

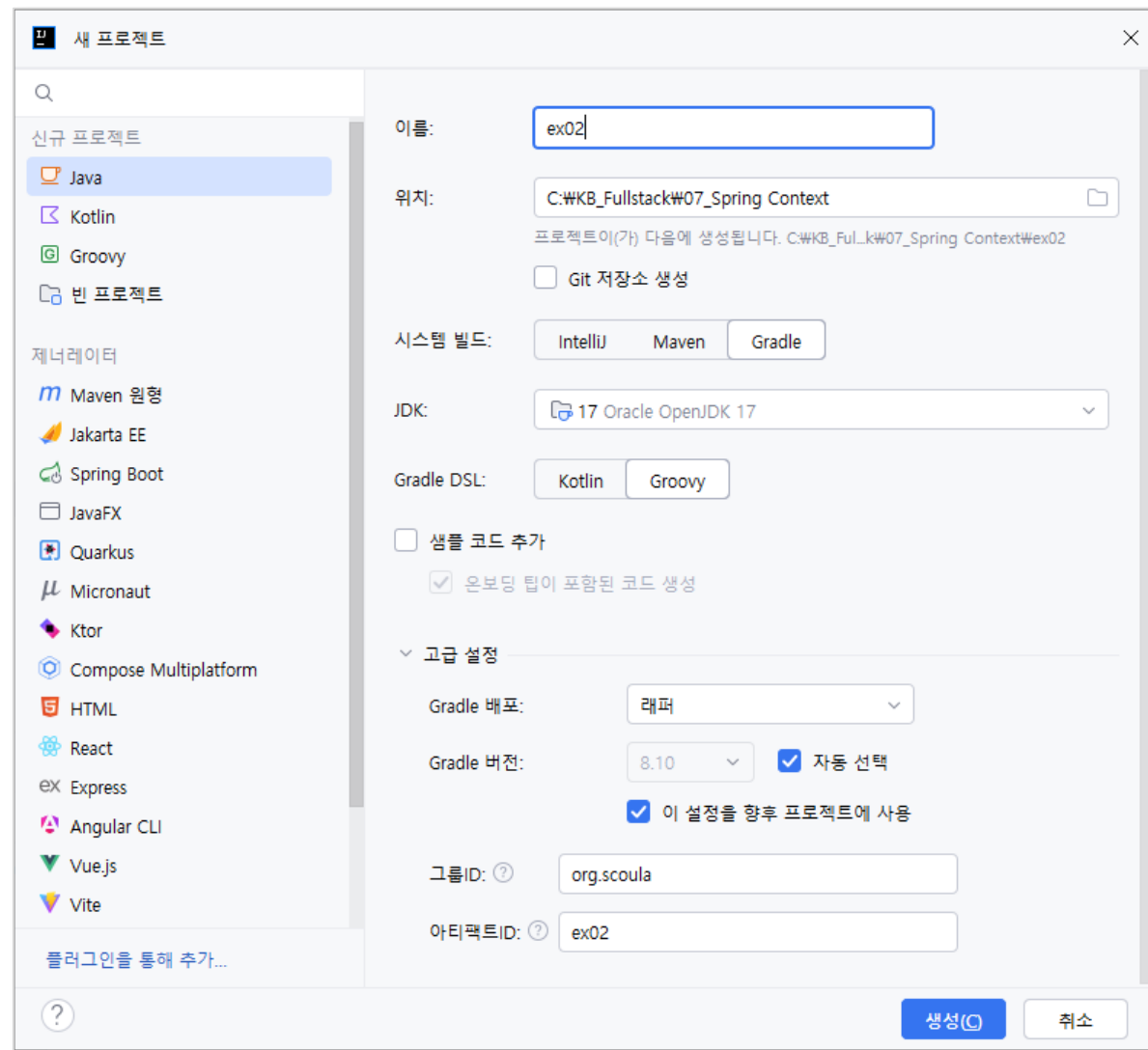
2025년 상반기 K-디지털 트레이닝

스프링 컨텍스트- 빈 작성

[KB] IT's Your Life

✓ 프로젝트 생성

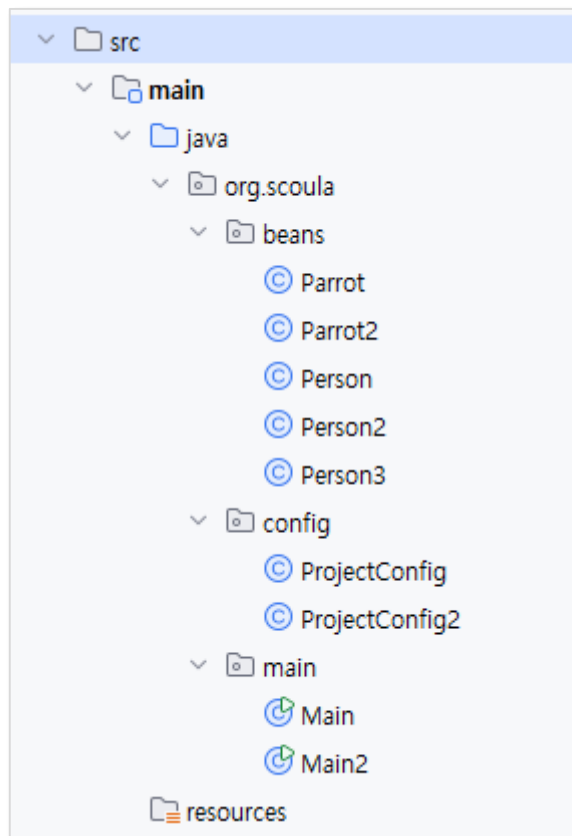
○ 프로젝트명: ex02



build.gradle

```
plugins {  
    id 'java'  
}  
  
group = 'org.scoula'  
version = '1.0-SNAPSHOT'  
  
repositories {  
    mavenCentral()  
}  
  
dependencies {  
    implementation 'org.springframework:spring-context:5.3.37'  
    implementation 'javax.annotation:javax.annotation-api:1.3.2'  
  
    testImplementation platform('org.junit:junit-bom:5.10.0')  
    testImplementation 'org.junit.jupiter:junit-jupiter'  
}  
  
test {  
    useJUnitPlatform()  
}
```

✓ 실습 환경

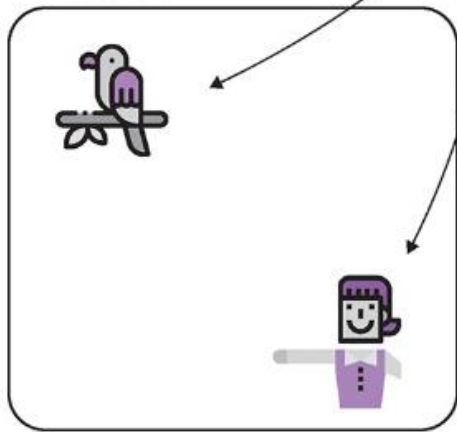


✔ 스프링 빈 간의 관계 설정

how?

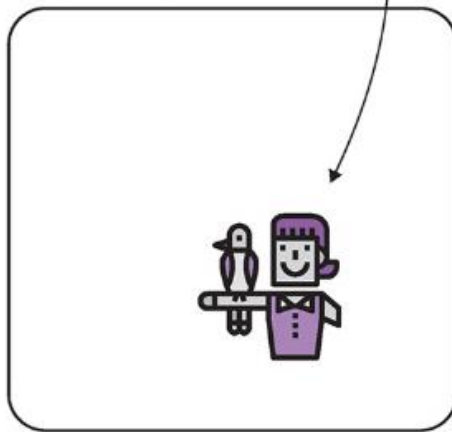
1단계:
스프링 컨텍스트에 앵무새와
사람을 빈으로 추가한다.

스프링 컨텍스트

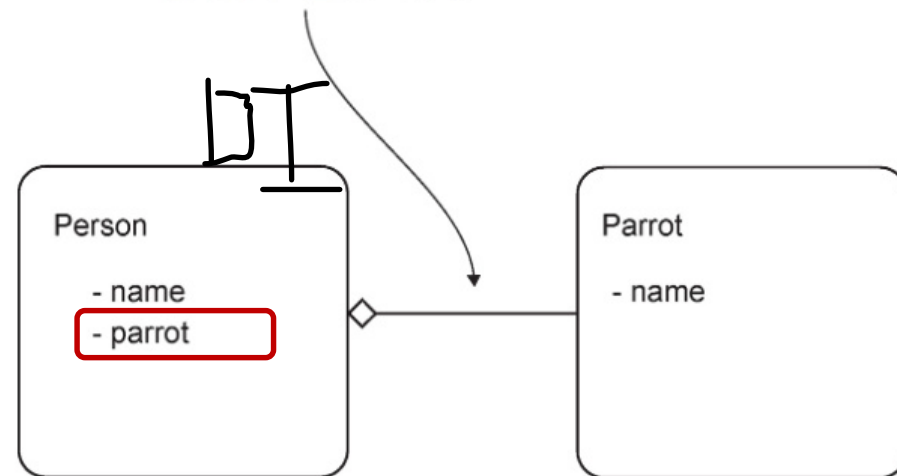


2단계:
사람이 앵무새를
소유하도록 설정한다.

스프링 컨텍스트



사람이 앵무새를 갖고 있다.



beans.Parrot.java

```
package org.scoula.beans;

public class Parrot {
    private String name;

    public String getName() {
        return name;
    }

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

    @Override
    public String toString() {
        return "Parrot : " + name;
    }
}
```

beans.Person.java

```
package org.scoula.beans;
```

```
public class Person {
```

의존성 유발 필드

```
    private String name;
```

```
    private Parrot parrot;
```

의존성.

```
    public String getName() {
```

```
        return name;
```

```
    }
```

의존성 주입 생산자

```
    public void setName(String name) {
```

```
        this.name = name;
```

```
    }
```

```
    public Parrot getParrot() {
```

```
        return parrot;
```

```
    }
```

```
    public void setParrot(Parrot parrot) {
```

```
        this.parrot = parrot;
```

```
    }
```

의존성 주입 setter

```
}
```

config.ProjectConfig.java

```
package org.scoula.config;

import org.scoula.beans.Parrot;
import org.scoula.beans.Person;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ProjectConfig {

    @Bean
    public Parrot parrot() {
        Parrot p = new Parrot();
        p.setName("Koko");
        return p;
    }

    @Bean
    public Person person() {
        Person p = new Person();
        p.setName("Ella");
        return p;
    }
}
```


main.Main.java

```
package org.scoula.main;

import org.scoula.beans.Parrot;
import org.scoula.beans.Person;
import org.scoula.config.ProjectConfig;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Main {

    public static void main(String[] args) {
        var context = new AnnotationConfigApplicationContext(ProjectConfig.class);

        Person person = context.getBean(Person.class);

        Parrot parrot = context.getBean(Parrot.class);

        System.out.println("Person's name: " + person.getName());
        System.out.println("Parrot's name: " + parrot.getName());

        System.out.println("Person's parrot: " + person.getParrot());
    }
}
```

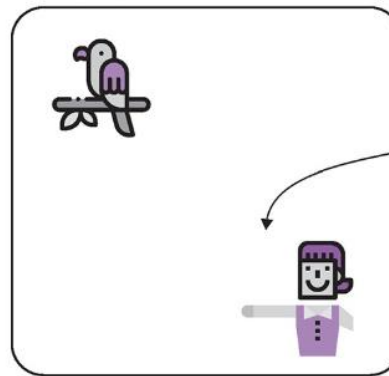
Person's name: Ella

Parrot's name: Koko

Person's parrot: null

✓ 관계 안 생김!

스프링 컨텍스트



사람이 아직 앵무새를 소유하고 있지 않다.

두 빈은 컨텍스트에 있지만
서로 연결되어 있지 않다.

- ✓ 두 @Bean 메서드 간 직접 메서드를 호출하는 빈 작성

config.ProjectConfig.java

```
@Configuration
public class ProjectConfig {

    @Bean
    public Parrot parrot() {
        Parrot p = new Parrot();
        p.setName("Koko");
        return p;
    }

    @Bean
    public Person person() {
        Person p = new Person();
        p.setName("Ella");
        p.setParrot(parrot()); ✓
        return p;
    }
}
```

메서드를 호출해서 패럿 인스턴스를 얻음

Person's name: Ella
Parrot's name: Koko
Person's parrot: Parrot : Koko ✓

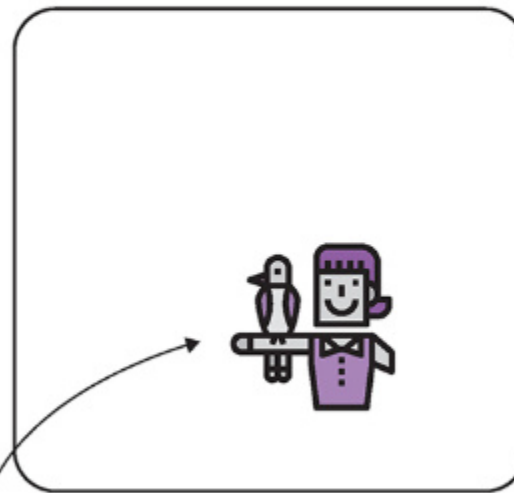
```
@Configuration  
public class ProjectConfig {
```

```
    @Bean  
    public Parrot parrot() {  
        Parrot p = new Parrot();  
        p.setName("Koko");  
        return p;  
    }
```

```
    @Bean  
    public Person person() {  
        Person p = new Person();  
        p.setName("Ella");  
        p.setParrot(parrot());  
        return p;  
    }
```

설정하려는 빈을 반환하는 메서드를
직접 호출하여 사람 빈과 앵무새 빈 간
관계를 정의한다.

스프링 컨텍스트



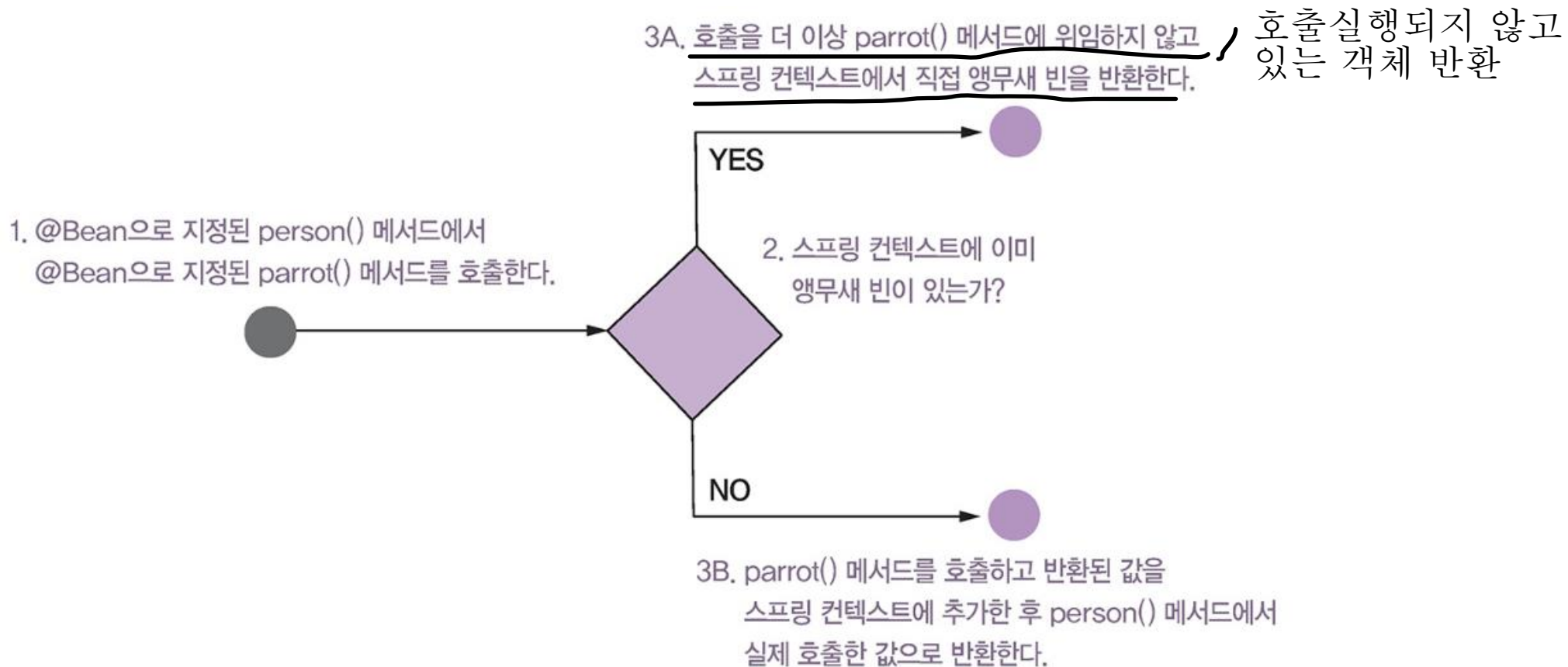
그 결과 두 빈 사이에 has-A 관계가 생성되었다.
사람은 앵무새를 소유하고 있다.

빈 메서드는
독특함.
바로 호출되지 않음.

먼저 컨텍스트에
해당 메소드로 인해
등록된 객체가 이미
등록되어있는지
확인부터한다.

저렇게 호출하면
그냥 호출의미가 아니라
이 메소드로 인해서
생긴 객체를 반환하라는
뜻.

컨텍스트에 등록된 객체가
반환된다. 이미 있다면 호출실행될 필요 없이. 싱글톤.



1. 스프링은 parrot() 메서드를 호출하여 앵무새 빈을 생성하고 컨텍스트에 추가한다.

스프링

```
@Configuration
public class ProjectConfig {
```

```
@Bean
```

```
public Parrot parrot() {
    Parrot p = new Parrot();
    p.setName("Koko");
    return p;
}
```

```
@Bean
```

```
public Person person() {
    Person p = new Person();
    p.setName("Ella");
    p.setParrot(parrot());
    return p;
}
```

2. 스프링은 person() 메서드를 호출하여 사람 빈을 생성하고 컨텍스트에 추가한다.

3. 스프링이 사람 빈을 생성할 때
여기에서 두 번째 Parrot 인스턴스가
생성된다는 의미일까? 아니!

✓ @Bean 메서드의 매개변수로 빈 와이어링하기

2번째
방법

```
@Configuration  
public class ProjectConfig {
```

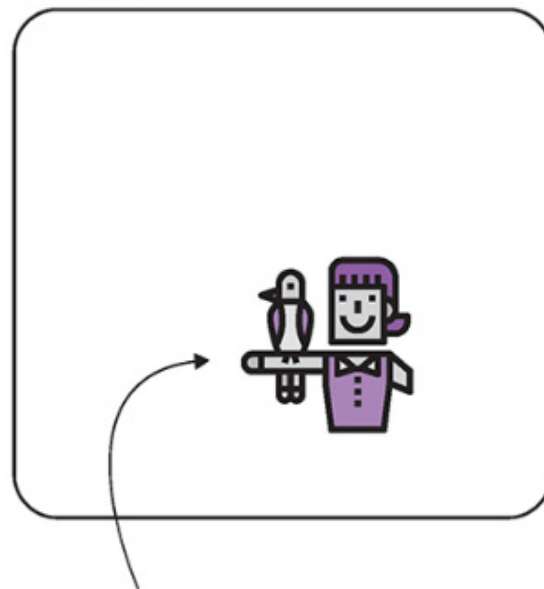
```
    @Bean  
    public Parrot parrot() {  
        Parrot p = new Parrot();  
        p.setName("Koko");  
        return p;  
    }
```

```
    @Bean  
    public Person person(Parrot parrot) {  
        Person p = new Person();  
        p.setName("Ella");  
        p.setParrot(parrot);  
        return p;  
    }  
}
```

스프링이 전달한 참조로
사람의 속성 값을 설정한다.

메서드에 대한 매개변수를 정의하여
스프링 컨텍스트에서 빈을 제공하도록
스프링에 지시한다.

스프링 컨텍스트



그 결과 두 빈 사이에 has-A 관계가 생성된다.
사람은 앵무새를 소유하고 있다.

저렇게 매개변수처럼
의존성이 필요하다
라고만 코딩해도

특별한 추가 행동 없이
알아서
저렇게만 표시한것만
으로도
스프링이 알아서
컨텍스트에있는
DI후보를 필요한 곳에
DI해준다!

config.ProjectConfig.java

```
@Configuration
public class ProjectConfig {

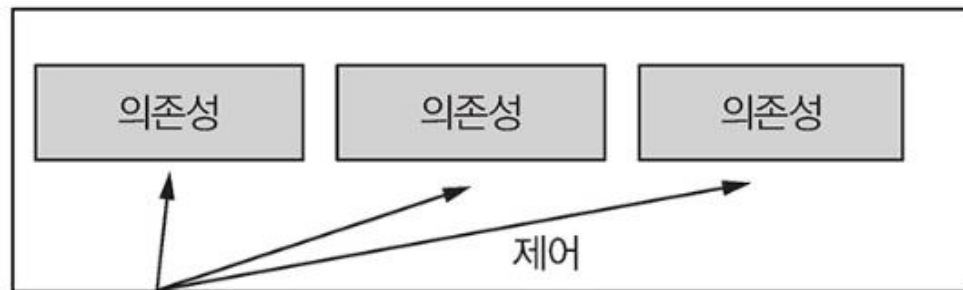
    @Bean
    public Parrot parrot() {
        Parrot p = new Parrot();
        p.setName("Koko");
        return p;
    }

    @Bean
    public Person person(Parrot parrot) {
        Person p = new Person();
        p.setName("Ella");
        p.setParrot(parrot);
        return p;
    }
}
```

Person's name: Ella
Parrot's name: Koko
Person's parrot: Parrot : Koko

DI시

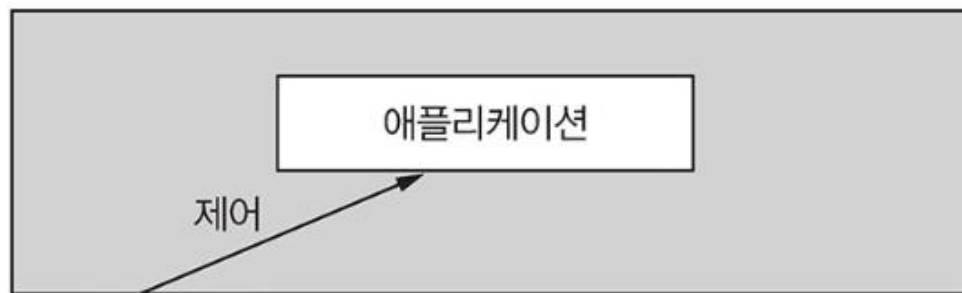
제어의 역전.
내가 제어X 외부에서 제어를 주입
IoC를 사용하지 않을 때



애플리케이션은 필요한 의존성을
실행하고 제어(사용)한다.

inversion of control

IoC를 사용할 때



애플리케이션은 프레임워크(의존성)로
제어되어 실행된다.

프레임워크(의존성)

@Autowired 애너테이션을 사용한 빈 주입

✓ @Autowired로 클래스 필드를 이용한 값 주입

```
@Component  
public class Person {  
  
    private String name = "Ella";
```

스테레오타입 애너테이션인 @Component는 스프링이 이 클래스(Person) 타입의 빈을 생성하고 추가하도록 지시한다.

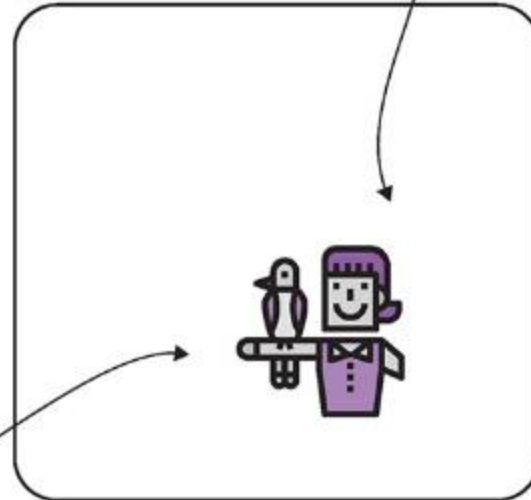
@Bean 자동화
@Bean으로
수동으로 메서드를
등록 안해도 돼

```
    @Autowired  
    private Parrot parrot;  
  
    // ...  
}
```

자동으로 엮어달라
별다른 외의 코드 없이
자동으로 의존성 주입됨
스프링이 의존성 주입함

스프링이 스프링 컨텍스트에서 빈을 가져와 @Autowired 애너테이션된 필드 값에 직접 설정하도록 지시한다. 이렇게 하면 두 빈 사이에 관계가 설정된다.

스프링 컨텍스트



beans.Parrot2.java

```
package org.scoula.beans;  
  
import org.springframework.stereotype.Component;
```

@Component 애도 등록해야지

```
public class Parrot2 {  
    private String name = "Koko"; ✓  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    @Override  
    public String toString() {  
        return "Parrot : " + name;  
    }  
}
```

beans.Person2.java

```
@Component  
public class Person2 {
```

```
    private String name = "Ella";
```

```
    @Autowired
```

```
    private Parrot2 parrot;
```

스프링에 의해서 의존성이 자동으로 채워짐.
스프링은 타입을 보고 알아서 채움. 아무 지시 사항 없을땐. 타입을 따르는게 디폴트

```
    public String getName() {  
        return name;  
    }
```

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

```
    public Parrot2 getParrot() {  
        return parrot;  
    }
```

```
    public void setParrot(Parrot2 parrot) {  
        this.parrot = parrot;  
    }
```

config.ProjectConfig2.java

```
package org.scoula.config;
```

```
import org.springframework.context.annotation.ComponentScan;
```

```
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
```

```
@ComponentScan(basePackages = "org.scoula.beans")
```

```
public class ProjectConfig2 {
```

```
}
```

✓ @Component 된 클래스들을 검색해야하는
위치알려주는 어노테이션

main.Main2.java

```
package org.scoula.main;

import org.scoula.beans.Person2;
import org.scoula.config.ProjectConfig2;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Main2 {

    public static void main(String[] args) {
        var context = new AnnotationConfigApplicationContext(ProjectConfig2.class);

        Person2 p = context.getBean(Person2.class);

        System.out.println("Person's name: " + p.getName());
        System.out.println("Person's parrot: " + p.getParrot());
    }
}
```

Person's name: Ella
Person's parrot: Parrot : Koko

✓ !!

@Autowired 애너테이션을 사용한 빈 주입

✓ @Autowired를 사용하여 생성자로 값 주입

@Component

```
public class Person {
```

```
    private String name = "Ella";
```

```
    private final Parrot parrot;
```

원래 소유하는 다른 객체 타입 필드는 final이 권장

@Autowired

```
public Person(Parrot parrot) {
```

```
    this.parrot = parrot;
```

```
}
```

```
// ...
```

```
}
```

별다른 코드 없이
person 생성할 때
생성자 호출될 때
필요한 의존성이
스프링에 의해 주입됨

스프링이 Person 타입의 빈을 생성할 때

@Autowired 애너테이션이 달린 생성자를 호출한다.

스프링은 매개변수 값으로 컨텍스트에서

Parrot 타입의 빈을 전달한다.

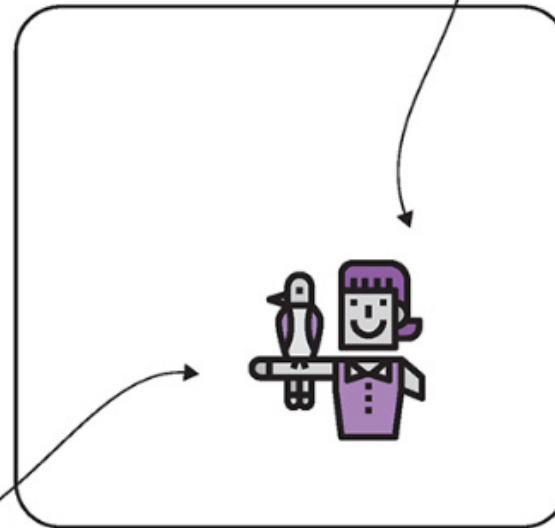
@Component 스테레오타입 애너테이션은

스프링이 이 클래스(Person) 타입의 컨텍스트에

빈을 생성하고 추가하도록 지시한다.

컨텍스트에
같은 타입의 빈을 생성하고 추가
이름은 클래스이름의 카멜케이스

스프링 컨텍스트



@Autowired 위치

1. 속성 필드 앞. final 타입이 권장인데

해당 권장시 어노테이션 못함. 옛날.

2. 생성자 => 이 타입의 매개변수 넣어달라.

전제조건 생성자가 1개만. 매개변수가 있다면

자동으로 매개변수를 통해서 DI.

3. setter. 메서드 앞. 매개변수에 넣어달라

beans.Person3.java

```
@Component
public class Person3 {

    private String name = "Ella";

    private final Parrot2 parrot; ✓

    @Autowired
    public Person3(Parrot2 parrot) { ✓
        this.parrot = parrot;
    }

    public String getName() {
        return name;
    }

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

    public Parrot2 getParrot() {
        return parrot;
    }

}
```


✓ setter를 이용한 의존성 주입 사용

```
@Component
public class Person {

    private String name = "Ella";

    private Parrot parrot; ✓

    public String getName() {
        return name;
    }

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

    public Parrot getParrot() {
        return parrot;
    }

    @Autowired
    public void setParrot(Parrot parrot) {
        this.parrot = parrot;
    }
}
```

✓
생성자처럼 setter도
의존성이 필요한게 스프링에 인된다면
dependency는 스프링이 inject한다

✓ 순환 의존성 교착상태 조심!

1. 스프링은 Parrot 타입의 빈을 생성해야 한다.

```
@Component  
public class Parrot {
```

```
    private final Person person;
```

```
    @Autowired  
    public Parrot(Person person) {  
        this.person = person;  
    }  
}
```

2. Parrot 클래스 생성자를 호출하려면 스프링은 Person 타입의 빈이 필요하다.

```
@Component  
public class Person {
```

```
    private final Parrot parrot;
```

```
    @Autowired  
    public Person(Parrot parrot) {  
        this.parrot = parrot;  
    }  
}
```

3. 스프링은 Person 타입의 빈을 생성하려고 한다.

4. Person 타입의 빈을 생성하기 위해 스프링은 Parrot 타입의 빈이 필요하다. 스프링은 이제 교착 상태에 빠졌다!

```
@Configuration  
public class ProjectConfig {
```

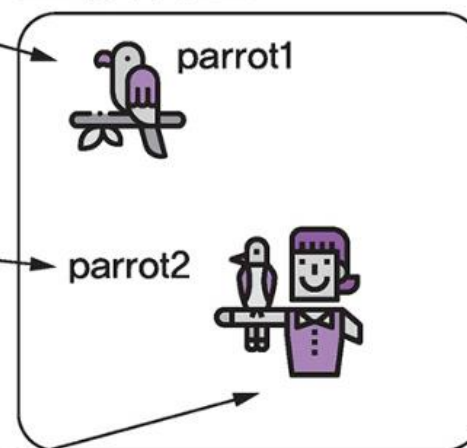
```
    @Bean  
    public Parrot parrot1() {  
        Parrot p = new Parrot();  
        p.setName("Koko");  
        return p;  
    }
```

```
    @Bean  
    public Parrot parrot2() {  
        Parrot p = new Parrot();  
        p.setName("Miki");  
        return p;  
    }
```

```
    @Bean  
    public Person person(Parrot parrot2) {  
        Person p = new Person();  
        p.setName("Ella");  
        p.setParrot(parrot2);  
        return p;  
    }  
}
```

우리는 컨텍스트에 빈 두 개를 정의한다.
이 빈 두 개는 이를 생성하는 메서드 이름인
parrot1과 parrot2를 갖는다.
또 Person 타입의 빈도 하나 정의한다.

스프링 컨텍스트



스프링은 우리가 정의한 매개변수 이름과
동일한 이름의 빈 값을 찾아 전달한다.

정확히 어느 상황에서 매개변수의
이름과 동일한 이름의 빈을 찾아
전달할까? 빈에서 빈을 찾을때?
@Component타입으로 찾을때 빼고?

같은 타입인 빈이 2개이상일때?

✓ @Qualifier로 대상 지정하기

- @Qualifier(value="빈 이름")으로 연결할 빈 이름 지정

```
public Person person(@Qualifier("parrot2") Parrot parrot) {  
    Person p = new Person();  
    p.setName("Ella");  
    p.setParrot(parrot);  
    return p;  
}
```

@Component된 클래스이름도 지정가능할까?

✓ 생성자 주입에서 대상 지정하기

```
@Component
public class Person {

    private String name = "Ella";

    private final Parrot parrot;

    public Person(Parrot parrot2) {
        this.parrot = parrot2;
    }

    public String getName() {
        return name;
    }

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

    public Parrot getParrot() {
        return parrot;
    }

}
```

```
public class Parrot {

    private String name;

    public String getName() {
        return name;
    }

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

    @Override
    public String toString() {
        return "Parrot : " + name;
    }

}
```

config.ProjectConfig.java

```
@Configuration
@ComponentScan(basePackages = "beans")
public class ProjectConfig {

    @Bean
    public Parrot parrot1() {
        Parrot p = new Parrot();
        p.setName("Koko");
        return p;
    }

    @Bean
    public Parrot parrot2() {
        Parrot p = new Parrot();
        p.setName("Miki");
        return p;
    }
}
```

@Component

```
public class Person {  
  
    private String name = "Ella";  
  
    private final Parrot parrot;  
  
    public Person(Parrot parrot2) {  
        this.parrot = parrot2;  
    }  
  
    // 코드 생략  
}
```

스프링이 Person 타입의 빈을 생성할 때
생성자의 매개변수 값도 전달해야 한다.
타입이 동일한 빈을 여러 개 사용할 수 있다면
스프링은 매개변수와 이름이 동일한 빈을 선택한다.

스프링 컨텍스트



parrot1

parrot2



```
@Component
public class Person {

    private String name = "Ella";

    private final Parrot parrot;

    public Person(@Qualifier("parrot2") Parrot parrot) {
        this.parrot = parrot;
    }

    public String getName() {
        return name;
    }

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

    public Parrot getParrot() {
        return parrot;
    }

}
```