Null 대신 Optional 클래스

11 장

By 박찬
역

Null 대신 Optional 클래스

토니 호어 – 알골 설계 (ALGOL W)을 설계하면서 null의 처음 등장
구현하기 쉬웠기 때문에 null을 도입했다

- 컴파일러의 자동 확인 기능으로 모든 참조를 안전하게 사용하는 것이 목표.
- 참조 및 예외로 값이 없는 상황을 가장 단순하게 구현할 수 있다고 판단

⇒ 자바를 포함해 최근 수십 년간 탄생한 대부분의 언어 설계에는 null 참조 개념을 포함한다.
⇒ 십억 달러짜리 실수라고 표현(이보다 클 수 있다.)

By 박찬

11.1 값이 없는 상황을 어떻게 처리할까?

```
public class Person {

  private Car car;

  public Car getCar() {
    return car;
  }

  public class Car {
    private Insurance insurance;

    public Insurance getInsurance() {
      return insurance;
    }
  }

  public class Insurance {
    private String name;

    public String getName() {
      return name;
    }

    public String getCarInsuranceName(Person person) {
      return person.getCar().getInsurance().getName();
    }
  }

}
```

어떤 문제가 발생할까??

차를 보유하지 않은 사람이 있다.

⇒ getCarInsuranceName을 호출하면

⇒ getInsurance()가 실행될때 Null 참조의 보험 정보를 반환하려 할 것이므로 NullPorinterException이 발생하며 중단

By 박찬열

11.1.1 보수적인 자세로 NullPointerException 줄이기

Null 안전 시도1: 깊은 의심

```
public String getCarInsuranceName(Person person) {
    if (person != null) {
        Car car = person.getCar();
        if (car != null) {
            Insurance insurance = car.getInsurance();
            if (insurance != null) {
                return insurance.getName();
            }
        }
    }
    return "Unknown";
}
```

- 모든 변수가 null인지 의심하므로 변수를 접근할 때마다 중첩된 if가 추가되며 들여쓰기 수준 증가
- 이와 같은 반복 패턴 코드를 '깊은 의심 (deep doubt)'이라고 부른다.
- 들여쓰기 수준이 증가해 코드 구조가 엉망이 되고 가독성이 떨어진다.

11.1.1 보수적인 자세로 NullPointerException 줄이기

Null 안전 시도2: 너무 많은 출구

```
public String getCarInsuranceName3(Person person) {
    if (person  == null) {
        return "Unknown";
    }

    Car car = person.getCar();
    if (car == null) {
        return "Unknown";
    }
    Insurance insurance = car.getInsurance();
    if (insurance == null) {
        return "Unknown";
    }
    return insurance.getName();
}
```

- 중첩 if문을 줄일수는 있지만 null 확인 코드마다 출구가 생긴다.

⇒ 값이 있거나 없음을 표현할 수 있는 방법이 필요!!

11.1.2 null 때문에 발생하는 문제

이외의 문제점들

- **에러의 근원이다 :** NullPointerException은 자바에서 가장 흔히 발생하는 에러다.
- **코드를 어지럽힌다 :** 중첩된 null확인 코드를 추가해야 하므로 가독성이 떨어진다.
- **아무 의미가 없다 :** null은 아무 의미도 표현하지 않는다.
- **자바 철학에 위배된다 :** 자바는 개발자로부터 모든 포인터를 숨겼지만 null포인터 만은 숨기지 못했다.
- **형식 시스템에 구멍을 만든다 :** null은 무형식이며 정보를 포함하고 있지 않으므로 모든 참조 형식에 null을 할당할 수 있다. 이런 식으로 null이 할당되면서 애초에 null의 사용 의도가 퇴색되고 있다.

11.1.3 다른 언어는 null 대신 무얼 사용하나?

그루비(groovy) – 네비게이션 연산자를 도입(?.)

```
def carInsuranceName = person?.car?.insurance?.name
```

- 어떤 사람은 자동차를 가지고 있지 않을 수 있으며 따라서 Person 객체의 car 참조는 null이 할당되어 있을 수 있다.
- 마찬가지로 자동차에 보험이 없을 수도 있다.
- 그루비 안전 네비게이션 연산자를 이용하면 호출 체인에 null인 참조가 있으면 결과로 예외 대신 null이 반환

하스켈(haskell), 스칼라(scala) – 선택형 값을 저장할 수 있는 형식 제공

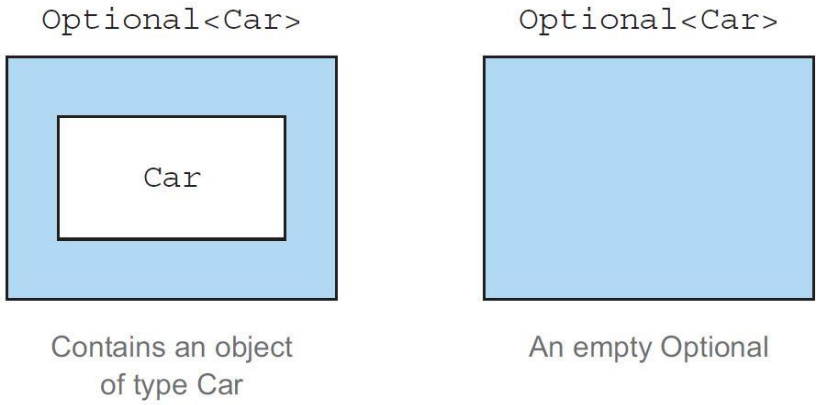
`Maybe 라는 형식을 제공` `Option[T]라는 구조를 제공`

- Maybe는 주어진 형식의 값을 갖거나 아니면 아무 값도 갖지 않을 수 있다.
- 스칼라도 T형식의 값을 갖거나 아무 값도 갖지 않을 수 있다.
- Option 형식에서 제공하는 연산을 사용해 값이 있는지 여부를 명시적으로 확인해야 한다.(null과 관련한 문제가 일어난 가능성 감소)

11.2 Optional 클래스 소개

하스켈과 스칼라의 영향을 받아서 java.util.Optional<T>라는 새로운 클래스 제공

- Optional은 선택형값을 캡슐화하는 클래스다.
- 예를 들어 어떤 사람이 차를 소유하고 있지 않다면 car 변수는 null을 가져야 한다. 하지만 Optional클래스가 값을 감싸기 때문에 값이 없으면 Optional.empty 메서드로 빈 Optional 객체를 반환한다.
- 또한 Optional을 사용함으로써 값이 없을 수 있음을 명시적으로 보여준다.



11.2 Optional 클래스 소개

```
public class Person {

  private Optional<Car> car;

  public Optional<Car> getCar() {
    return car;
  }

  public class Car {
    private Optional<Insurance> insurance;
    public Optional<Insurance> getInsurance() {
      return insurance;
    }
  }

  public class Insurance {
    private String name;

    public String getName() {
      return name;
    }
  }

}
```

데이터 모델 재정의

- 사람이 차를 소유하지 않을 수도 있으므로 Optional로 정의
- 자동차가 보험에 가입돼 있지 않을 수도 있으므로 Optional
- 보험회사는 반드시 이름이 있기 때문에 Optional X

11.3 Optional 적용 패턴

Optional로 감싼 값을 어떻게 사용할 수 있을까??

By 박찬열

11.3.1 Optional 객체 만들기

빈 Optional

```
Optional<Car> optCar = Optional.empty();
```

Null이 아닌 값으로 Optional 만들기

```
Optional<Car> optCar = Optional.of(car);
//car가 null이라면 즉시 NullPointerException이 발생
```

Null값으로 Optional 만들기

```
Optional<Car> optCar = Optional.ofNullable(car);
//car가 null이면 빈 Optional 객체가 반환된다.
```

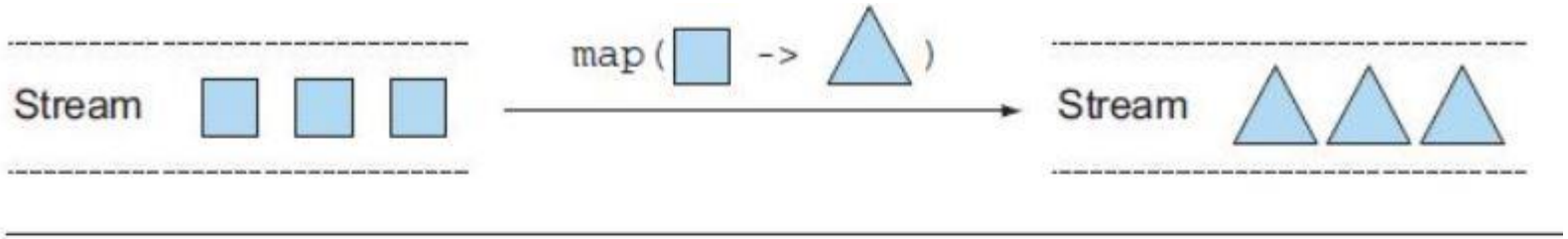
11.3.2 맵으로 Optional의 값을 추출하고 변환하기

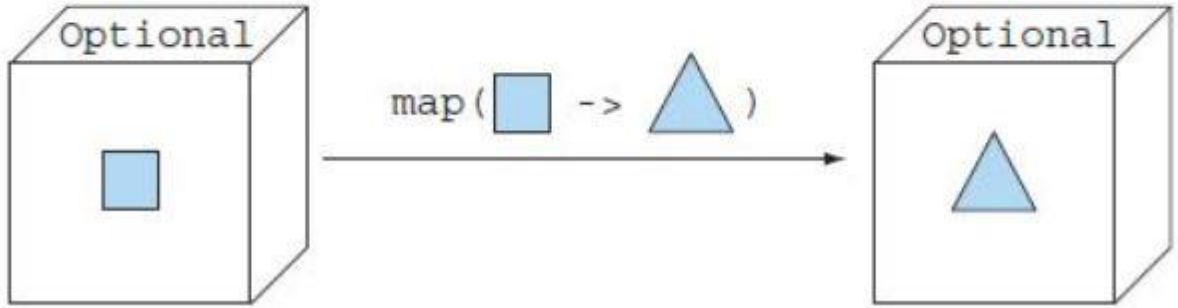
일반 객체에서 값 추출하기 – 객체가 존재하는지 확인후 추출

```
String name = null;
if(insurance != null) {
    name = insurance.getName();
}
```

이런 유형의 패턴에 사용할 수 있도록 Optional은 map메서드를 지원한다.

```
Optional<Insurance> optInsurance = Optional.ofNullable(insurance);
Optional<String> name = optInsurance.map(Insurance::getName);
```





- 스트림의 map 메서드와 개념적으로 비슷
- 요소의 개수가 한 개 또는 0개인 컬렉션으로 생각할 수 도 있다.

11.3.3 flatMap으로 Optional 객체 연결

Car, Insurance 모두 Optional type이라면?

```
public String getCarInsuranceName(Person person) {
    return person.getCar().getInsurance().getName();
}
```

Map을 이용한 코드 재구현 – 컴파일 x

```
Optional<Person> optPerson = Optional.of(person);
Optional<String> name = optPerson.map(Person::getCar)
                                 .map(Car::getInsurance)
                                 .map(Insurance::getName);
```

Optional<Optional<Car>>

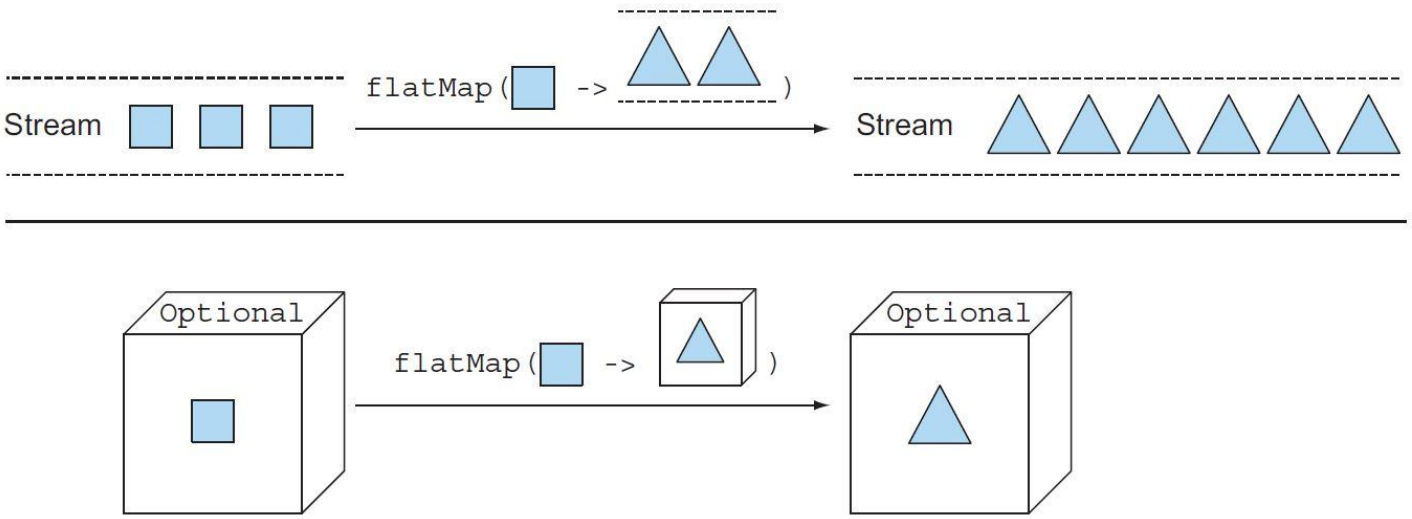
Optional<Optional<Optional<Car>>>

By 박찬

11.3.3 flatMap으로 Optional 객체 연결

해결방법 – flatMap 사용

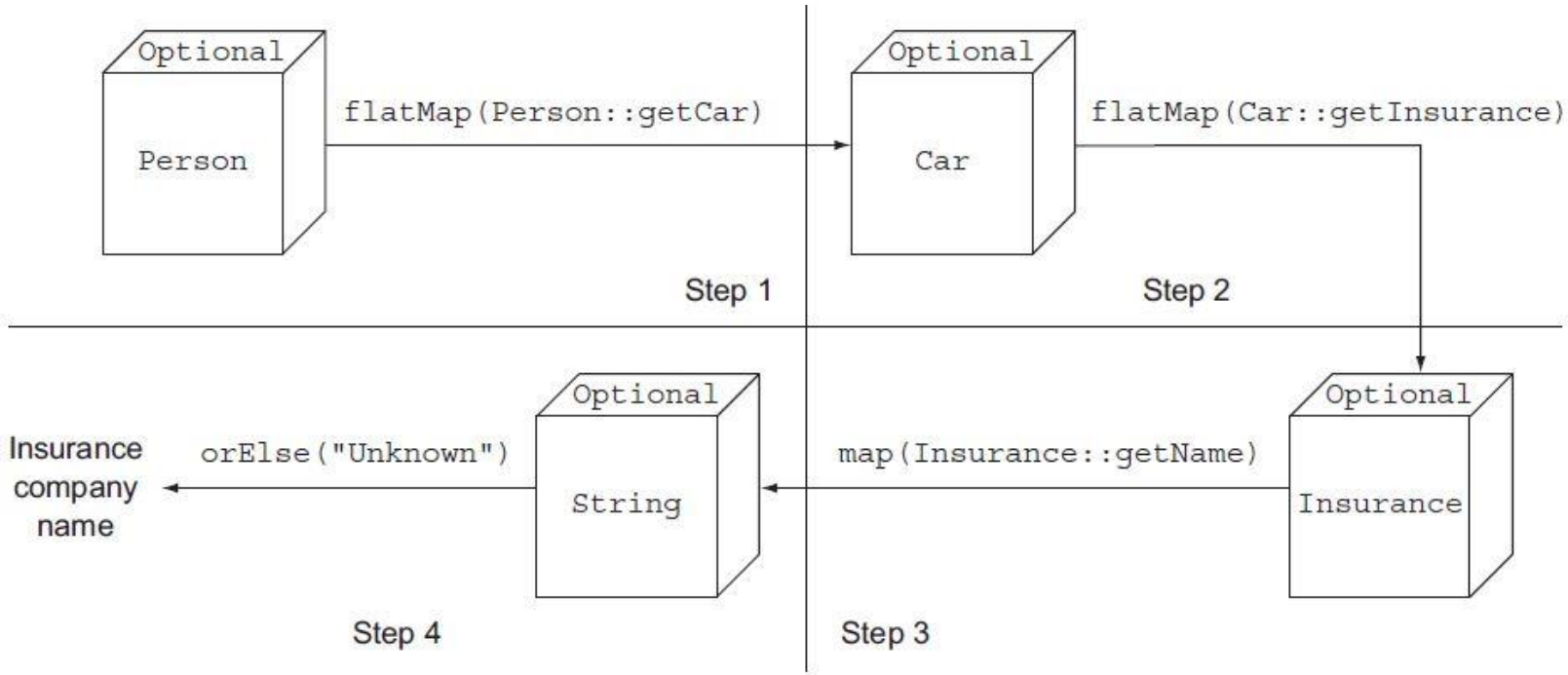
이차원 Optional을 일차원 Optional로 평준화



11.3.3 flatMap으로 Optional 객체 연결

Optional로 자동차의 보험회사 이름 찾기

```
public String getCarInsuranceName(Optional<Person> person) {
    return person.flatMap(Person::getCar)
                 .flatMap(Car::getInsurance)
                 .map(Insurance::getName)
                 orElse("Unknown") //결과 Optional이 비어있으면 기본값 사용
}
```



By 박찬

11.3.4 Optional 스트림 조작

사람 목록을 이용해 가입한 보험 회사 이름 찾기

```
public Set<String> getCarInsuranceNames(List<Person> persons) {
    return persons.stream()
        .map(Person::getCar)
        .map(optCar -> optCar.flatMap(Car::getInsurance))
        .map(optIns -> optIns.map(Insrance::getName))
        .flatMap(Optional::stream)
        .collect(toSet())
}
```

Stream<Optional<Car>>

Stream<Optional<Insurance>>

Stream<Optional<String>>

Stream<String>

11.3.5 디폴트 액션과 Optional 언랩

Get()

- 값이 있으면 해당 값을 반환하고 없으면 NoSuchElementException 발생
- 반듯시 값이 있다고 가정할 수 있는 상황이 아니면 사용x
- 이때 사용하게 되면 null 확인 코드를 넣는 상황과 크게 다르지 않다.

orElse(T other)

- 값을 포함하지 않을 때 기본값을 제공

orElseGet(Supplier<? Extends T> other)

- Optional이 비어있을 때만 기본값을 생성
- 디폴트 메서드를 통해서 실행해야 한다.

orElseThrow(Supplier<? Extends X> exceptionSupplier)

- Optional이 비어있을 때만 예외를 발생 get과 비슷 하지만 예외의 종류를 선택할 수 있다.

ifPresent(Consumer<? Super T> consumer)

- Optional이 비어있을 때만 예외를 발생 get과 비슷 하지만 예외의 종류를 선택할 수 있다.

By 박찬

11.3.6 두 Optional 합치기

가장 저렴한 보험료를 제공하는 보험회사를 찾는 로직

```
public Insurance findCheapestInsurance(Person person, Car car) {
    // 다양한 보험회사가 제공하는 서비스 조회
    // 모든 결과 데이터 비교
    return cheapestCompany;
}
```

```
public Optional<Insurance> nullSafeFindCheapestInsurance(
    Optional<Person> person, Optional<Car> car) {
    if (person.isPresent() && car.isPresent()) {
        return Optional.of(findCheapestInsurance(person.get(), car.get()));
    } else{
        return Optional.empty();
    }
}
```

만약 인수로 전달한 값 중 하나라도 비었으면 빈 Optional<Insurance>를 반환한다.

11.3.7 필터로 특정값 거르기

보험 회사 이름이 CambridgeInsurance 인지 확인해야 한다면?

Insurance 객체가 null인지 확인한 다음 getName 메서드 호출

```
Insurance insurance = ...;
if(insurance != null && "CambiridgeInsurance".equals(insurance.getName())) {
    System.out.prinln("ok");
}
```

Optional 객체에 filter 메서드를 이용할 수 있다.

```
Optional<Insurance> optInsurance = ...;
optInsurance.filter(insurance -> "CambridgeInsurance".equals(insurance.getName()))
            .ifPresent(x -> System.out.println("ok"));
```

By 박찬

11.4 Optional을 사용한 실용 예제

Optional 클래스를 효과적으로 이용하려면 잠재적으로 존재하지 않는 값의 처리 방법을 바꿔야 한다.

By 박찬

11.4.1 잠재적으로 null이 될 수 있는 대상을 Optional로 감싸기

Map<String, Object> 형식의 맵이 있는데 key로 값에 접근할때 해당 값이 없으면 null이 반환될 것이다.

Map에서 반환하는 값을 Optional로 감싸서 이를 개선할 수 있다.

```
Optional<Object> value = Optional.ofNullable(map.get("key"))
```

By 박찬

11.4.2 예외와 Optional 클래스

문자열을 정수로 바꾸지 못할 때 NumberFormatException을 발생시킨다.

```
public static Optional<Integer> stringToInt(String s) {
    try {
        return Optional.of(Integer.parseInt(s));
    } catch (NumberFormatException e) {
        return Optional.empty();
    }
}
```

11.4.3 기본형 Optional을 사용하지 말아야 하는 이유

- OptionalInt, OptionalLong, OptionalDouble가 존재
- Optional의 경우 최대 요소 수는 한 개 이므로 기본형 특화 클래스로 성능을 개선할 수 없다.
- 기본형 특화 Optional은 map, flatMap, filter를 지원하지 않는다.