Optional

Optional to rozwiązanie zaprojektowane w celu dostarczaniaalternatywy dla obiektu null oraz wszechobecnego wyjątku NullPointerException.

Jest wielu przeciwników wartości null, nawet twórca konceptu TonyHoare opisuje go jako „my bilion-dollar mistake”.

null jest bardzo często reprezentacją braku wartości, i w tym właśnie przypadku może zostać zastąpiony przez obiekt typu Optional.

Problem, który napotykamy podczas korzystania z nulla, towspomniany wcześniej NullPointerException. Jeśli spróbujesz wywołać jakąś metodę na referencji wskazującej na null, aplikacja wybucha.

Poprzez szereg metod klasy Optional, powinniśmy być w staniezabezpieczyć się przed przypadkowym odwołaniem do *null reference*.

Czym właściwie jest ten cały Optional ?

Jest to Generyczny (<T>) pojemnik na zmienną dowolnego typu, która może mieć wartość null

Jak tworzymy obiekty typu Optional ?

- empty() - tworzy pusty Optional z wartością null w środku.
- of(T value) - tworzy Optional z podaną wartością. W przypadku przekazania null dostaniemy nullPointerException (już w momencie przekazania parametru)
- ofNullable(T value) - również tworzy Optional z podaną wartością, ale w przypadku przekazania null nie zostanie zgłoszony wyjątek.
- Optional(T value) - konstruktor rzucający błąd w przypadkuprzekazania wartości null. (prywatny, wywoływany przezOptional.of)

Dwie metody pozwalające nam pobrać wartość obiektu

- isPresent() – boolean, sprawdza czy obiekt w środku jest wartością, false jeśli null
- get() – pobranie przechowywanego obiektu. Jeśli niedostępny – NoSuchElementException (powinno być:getOrElseThrowNoSuchElementException)

Przykład użycia

```
private String getCompanyFirstUserName(final Holding holding) {
    if (holding != null) {
        final Company company = holding.getCompanies().get(0);
        if (company != null && company.getUsers() != null) {
            final User user = company.getUsers().get(0);
            if (user != null && user.getFirstName() != null) {
                final String result = user.getFirstName();
                if (result.length() > 0) {
                    return result;
                }
            }
        }
    }
    return "not found";
}
```

może zostać zastąpione przez:

```
   private String getCompanyFirstUserName(final Holding holding) {
        return Optional.ofNullable(holding)
            .map(Holding::getCompanies)
            .map(Vector::firstElement)
            .map(Company::getUsers)
            .map(Vector::firstElement)
            .map(User::getFirstName)
            .filter(name -> name.length() > 0)
            .orElse("not found");
    }
```

Gdzie nie używać Optional?

pola w klasach DTO nie powinny być deklarowane jako Optional

```
private Optional<HashMap<String, Integer>> data;
```

ponieważ zaciemnia to obraz oraz ogranicza możliwości serializacji klas. Zamiast tego można to zaimplementować w getterze

```
private Optional<HashMap<String, Integer>> getData() {
    return Optional.ofNullable(data);
}
```

Podobnie rzecz ma się w stosunku to przekazywania jako parametr konstruktora lub funkji – lepiej

przekazać zwykły obiekt i w środku metody opakować w Optional.

```
private Component component;

public MyClass(Component c) {
    this.component = Optional.ofNullable(c).orElse(new Component());
}
```