요즘엔 스프링말고
스프링 부트를 지워해주는 추세여서

스프링 프로젝트 생성시
수동으로 해야할 일이 있음

KB금융그룹 | 행복을 금융파트너

2025년 상반기 K-디지털 트레이닝

스프링 컨텍스트- 빈 정의

[KB] IT's Your Life

현재 상황을 캡슐화ㅎ하여
현재 상황을 관리하는 객체

스프링에서의 현재 상황
==
인스턴스 관리.

DI를 위해서.
관리되는 인스턴스는 DI를 위한 후보들.

나중에 필요할 때 요구되는 이슨턴스를
dependency injection

스프링이란

스프링 사용 이유 : 어플리케이션의 로직 자체에 더 집중할 수 있도록 도와주는 프레임웤. 불가피하게 작성해야만 했던 불필요하게 복잡한 코딩 안해도 되어 복잡성을 낮춤

자바 애플리케이션 개발을 위한 경량 프레임워크

다른 프레임웤과 다르게 스프링은 어떠한 자바 엎플리케이션 개발에든 사용될 수 있다

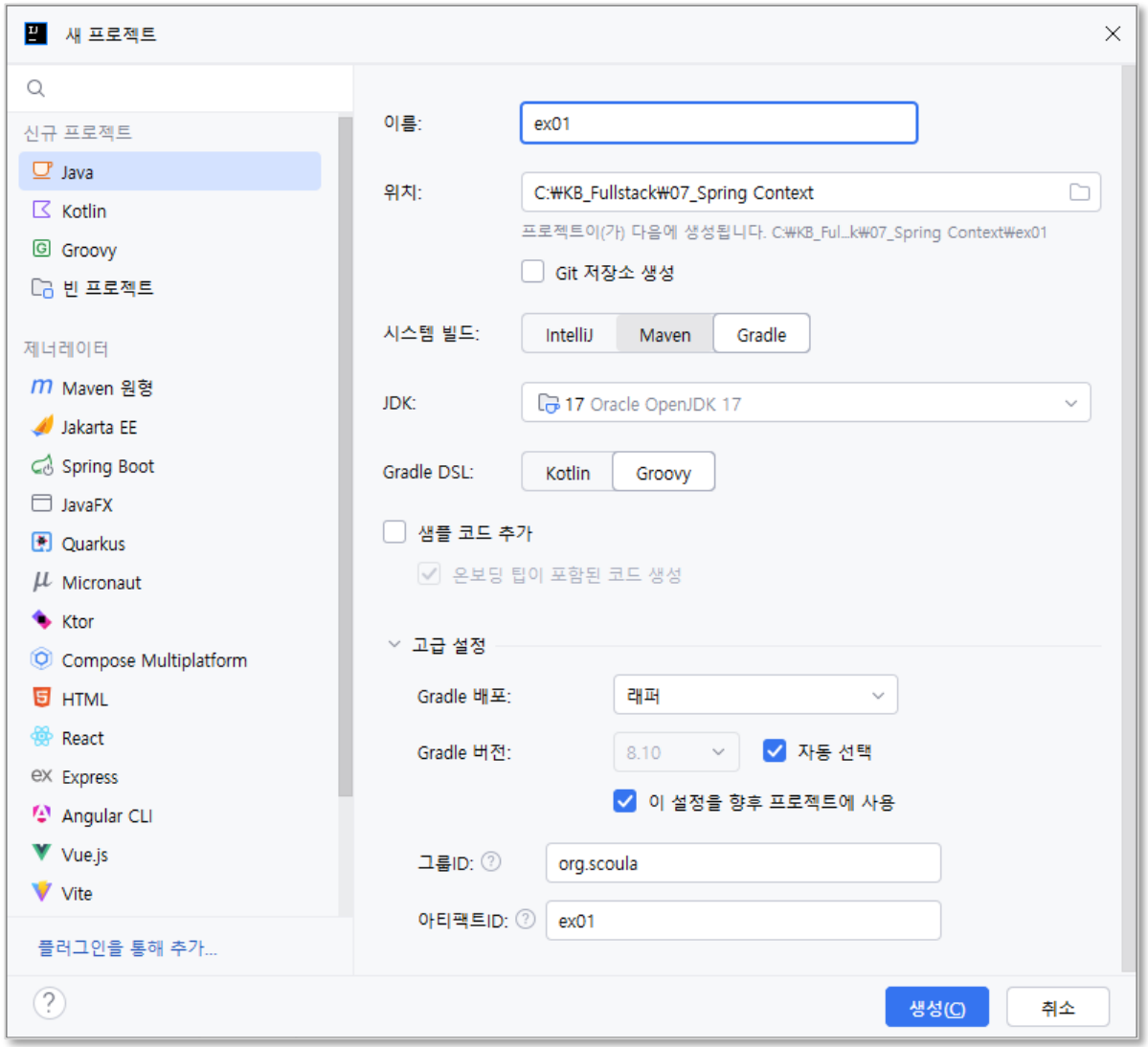
가장 최소한의 영향, 즉 스프링코어가 제공하는 다양한 기능을 사용하기 위해 아주 적은 코드 변경만이 필요하다.

핵심 2가지
제어의 역전 : 컴포넌트 의존성의 생성과 관리를 외부화
의존성이 필요할때(주입이 필요할때) 런타임에 스프링 프로세스에 의해서 주입된다.
의존성 주입 : 필요한 모든 의존성을 넣어서 인스턴스를 제공한다.
AOP와 같은 강력한 개념을 제공하기에 필수조건이다.

또다른 특징
스프링 컨테이너 설정정보를 참고로해서 앱을 구성하는 오브젝트를 생성하고 관리한다.
스프링을 사용하기 위해서 스프링 컨테이너를 다루는 방법과 설정정보를 작성하는 방법을 알아야 한다.
AOP : 코드에 이리저리 흩어져있는 부가적인 기능을 독립적으로 모듈화하는 프로그래밍 모델. 깔끔한 코드를 유지할 수 있게 도와준다
로깅, 트랜잭션, 인증과 같이 여러 클래스에 걸쳐 공통적으로 필요한 로직을 분리, 모듈화하여 BL과 부가적인 기능을 분리하여 oop의 한계를 극복한 프로그래밍 패러다임. 코드재사용성 유지보수 용이성 코드가독성.
POP 프로그래밍을 지향. Plain Old Java Object 순수 자바만을 통해서 생성한 객체. java 스펙에 정의된 기술만 사용한다는 의미.
외부기술이나 규약의 변화에 얽매이지 않는다. 객체지향 설계에 제한이 안걸린다.

KB 국민은행

프로젝트 생성


새 프로젝트
신규 프로젝트
Java
Kotlin
Groovy
빈 프로젝트
제너레이터
Maven 원형
Jakarta EE
Spring Boot
JavaFX
Quarkus
Micronaut
Ktor
Compose Multiplatform
HTML
React
Express
Angular CLI
Vue.js
Vite
플러그인을 통해 추가...
이름:
ex01
위치:
C:₩KB_Fullstack₩07_Spring Context
프로젝트이(가) 다음에 생성됩니다. C:₩KB_Ful..k₩07_Spring Context₩ex01
Git 저장소 생성
시스템 빌드:
IntelliJ
Maven
Gradle
JDK:
17 Oracle OpenJDK 17
Gradle DSL:
Kotlin
Groovy
샘플 코드 추가
온보딩 팁이 포함된 코드 생성
고급 설정
Gradle 배포:
래퍼
Gradle 버전:
8.10
자동 선택
이 설정을 향후 프로젝트에 사용
그룹ID:
org.scoula
아티팩트ID:
ex01
생성(C)
취소

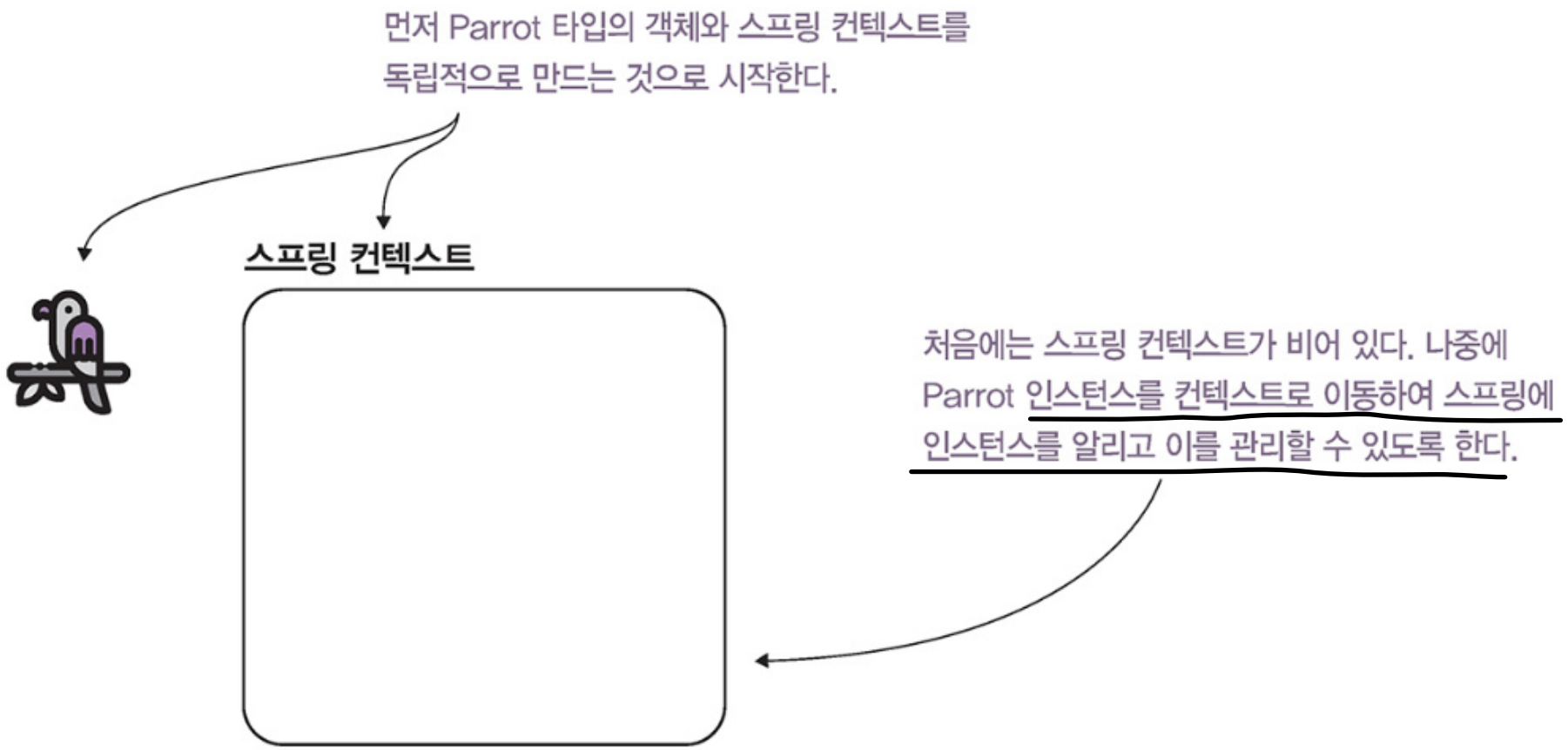
프로젝트 생성

build.gradle

```
implementation 'org.springframework:spring-context:5.3.37'
```

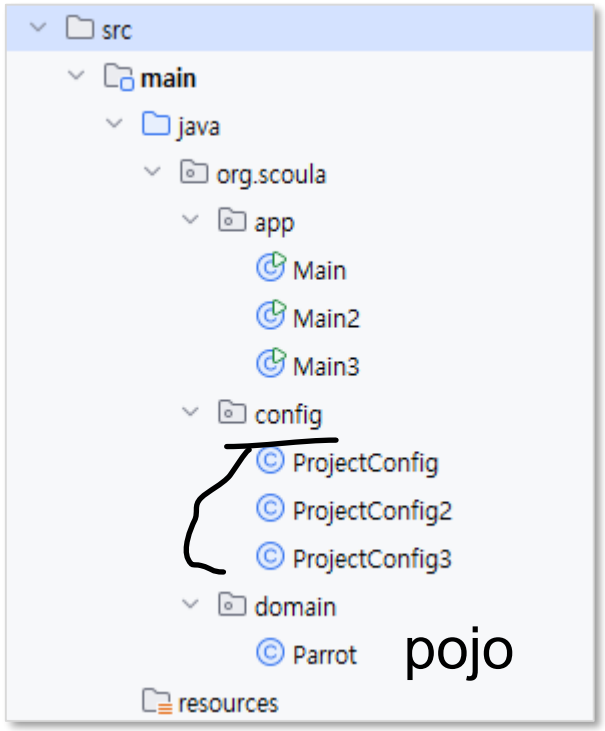
스프링 컨텍스트에 새로운 빈 추가

스프링이 하는 대표적인 일은 DI 의존성 주입이다. 스프링 프레임웍없이는 사용자가 직접 필요한 의존성(필요한 객체를) 메서드든 생성자든 통해서 주입해줘야하는데 이런거에 신경쓰면 본 로직에 집중못하고 부가적으로 할일이 많아진다. 스프링 프레임웍이 DI 의존성 주입 역할을 대신해준다.
DI의 재료 후보들을 컨텍스트라는 공간에 등록 관리시켜놓고 코드 진행에 있어서 의존성이 필요할때 컨텍스트에 있는 후보를 꺼내서 주입한다.



스프링 컨텍스트에 새로운 빈 추가

실습 환경



스프링 컨텍스트에 새로운 빈 추가

domain.Parrot.java

```
package org.scoula.domain;

public class Parrot {
    private String name;

    public String getName() {
        return name;
    }

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

}
```

디폴트 생성자가 반드시 있어야 함

스프링 컨텍스트에 새로운 빈 추가

Main.java

```
package org.scoula.app;

import org.scoula.domain.Parrot;

public class Main {

    public static void main(String[] args) {

        Parrot p = new Parrot();
    }
}
```

스프링 컨텍스트에 새로운 빈 추가

당신이 한 것

Parrot 인스턴스를 생성했지만
스프링 컨텍스트 외부에 있다.

스프링 컨텍스트

스프링 컨텍스트를 정의했지만 현재 비어 있다.

당신이 하려는 것

스프링 컨텍스트에 Parrot 인스턴스를 추가하면
스프링이 해당 인스턴스를 볼 수 있다.

스프링 컨텍스트

가지고 있다가
요청하면
뱉어줌

스프링 컨텍스트에 새로운 빈 추가

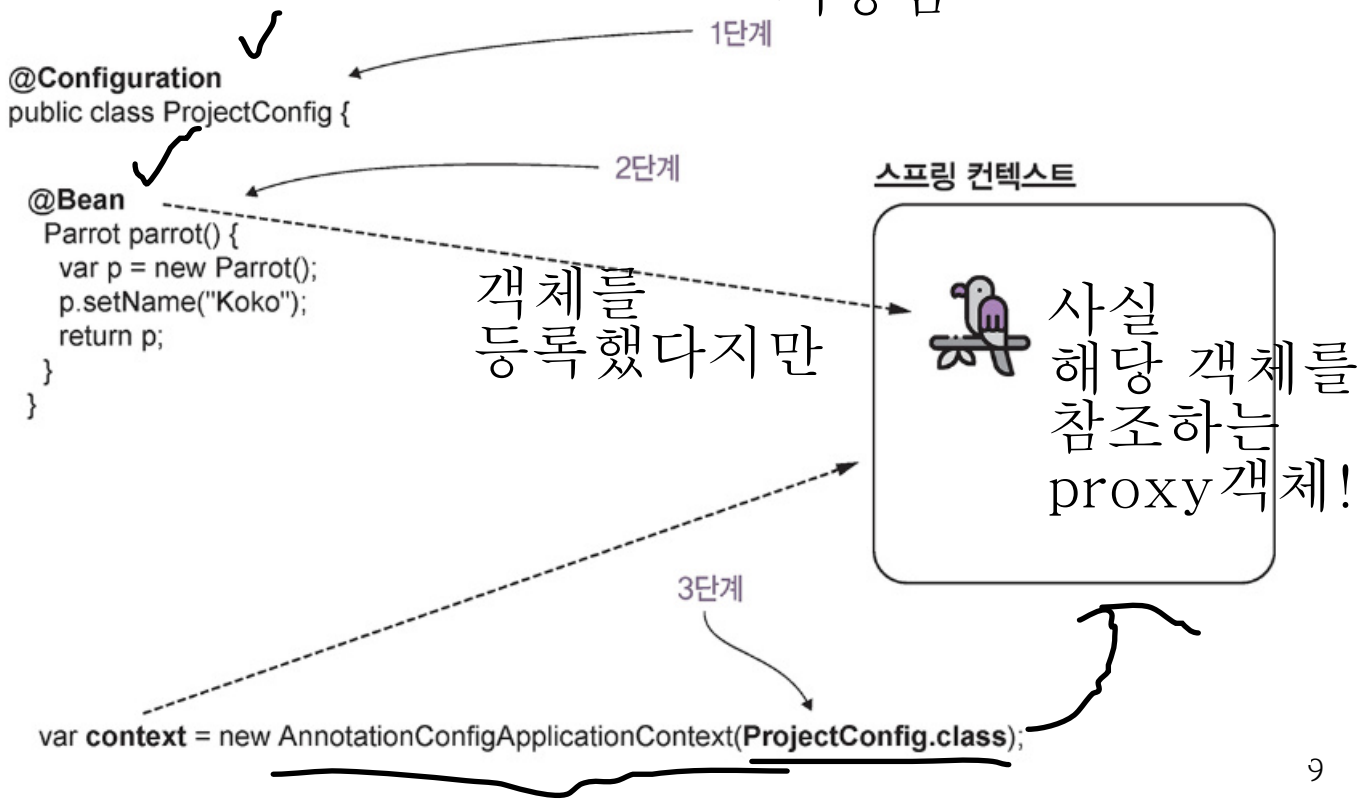
등록 방법에는
2가지가 있다.
xml기반은
미흡하고
작성하기 까다롭다
java 설정 파일로
해보자
자바 스타일로할때는
어노테이션을 많이
사용함

@Bean 애너테이션을 사용하여 스프링 컨텍스트에 빈 추가

- 스프링은 Bean으로 등록된 객체만 관리할 수 있음
- 스프링 컨텍스트에 빈을 추가하는 단계

1. @Configuration으로 구성 클래스 정의
   - 스프링 컨텍스트 구성 시 사용
2. 컨텍스트에 추가하려는 객체 인스턴스를 반환하는 메서드를 구성하는 클래스에 추가, 해당 메서드에 @Bean 애너테이션 추가
3. 스프링이 1에서 정의한 구성 클래스 사용

커피콩을 의미
객체 한 알맹이다.



스프링 컨텍스트에 새로운 빈 추가

1단계: 프로젝트에서 구성 클래스 정의하기

config.ProjectConfig.java

```
package org.scoula.config;

import org.springframework.context.annotation.Configuration;

@Configuration
public class ProjectConfig {

}
```

스프링 컨텍스트에 새로운 빈 추가

2단계: 빈을 반환하는 메서드를 생성하고 Bean 애너테이션을 메서드에 추가하기

config.ProjectConfig.java

```
package org.scoula.config;

import org.scoula.domain.Parrot;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

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

}
```

맵객체로 관리함.

키 값 쌍

키: 메서드명이 키. "parrot"
값: 인스턴스 참조

키: 타입 "Parrot.class"
값: 인스턴스 참조

프로토타입패턴을 활용할 수 있겠다

스프링 컨텍스트에 새로운 빈 추가

3단계: 새로 생성된 구성 클래스로 스프링이 컨텍스트를 초기화하도록 만들기

Main.java

```
package org.scoula.app;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Main {

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

인자로 들어간
해당 클래스의 @Bean이
붙은 메서드들을 한번씩
실행함. => 컨텍스트 만들어짐

- AnnotationConfigApplicationContext(구성클래스);
  - 구성 클래스로 컨텍스트를 만들도록 하는 클래스
  - 구성 클래스의 class를 매개변수로 지정

스프링 컨텍스트에 새로운 빈 추가

컨텍스트에서 원하는 빈 객체 추출하기

app.Main.java

```
package org.scoula.app;

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

public class Main {

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

        Parrot p = context.getBean(Parrot.class);
        System.out.println(p.getName());

    }
}
```

타입을 통해서 필요로하는 인스턴스 얻고 있음 ✓

Parrot p2=context.getBean(Parrot.class); // 같은 객체 참조 값이냐 아니면 복제품이냐. 전자이다! 주소가 같다. 해쉬코드값도

필요로 하는 객체가 있을 때마다 뱉어주는데 그게 실객체가 같다 == 싱글톤으로 운영됨.

동일한 타입인데 새로운 인스턴스를 얻고 싶을땐? 그렇게 하는 방법이 또있어. 싱글톤으로 운영이 디폴트. 프로토타입?으로 지정하면 (패턴 말하는 건가) 얻을때마다 새로운 인스턴스

스프링 컨텍스트에 새로운 빈 추가

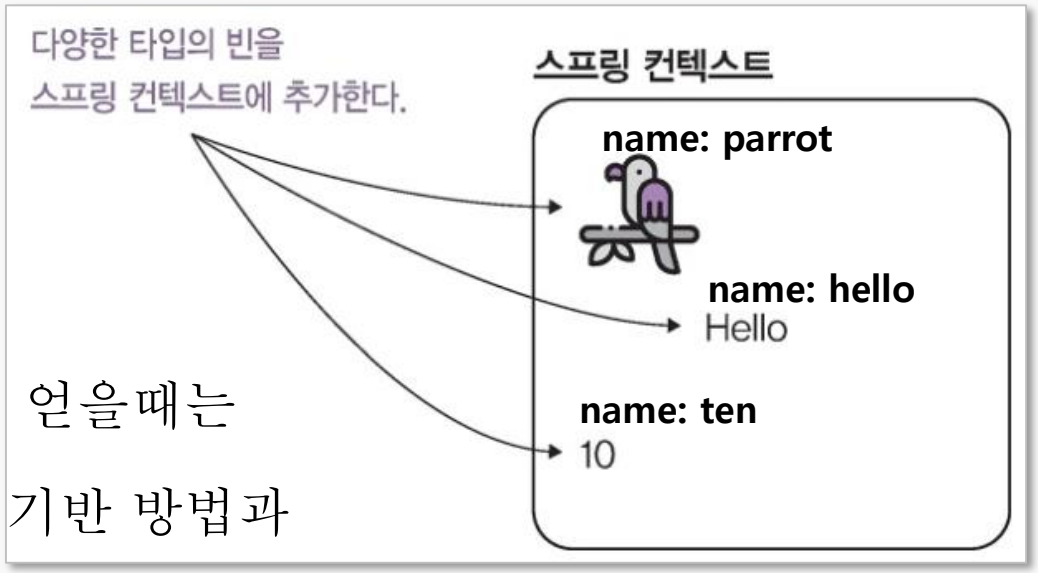
config.ProjectConfig.java

```
@Configuration
public class ProjectConfig {

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

    @Bean
    String hello() {
        return "Hello";
    }

    @Bean
    Integer ten() {
        return 10;
    }
}
```



객체 얻을때는

타입기반 방법과

이름기반 방법이
있다

스프링 컨텍스트에 새로운 빈 추가

Main.java

```
public class Main {

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

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

        System.out.println(p.getName());

        String s = context.getBean(String.class);

        System.out.println(s);

        Integer n = context.getBean(Integer.class);

        System.out.println(n);
    }
}
```

매개변수와 반환 타입이 같은 제네릭타입이다.

캐스팅 없이 바로 받아서 쓸 수 있는 이유는

매개변수 타입에 따라 반환 타입이 맞춰진다

```
Koko
Hello
10
```

- 동일한 타입에 대해서는 1개의 Bean만 추출할 수 있음

스프링 컨텍스트에 새로운 빈 추가

스프링 컨텍스트에 동일한 타입의 빈 여러 개 추가하기

config.ProjectConfig2.java

```
@Configuration
public class ProjectConfig2 {

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

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

    @Bean
    Parrot parrot3() {
        var p = new Parrot();
        p.setName("Riki");
        return p;
    }
}
```

스프링 컨텍스트에 새로운 빈 추가

스프링 컨텍스트에 동일한 타입의 빈 추출하기

Main2.java

```
public class Main2 {

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

        Parrot p = context.getBean(Parrot.class);  // 예외 발생 !!!
        System.out.println(p.getName());

    }
}
```

- Parrot 타입으로 인스턴스가 3개 등록되어 있음 → 3개 중 어느 것을 참조할지 결정할 수 없어 예외 발생

- 타입 대신 빈의 이름으로 선택해야 함
  - @Bean 등록 시 사용한 메서드명이 빈의 기본 이름으로 등록됨
  - @Bean(name="") 또는 @Bean(value="")를 사용하여 이름 지정 가능

스프링 컨텍스트에 새로운 빈 추가

config.ProjectConfig2.java

```
@Configuration
public class ProjectConfig2 {

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

    @Bean(name = "miki")  // 빈의 이름 등록 @Bean(value="miki"), @Bean("miki")
    Parrot parrot2() {
        var p = new Parrot();
        p.setName("Miki");
        return p;
    }

    @Bean
    Parrot parrot3() {
        var p = new Parrot();
        p.setName("Riki");
        return p;
    }
}
```

스프링 컨텍스트에 새로운 빈 추가

스프링 컨텍스트에 동일한 타입의 빈 추출하기

Main2.java

```
public class Main2 {

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

        Parrot p = context.getBean("miki", Parrot.class);
        System.out.println(p.getName());

    }
}
```

```
Miki
```

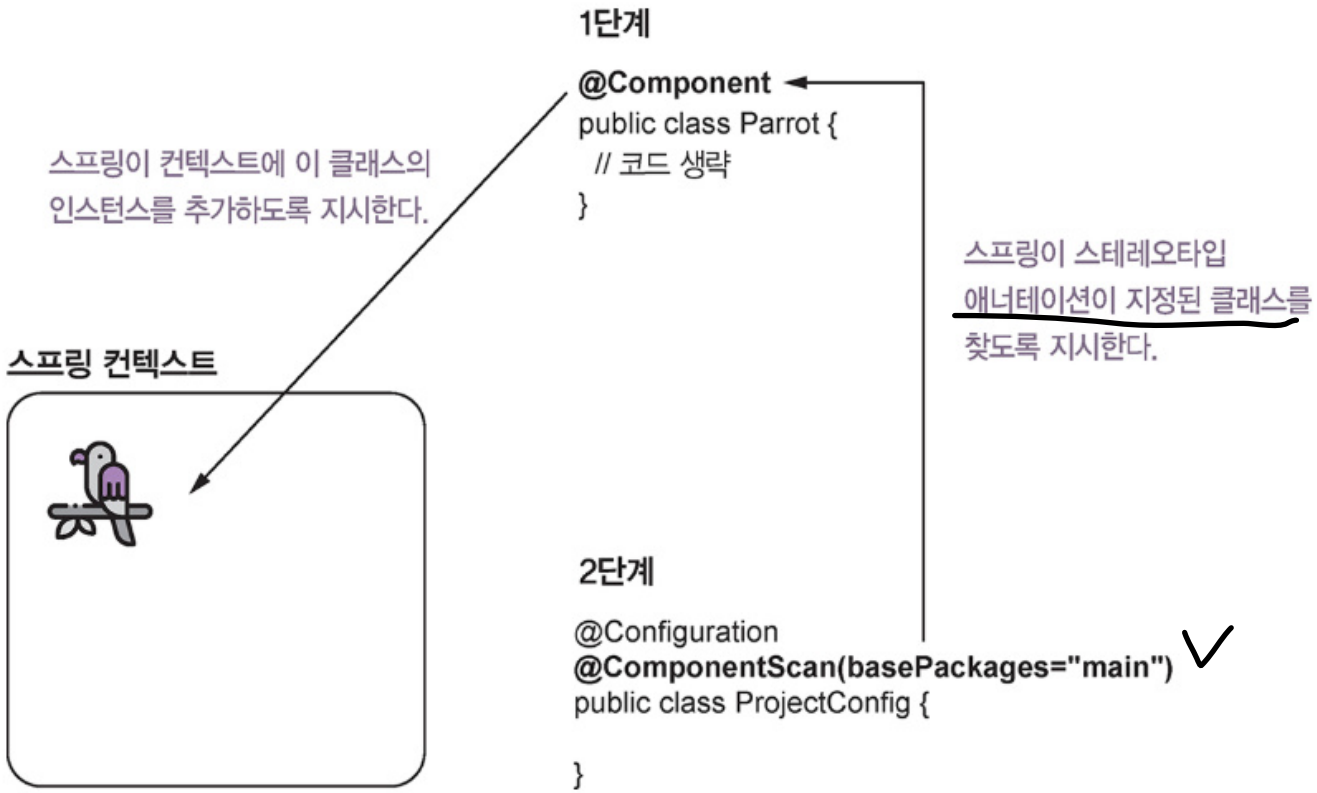
- context.getBean(빈이름, 타입.class);
  - 타입.class : 리턴 타입으로 사용할 클래스의 class

스프링 컨텍스트에 새로운 빈 추가

스테레오타입 애너테이션은 스프링 프레임워크에서 클래스를 스프링 컨테이너가 관리하는 빈(Bean)으로 등록하기 위해 사용되는 애너테이션입니다.
주로 @Component, @Controller, @Service, @Repository 등이 있으며, 각 애너테이션은 특정 계층의 역할을 나타내어 코드를 더 명확하게 만들고 이해하기 쉽게 해줍니다.

스테레오타입 애너테이션으로 스프링 컨텍스트에 빈 추가

1. @Component 애너테이션
   - 스프링이 컨텍스트에 인스턴스를 추가할 클래스를 표시
2. 구성 클래스 위에 @ComponentScan
   - 애너테이션으로 표시한 클래스를 어디에서 찾을 수 있는지 스프링에 지시



스프링 컨텍스트에 새로운 빈 추가

Parrot.java

```
package org.scoula.domain;

import org.springframework.stereotype.Component;

@Component      // 디폴트 컴포넌트의 name: 클래스명의 camelCase - parrot
public class Parrot {

    private String name;

    public String getName() {
        return name;
    }

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

스프링 컨텍스트에 새로운 빈 추가

config.ProjectConfig3.java

```
package org.scoula.config;

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

@Configuration
@ComponentScan(basePackages = "org.scoula.domain")
public class ProjectConfig3 {

}
```

해당 패키지 뿐만 아니라 하위도메인까지 포함

@Component 어노테이션이 붙은 애들을 다 찾아서 자동 등록함.

일일히 Bean할 필요 없어요

언제 @component @bean을 쓸까

@component 편하지만 속성이 nul 디폴트

@bean 뭔가 설정이 필요할때 사용

1. 속성 초기화가 필요하면 bean

2. 라이브러리 일반 객체를 빈 등록할때 bean

스프링 컨텍스트에 새로운 빈 추가

Main3.java

```
package org.scoula.app;

import org.scoula.config.ProjectConfig3;
import org.scoula.domain.Parrot;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Main3 {

    public static void main(String[] args) {
        var context = new AnnotationConfigApplicationContext(ProjectConfig3.class);

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

        System.out.println(p);
        System.out.println(p.getName());
    }
}
```

```
org.scoula.domain.Parrot@d6e7bab
null
```

앞서 말했듯 속성 디폴트null

스프링 컨텍스트에 새로운 빈 추가

- **PostConstruct를 사용하여 인스턴스 생성 후 관리하기** 생성 이후에 호출해라
  - @Bean은 인스턴스 생성 후 후처리 가능
  - @Component는 생성 후 후처리 불가

  - javax.annotation-api에서 정의한 @PostConstruct를 사용하여 후처리 메서드 지정
    - implementation 'javax.annotation:javax.annotation-api:1.3.2'

스프링 컨텍스트에 새로운 빈 추가

Parrot.java

```
package org.scoula.domain;

import org.springframework.stereotype.Component;
import javax.annotation.PostConstruct;

@Component
public class Parrot {

    private String name;

    @PostConstruct
    public void init() {
        this.name = "Kiki";
    }

    public String getName() {
        return name;
    }

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

생성후 호출될 메스드

스프링 컨텍스트에 새로운 빈 추가

Parrot.java

```
public class Main {

    public static void main(String[] args) {
        var context = new AnnotationConfigApplicationContext(ProjectConfig3.class);

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

        System.out.println(p);
        System.out.println(p.getName());
    }
}
```

```
org.scoula.domain.Parrot@223191a6
Kiki
```