Podstawy Java

Optional oraz Lambda

Optional

Optional - został stworzony z zamysłem zredukowaniu ilości wystąpień błędu NullPointerException, który jest bardzo częstym błędem występującym w kodzie. Do wyjaśnienia na przykładzie klasy Optional, posłużę się modelem dziennika (szkolnego):

```
public class Journal {
    private List<Student> studentList = new ArrayList<>();

    public void addStudent(Student t) {
        studentList.add(t);
    }

    public Student getStudentAtIndex(int index) {
        if (studentList.size() <= index) {
            return null;
        } else if (index < 0) {
            return null;
        }

        return studentList.get(index);
    }
}
```

W klasie dodałem metodę pobrania **i-tego** studenta z dziennika metodą **getStudentAtIndex**, której parametrem jest **index** – i. Jeśli podany student się nie znajdzie, bardzo częstą praktyką jest zwrócić wartość **null**, co wymusza dokonania sprawdzenia czy wartość jest **null'em** i obsługi tego wyjątku na osobie która używa tej metody ponieważ jeśli użyję obiektu Journal w następujący sposób:

```
Journal j = new Journal();
Student st = j.getStudentAtIndex(1);

System.out.println(st.getAge());
```

To uruchomienie kodu spowoduje wyjątek **NullPointerException** ponieważ używam metody **getAge** na obiekcie null (student o indeksie 1 nie istnieje w dzienniku).

Optional

Do zabezpieczenia przed tego typu błędami pomoże nam klasa **Optional**, która jest klasą generyczną, czyli w deklaracji obiektu możemy podać typ jaki będzie przechowywał obiekt **Optional**. Użyjemy go w metodzie pobrania studenta:

```
public Optional<Student> getStudentAtIndex(int index) {
    if (studentList.size() <= index) {
        return Optional.empty();
    } else if (index < 0) {
        return Optional.empty();
    }

    return Optional.ofNullable(studentList.get(index));
}
```

Optional mówi nam, że zwracany obiekt może przyjmować wartość, ale może jej również nie przyjmować i być wartością null. Pomaga nam zadeklarować intencję i zwrócić uwagę na fakt, że wynikiem metody może być również wartość **null.** Do użycia Optionala:

```
Journal j = new Journal();
j.addStudent(new Student());
Optional<Student> st = j.getStudentAtIndex(3);

if (st.isPresent()) {
    Student student = st.get();
    System.out.println(student.getName());
}
```

Ponieważ wiem, że wartość zwracana to Optional, nie zapomnę o obsłużeniu przypadku, kiedy to podany student nie zostanie mi zwrócony. Metoda **isPresent** mówi nam czy obiekt zawiera wartość, natomiast metoda **get** pobiera ją z obiektu **Optional**.

Optional & Lambda

Alternatywnie dopuszczalne jest użycie innej metody z Optional. Możemy użyć metody **ifPresent** która przyjmuje jako parametr **obiekt na którym wywoła metodę accept**. Ten obiekt musi implementować interfejs Consumer, który posiada tą metodę. Klasa Consumer jest generyczna. W zależności od typu mu zadeklarowanego, taki będzie typ parametru w metodzie **accept** tego obiektu.

```
Consumer<Student> consumer = new Consumer<Student>() {
    @Override
    public void accept(Student student) {
        System.out.println(student.getName());
    }
};

st.ifPresent(consumer);
```

Metoda **ifPresent** spowoduje wywołanie metody accept tego obiektu, a w nim możemy dopisać obsługę jego kodu. Bardziej popularne jest użycie tego kodu z anonimową deklaracją obiektu:

```
st.ifPresent(new Consumer<Student>() {
    @Override
    public void accept(Student student) {
        System.out.println(student.getName());
    }
});
```

A najkrótsza metoda to z użyciem Lambdy (skróconego zapisu):

```
st.ifPresent(student -> {
    System.out.println(student.getName());
});
```

Optional & Lambda

Na rysunku zaznaczone powiązania: