Optional 프로그래밍

Sugwoo Byun

1 개념

`https://docs.oracle.com/javase/10/docs/api/java/util/Optional.html`

1.1 컨테이너(Container) 타입

- 자바에서 타입은 클래스나 인터페이스로서 정의된다. 제너릭스 타입(Generics Type)은 타입 변수를 인수로 갖는 클래스나 인스턴스를 의미한다. 타입 변수는 `T, U, E` 등의 대문자로 표현된다. `Optional<T>` 등의 클래스나 `Function<T,R>` 등의 인터페이스가 제너릭스의 예이다.
- 타입 변수는 아직 정해지지 않은 임의의 타입을 의미하며, 어떤 특정 타입으로 실례화(instantiation) 된다. 이때, 실례화될 수 있는 타입은 Wrapper 타입이어야 한다. `int, short, byte, long, Float, double, boolean, char` 는 primitive 타입이므로 실례화될 수 없으며, 그에 대응하는 wrapper 인 `Integer, Boolean ...` 등의 타입에 대해서는 실례화가 가능하다.
- 제너릭스 타입은 프로그래밍언어에서 일반적으로 Parameterized Type 이라는 용어로 불리우고 있으며, 또 다른 측면에서는 컨테이너(container) 타입이라고도 한다.
- 대략적으로, 타입은 값들(values)의 모음(collection) 이라고 해석된다. 예를 들어, `Boolean` 타입은 원소로서 `true, false` 의 두 개의 값을 갖는다 (엄밀하게 이야기해서, 자바에서는 이것이 `Boolean` 이 아닌 primitive 타입의 `boolean` 이 되어야 하므로 이론과 약간의 차이가 있음). 이때 `Optional<Boolean>` 은 `true` 나 `false` 를 `Optional` 컨테이너 안에 넣은 것으로 볼 수 있다.
- 컨테이너는 다른 용어로서 문맥(context) 라고도 한다. 디양한 목적으로 여러 형태의 컨테이너들이 정의될 수 있다. 새로운 클래스나 인터페이스를 이해할 때는 그 의도와 목적을 정확히 이해하는 것이 중요하다. 예를 들어, `Integer` 와 `List<Integer>` , `Optional<Integer>` 는 어떤 차이가 있는가?

1.2 `null` 과 `Optional.empty()` 의 차이

- 일반적으로 프로그래밍 언어에서는 **정의되지 않은 값(undefined value)** 라는 개념이 존재한다. 타입의 구성원이 값이므로 이 정의되지 않은 값은 타입의 원소가 될 수 없다. 정의되지 않은 대표적인 예로서 `x / 0` 와 같은 산술식으로서, 이것의 타입을 `Integer` 라고 할 수는 없다.
- 정의되지 않은 값의 예는 매우 다양하다. 예를 들어, 파일을 open 하는 함수는 결과 값으로 주어진 `filePoint` 를 return 하는데, 만약 주어진 파일이 존재하지 않는다면 값은 정의되지 않은 값으로 해석될 수 있다. 또한, empty 리스트에서 어떤 값을 읽는 경우도 정의되지 않은 값으로 해석될 수 있다.
- **정의되지 않은 값** 은 매우 다양한 형태로 존재하는 데, 이들은 모두 동일하게 취급되며, 이론적으로 $\perp$ 기호로 표현한다. 타입 `T` 를 집합 T 로 표시할 때, `Optional<T>` 타입의 의미는 $T \cup \{\perp\}$ 으로 설명될 수 있다. 즉, 기존 값 외에 추가적으로 **정의되지 않은 값** 을 표현하는 타입이라고 볼 수 있다. 예를 들어, `Optional<Boolean>` 타입의 원소는 $true, false, \perp$ 로 구성되어 있다고 볼 수 있다.
- 그렇다면 자바에서 $\perp$ 은 어떻게 표현될까? 자바에서 이것은 구문이 아닌 메소드 `Optional.empty()` 으로서 표현한다. 여기서 주목할 점은 $\perp$ 은 타입에 상관 없이 **모든 정의되지 않은 값** 을 표현하므로, `Optional.empty()` 에서 특정 타입 `T` 를 명시하지 않는다.

- 그렇다면 자바의 null 은 무엇인가? null 과 Optioal.empty() 는 다른가? 자바에서 null 은 primtive 타입을 가질 수는 없으며, 그것의 wrapper 타입으로서 표현된다. null 이 Integer 등의 wrapper 타입을 갖는다면, null 이 Integer 타입의 원소란 뜻인가? 자바 문서에 따르면 null 은 특별히 취급되어, 특정 타입의 변수에 할당될 수는 있으나, 그렇다고 해서 null 이 그 타입의 값인 것은 아니라고 명시되어 있다. 이 상황은 자바의 instanceof 를 이용하여 확인할 수 있다. 아래의 예를 실습해 보기 바란다.

```
jshell> int x = null                // Error:
jshell> Integer x = null            // OK, x is a pointer of an object of Integer
jshell> Boolean y = null            // OK
jshell> String  z = null            // OK
jshell> Integer x2 = new Integer(2) // x2 is an instance of Integer
jshell> x2 instanceof Integer       // true
jshell> x instanceof Integer        // false
jshell> Integer a                   // a ==> null
jshell> a == null                   // true
```

위의 예에서, null 은 wrapper 타입의 변수 x 에 할당될 수는 있으나, 그렇다고 해서 x 가 Integer 의 객체는 아닌 것이다. 대신 x 는 Integer 객체에 대한 메모리 공간이 할당될 때 이를 포인트하는 포인터 (reference)의 역할을 한다. x 에 null 을 할당하는 것은 단지 타입 선언만 한 것과 동일하다. 위의 예에서 a 와 null 이 동일하다는 것을 확인할 수 있다.

1.3 Optional.ofNullable() 메소드

- 이 메소드는 T 타입의 값을 Optional<T> 형태의 컨테이너에 넣는 기능을 한다. 다음 첫 번째 예는 정수 값 10을 이 컨테이너로 넣는 상황으로 설명될 수 있다. 그런데 이 메소드는 마지막 예에서 보여 주듯이, null 에 대해서는 이를 $\perp$ (즉, Optional.empty()) 으로 바꿔서 넣어 준다. null 은 Optional 의 원소가 아니며, 이는 아직 배정되지 않는 메모리 공간의 포인터를 의미한다. Optional.ofNullable() 메소드와 관련하여 매뉴얼을 참조하기 바란다.

```
jshell> Optional<Integer> n = Optional.ofNullable(new Integer(10))
jshell> null == null                                    // true
jshell> Optional.empty() == Optional.empty()            // true
jshell> null == Optional.empty()                        // false
jshell> Optional.ofNullable(null) == Optional.empty()  // true
```

2 컨테이너 타입 연산

2.1 기본 원리

- 컨테이너 타입 Optional<T> 는 T 타입의 값을 Optional 컨테이너 안에 넣은 것으로 해석할 수 있다. Optional<T> 타입의 수식을 연산하는 방법은 Optional 컨테이너에서 T 에 해당되는 값을 꺼내서, T 타입에 해당되는 메소드를 적용하여 연산한 다음, 필요에 따라 이 값을 다시 Optional 컨테이너에 넣는 것이다.

- 이 방법은 Optional 컨테이너뿐만 아니라 모든 제너릭스 타입 Generics <T> 형태의 컨테이너에 대해서 동일하게 적용된다. 예를 들어, List<Integer> 에서도 특정 인덱스에 위치한 Integer 타입의 값을 꺼내서, Integer 타입에 대해 적용할 수 있는 메소드를 적용하여 계산한 다음, 다시 이 결과를 List 컨테이너에 넣는 방법이 적용된다.

- 따라서 모든 제너릭스 타입의 계산을 위해서는 해당 컨테이너에 값을 넣는 메소드와 컨테이너로부터 값을 꺼내오는 메소드가 필요하게 된다.

2.2 Optional<T> 컨테이너에 값 넣기 메소드

Optional 컨테이너에 값을 넣을 때 사용될 수 있는 메소드는 3개로서 모두 static 이다.

- static Optional<T> Optional.empty()

이 메소드는 undefined value($\perp$) 를 임의의 Optional<T> 컨테이너에 넣는다.

- static Optional<T> Optional.of(T value)

이 메소드는 null 이 아닌 value를 컨테이너 안에 넣는다. 만약 null 일 경우 NPE(NullPointerException) 가 발생한다.

- static Optional<T> Optional.ofNullable(T value)

이 메소드의 인수인 value의 값은 null 일 수도 있고, 아닐 수도 있다. null 일 경우, 컨테이너 안에 들어가는 값은 Optional.empty() 가 되며, null 이 아닐 경우 그 값을 그대로 넣는다.

2.3 Optional<T> 컨테이너로부터 값 꺼내기 메소드

컨테이너로부터 T 타입의 값을 꺼내오는 메소드는 모두 instance 메소드로서 get(), orElse(), orElseGet(), orElseThrow(), orElseThrow() 등이 있다. 여기서는 그 중에 대표적인 두 개의 사용법을 소개한다.

- T get() 메소드

이 메소드는 x.get() 형태로 수행되어, $\perp$ 이 아닌 T 타입의 값을 꺼내 온다.

- T orElse(T value) 메소드

이 메소드는 x.orElse(value) 형태로 수행된다. x 의 값이 $\perp$ 일 경우, 수행 결과는 인수로 주어진 value 가 된다. x 의 값이 $\perp$ 이 아닐 경우, 컨테이너 안에 들어 있던 T 타입의 값이 복귀 값이 된다.

2.4 기타 주요 메소드들

- isPresent() 메소드

이 메소드는 x.isPresent() 형태로 수행되는데, x 의 값이 $\perp$ 이면 true, 아니면 false 의 값이 결과 값이 된다.

- Optional<U> map(Function<T,U> mapper) 메소드

이 메소드는 x.map(f) 의 형태로 수행되며, f 의 입력 타입이 T, 출력 타입이 U 일때, 이 메소드 수행의 결과 값은 Optional<U> 타입의 값을 갖는다. x 가 $\perp$ 인 경우 결과 값 또한 $\perp$ 이 된다. $\perp$ 이 아닌 경우, 컨테이너로부터 T 타입의 값을 꺼내서 이 값에 mapper 함수를 적용하여 U 타입의 값을 얻게 되며, 이 값을 다시 컨테어너 안에 넣어 Optional<U> 타입의 결과를 갖는다. 이 메소드는 매우 효과적이므로 잘 익혀 사용할 필요가 있다. 이 메소드는 컨테이너 문맥에 있는 값에 mapper 함수를 적용하도록 하는 기능을 한다. 즉, 컨테이너 안에 있는 값의 처리를 마치 컨터이너 밖에 있는 것 처럼 표현할 수 있도록 하며, 처리될 값이 null 인지의 여부를 검토하는 과정은 내부적으로 처리되며, 이 내용이 외부적으로는 표현하지 않으므로 코딩의 간결성을 갖게 된다.

3 예제 프로그램

http://www.daleseo.com/java8-optional-before/ 에 소개된 예제를 소개한다. 다음과 같이 웹 쇼핑몰 등에서, 주문을 한 고객이 살고 있는 도시를 찾는 프로그램을 고려한다. 이를 위한 코드는 대략 다음과 같이 표현할 수 있다.

3.1 기본 Logic

```
/* 주문을 한 회원이 살고 있는 도시를 반환한다 */
public String getCityOfMemberFromOrder(Order order) {
  return order.getMember().getAddress().getCity();
}
```

3.2 NPE를 방어하는 코드

위의 코드는 NPE 문제가 발생할 수 있으므로, 이를 방어하기 위한 코드로서, 생성되는 객체가 `null` 인지를 점검하는 부분이 첨가된다.

```
public String getCityOfMemberFromOrder(Order order) {
  if (order != null) {
    Member member = order.getMember();
    if (member != null) {
      Address address = member.getAddress();
      if (address != null) {
        String city = address.getCity();
        if (city != null) {
          return city;
        }
      }
    }
  }
  return "Seoul"; // default
}
```

3.3 `Optional` 과 `map` 을 이용하는 코드

- 위의 코드는 기본 프로그램 로직에 NPE 문제 해결을 위한 기능이 첨가 되었다. 그러나 이런 형태의 코드는 프로그램의 가독성이 저하되는 문제가 발생한다. 이를 처리하는 방법으로서 `Optional` 타입과 함께 `map` 을 이용함으로써 이 문제를 해결할 수 있다. 이에 대한 코드는 다음과 같다.
- 이 코드는 NPE 처리 과정을 `map` 안에 내재함으로써 이 과정이 겉으로 드러나지 않는다. 이 코드는 외형적으로 볼 때 기본 로직 만을 표현하는 것과 같은 느낌을 주고 있으므로, 가독성이 좋으면서도 NPE 문제를 해결하고 있다.

```
Optional.ofNullable(order)
    .map(Order::getMember)
    .map(Member::getAddress)
    .map(Address::getCity)
    .orElse("Seoul");
```

3.4 수행 가능하도록 정리된 코드

```
import java.util.Optional;

class Order {
   private int id; private int date; private Member member;

   Order (int id, int date, Member member) {
       this.id = id;  this.date = date;  this.member = member;
   }
```

```
   int getId() { return this.id; }
   int getDate() { return this.date; }
   Member getMember() { return this.member; }
}

class Member {
    private int id;  private String name; private Address address;

    Member(int id, String name, Address address) {
        this.id = id;  this.name = name;  this.address = address;
    }

    int getId() { return this.id; }
    String getName() { return this.name; }
    Address getAddress() { return this.address; }
}

class Address {
private String street; private String city; private int zipcode;

    Address(String street, String city, int zipcode)  {
        this.street = street;  this.city = city;  this.zipcode = zipcode;
    }

    String getStreet() { return this.street; }
    String getCity() { return this.city; }
    int getZipcode() { return this.zipcode; }
}

public class NpeOrder {
    public static void main(String[] args) {
  Order order; Member memb;  Address addr;

  addr = new Address("Daeyeong-Dong", "Busan", 48434);
        memb = new Member(100, "HongGilDong", addr);
  // order = new Order(200, 20170530, null);  // NPE
  order = new Order(200, 20170530, memb);

  System.out.println(getCityOfMemberFromOrder(order));
  System.out.println(getCityOfMemberFromOrder2(order));
  System.out.println(getCityOfMemberFromOrder(null));
  System.out.println(getCityOfMemberFromOrder2(null));
    }

    public static String getCityOfMemberFromOrder(Order order) {
  // return order.getMember().getAddress().getCity(); // no protection

  if (order != null) {
    Member member = order.getMember();
    if (member != null) {
      Address address = member.getAddress();
      if (address != null) {
```

```
        String city = address.getCity();
        if (city != null) {
          return city;
        }
      }
    }
  }
  return "Seoul"; // default
    }

    public static String getCityOfMemberFromOrder2(Order order) {
  return Optional.ofNullable(order)
      .map(Order::getMember)
      .map(Member::getAddress)
      .map(Address::getCity)
      .orElse("Seoul");
    }
}
```