05

Inicialização de property por delegação

Transcrição

[00:00] Uma outra abordagem que o Kotlin nos oferece, diferentemente do "lateinit" é por meio de um recurso chamado de properties delegadas ou, tecnicamente, conhecido como delegated properties. O que seria essa properties delegadas, como que ela funciona para a gente?

[00:14] Basicamente, as properties delegadas, é uma técnica, na qual, o Kotlin permite com que a gente delegue a responsabilidade de inicialização, para alguma estratégia, comum no nosso dia-a-dia. Essa estratégias comum no nosso dia a dia, o Kotlin já fez essa implementação para a gente. No caso, são algumas estratégias, não é apenas uma, a gente vai utilizar uma delas. Qual a estratégia que a gente vai utilizar, para atender essa nossa necessidade, qual a estratégia que a gente pode utilizar para isso?

[00:41] O Kotlin nos oferece uma estratégia chamada de inicialização preguiçosa, ou seja, quando a gente utiliza essa delegação de inicialização preguiçosa, a gente vai ter a capacidade de inicializar a nossa property, apenas no momento que a gente precisar dela. É por meio dessa estratégia que a gente vai conseguir ter um comportamento similar, com o que a gente viu aqui no "lateinit", que é uma inicialização atrasada, que vai acontecer só depois, só que, agora, apenas quando a gente for utilizar a variável, que ela vai ser inicializada.

[01:07] A gente vai indicar qual é o valor que a gente espera dessa inicialização. Então ela não vai ocorrer, justamente, no momento que a nossa classe é construída. Agora que eu falei para vocês sobre essa estratégia que a gente vai utilizar, como que a gente pode implementar ela?

[01:21] Basicamente, quando a gente está utilizando uma delegação, no caso, uma property delegada, a gente precisa indicar que a reinicialização dela vai ser feita por alguém, como que a gente indica isso aqui no Kotlin? A gente utiliza a palavra-chave "by", partir desse "by", a gente está indicando o seguinte, a sua inicialização está sendo delegada para alguém, e agora a gente tem que definir qual é a estratégia que a gente vai utilizar. Como eu falei, a gente já tem implementações prontas para isso.

[01:45] A gente vai utilizar a implementação da delegação por preguiça, ou preguiçosa, como seria isso então? "lazy", quando a gente utiliza esse "lazy", a gente está indicando que, a inicialização está sendo feita por uma estratégia que vai ser preguiçosa. Aqui ele até mostra as implementações que tem no "lazy", a princípio, ele mostra a expressão lambda, que é justamente a que a gente vai utilizar, porque a gente só vai fazer uma inicialização bem direta, bem objetiva, a gente não vai precisar passar outros parâmetros nesse nosso contexto.

[02:11] Vamos usar, primeiro, essa aqui da expressão lambda, só que reparem, que no momento que a gente fez essa parte da inicialização pelo "lazy", a gente teve algumas peculiaridades, que a gente tem que lidar, por questões de problema de compilação. O primeiro deles, que a gente vai resolver é, justamente, essa parte entre o "lateinit" e a delegação "by lazy", quando a gente está usando a delegação "by lazy" ou "lateinit", eles não são compatíveis, portanto, a gente tem que utilizar ou um, ou outro.

[02:36] Nesse caso a gente está utilizando a delegação, portanto, o "lateinit" vai embora. Essa é a primeira abordagem que a gente precisa resolver. Outro caso, é o seguinte, quando a gente não coloca nada aqui no nosso "lazy", por padrão, o que ele vai devolver, ele vai devolver justamente o "unit". Se a gente colocar, por exemplo, a "unit", ele vai lá e já devolve para a gente o valor, só que agora tem outro detalhe, o outro detalhe é o seguinte, quando a gente está utilizando o "lazy", a gente não pode utilizar properties, no caso, que são "var.

[03:03] Ele, no caso, ele obriga que a gente utilize "val", ou seja, esse valor nunca vai ser mudado, por que isso acontece? Porque, ele executa, diversas coisas aqui dentro do escopo dele e ele lembra a última execução dele, ele lembra apenas aquilo que ele devolveu. Justamente por essa questão de lembrar apenas o que ele devolveu, já que ele vai executar apenas uma única vez, ele faz com que a gente utilize o "val", porque não vai mudar o valor, não vai executar novamente, aqui a gente tem que utilizar o "val", justamente por isso.

[03:30] Agora, reparem que quando a gente não coloca nada e a gente coloca, aqui, o retorno como "unit", ele vai lá e compila, porque, realmente, aqui a gente não tá devolvendo nada. Então, por padrão, ele vai devolver o "unit" para a gente. O que a gente pode fazer, agora, para poder devolver, aqui, o nosso valor, que a gente espera? Basicamente, a gente pode vir aqui e colar aqui, o valor que a gente espera, que é justamente uma "view", ele até está reclamando, que agora a gente está devolvendo uma "view".

[03:53] Portanto, aqui, o que a gente espera, é uma "view". Inclusive, quando a gente tem essa abordagem, a gente, necessariamente, nem precisa indicar que isso daqui é uma "view", a gente pode simplesmente omitir esse tipo, porque ele já sabe que a gente está lidando com uma "view" aqui. A gente pode ver o parâmetro, apertar, por exemplo, o Ctrl+Q, vamos ver, a gente tem uma view aqui, a "viewDaActivity", porque aqui no final, ele já sabe que é uma "view", ele já vai lá e infere para a gente.

[04:17] Esse que a delegação por preguiça, o "by lazy". O que tá acontecendo aqui, só para poder recapitular, quando a gente está utilizando esse tipo de inicialização, a gente vai fazer uma inicialização que vai ser feita, delegada por alguém, por uma estratégia, que é a estratégia preguiçosa e ela, só vai ser executada, no momento em que a gente, realmente, utilizar essa property. Essa inicialização, essa informação que a gente está colocando na property "viewDaActivity", ela só vai ser inserida no momento que a gente utilizar essa "view" aqui.

[04:46] Vamos lá, vamos testar, para ver se realmente está funcionando, eu não estou inicializando em outro lugar, eu vou até apagar essa parte aqui, vou até apagar agora aqui, porque a gente está inicializando nesse momento aqui do "lazy". Vamos ver o que acontece, Alt+Shift+F10, vamos executar, veja que o Android Studio conseguiu executar para gente e, olha o que acontece, a nossa aplicação está funcionando.

[06:06] A gente consegue inserir uma transação aqui, a gente consegue alterar aqui também, ela tá funcionando como a gente já viu anteriormente, eu vou colocar uma despesa, para gente testar uma coisa diferente, para não ficar só na receita. A gente consegue fazer exatamente o mesmo comportamento que a gente tinha antes, a diferença é que, agora, a gente sabe com o que a gente está inicializando, a gente está garantindo essa inicialização e ela só está sendo feita, no momento que essa variável é utilizada.

[05:30] É só nesse momento que essa inicialização está sendo feita, mas agora abrem diversas dúvidas, para ver se realmente, tudo que eu falei para vocês, faz sentido, que aqui só é executado uma vez, que ele lembra desse valor. Para isso a gente pode até fazer um teste, a gente pode vir no nosso "onCreate", por exemplo, e fazer o seguinte, a gente pode fazer uns logs aqui, vou fazer uma cópia de três logs, que vai testar o nosso "lazy", "teste lazy 1", aqui o que eu vou fazer?

[05:55] Eu vou pegar a nossa property, que é a "viewDaActivity", e vou imprimir o "toSpring" dela, para a gente ver qual é o objeto que está sendo referenciado. Aqui vou fazer uma cópia desse log, três vezes, para ver se realmente o que eu falei para vocês, faz sentido. Vamos lá, o "lazy 1" aqui, agora o "lazy 3" e aqui dentro, desse escopo do "lazy", eu vou colocar um outro log, para gente ver ele sendo executado, para ver quando é chamado o "lazy", entre outras coisas que acabei não comentando com vocês.

[06:22] "teste lazy", esse vai ser o teste "lazy" 0, já que vai ser o Inicial, é assim que a gente espera, e aqui eu vou falar o que é a "inicialização do lazy". Agora só para gente ver se realmente, tudo que eu acabei mostrando para vocês faz sentido, a gente fazer esse teste, vamos lá, Alt+Shift+F10, veja que o Android Studio conseguiu executar para a gente, vamos ver aqui no nosso "Logcat" e aqui eu vou filtrar por "teste", acho que só o teste ele já vai, o espaço ele não conseguiu pegar.

[06:56] Mas olha o que aconteceu, vamos lá, vamos tentar captar o que aconteceu aqui de acordo com o nosso teste. Reparem que a princípio, quando a gente fez a primeira utilização aqui da nossa "viewDaActivity", a inicialização do "lazy" aconteceu, como eu havia comentado com vocês. Em seguida, ele foi lá e imprimiu o "toString", depois que ele utilizou a "viewDaActivity", novamente, a inicialização não foi feita de novo, ele lembrou do valor, ele lembrou desse valor "window.decorView".

[07:25] Ele está utilizando esse valor agora, para a view da property, porque ela já foi inicializada e ela não vai modificar mais o valor e, novamente, ele veio com o "toString", a gente pode ver, também, que no terceiro teste, também isso ocorreu. A inicialização pela estratégia "lazy", uma estratégia preguiçosa, nos permite com que a gente tenha o mesmo comportamento do "lateinit", que permite que a gente inicialize a nossa variável em algum outro momento.

[07:47] Só que esse momento é, justamente, quando a gente vai utilizar a nossa variável, a nossa property, nesse caso. E a inicialização já está bem clara, qual é o valor dela aqui, a gente já sabe, a gente está obrigando essa inicialização, e ela só está ocorrendo, quando a gente precisa da nossa variável, que é um caso que a gente precisa nesse momento, que ela não poderia ser inicializada, aqui, no momento que a gente construiu a classe, e ela só poderia ser inicializada dentro dos membros.

[08:08] Que já passaram no "onCreate" e já passava no ciclo de vida que a gente já tinha a "view", a gente conseguia fazer isso. Essa que é a parte da delegação de properties e é essa estratégia que a gente utilizou, que é a estratégia de delegação por preguiça, até mais!