**01 스프링 컨텍스트**

스프링 : 어떠한 자바 어플리케이션 개발할 수 있는 경량 프레임워크. 여기서 경량은 개발자의 부담.

나오는 개념 : dependency injection, 스프링 컨텍스트, Plain Old Java Object, bean

암기 키워드 : @Configuration, @Bean, AnnotationConfigApplicationContext, getBean, @Component, @ComponentScan, @PostConstruct

컨텍스트에 들어갈 수 있는 인스턴스의 클래스 조건   
== 스프링이 알아서 필요 인스턴스(의존성)을 주입해줄 수 있는 클래스 조건 : 디폴트 생성자 + setter

컨텍스트에 빈들을 등록하여 사용한다.

스프링 컨텍스트를 구성하려면 @Configuration으로 구성 클래스를 정의. 한구성클래스 == 한컨텍스트

Spring context이용을 위한 build.gradle  
implementation ‘org.springframework:spring-context:5.3.37’

빈이 되면 해당 타입의 객체는 싱글톤으로 운영되는게 디폴트이다.

컨텍스트의 주입 의존성 후보들은 빈으로 등록되어야 한다.

빈으로 등록하는 방법1 : @Configuration된 클래스의 멤버 메서드가 객체를 반환하게 하고, 메서드에 @Bean 어노테이션을 적용하면 된다. 이때 빈(의키)은 메서드의 이름으로 컨텍스트에 등록된다. 메소드로 반환하기 전에 객체에 이거 저것 세팅을 할 수 있다. 세터, 연산 등등.  
이때 빈의 키를 메서드의 이름이 아닌 다른 별도의 값으로 등록하고 싶다면 @Bean(name=”별도이름”)을 적용하면 된다.

빈으로 등록하는 방법2 : @Component된 클래스의 타입은 @Configuration된 클래스에 클래스 레벨  
@CompnentScan(basePackages=”@Component된 클래스가 위치한 패키지 이름(경로)”)에 의해서 스캔되어 지정패키지포함 하위패키지까지 @Component된 클래스 타입들을 모두 찾아서 모두 빈으로 컨텍스트에 등록한다. 이때 빈(의키)은 @Component된 클래스 이름의 캐멀케이스로 컨텍스트에 등록된다.

등록된 빈을 가져다 쓰는 방법1 : 명시적으로 빈을 꺼내서 사용하는 방법이다.  
빈으로등록된클래스타입 객체이름=context.getBean(빈으로등록된클래스타입.class);  
클래스 정보를 넘겨주면 컨텍스트가 알아서 bean이 생성되었는지 확인하고 있으면 그것을 반환, 없으면 해당 bean 객체를 생성하는 메서드를 호출하여 bean객체를 반환한다.  
명시적으로 빈을 받을 때 같은 빈으로 등록된클래스타입을 반환하는 메서드가 여러 개라면 아까 인자로 넘겨준 “비능로등록된클래스타입.class”정보 만으로 어떤 bean객체를 생성하고 반환하는지에 대한 정보가 부족하니 이럴 땐  
context.getBean(빈의키문자열,빈으로등록된클래스타입.class); 을 이용하면 된다.

메서드로 반환된 bean객체를 반환하는 메서드를 사용하는 bean객체와 달리  
@Component 클래스 레벨 어노테이션으로 타입이 등록된 bean객체는 이것 저것 세팅을 할 수 없다.  
@Component 클래스 레벨 어노테이션으로 타입이 등록된 bean객체는 생성자, 세터를 거쳐 생성되어 제공된다(물론 생성된 객체가 없다면)(있으면 그거 그냥 제공).  
그래서 생성자로 인해서 초기화 되지 않는 필드 값, 혹은 제공되기 전 동작을 수행하지 않은 채로 제공된다.  
@Component 클래스 레벨 어노테이션으로 타입이 등록된 bean객체도 메서드로 반환된 bean객체를 반환하는 메서드처럼 제공하기 전에 이것 저것 세팅하고 싶다면, @PostConstruct 메서드 레벨 어노테이션을 사용하면 된다.  
@PostConstruct는 생성자 호출 이후 호출되는 메서드에게 붙이는 어노테이션이다. 해당 메서드에서 하고 싶은 세팅을 하면 된다!

@PostConstruct를 사용하기 위한 build.gradle  
implementation ‘javax.annotation:javax.annotation-api:1.3.2’

**02 스프링 컨텍스트**

암기 키워드 : @Autowired, @Qualifier

등록된 빈을 가져다 쓰는 방법2 : 암시적으로 빈을 꺼내 사용하는 방법이며, 의존성이 생길 때, 컨텍스트가 의존성 후보를 주입해야 하는 상황일 때 많이 일어난다.

등록된 빈을 가져다 쓰는 방법2-1 : @Bean이 적용된 메서드를 직접 호출하는 방법.  
@Bean 메서드1시그니처. 메서드1이름( ); 호출.  
여기서 주의해야할게 보통 메서드 호출하듯이 동작하지 않는다.  
먼저 컨텍스트에 해당 메소드로 인해 생성된 객체가 이미 있는지 확인하고 있으면 바로 주입, 없으면 해당 메소드 호출하여 bean객체 제공.

등록된 빈을 가져다 쓰는 방법2-2 : bean 와이어링. @Bean적용된 메소드의 매개변수의 타입을 빈으로 등록된 타입을 사용하는 것.   
@Bean public Person person(빈으로등록된타입Parrot parrot){ … }. 이는 컨텍스트에 “의존성이 생겼어”라고 하는 것과 같다. 컨텍스트는 의존성을 주입한다. 이미 생성된 해당 타입의 객체가 있다면 그걸 제공하고 없다면 @Bean메서드든 @Component클래스든 등록된 것에서 객체를 생성하고 제공한다.  
이때 매개변수의 타입으로 등록된 빈이 2개 이상이라면 빈의 이름(키)가 매개변수 이름과 같은 bean객체가 제공된다.

등록된 빈을 가져다 쓰는 방법2-3 : @Autowired bean 와이어링. @Autowired 필드 레벨 어노테이션이 적용된 멤버가 컨텍스트에 의해 자동으로 주입된다.   
이때도 (필요한 의존성)멤버의 타입으로 등록된 bean객체가 2개 이상이면 빈의 이름(키)가 멤버 이름과 같은 bean객체가 제공된다. 의존성 멤버는 final인게 권장사항이기에 2-3 방법은 권장사항이 아니다.

등록된 빈을 가져다 쓰는 방법2-4 : @Autowired bean 와이어링. @Autowired 메소드 레벨 어노테이션이 적용된 생성자의 매개변수의 타입을 빈으로 등록된 타입을 사용하는 것.   
코드 내용은 (필요한 의존성)멤버를 채우는 코드 내용이다. 즉 권장사항을 지키며 의존성을 주입할 수 있는 똑똑한 방법이다.  
이때 전제조건은 클래스에 생성자는 해당 생성자 단 하나만 있어야 한다. 컨텍스트의 bean객체 제공은 2-2, 2-3과 같다.

등록된 빈을 가져다 쓰는 방법2-5 : @Autowired bean 와이어링. @Autowired 메소드 레벨 어노테이션이 적용된 (필요한 의존성)멤버의 세터의 매개변수의 타입을 빈으로 등록된 타입을 사용하는 것. 이때 컨텍스트의 bean객체 제공은 2-2, 2-3, 2-4와 같다.

주의 순환 의존성! : 생성자를 통한 bean 와이어링은 순환되어 교착 상태에 빠질 수 있다! 서로의 생성자에서 서로의 타입을 매개변수로 받으려 한다면, 컨텍스트가 의존성을 주입하려는 과정에서 즉 의존성 객체를 생성하려 과정에서 무한으로 서로의 생성자가 번갈아 가며 호출되며 교착 상태에 이른다.

실무에서는 등로된 빈을 가져다 쓸 때 1번 방법과 2-4번 방법을 제일 많이 쓴다.  
1번일 경우 외부에서 정의한 객체를 활용할 경우 ???config에서 직접 @Bean하여 많이 쓰고,  
2-4번일 경우 생성자에다가 적용해도 되지만 보통 롬복으로 생성자를 생성함. 그리고 생성자가 하나일 경우(final필드 멤버만 채우는 @RequiredArgConstructor형식일 경우) @Autowired 명시적으로 적용안해도 된다.

등록된 빈들 중 선택하여 가져다 쓰는 방법 : 등록된 빈을 가져다 쓰는 방법 2-2, 3, 4, 5에서 컨텍스트에 등록된 빈들의 선택 과정에 대해서 설명했다. 매개변수의 명을 마음대로 못하고 불안한게 단점인 방법이다. @Qualifier 변수 레벨 어노테이션을 사용하면 컨텍스트에 등록된 빈의 키를 통해 명시적으로 주입해 달라는 bean객체(의존성)을 요구?할 수 있다.

@Qualifier(컨텍스트에등록된빈의키문자열) Parrot parrot. 필드멤버, 매개변수 등등.  
Parrot이라는 타입인 등록된 bean객체가 여러 개여도 키 문자열로 어느 것인지 명시되어 있음.