03

Salvando objetos de forma estática

Transcrição

[00:00] Considerando a necessidade que a gente viu de manter essa nossa property "transacoes" de forma estática aqui na memória como que a gente pode estar fazendo isso aqui no Kotlin?

[00:08] Basicamente a gente pode estar pensando da seguinte maneira. Lá no Java a gente utilizando o Word static, então basicamente a gente pode vir aqui na frente e colocar um static. A gente pode pensar dessa maneira. Só que o que acontece? O Kotlin nem tem essa keyword chamada de static, ele nem sabe o que é isso, ou seja, não temos static aqui no Kotlin.

[00:26] Então como que a gente pode estar fazendo pra gente ter exatamente o mesmo comportamento do satitc que a gente tinha lá no Java? O Kotlin, ele chama esse tipo de comportamento como objetos companheiros. De forma técnica, isso é conhecido como companion object. A gente precisa transformar as nossas transições, no caso a nossa property "transacoes" em um companion object.

[00:47] Então como a gente pode transformar um companion object dentro de uma classe aqui? Como que a gente pode estar fazendo isso? Pra isso, a gente vai escrever a seguinte keyword "companion", e olha só ele ate mostra pra gente, ele até dá um spoiler do que a gente espera, que é o "companion object".

[01:04] Dentro desse companion object a gente pode colocar tudo que a gente espera que fique de forma estática, o que a gente espera nesse caso? A gente espera que a nossa lista de transações aqui seja mantida de forma estática. Repara que quando a gente fez a modificação, não teve nenhuma alteração aqui na nossa "TransacaoDAO", ele ia estar compilando da mesma maneira.

[01:21] Só que se a gente entrar aqui na Activity, vamos ver como que ficou? Olha só, aqui a gente tem o ponto de reclamação, um ponto que não está compilando, vamos subir lá? Subindo aqui, ele não tá compilando agora nesse momento que a gente chama as transações baseando-se no objetos das transações DAO, no nosso objeto DAO, o nosso DAO não está conseguindo pegar aqui as transações.

[01:44] Porque que não tá conseguindo? Porque o companion object, ele tem o mesmo aspecto que a gente tem no static, e no caso ele não fica atrelado ao nosso objeto. Portanto, isso daqui não pode ser chamado por meio de um objeto, ele tem que ser chamado por meio da classe que tá invocando ele. Em outras palavras, pra gente conseguir fazer uma chamada de uma property ou de uma função que seja como companion object, a gente vai ter que chamar a partir de sua classe.

[02:07] Então a gente vai ter que chegar aqui e usar a referência da classe para poder utilizar as nossas transações agora, porque agora elas não estão mantidas dentro da memória de forma estática. Vamos testar agora, pra ver se agora essas transações são mantidas? Vamos lá, Alt+Shift+F10, e veja que o Android Studio conseguiu executar. Vamos agora testar aqui, eu até deixei em modo portait, porque é legal da gente testar também.

[02:29] Então portait aqui, estou adicionando uma transação. E olha só, eu adicionei uma transação, agora eu vou virar de novo aqui para nossa posição normal e olha só, a nossa transação é mantida. Eu vou adicionar mais um aqui só para a gente ver se realmente tá funcionando? Agora uma despesa, vamos lá, agora em modo portrait, agora vou pro landscape, então voltando aqui. Olha só, ainda as transações são mantidas.

[02:49] Então de fato agora as nossas transações, elas estão dentro da memória, elas não estão atreladas ao objeto. Portanto por mais que a gente perca o objeto no momento em que a Activity é destruída as transações são mantidas na

memória, elas estão reservadas lá pra gente. Só que por mais que a gente tenha feito essa solução aqui, ainda tem uns detalhes que a gente precisa tomar cuidado quando a gente tá lidando com esse tipo de solução, ou seja, disponibilizando uma lista estática para todo mundo, de maneira global.

[03:17] E qual esse cuidado que a gente tem que tomar? Perceba que aqui na nossa implementação, vou dar um Ctrl+B aqui nas transações a gente está devolvendo uma MutabelList, qual o risco a gente tem quando tem uma MutableList? Se a gente vir aqui na Activity, vamos tentar fazer o seguinte. Vamos chegar aqui, vamos criar uma transação qualquer, "Transacao", eu vou colocar uma transação de teste mesmo.

[03:39] Então um BigDecimal de 100, um "Tipo.RECEITA" por exemplo. É uma transação de teste, então é claro, deixa eu só colocar aqui o limite parameter pra entender que isso daqui refere-se ao tipo, porque não é esse parâmetro que ele espera nesse momento. Agora eu vou colocar aqui como um objeto qualquer, "transacaoDeTest" só para gente testar mesmo, e para vocês entenderem, por exemplo, quais são os riscos que tem ao disponibilizar esse tipo de lista para qualquer tipo de membro dentro da nossa aplicação.

[04:09] Agora, vamos chegar aqui nas transações e olha só o que a gente tem capacidade. A gente tem capacidade, por exemplo, de adicionar elementos. Então, a gente pode adicionar ser essa transação de teste. E olha o quão bizarro fica a nossa aplicação quando a gente tem uma lista estática, e a gente consegue modificar fora do nosso controle, ou seja, qualquer um pode chegar e modificar.

[04:31] Olha só o que acontece, Alt+Shift+F10, veja só, o Android Studio conseguiu executar. Olha só o que acontece, vamos lá, ele já criou uma transação aqui pra gente, ele já conseguiu mexer ali na nossa estrutura de dados que a gente está tendo ali por debaixo, que tá mantendo as nossas transações.

[04:48] Inclusive vamos rotacionar pra ver o que acontece, rotacionando ele adicionou outra. Quando a gente rotaciona de novo, ele adiciona outra, ou seja, ele está manipulando a nossa lista. Ele está fazendo o que ele quiser com a nossa lista e não é o comportamento esperado. A gente está criando de fato um bug aqui. Porque qualquer um pode fazer isso, qualquer um pode modificar, pode manipular da maneira que quiser.

[05:08] Portanto, quando a gente tá lidando com esse tipo de objeto estático a gente tem que tomar bastante cuidado com quem a gente está disponibilizando, porque qualquer um vai ter acesso, qualquer um vai modificar na hora que quiser, e a gente não tem controle muito disso.

[05:19] Então considerando esse aspecto que a gente agora precisa lidar com objeto estático, mas também não pode permitir que alguém ou qualquer um modifique a nossa estrutura, como que a gente pode estar fazendo para evitar esse tipo de situação? A princípio a gente pode fazer o seguinte. O primeiro passo de todos é justamente definir que esse cara aqui, ele não vai ser acessível por ninguém a não ser ele mesmo, ou seja, deixar ele privado.

[05:42] A gente tem a capacidade de deixar o objeto companheiro ou o companion object privado também. Só que a gente entra naquele caso que se a gente deixa ele privado, olha só o que acontece, não vai ter acesso. Ninguém via ter acesso a ele. O que a gente precisa fazer? Agora que a gente deixou privado, a gente precisa disponibilizar um objeto para alguém conseguir acessar, e a princípio a gente fez da seguinte maneira.

[06:01] A gente pensa que a gente pode tá criando agora uma "transacoes" aqui baseando-se no nosso companion object. Que a princípio a gente vai imaginar que agora vai ser uma cópia que vai dar para fora. Só que agora entra alguns detalhes, pra gente conseguir mandar esse carinha aqui, repare que ele já sabe que esse "transacoes" se refere ao companion object, ele não é uma nova property.

[06:22] Então qual que é o caso que a gente pode fazer aqui? A gente pode criar, por exemplo, uma property com nome diferente, "transacoesPublicas", a gente pode criar a "transacoesPublicas" que vai ser a property pública pra todo

mundo. Todo mundo vai ter acesso a ela. A gente pode estar fazendo. Só que repara que "transacoesPublicas" não é um nome assim que a gente espera pra poder dar todas as transações, é comum a gente liberar essas transações.

[06:42] Então com que a gente pode fazer para distinguir essa property que a gente tem que é uma property normal, uma property da instância da "TransacaoDAO" com o nosso companion object que também tem o mesmo nome que é "transacoes". Basicamente o companion object, o nome dele de modo geral, o nome composto dele, ele vem com esse prefixo chamado de "Companion".

[07:06] Então esse "Companion" é justamente a forma como a gente pode acessar de modo composto todos os nossos companion objects, todo mundo que tiver dentro desse "companion object", ele vai ter esse prefixo companion. É assim que a gente consegue distinguir o nosso companion object "transacoes" dessa nossa property "transacoes", que ela refere-se justamente ao objeto.

[07:24] Mas é claro, quando a gente tem esse comportamento a gente toma muito cuidado. Porque antes, a gente utilizava o nosso companion object como se fosse uma property aqui do nosso "transacaoDAO" e agora a gente tem a nossa property, e temos o companion object aqui. E olha só quem a gente tá acessando aqui, dentro do adiciona, altera e remove. A gente tá usando aqui a nossa property que a gente criou.

[07:44] Portanto, a gente não tá inserindo aqui dentro o nosso companion object, que é justamente aquela nossa variável que é estática, que vai manter as nossas informações. Portanto, agora que a gente tá usando esse tipo de recurso pra poder permitir o acesso à property pública para todos os membros de fora, mas internamente a gente quer que usa o companion object, a gente vai ter que chegar aqui e colocar "Companion" para todos.

[08:04] Então a gente vai lá, "Companion" aqui, "Companion" aqui, e "Companion" aqui. Agora, internamente a gente quer que ele acesse o companion object, só que quem via acessar de fora essa lista de transações, aí sim ele vai pegar ela apenas e não vai nem saber que existe aqui dentro. É isso o comportamento que a gente espera de fato.

[08:22] Então agora a gente está fazendo essa transferência do companion object pra uma property pra ser acessada apenas pelo objeto, e vamos ver o que acontece. Agora veja que parou de compilar novamente. Porque pra acessar essas transações, essa property, a gente precisa de uma instância. Ela não tá sendo estática, ela não está na memória, portanto, novamente o nosso "dao".

[08:41] Então o "dao" novamente, ele vai lá e permite acessar. Agora tem um detalhe, claro vamos tentar ver o que acontece primeiro, vamos ver se tudo funciona. Vamos ver se esse comportamento aqui ainda mantém. Vamos ver se isso ainda funciona, porque ainda a gente tá conseguindo chamar, mas vamos ver se ele consegue afetar internamente o nosso companion object.

[08:59] Essa que é a ideia desse teste, vamos lá, Alt+Shift+F10, veja que ele conseguiu executar, vamos testar aqui. Olha só, ele criou aqui novamente, porque de fato ele tá mexendo ali na estrutura dele. Agora vamos rotacionar, olha só ele ainda tá mantendo aquele mesmo comportamento. Ele ainda tá adicionando e afetando a nossa lista internamente, ele está afetando o nosso companion object, ele tá persistindo ainda essas transações. Por que isso acontece?

[09:27] É o seguinte, esse companion object, quando a gente manda pra cá, quando ele atribui pra essa property, ele tá mandando e fato a referência desse companion object. Então por mais que ele esteja privado, por mais que a gente privou ele aqui, a gente tá mandando diretamente a referência desse cara, não é uma cópia.

[09:43] Considerado justamente esse aspecto, a gente precisa de um mecanismo diferente para gente conseguir proteger aqui esse nosso carinha, essa nossa MutabelList. Então agora que eu até dei destaque para isso, o que que a gente pode fazer é o seguinte. Já que a gente precisa mandar esse companion object, a gente precisa mandar os dados dele, a gente pode mandar através da interface MutableList, a gente pode mandar a interface "List".

[10:08] Porque lembra que essa interface, ela é imutável, portanto se alguém pegar essa "transacoes" aqui ela não vai ter a capacidade de adicionar, remover ou alterar. Ela não vai ter a capacidade de mudar essa estrutura, por causa da interface "List". E a gente mantém exatamente o mesmo comportamento que a gente tava fazendo aqui internamente. A gente mantém exatamente a mesma coisa.

[10:27] Voltando aqui, ele para de compilar essa parte adicional, porque agora a gente não permite isso. Então ele não consegue fazer isso apenas recebendo o nosso carinha aqui, mas é claro, entra aquelas situações, ah mas ele pode tentar fazer uma conversão, certo? Realmente, ele pode tentar fazer. Vamos ver o que acontece quando ele tenta fazer isso.

[10:46] Então, por exemplo, ele pode chegar aqui e falar o seguinte, eu sei que você tá mandando uma list, mas eu espero uma MutableList. A princípio não vai compilar, por quê? Porque aqui a gente tem uma lista, a gente não tem uma MutableList. Só que ele pode chegar aqui e fazer uma função de conversão, que é a "toMutableList", ele pode fazer isso, o Kotlin permite isso. E o que acontece? A gente precisa testar pra ver.

[11:09] Vamos ver, Alt+Shift+F10, veja que o Android Studio conseguiu executar. E olha só, ele conseguiu colocar a transação dele. Agora vamos voltar aqui, ele conseguiu manter apenas a única transação que ele colocou aqui. Em outras palavras, o que tá acontecendo é o seguinte. No momento que ele fez o "toMutableList", ele fez uma cópia dessa referência. Então já não é mais essa referência que ele tem.

[11:33] Então mesmo que ele adicione uma transação, olha só o que vai acontecer. Ele vai adicionar uma transação aqui, olha só, não apareceu mais a transação. Por que não apareceu? Porque a transação é feita lá no nosso "dao", só que ele não tem mais acesso aquela referência daquela lista, ele não tem mais acesso a ela.

[11:49] Então, ele não consegue mais manipular da maneira que ele quer, ele até não consegue ter o mesmo comportamento que ele espera, que é adicionar, aparecer a transação, entra ouras coisas que a gente viu.

[11:57] Então dessa forma, que a gente protege os nosso companion objects. A gente consegue proteger eles fazendo com que a interface deles seja uma interface imutável. Aí a gente não precisa se preocupar, tô tendo que mandar meu companion object, e ele tá pegando a referência dele, ele tá alterando da maneira que ele quer, a gente não precisa mais se preocupar com isso.

[12:15] Inclusive, agora vamos voltar tudo como tava antes, pra poder testar e ver se tudo funciona. Vamos tirar aqui esse MutableList, ou se quiser deixar List pra ver qual que é a interface também, pode deixar, não tem nenhum problema. Mas vamos deixar sem porque a gente já sabe que é uma lista de transações, vamos apagar esse carinha aqui. E vamos se se tudo que a gente colocou na nossa app tá funcionando.

[12:34] Alt+Shift+F10, veja que ele conseguiu executar, vamos testar aqui. Então a princípio a gente não tem nenhuma transação, vamos lá, adicionamos aqui, adicionei. Beleza, tô adicionando, eu vou alterar, vou colocar aqui R$ 1100, vou colocar uma data diferente, 16, eu vou colocar aqui uma categoria diferente, pagamento. [12:57] E olha só ele tá funcionando a alteração, vamos tentar remover, removeu, tudo funcionando. Agora aquele comportamento que a gente viu que tava sendo problemático. Vou adicionar aqui uma transação qualquer, vou adicionar outra transação aqui no valor de R$ 600, tem uma receita e uma despesa.

[13:12] Vamos rotacionar, olha só, ele ainda mantém. Se a gente adicionar nesse modo rotacionado, que é o modo landscape, que é o modo paisagem, vamos ver, ele tá adicionando também. Se a gente voltar agora aqui, olha só, ele ainda mantém.

[13:24] Então, a gente viu essa técnica de poder manter essas transações, e também a gente aprendeu como a gente pode deixar isso em memória de forma estática aqui no Kotlin. Só um último feedback dessa parte da implementação que a gente fez. Essa implementação aqui, ela não é ideal no dia a dia de Android.

[13:40] A implementação ideal é utilizando aqueles recursos de storage do Android, que é justamente um SQLite, entre outros que existem por aí. Mas só pra gente entender que no Kotlin tem essa possibilidade de forma estática, e ver como que a gente pode estar utilizando aqui, e os cuidados que a gente tem que tomar. Principalmente quando a gente utiliza essas referências como é o caso de lista, dentre outras coisas que a gente tem aqui no nosso dia a dia.

[14:00] Então era esse feedback que eu queria passar pra vocês. Eu espero que vocês tenham gostado, e até mais.