Класс Optional

Класс Optional

При работе с объектами и ссылками на них в идеальном случае каждая ссылка должна указывать на объект. Но в реальности это не всегда так.

Посмотрите на код:

```
Map<Integer, String> numbers = new HashMap<>();
numbers.put(1, "Один");
numbers.put(2, "Два");
numbers.put(3, "Три");
```

Здесь у вас есть Map, которая хранит соответствие числа и его наименования. Теперь представьте, что кто-то захотел получить наименование числа 5:

```
System.out.println(numbers.get(5));
```

Изображения: Даниил Пилипенко

Класс Optional

Для этого числа нет значения, поэтому метод get вернёт null. А теперь представьте, что вы не знаете об этом и попробуйте вывести наименование числа «5» в верхнем регистре:

```
System.out.println(numbers.get(5).toUpperCase);
```

При запуске программы получите ошибку:

```
Exception in thread "main" java.lang.NullPointerException
```

Изображения: Даниил Пилипенко

NullPointerException

NullPointerException — одна из самых распространённых ошибок в работе Java-программиста. Она может возникнуть в нескольких случаях:

- вызов метода у экземпляра объекта, который является null
- доступ или изменение поля у экземпляра объекта, который является null
- вызов метода length для получения длины массива, который является null
- доступ или изменение элемента массива, который является null

Класс java.util.Optional

В JDK 8 был представлен класс java.util.Optional, который позволяет более гибко обрабатывать нулевые ссылки. Фактически он представляет собой объект-контейнер, в котором содержится значение, которое может быть null.

Отсюда и название класса — у вас есть две опции состояния объекта — пустое и непустое. И понять, в каком вы состоянии, можно, вызвав встроенные методы класса Optional.

С помощью Optional вы показываете тем, кто использует ваш код: «Внимание! Этого объекта может не быть, нужно обрабатывать код с осторожностью».

Методы класса Optional

- Метод **Optional.of** возвращает вам Optional, содержащий ваш объект. Если объект будет равен null, то метод of выбросит NullPointerException. Применение этого метода гарантирует то, что optional будет не пустой
- Метод **Optional.ofNullable** служит для работы с объектами, которые могут быть null. Этот код вернёт вам пустой Optional
- Метод **Optional.empty** — возвращает пустой Optional-объект
- Методы **ifPresent** и **isPresent** необходимы для непосредственной работы с Optional

Получение объекта из Optional

- Самый первый и очевидный способ — это вызов **метода get()**. Этот метод возвращает содержимое Optional. У этого способа есть свой недостаток. Если Optional окажется пустым, то программа выбросит исключение **NoSuchElementException**
- Если Optional будет пустым, очевидным решением будет возвращать какое-то значение по умолчанию. Это можно сделать с помощью **метода orElse()**
- Также можно не только вернуть значение, но и указать на метод, который вернёт нужное значение. Или указать на исключение, которое необходимо выбросить с помощью **метода orElseThrow()**
- **Метод filter** возвращает Optional с исходным объектом, если условие возвращает true, и пустой Optional, если условие возвращает false

Ассерты

В Java 14 появился ещё один способ работы с nullable-значениями — так называемые ассерты. Для того чтобы проверить переменную на null, достаточно написать ключевое слово assert и условие.

```
Computer computer = new Computer();
assert computer != null;
```

Там, где вы написали assert, произойдёт проверка, и если условие вернёт false, то программа выбросит исключение:

```
Exception in thread "main" java.lang.AssertionError:
```

Также можно указать сообщение, которое увидит пользователь в случае ошибки:

```
Computer computer = new Computer();
assert computer != null : “computer is null”
```

Изображения: Даниил Пилипенко

Выводы

С помощью класса Optional и его инструментов вы можете существенно сократить количество написанного кода и избавиться от лишних проверок на null.