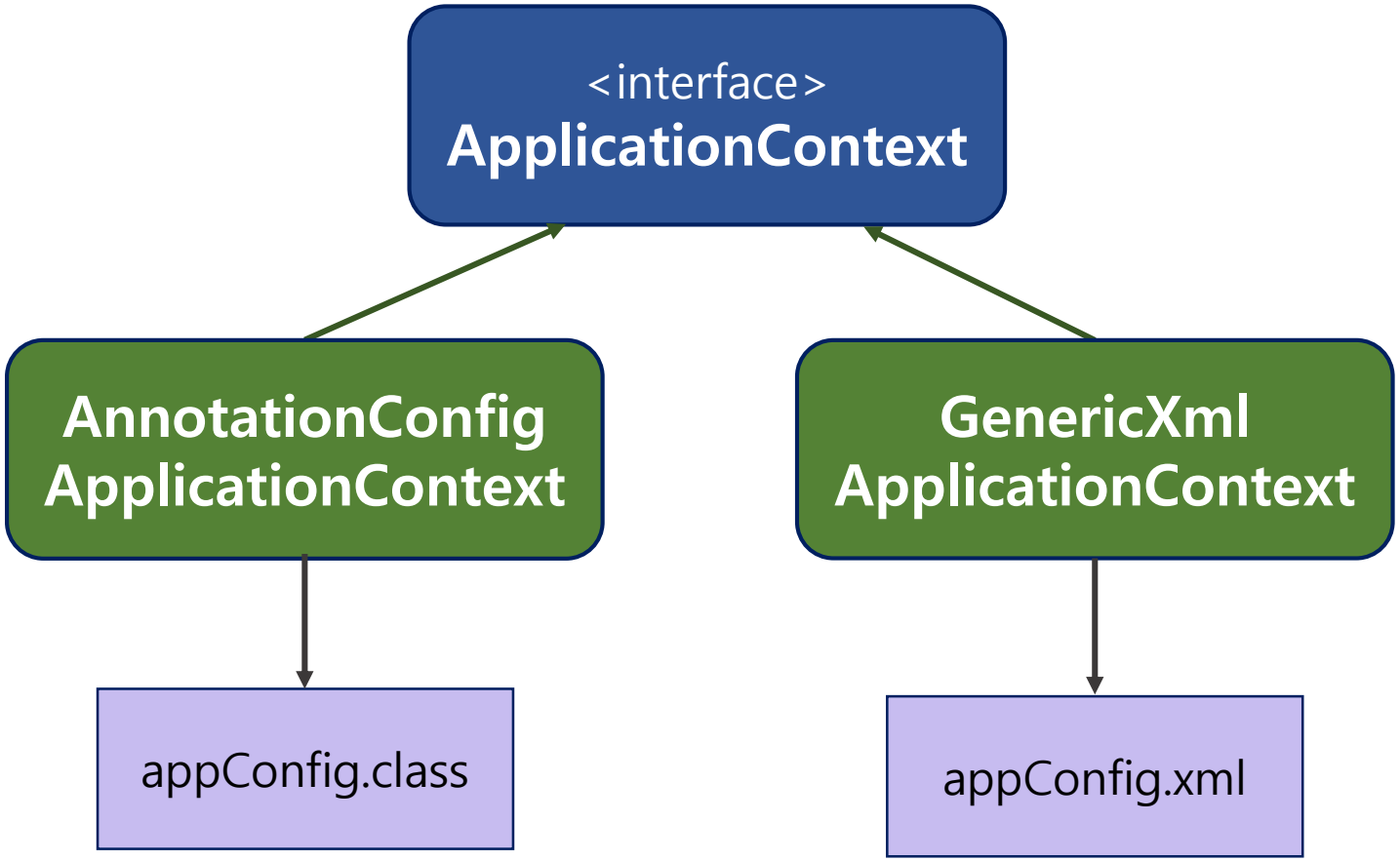
스프링 컨테이너 종류

스프링은 두가지 타입의 컨테이너를 제공한다.

BeanFactory 를 구현한 클래스로 컨테이너에 접근할 수 있고, ApplicationContext 를 구현한 클래스로 컨테이너에 접근할 수 있다. 하지만, 보통 BeanFactory를 구현한 클래스로는 컨테이너에 접근하지 않는다.

BeanFactory 외에 다른 인터페이스들을 상속받으면서 더 많은 기능을 지원하는 ApplicationContext 를 구현한 클래스로 컨테이너에 접근하도록 한다.

스프링 컨테이너 종류



AnnotationConfigApplicationContext 는 AnnotatedBeanDefinitionReader 를 사용해서 AppConfig.class 파일을 읽고 BeanDefinition 을 생성한다.

GenericXmlApplicationContext 는 XmlBeanDefinitionReader 를 사용해서 appConfig.xml 설정 정보를 읽고 BeanDefinition 을 생성한다.

객체 생성 및 조립

```java
public class Student {

    private int studentID;
    private String name;

    public Student() {

    }

    public int getStudentID() {
        return studentID;
    }

    public void setStudentID(int studentID) {
        this.studentID = studentID;
    }

    public String getName() {
        return name;
    }


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

}
```

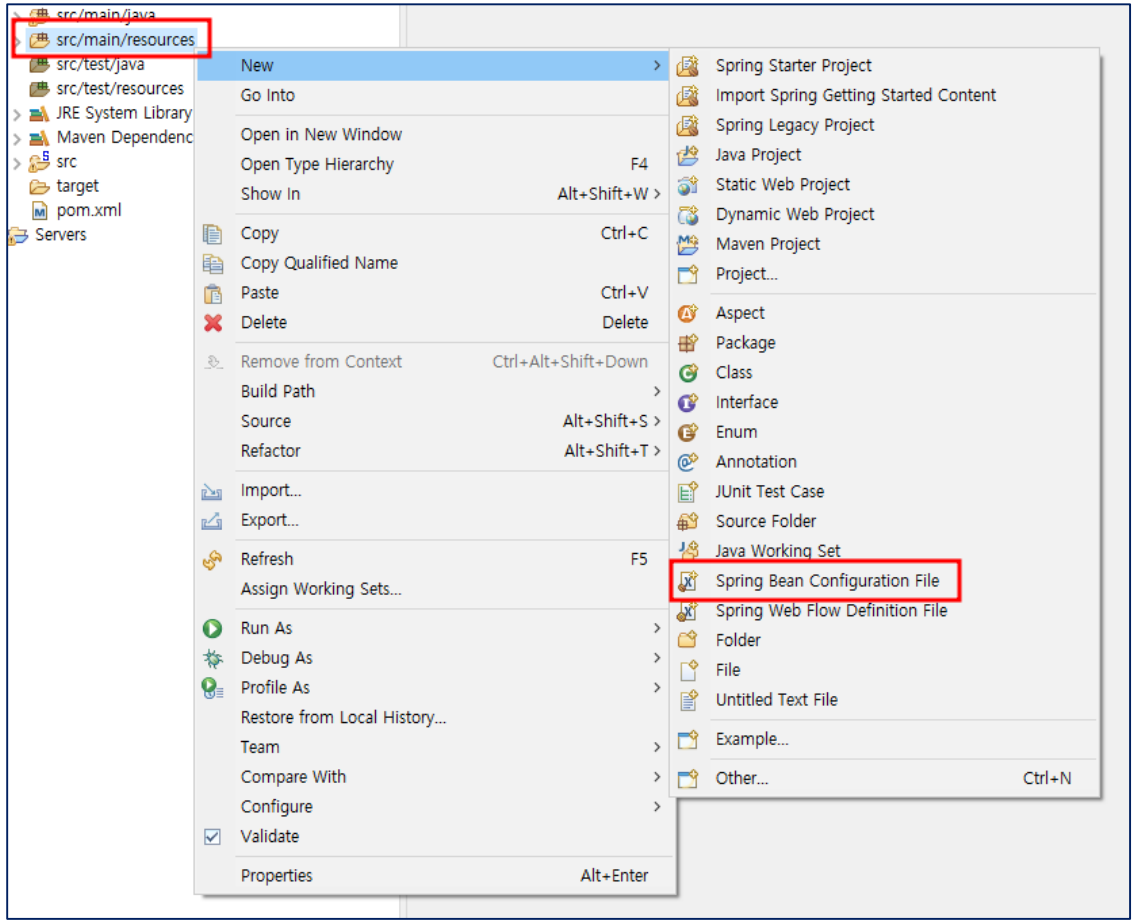
```java
public class StudentService {

    private Student student;

    public void setStudent(Student student) {
        this.student = student;
    }
```

스프링은 객체를 생성하고 조립하는 것이라고 볼 수 있다. 외부에서 생성한 객체 및 데이터를 저장할 필드를 선언을 하고 Setter 메소드와 생성자를 통하여 사용하고자 하는 객체 및 데이터의 주입을 받는다.

객체 생성 및 조립



스프링은 객체를 외부에서 생성하여 주입하는 방식을 사용한다. src/main/resources 폴더에 스프링 설정파일을 만들도록 한다.

객체 생성 및 조립

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">


    <bean id="student" class="com.spring.di.Student">
        <property name="studentID">
            <value>12345</value>
        </property>

        <property name="name" value="홍길동" />

    </bean>


    <bean id="studentService" class="com.spring.di.StudentService">
        <property name="student">
            <ref bean="student"/>
        </property>
    </bean>


</beans>
```

스프링 설정 파일에서 <bean> 태그를 사용하여 해당 빈 객체를 생성한다.

id 속성에는 빈 객체 이름을 명시하고 class 속성은 패키지를 포함하여 해당 빈 클래스의 완전한 이름을 명시한다.

<property> 태그를 사용하여 빈 객체의 속성에 객체 및 데이터를 주입하여 객체의 기능을 사용하도록 한다.

객체 생성 및 조립

```java
public class StudentMain {

    public static void main(String[] args) {


        GenericXmlApplicationContext context =
                new GenericXmlApplicationContext("classpath:applicationContext.xml");

        StudentService service = context.getBean("studentService", StudentService.class);

        service.studentProfile();

        context.close();

    }

}
```

GenericXmlApplicationContext 클래스의 생성자에 설정파일의 이름을 문자열로 주어 스프링 설정파일을 읽어와 컨테이너를 생성한다. 스프링 컨테이너에서 빈 객체를 얻을 때는 getBean 메소드로 빈 객체를 얻을 수 있다.

의존 자동주입 @Autowired

Setter 메소드와 생성자를 통해서 의존성을 주입을 컨테이너에서 직접 할 수 있지만, 스프링은 의존 자동주입 기능이 있다. 자동 주입 기능을 사용하면 스프링이 알아서 의존 객체를 찾아서 주입한다. 자동주입 기능을 사용하는 것은 간단하다.

의존 주입 대상에 @Autowired 어노테이션을 사용하여 주입할 수 있다.

@Autowired 는 필요한 의존 객체의 "타입" 에 해당하는 빈을 찾아 주입한다.

(필드, 생성자, setter 메소드) 주입이 있다.

의존 자동주입 @Autowired

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www
        http://www.springframework.org/schema/context http://www.springframew

    <context:annotation-config />
```

스프링 컨테이너에 등록된 bean에 대해서 Annotation을 활성화해주는 태그이다.

@Autowired 및 @Qualifier Annotation을 해석해서 자동 주입을 하도록 한다.

@Autowired 필드 주입

```
@Autowired
private Member member;
```

필드주입은 멤버필드에 @Autowired 어노테이션을 선언하여 주입받는 방법이다.

◈ 가장 간단한 선언 방식이다.

하지만 의존 관계가 보이지 않아 추상적이고, 이로 인해 의존성 관계가 복잡해질 수 있다. 또한 의존성 주입 대상 필드에 final 선언이 불가능하다.

@Autowired 생성자 주입

생성자 주입은 생성자에 의존성 주입을 받고자 하는 필드에 객체생성시 생성자에 주입받는 방법으로, 권고되는 방법 이다.

- ◈ 해당하는 빈객체 없이 인스턴스를 만들지 못하도록 강제한다.
- ◈ Spring 4.3 이상부터 생성자가 하나일때 @Autowired를 사용하지 않아도 된다.
- ◈ 의존성 주입 대상의 필드를 final 로 선언할 수 있다.

순환 참조로 주입을 할 경우는 생성자 주입으로 해결하기 어렵기 때문에 이러한 경우에는 나머지 주입 방법 중에 하나를 사용한다.

@Autowired 메소드 주입

메소드 주입은 메소드에 @Autowired 어노테이션을 선언하여 주입받는 방법이다.

◈ 의존성이 선택적으로 필요한 경우에 사용한다.
◈ 의존성 주입대상을 선택적으로 나눠서 주입 할 수 있게 부담을 덜어준다.

의존성 주입 대상 필드가 final 선언 불가능하다. 따라거 생성자 주입 방법과 Setter 주입 방법을 적절하게 상황에 맞게 분배하여 사용하도록 한다.