Hi there. Null. How painful is that and why should it be avoided?

First, you have to handle it. Whatever you do, you should keep in mind every single object may be null at any time. So, you have to use the if-else or try-catch statement to perform the null checks. If the object turns out to be null we have to handle it in some special way if not we just do what we wanted. This crap pollutes the business logic with useless technical details. The program becomes less readable and maintainable what makes it expensive and unpleasant to work with. It's especially painful when we talk about lambdas or collection processing because the null checks can double such type of code that pisses people off.

Second, null reference ruins all type concept. Well, each type provides kinda contract, which contains the list of fields and methods, which every object of the type must have. So, each typed entity must fulfill the contract. The main purpose of that is to let you trust your own code. You never get no such method exception if types work fine. But, null doesn't think so. It's like a demon just pretends to be a typed object but when the contract must be fulfilled it reviles itself and fucks you up throwing null pointer exception. So, no trust to objects, welcome to the null check world of shit.

The root of the problem is that in many old languages like java, C# and C++ reference types are nullable by default. Null itself is nothing but a pointer to the first cell in your memory. It is nothing it doesn't mean anything. Almost every programmer nowadays have a deal with abstractions, but null breaks everything. I'm glad Kotlin, Dart and Swift got rid of that, so it looks like we're going right way.

So, how to avoid this crap?

First, null Object. Beautiful solution but it is not always possible. Just use some default value wherever you'd use null. It works wonderful if we're talking about a collection. If there's no way to fill it, just leave it empty and everyone who can write a bubble sort will handle such case.

Fail Fast. Sometimes you just cannot handle the situation yourself. So, just throw an exception. Be careful here, exceptions build stack trace what may totally ruin all performance. Besides, it's not a good idea to throw it as a business value, if you wanna do it think twice.

Nullable. In this solution you just say explicitly value may not exist. So, null just won't be able to hide among normal objects. In java, we use Optional class for such purpose. It doesn't just show, but forces you to check if the value exist, or it fails really fast.

Thank you for watching. Like the video if you enjoy it, or dislike if you don't. Leave a comment below. Please always avoid null. And remember the code ist das Essen, und wir sind die Jager. Farewell.