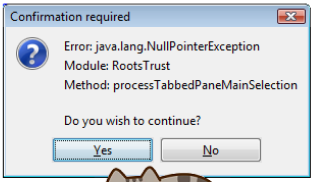
10 способов эффективно справиться с Null в Java



Когда я только начал программировать на Java, Null очень быстро стал моим главным врагом. Сейчас я могу с уверенностью сказать, что битва между мной и Null с успехом завершилась. Большинство NullPointerExceptions случаются по неосторожности, и, чтобы помочь вам научиться справляться с ними, я подготовил небольшой список того, что я обычно делаю, когда встречаюсь Null.

Не усложняйте

Справиться с Null бывает порой непросто, поэтому мы должны писать код настолько чисто и понятно, насколько это возможно. Например, использование методов Objects, класса Optional или даже отдельного метода, использующего Optional, вместо простой проверки на Null на деле оказывается очень плохим решением.

if (Optional.ofNullable(myVariable).isPresent()) // плохо

if (Objects.nonNull(myVariable)) // лучше, но все равно плохо

if (myVariable != null) // хорошо

Используйте методы Objects в качестве предикатов Stream

Хотя использование Objects.isNull и Objects.nonNull не очень хорошо подходит при обычной проверки на Null, однако они отлично подойдут для использования в потоках.

myStream.filter(Objects::nonNull)

myStream.anyMatch(Objects::isNull)

Никогда не передавайте Null в качестве аргумента

И вот почему:

* Нужно разбираться, сможет ли наша функция и следующие после нее функции справиться со значением Null.
* При изменении таких функций нужно быть очень аккуратным, ведь, если что-то пойдет не так, нам придется перелапачивать весь код в попытке найти ошибку.

Данных проблем можно избежать, если никогда не использовать Null в качестве аргумента функции.

void kill() {

kill(self);

}

void kill(Person person) {

person.setDeathTime(now());

}

Проверяйте аргументы публичных API

Конечно, вы и ваша команда можете придерживаться принципа — «никогда не использовать Null в качестве аргумента функции», но что, если у вас есть свой публичный API, и его пользователи даже не слышали о таком принципе? Поэтому лучше всегда проверять на корректность аргументы, передаваемые в ваш API.

public Foo(Bar bar, Baz baz) {

this.bar = Objects.requireNonNull(bar, "bar must not be null");

this.baz = Objects.requireNonNull(baz, "baz must not be null");

}

Эффективно используйте Optional

До Java 8 частой практикой было то, что метод возвращал Null, если значение отсутствовало. Теперь у нас есть класс Optional, который предупреждает вас, что возвращаемое значение может быть упущено.

Optional<String> makingYouCheck() {

// stuff

}

makingYouCheck().orElseThrow(ScrewYouException::new);

Возвращайте пустые коллекции вместо Null

Мы уже знаем, что возвращать Null — плохо и, что избежать этого нам поможет класс Optional. Но вещи меняются, когда дело доходит до коллекций, ведь коллекции могут содержать любое количество элементов, в том числе и 0. Тем более в Collections есть специальные методы emptyXxx, которые могут возвращать такие коллекции. Поэтому лучше использовать коллекции, нежели усложнять вещи с Optional или возвращать Null.

List<String> findSomething() {

if (someCondition) {

return Collections.emptyList();

}

// stuff

}

Optional не подходит для полей

Согласно правилам [инкапсуляции](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) вы должны иметь полный контроль над значениями полей, включая Null. С другой стороны, что если мы сделаем так, что поля могут и не иметь значений? Тогда это приведет к нескольким проблемам:

* Как писать конструктор или сеттер для такого поля?
* Нам придется иметь дело с Optional, даже, если мы уверены, что значение существует.
* Как AutoMapper’ы будут справляться с такими полями?

Вывод: не используйте Optional с полями.

Используйте исключения вместо Null

Если что-то пошло не так: всегда используйте исключения, вместо того, чтобы возвращать Null.

Тестируйте свой код

Тестирование — хороший способ обнаружить NPE и исправить их. Поэтому никогда не стоит выпускать проект, предварительно не протестировав его.

Двойная проверка

Каждый раз, когда вы предполагаете, что какое-то выражение не может оказаться Null — дважды проверьте ваши выводы. Используйте документацию к API, спросите коллег, задайте вопрос на Stack Overflow. И всегда помните, что лучше перестраховаться, чем потом возиться с Null.