Custom-Annotation 작성법

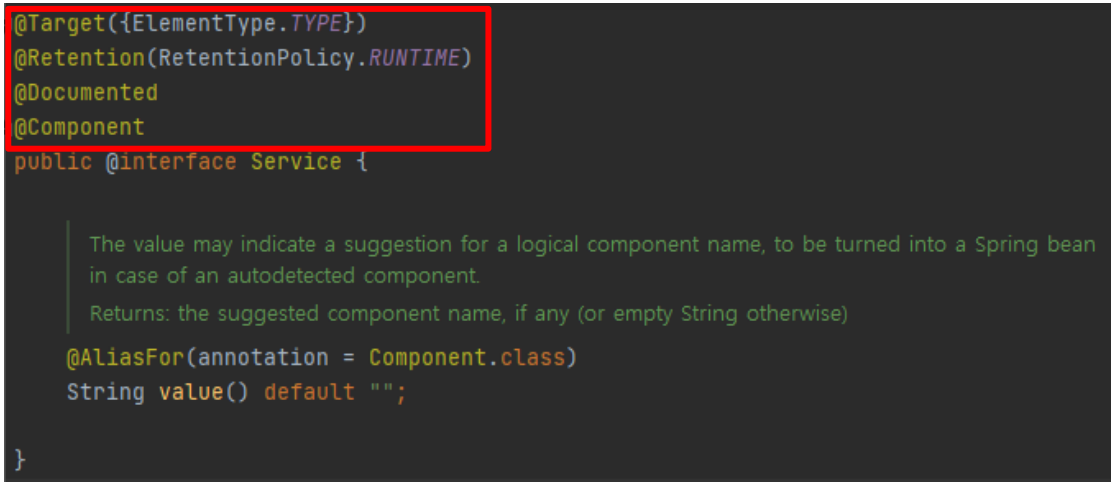
스터디 3회차 발표 - ES

Target? Retention? Documented?

```java
package hello.core.scan.filter;

import java.lang.annotation.*;

@Target(ElementType.TYPE) // 어노테이션을 작성할 곳
@Retention(RetentionPolicy.RUNTIME) // 어노테이션의 지속 시간
@Documented // Java doc 문서화 여부
public @interface MyIncludeComponent {

}
```

```java
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Indexed
public @interface Component {

    /**
     * The value may indicate a suggestion for a logical component name, to be turned into a Spring bean
     * in case of an autodetected component.
     * Returns: the suggested component name, if any (or empty String otherwise)
     */
    String value() default "";

}
```

meta-annotation



- annotation은 상속X, 인터페이스 구현X, 여러 개의 어노테이션에 공통적 속성을 부여하려면?
- 다른 annotation 에서도 사용되는 annotation (주로 custom-annotation 생성시 사용)
- 어노테이션에 정의에 부여된 어노테이션
- @Service = 빈 등록 위해 @Component 내포 -> @Component가 meta-annotation

@Retention

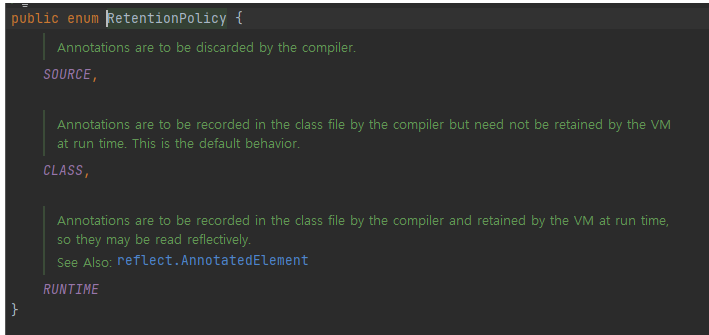
```java
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Indexed
public @interface Component {

    The value may indicate a suggestion for a logical component name, to be turned into a Spring bean
    in case of an autodetected component.
    Returns: the suggested component name, if any (or empty String otherwise)
    String value() default "";

}
```

- 어노테이션의 라이프 사이클을 정하는 것
  (어디까지 보유할 것인지?)
- RetentionPolicy : 어노테이션의 메모리 보유 범위 값을 결정하는 enum 클래스

```java
public enum RetentionPolicy {
    Annotations are to be discarded by the compiler.
    SOURCE,

    Annotations are to be recorded in the class file by the compiler but need not be retained by the VM
    at run time. This is the default behavior.
    CLASS,

    Annotations are to be recorded in the class file by the compiler and retained by the VM at run time,
    so they may be read reflectively.
    See Also: reflect.AnnotatedElement
    RUNTIME
}
```

@Retention -> RetentionPolicy

- RetentionPolicy
  - SOURCE : 소스코드(.java)까지 남아있는다
    = 컴파일 할 때 어노테이션의 메모리를 버림
  - CLASS : 디폴트값, 클래스 파일(.class)까지 남아있는다 = 바이트 코드
    = 런타임시 사라짐, Reflection 사용 불가
  - RUNTIME : 런타임까지 남아있는다
    = JVM이 자바 바이트코드 담긴 class 파일에서 런타임환경 구성하고 런타임 종료할 때까지 메모리가 살아있다
    = Reflection 사용 가능



@Retention -> RetentionPolicy

- SOURCE

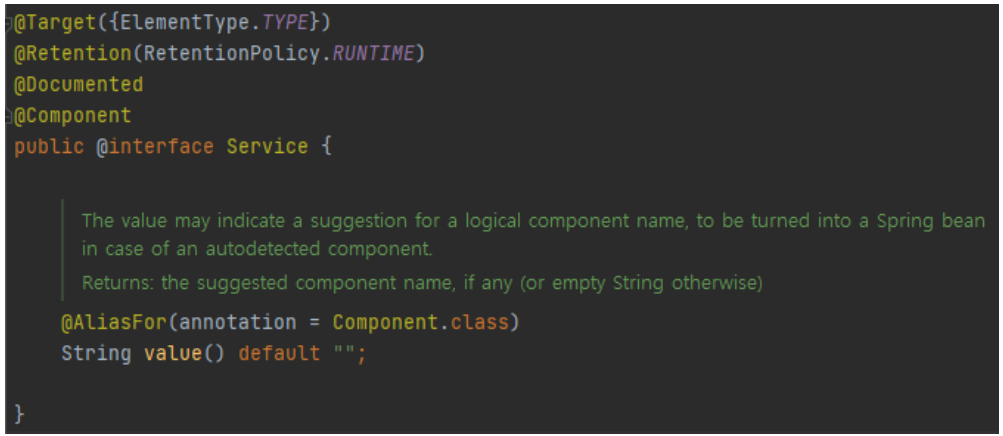
```
@Target({ElementType.FIELD, ElementType.TYPE})
@Retention(RetentionPolicy.SOURCE)
public @interface Getter {
    If you want your getter to be non-public, you can specify an alternate access level here.
    Returns: The getter method will be generated with this access modifier.
    lombok.AccessLevel value() default lombok.AccessLevel.PUBLIC;
```

  - Getter / Setter : 롬복이 바이트 '코드를 생성'해서 넣어준다, 따라서 바이트코드에 어노테이션 정보 필요X

    → RetentionPolicy.SOURCE (.java 까지만 유지)

@Retention -> RetentionPolicy

- RUNTIME



```java
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component
public @interface Service {

    The value may indicate a suggestion for a logical component name, to be turned into a Spring bean
    in case of an autodetected component.
    Returns: the suggested component name, if any (or empty String otherwise)
    @AliasFor(annotation = Component.class)
    String value() default "";

}
```

  - Reflection API 등을 이용해 런타임 중 어노테이션 정보를 가져올 수 있음
  - @Controller, @Service, @Autowired

    → 스프링 실행될 때 컴포넌트 스캔이 가능해야 하기 때문에 RUNTIME

    (스프링이 내부적으로 Reflection 활용해 어노테이션 붙은 컴포넌트만 가져옴)

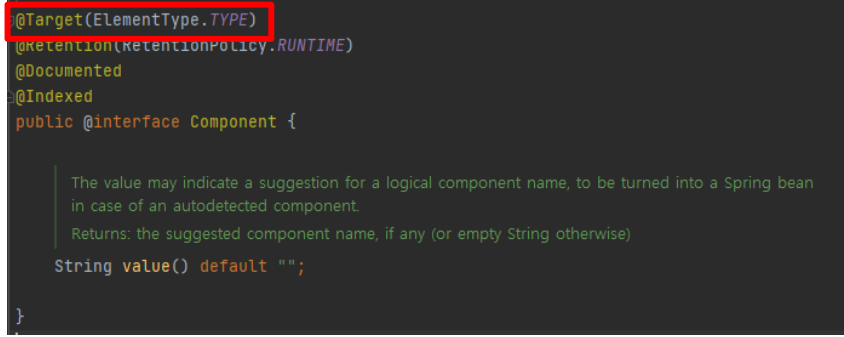
@Retention -> RetentionPolicy

- CLASS

```
@Target({ElementType.FIELD, ElementType.METHOD, ElementType.PARAMETER, ElementType.LOCAL_VARIABLE, ElementType.TYPE_USE})
@Retention(RetentionPolicy.CLASS)
@Documented
public @interface NonNull {
}
```

  - Maven/Gradle 로 받은 라이브러리(.jar) 파일엔 소스가 포함X (only .class)
  - 라이브러리에도 타입체커, IDE 부가기능 사용 위함
  - (만약 source 였다면? 컴파일된 라이브러리 .jar엔 어노테이션 정보가 x)
  - 그 외에도 클래스로딩시 사용하고 싶을 때...

@Target



- annotation이 어디에 적용될지?
- default 값 : 모든 대상
- ElementType.PACKAGE : 패키지 선언시 사용 = 패키지에만 어노테이션 달 수 있음
- ElementType.TYPE : 타입 선언시 사용(class, interface, enum…)
- ElementType.CONSTRUCTOR : 생성자 선언시 사용
- ElementType.FIELD : 멤버 변수 선언시 사용
- ElementType.LOCAL_VARIABLE : 지역 변수 선언시 사용
- ElementType.METHOD : 메서드 선언시 사용
- ElementType.PARAMETER : 파라미터 선언시 사용
- ……

Custom-annotation 규칙

- @interface 정의
  - java.lang.Annotation 인터페이스 상속하기 때문
  - 다른 클래스, 인터페이스 상속 x
- 파라미터 멤버 접근자 public 또는 default
- 파라미터 멤버 byte, short, char, int, float, double, boolean 기본타입
  + String, Enum, Class, 어노테이션만 사용 가능
- 클래스 메소드, 필드에 관한 어노테이션 정보를 얻기 위해선 리플렉션 API를 통해서만 가능 (RetentionPolicy.RUNTIME 시 가능)

사용 예제

- SNS 서비스에 접속하는 기능 제공하는 빈을 AOP 포인트컷으로 지정하고 싶어요, 구분을 편하게 할 수 없을까요?

```java
@Component
public @interface SnsConnector {
    // 해당 어노테이션 사용하면 클래스마다 @Component 붙이지 않아도
    // 자동으로 빈 등록 가능
}
```

```java
@SnsConnector
public class KakaoConnector {
}
```