05

Realizando chamadas seguras

Transcrição

[00:00] Como vimos anteriormente, essa abordagem na qual a gente colocou o "!!" e uma property, que pode ter o valor nulo, a gente viu que é muito arriscado esse tipo de abordagem, porque no momento em que a gente tem o valor da "view", tudo funciona normalmente, mas no momento que a gente esqueceu do valor e ele foi "null", a gente tomou Null Pointer Exception.

[00:17] Em outras palavras, quando a gente está lidando com esse tipo de situação, a primeira coisa que a gente precisa fazer é, justamente, evitar isso o máximo possível, ou melhor, não usar. Portanto, vamos apagar essa abordagem que a gente viu aqui, que permite com que o nosso código que tenha o valor "null" seja executado sem nenhum tipo de tratamento e vamos usar uma abordagem diferente, para poder lidar com esse tipo de situação. O que seria essa abordagem diferente?

[00:39] A ideia é fazer com que, essa execução da "view", chamando os seus membros, só seja chamada de uma maneira segura, o que seria uma maneira segura? Uma maneira da qual a gente garante que o valor dentro dela, por mais que ele possa ser "null", a gente vai garantir que ele não é "null", a gente só vai executar esse código, se ele não "null", a gente vai ter que fazer essa garantia. E é justamente isso que a gente vai ver agora. Como que a gente pode fazer para poder pegar essa nossa property, e indicar que ela não é "null"?

[01:06] A gente pode fazer da seguinte maneira, a gente pode vir aqui e dar um espaço nesse "totalReceita", que a gente está pegando e fazer um teste, como a gente está acostumado em programação, por meio da estrutura do "if". Então o "if" aqui e, agora, a gente pode chegar aqui e falar o seguinte, "view", se você for diferente de "null", eu quero escutar todo esse código, considerando que você não seja "null", é desta maneira que a gente consegue fazer aqui no Kotlin.

[01:33] Aliás, uma das maneiras que a gente consegue fazer no Kotlin, para garantir que um valor que pode ser "null", na verdade, não está sendo "null", neste momento. Esse código que a gente tem um "with" e, aqui, que a gente faz todas as atribuições, a gente pode colocar dentro desse "if", que garante para a gente, e o nosso código, agora, volta a compilar, porque isso acontece?

[01:54] Porque no Kotlin, quando ele faz esse tipo de teste, que está garantido que esse valor não é "null", ele faz uma conversão inteligente para a gente, o que significa essa conversão inteligente? Ele trata essa "view", como um valor que não é "null", ele já faz essa conversão, essa conversão, tecnicamente falando, é conhecida como SmartCast. Se a gente olhar aqui, essa "mensagenzinha" aqui do balãozinho, que ele está dando como spoiler, ele até fala que foi feito um SmartCast para o "android.view.View", que é a classe.

[02:23] Só que a ".View" que é a classe, para um valor que não é "null", ele já conseguiu garantir isso para a gente. Então o Kotlin é bem bacana nessa parte de facilitar a vida do programador e, tentar também, converter as coisas automaticamente, considerando que a gente já fez o teste aqui, a gente não precisa mais ter que provar mais nada, a gente está provando neste momento, essa é uma das abordagens fazer isso.

[02:44] Só que, agora, percebam que a gente fez um código, que é bem similar, também, no Java, que é uma das maneiras que a gente também faria uma garantia de, o valor ser "null" ou não, lá no Java e, como a gente viu, o Kotlin tem bastante essa abordagem, de tentar ser um código um pouquinho mais sucinto, um pouquinho mais fácil, para o programador. Então, essa é uma abordagem válida, só que existem outras técnicas que a gente pode aplicar, aqui, para poder, também, considerar que o nosso valor não é "null" e fazer uma chamada segura.

[03:12] Vamos ver como são essas técnicas, deixa eu apagar novamente o "if" e vamos ver como seria, estou apagando aqui, eu vou usar o atalho Ctrl+Alt+L para ele para ele organizar o código para a gente, voltamos para o mesmo cenário de anteriormente, a gente viu que a gente tem o "if" e, agora, a gente vai ver uma outra abordagem, que é conhecida também como "chamada segura", ou, no caso, tecnicamente, a "safe call", como funciona essa "chamada segura"?

[03:36] Basicamente, a gente coloca, por exemplo, uma "?" aqui, e nesse momento que a gente coloca uma "?" e uma variável, que, pode ser nula, como é o caso da nossa "view", a gente consegue chamar os seus membros sem nenhum problema, porque a gente está garantido, que essa chamada aqui, que essa variável "view", não é "null", dessa maneira, chamando a "safe call", que é por meio dessa "?".

[03:58] Só que, agora, tem alguns detalhes, quando a gente faz uma "safe call", por exemplo, a gente garante que, a "view", que é quem chamou ela, não é "null", só que os seus membros não estão sendo garantidos que não são "null". Portanto, esse "resumo_card_view", aliás, "resumo_card_receita", que é aquela nossa "textView", agora, é um valor que pode ser "null", portanto, por mais que a gente tenha chamado essa "safe call" aqui, todo esse código dentro do "if", já não pode mais ser executado.

[04:26] Porque isso acontece? Porque a gente não consegue, também, garantir, apenas com essa "safe call", que o "resumo_card_receita" não é "null", como a gente pode ajustar essa abordagem? Se a gente perceber, a gente vai, meio que, sacrificar o nosso "if", porque a gente não consegue fazer, por exemplo, uma chamada de "safe call", aqui, novamente, a gente não consegue fazer isso. Então, para a gente conseguir usar a "safe call", da maneira como a gente está utilizando até o momento, a gente vai fazer um código, da seguinte maneira.

[04:52] Só para mostrar para vocês como funciona, a gente vai pegar essa primeira chamada da "safe call" e, aqui, a gente pode fazer, novamente, uma chamada da "safe call", que é chamada de "safe call" encadeada, ou então, uma "chain" dela, e, agora, a gente vai lá e pode chamar os seus membros sem nenhum problema, a "safe call" funciona dessa maneira, todo mundo que a gente chama de membro, a gente precisa garantir que eles não são "null".

[05:16] Por mais que a gente só tenha garantido que o primeiro não é "null", a gente também precisa garantir o segundo, mesmo que eles não sejam "null" por padrão, eles acabam sendo considerados. Vamos só pegar essa parte para a gente poder ver como funciona, "view?.resumo_card_receita?.text", aqui, "totalReceita", só para a gente ver como que fica, olha como que fica, utilizando apenas a "safe call". Dessa maneira a gente tá garantido que, ele só vai ser executado, quando todas as chamadas forem seguras.

[05:54] Basicamente, por debaixo dos panos, ele faz um teste aqui na "view", testou a "view", ele vai lá e faz um teste no "resumo_card_receita" e vai lá e executa. Basicamente, por debaixo dos panos, ele faz isso, só que reparem como ficou o nosso código, a gente até sacrificou o nosso "with" para isso, que não é um dos comportamentos que a gente espera, o que a gente espera, é fazer algo bem similar ao que a gente viu lá no nosso "if", que permitiu executar o nosso código, como a gente espera.

[06:20] Que é por meio do "if", reutilizar essa chamada de objeto e tudo que a gente fez anteriormente, como a gente pode fazer para, executar um código, que vai considerar que a apenas a "view" é segura e o resto pode não precisa verificar mais nada, da mesma maneira que a gente viu lá no "if"?

[06:37] Deixa eu só voltar o código, voltando, uma das maneiras que a gente pode fazer isso, que é bem comum, também, na comunidade do Kotlin, é justamente fazendo a seguinte chamada. A gente pega, por exemplo, a "view", que o único componente, aqui, no caso, que é a única variável que precisa realmente ser garantida de "null" e a gente vai lá, e faz uma "safe call", fizemos uma "safe call". Agora, a gente pode chamar, uma função, que é uma função bem genérica, que pode ser chamado por qualquer tipo de objeto, que é chamada de "let".

[07:07] Veja que, nessa função "let", ela já faz uma implementação de expressão lambda, assim como a gente viu lá na nossa collections, e o que está acontecendo aqui dentro? Essa "let" quando ela é chamada, ela pega, justamente, esse

objeto que está chamando ela, passa para dentro do escopo dela, então, o que está tendo de diferente aqui, já que ela tá só chamando o objeto e passando para o escopo, o que está tendo de diferente?

[07:27] A diferença é a seguinte, que no momento em que a gente fez essa "safe call", a gente garante que a "view" não é mais "null". A gente tem uma "view" que não é nula e, se a gente mandar uma "view" que não é nula, para o escopo de uma função, que vai receber ela mesmo, a gente vai ter a "view" como não nula. Portanto, a gente pode chegar aqui e colocar, por exemplo, que todo o código que for dentro desse "let", que vai receber a "view", que não é mais nula, vai ser executado dessa maneira.

[07:53] É uma das técnicas que a gente pode utilizar, para executar um código, que a gente quer considerar que um objeto que pode ser "null", não seja "null", é usando essa abordagem de "safe call", junto com essa chamada do "let", que vai pegar o objeto que chamou ela e vai mandar para dentro do escopo dela. Tanto que, se a gente tirar agora, por exemplo, essa "view", ele considera novamente que a nossa "view" é um tipo, que pode ser "null", mas agora, quando a gente faz isso, ela considera que a nossa "view" é normal, porque?

[08:24] Porque a gente está mandando ela para dentro do "let", quando a gente testa e garante que ela não é mais um valor nulo, por meio da "safe call", inclusive, quando a gente manda essa variável para cá, quando a gente manda esse objeto para dentro do "let", a gente pode, até mesmo, usar o nosso objeto subintendido, que é o próprio "it", a gente já entende que o "it", é o próprio caminho que está sendo chamado aqui pelo "let". Então, é uma da abordagens que a gente pode fazer.

[08:46] Inclusive, agora que a gente viu essa abordagem do "let", vamos só colocar, todo esse código de "let" para os outros membros, também, e vamos ver o que acontece. Vamos ver aqui, testar para ver se realmente está funcionando essa parte do "let", vamos lá. Agora eu estou colocando para parte da despesa, se você quiser manter a "view" ou o "it", fica à vontade em manter da maneira que você preferir, aqui, temos o "let", agora vamos colocar o "let" aqui na parte do total, claro, deixa só eu colocar, também, aqui, a "view".

[09:17] "view?" e aqui eu coloco ".let", agora, eu vou lá, e coloco o nosso código. Olha o que a gente fez, a gente colocou, em todos os pontos que tinham um risco de ser um valor "null", com essa chamada do "let", garantindo com uma "safe call", a nossa property view. Agora a gente conseguiu garantir esse código, eles só serão executados, apenas quando a "view" não for valor "null", ficou bem mais sucinto do que só colocar aquela estrutura do "if□, usando operadores lógicos.

[09:47] É das técnicas, até bem comum aqui no Kotlin, é bem comum, na comunidade do Kotlin, utilizar esse tipo de técnica para poder garantir, que, um valor, que pode ser "null", seja executado aqui, sem que seja "null", usando "it" e outras técnicas que existem por aí. Agora que a gente conseguiu fazer essa modificação, vamos ver o que acontece, Alt+Shift+F10, vem aqui, veja que o Android Studio conseguiu executar.

[10:12] Reparem o que acontece, por mais que a gente tenha segurado, ele não quebrou nossa App, isso é algo bem legal, ele não quebrou App. Só que muitas das coisas, aqui, não foram executadas, como eu havia comentado com vocês, o código "adiciona_receita", por exemplo, não foi executado, o código "adicionaDespesa", também não e nem o total, eles não foram executados. Então, por mais que a gente tenha garantido essa parte de não quebrar a App, ainda assim, a gente apresentou um aspecto que não é tão legal para o nosso usuário.

[10:39] A gente não resolveu todos os problemas, a gente só garantiu que, o nosso código, não quebrasse. Por mais que a gente tenha feito todos esses processos, não é uma maneira ideal da gente lidar com a situação que estamos tendo agora, que é, justamente, esquecer de inicializar uma variável. A gente teve só um aviso prévio de que, o comportamento que a gente espera, não foi executado, mesmo assim, ainda não é uma maneira ideal.

[11:00] Inclusive, se a gente chegar aqui e tentar adicionar uma transação, ele vai lá e quebra, porque ele quebra? Porque, se a gente voltar, aqui, na nossa Activity, reparem que aqui, ele tá tentando fazer esse cast. E quando ele tem

essa parte aqui do cast, ele não entra naquela questão do "Null Safety", como a gente viu anteriormente, de assegurar com que, essa operação, só pode acontecer caso o valor não seja nulo, ele não tem essa abordagem.

[11:25] Se a gente até ver, aqui, o erro que a gente tomou, a gente tomou o erro de cast, que ele não pode fazer um cast de um valor que é nulo. Então, por mais que o Kotlin esteja ali, para ajudar a gente a evitar o máximo possível de Null Pointer Exception, a gente corre bastante risco, quando a gente tá usando o código que tende a ser "null", ou seja, usar esse tipo de abordagem para, simplesmente, colocar um valor aqui na property, já que, a gente não pode inicializar nesse momento, não é a abordagem para esse tipo de cenário.

[11:56] Visando evitar esse tipo de situação, logo mais, a gente ver, algumas técnicas que a gente pode utilizar, para evitar com que a gente tenha que lidar com esse tipo de valor. Porque a gente viu o tanto de trabalho que a gente teve, por exemplo, no nosso "ResumoView", a gente teve que perceber um valor que não é "null", a gente teve que garantir todos os pontos que o código não está sendo "null", para poder executar os membros dele, a gente teve um trabalho e tanto.

[12:20] Então, por mais que o Kotlin nos avisa, ele evita que a gente tome esses erros, a gente tem muito trabalho e mesmo assim a gente corre riscos, considerando todos esses aspectos, logo mais a gente vai ver alguma técnica ou algumas técnicas, que evitam esse tipo de abordagem, que a gente tenha que lidar com valores nulos, esse era só um feedback que eu queria passar para vocês, que é algo que pode ser bem comum, no momento que você estiver lidando com código Kotlin.

[12:42] Porque, de repente, você que fazer uma conversão de um código que está em Java, você vai ter que lidar com valor nulo em alguns momentos e você vai saber como você pode lidar aqui dentro do Kotlin, qual é a abordagem que ele considera, quando ele tá lidando com os valores nulos, porque, de repente, como é o caso aqui do Bundle, a gente não tem como ter controle, que esse Bundle sempre vai ser um valor real ou que não vai ser nulo, a gente vai ter que lidar com ele.

[13:03] Por mais que, quando a gente escrever o nosso código, a gente vai evitar o máximo possível essa abordagem, os códigos que a gente utilizar, que a gente pode receber valor nulo, a gente sabe como lidar. A gente sabe que a gente tem muito trabalho ao lidar com esse tipo de coisa, a gente vai ver como lidar de uma maneira mais fácil e objetiva.

[13:21] Mas é importante, também, a gente saber essa parte do Kotlin, que é uma parte bem bacana, que evita com que o programador tome Null Pointer Exception de graça, sem saber o que tá acontecendo no código dele. Era isso que eu queria passar para vocês e até mais!