Optional
Assertion
Spring Bean
Dependency Injection

2021.08.22 Spring Study 2week

유채린

chaerin.du.ub@gmail.com

해당 내용은 인프런 강의 수강 후, 개인적으로 작성한 내용입니다.
잘못된 내용에 대하여 피드백주시면 감사하겠습니다!

(인프런 아이콘 클릭 시, 참고한 인프런 강의 사이트로 이동합니다.)

Optional<T> ?

import java.util.Optional;

*null 일 수도 또는 null이 아닌 value*를 갖고 있는 container object

Integer클래스처럼 'T'타입의 객체를 포장해 주는 Wrapper class
따라서 Optional 인스턴스는 모든 타입의 참조 변수를 저장할 수 있다.

NullPointerException를 Optional 내장 메소드로 간단히 회피할 수 있다.
(모든 메서드가 그런 건 아님!)

👉 Optional 객체 생성

of()	ofNullable()
명시된 값이 null이 아니면 명시된 값을 가지는 Optional 객체를 반환 **[예시]** **Optional<String>** opt = **Optional.**ofNullable("자바 Optional 객체"); **System.**out.println(opt.get());	
null이 저장되면 NullPointerException예외가 발생	명시된 값이 null이면 비어있는 Optional 객체를 반환

👉 Optioanl 객체 접근: get()

get() 메소드를 호출하기 전에 **isPresent() 메소드**를 사용하여
Optional 객체에 **저장된 값이 null인지 아닌지를 먼저 확인한 후 호출**하기

🔎🤔

empty()	isPresent()
아무런 값도 가지지 않는 비어있는 Optional 객체를 반환	저장된 값이 존재하면 true를 반환, 값이 존재하지 않으면 false를 반환

👉 null 값 대체하기

orElse()	값이 존재하지 않으면 인수로 전달된 값을 반환
orElseGet()	값이 존재하지 않으면 인수로 전달된 람다 표현식의 결과값을 반환
orElseThrow()	값이 존재하지 않으면 인수로 전달된 예외를 발생

출처: http://tcpschool.com/java/java_stream_optional
https://docs.oracle.com/javase/8/docs/api/

🔎🤔 orElse vs. orElseGet ?

orElse는 null이든 아니든 항상 불린다.
orElseGet은 null일 때만 불린다.

🤔🤔🤔

그럼 같은거 아냐?

아닙니다. 아래의 경우를 생각해봅시다.

```
public void ohMyGod() {
        String username = null;
        String result1 = Optional.ofNullable(username).orElse(getDefaultName());
        System.out.println(result1);

        String result2 = Optional.ofNullable(username).orElseGet(() -> getDefaultName());
        System.out.println(result2);
}

private String getDefaultName() {
        return "no name";
}
```

이 경우에 결과는 어떻게 될까요?
네. 둘다 "no name" 입니다. 결과는 같지만 여긴 굉장한 차이가 있습니다.
orElse의 경우는 "값"을 취합니다.
orElseGet은 "Supplier"를 취하죠.

위의 예시는 아래 코드로 다시 쓰여질 수 있습니다.

```
String username = null;
String defaultName = getDefaultName(); // 여기가 다릅니다.
String result1 = Optional.ofNullable(username).orElse(defaultName);
System.out.println(result1);

Supplier<String> supplier = () -> getDefaultName(); // 여기요.
String result2 = Optional.ofNullable(username).orElseGet(supplier);
System.out.println(result2);
```

아시겠나요? 네니요..😂

🔎🤔 Supplier ?

함수적 인터페이스
메소드를 호출하는 곳으로 데이터를 리턴(공급)하는 역할

```
40 @FunctionalInterface
41 public interface Supplier<T> {
42
43     /**
44      * Gets a result.
45      *
46      * @return a result
47      */
48     T get();
49 }
```

매개변수를 받지 않고
단순히 무엇인가를 반환하는
추상 메서드가 존재

출처: http://tcpschool.com/java/java_stream_optional
https://docs.oracle.com/javase/8/docs/api/
https://cfdf.tistory.com/34
https://m.blog.naver.com/zzang9ha/222087025042

🤔🤔🤔 회원가입 예시

큰 일 날 뻔한 사례

```
public User findByUsername(String name) {
        return userRepository.findByName(name).orElse(createUserWithName(name));
}

private User createUserWithName(String name) {
        User newUser = new User();
        newUser.setName(name)
        return userRepository.save(user);
}
```

사용자 이름을 찾고 없다면 유저 생성

(대충 이런 상황이었습니다.)

만약 user 테이블의 name이 unique였다면?

네. 맞습니다. 장애입니다. 왜냐면 아래랑 같기 때문이죠.

```
public User findByUsername(String name) {
        User newUser = createUserWithName(name); // ㅠㅠ
        return userRepository.findByName(name).orElse(newUser);
}

private User createUserWithName(String name) {
        User newUser = new User();
        newUser.setName(name)
        return userRepository.save(user);
}
```

우선!! 사용자 이름으로 유저 생성한 다음, 사용자를 찾는다.
-> 이름이 unique 조건이라면 오류 발생!!

orElseGet을 사용하여, 사용자가 없는 경우에만 유저 생성하도록 한다.
→ onElse는 null이든 아니든 생성하기 때문에 오류 발생하고
onElseGet은 null일 때만 생성하기 때문에 오류가 발생하지 않는다.

이해가 되시나요? 네..😂

👉 기본 타입의 Optional 클래스

OptionalInt, OptionalLong, OptionalDouble 클래스

반환 타입이 Optional<T> 타입이 아니라
해당 기본 타입이라는 사실만 제외하면 거의 모든 면에서 비슷하다.

각 클래스별로 저장된 값에 접근할 수 있는 메소드가 제공된다.
-> getAsInt(), getAsLong(), getAsDouble()

◆ isPresent()와 orEleseGet() 두 메소드가 많이 사용된다고 합니다!

✨자바8에 추가된 4가지의 **함수형 인터페이스**

Function<T, R> **R apply(T t);**	**T를 인자로 받고, R을 리턴**합니다.
Consumer<T> **void accept(T t);**	**T를 인자로 받고, 이를 소모**합니다.(리턴 X)
Predicate<T> **boolean test(T t);**	**T를 인자로 받고, Boolean형을 리턴**합니다.
Supplier<T> **T get();**	**T를 리턴합니다.**(Lazy Evaluation)

출처: http://tcpschool.com/java/java_stream_optional
https://docs.oracle.com/javase/8/docs/api/
https://cfdf.tistory.com/34
https://m.blog.naver.com/zzang9ha/222087025042

Assertion?

코드가 실행될 때 반드시 어떤 값일지
확신하는 값, 범위 또는 확실한 클래스의 상태 등을 체크하여
프로그램의 신뢰성을 높이기 위해서 사용된다. ✨TEST

Assertions은 static import가 가능하다.
Static import 하고 난 뒤에 Assertions. 없이 바로 assertThat 사용 가능

```
import static org.assertj.core.api.Assertions.*;
...
assertThat(member).isEqualTo(result);
...
```

String Bean & Dependency Injection

스프링 빈 -> 스프링 컨테이너가 관리하는 자바 객체
ApplicationContext가 만들어서 그 안에 담고 있는 객체를 의미

👉 스프링 빈 등록 2가지 방법

1. 컴포넌트 스캔과 자동 의존관계 설정
→ @Controller, @Service, @Repository 셋 다 @Component 어노테이션을 갖고 있음
-> 자동으로 스프링 빈 등록

2. 자바 코드로 직접 스프링 빈 등록하기
→ 자바 클래스 생성 후, 클래스에 @Configuration 어노테이션을 붙여준다.
그리고 service, repository를 각각 @Bean 어노테이션을 붙여 스프링 뜰 때 객체를 생성하도록 한다.

실무에서는 주로 정형화된 컨트롤러, 서비스, 리포지토리 같은 코드는 컴포넌트 스캔을 사용한다. **그리고 정형화 되지 않거나, 상황에 따라 구현 클래스를 변경**해야 하면 설정을 통해 스프링 빈으로 등록한다.
예) 기존 레퍼지토리를 다른 레퍼지토리로 변경할 때 기존의 운영중인 코드를 하나도 수정하지 않고 바꿔치기 할 수 있다.

👉 Dependency Injection

생성자 주입	생성자를 통해서 멤버 서비스가 멤버 컨트롤러에 주입된다. (권장)
필드 주입	생성자를 빼고 필드에다가 @Autowired 선언해서 삽입 (intelliJ가 별로 추천하진 않음)
Setter 주입	setter method에 @Autowired 어노테이션 선언 -> 단점: 누군가가 멤버 컨트롤러 선언할 때 public으로 열려 있어야 한다. -> public 노출로 문제 발생 위험

단축키

	Mac	Window
Project Structure	Command + ;	Ctrl + Alt + Shift + S
Settings	Command + ,	Ctrl + Alt + S
Java method parameter 정보 조회	Command + P	Ctrl + P
자동완성	Command + Shift + Enter	Ctrl + Shift + Enter
Getter/setter/constructor ..	Command + N	Alt + Insert
선택한 변수명 전체 변경	Shift + F6	Shift + F6
변수명 리팩토링	Command + Opt + V	Ctrl + Alt + V
JUnit 단위테스트 생성	Command + Shift + T	Ctrl + Shift + T
이전 실행문 재실행	Ctrl + r	Shift + F10

참고
https://gmlwjd9405.github.io/2019/05/21/intellij-shortkey.html