lombok 기능정리 (2) - @NoArgsConstructor, @RequiredArgsConstructor, @AllArgsConstructor

태그

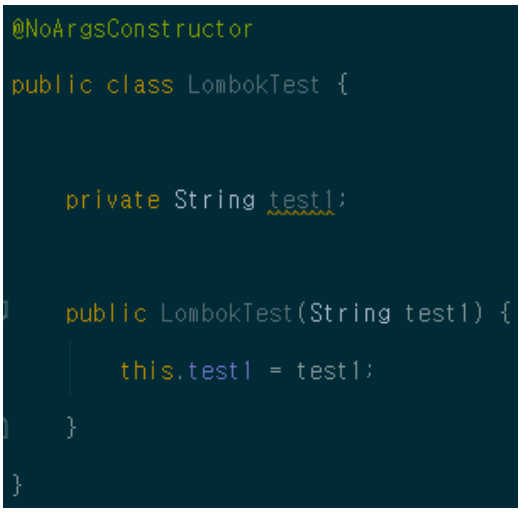
@NoArgsConstructor/@RequiredArgsConstructor/ @AllArgsConstructor

생성자를 자동으로 생성해주는 애노테이션들입니다.

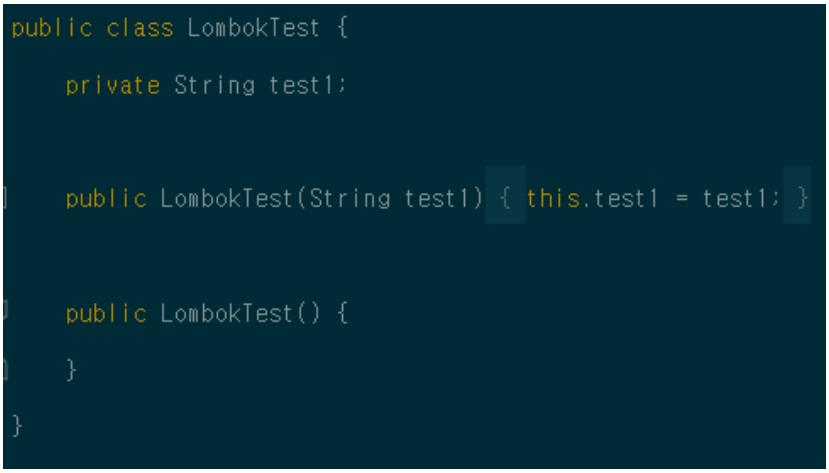
@NoArgsConstructor

이 애노테이션은 파라미터가 없는 생성자를 생성합니다.

lombok 코드



순수 자바 코드



파라미터가 없는 생성자를 자동으로 생성해주는 것을 볼 수 있습니다.

@NoArgsContructor를 사용할 때 주의점

1. 필드들이 final로 생성되어 있는 경우에는 필드를 초기화 할 수 없기 때문에 생성자를 만들 수 없고 에러가 발생하게 됩니다.

이 때는 @NoArgsConstructor(force = true) 옵션을 이용해서 final 필드를 0, false, null 등으로 초기화를 강제로 시켜서 생성자를 만들 수 있습니다.

2. @NonNull 같이 필드에 제약조건이 설정되어 있는 경우, 생성자내 null-check 로직이 생성되지 않습니다.

후에 초기화를 진행하기 전까지 null-check 로직이 발생하지 않는 점을 염두하고 코드를 개발해야 합니다.

@RequiredArgsConstructor

이 애노테이션은 추가 작업을 필요로 하는 필드에 대한 생성자를 생성하는 애노테이션입니다.

초기화 되지 않은 모든 final 필드, @NonNull로 마크돼있는 모든 필드들에 대한 생성자를 자동으로 생성해줍니다.

@NonNull로 마크돼있는 필드들은 null-check가 추가적으로 생성되며, @NonNull이 마크되어 있지만, 파라미터에서 null값이 들어온다면 생성자에서 NullPointerException이 발생합니다.

파라미터의 순서는 클래스에 있는 필드 순서에 맞춰서 생성됩니다.

롬복 코드

```java
@RequiredArgsConstructor
public class LombokTest {

    private String test1;
    private final String test2;
    @NonNull
    private String test3;
}
```

순수 자바 코드

```java
public class LombokTest {
    private String test1;
    private final String test2;
    @NonNull
    private String test3;

    public LombokTest(String test2, @NonNull String test3) {
        if (test3 == null) {
            throw new NullPointerException("test3 is marked non-null but is null");
        } else {
            this.test2 = test2;
            this.test3 = test3;
        }
    }
}
```

final 변수와 @NonNull 마크 변수에 해당하는 생성자가 생성된 것을 보실 수 있습니다.

그리고 null-check 로직이 자동적으로 생성된 것도 보실 수 있습니다.

@AllArgsConstructor

이 애노테이션은 클래스에 존재하는 모든 필드에 대한 생성자를 자동으로 생성해줍니다.

만약 필드중에서 @NonNull 애노테이션이 마크되어 있다면 생성자 내에서 null-check 로직을 자동적으로 생성해줍니다.

롬복 코드

```
@AllArgsConstructor
public class LombokTest {

    private String test1;
    private String test2;
    @NonNull
    private String test3;
}
```

순수 자바 코드

```
public class LombokTest {
    private String test1;
    private String test2;
    @NonNull
    private String test3;

    public LombokTest(String test1, String test2, @NonNull String test3) {
        if (test3 == null) {
            throw new NullPointerException("test3 is marked non-null but is null");
        } else {
            this.test1 = test1;
            this.test2 = test2;
            this.test3 = test3;
        }
    }
}
```

모든 필드를 가진 생성자를 자동으로 생성해주는 것을 볼 수 있습니다.

staticName 옵션을 이용한 static factory 메소드 생성

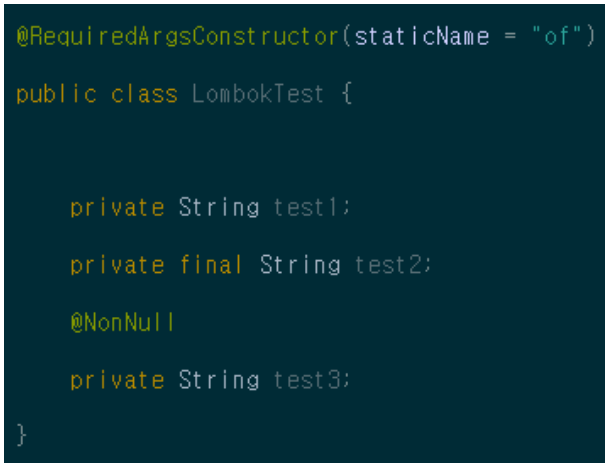
위의 세 생성자 관련 애노테이션은 static factory를 만들 수 있는 옵션이 있습니다.

staticName이라는 옵션을 사용하여 생성자를 private으로 생성하고, private 생성자를 감싸고 있는 static factory 메소드를 추가할 수 있습니다.

아래와 같이 옵션을 사용할 수 있습니다.

ex) @RequiredArgsConstructor(staticName = “of”)

롬복 코드



순수 자바 코드

```java
public class LombokTest {

    private String test1;

    private final String test2;

    @NonNull

    private String test3;


    private LombokTest(String test2, @NonNull String test3) {

        if (test3 == null) {

            throw new NullPointerException("test3 is marked non-null but is null");

        } else {

            this.test2 = test2;

            this.test3 = test3;

        }

    }


    public static LombokTest of(String test2, @NonNull String test3) {

        return new LombokTest(test2, test3);

    }

}
```

private 생성자가 생성되고, private 생성자를 감싸는 static factory 메소드가 생성된 것을 볼 수 있습니다.

생성자 관련 애노테이션을 사용할 때 주의사항

1. static 필드들은 스킵됩니다.

2. 파라미터의 순서는 클래스에 있는 필드 순서에 맞춰서 생성하기 때문에 매우 주의해야 한다.

→ 만약 클래스 필드의 순서를 바꾸었을 때 롬복이 자동적으로 생성자 파라미터 순서를 바꾸어 주기 때문에 주의하지 않으면 버그가 발생할 수 있다.

3. AccessLevel을 꼭 설정해주어야 합니다.

→ 세 애노테이션 모두 접근 제한자를 AccessLevel로 설정할 수 있습니다.

기본값은 public이지만 필요로 따라서 접근제한자를 설정해주어야 합니다.

출처