01

Realizando herança

Transcrição

[00:00] Agora que você finalizou a refaturação da nossa Activity, vamos dar uma olhada como é que ficou os códigos dos dialog quem a gente criou, ou seja, do "AdicionaTransacaoDialog" e do "AlteraTransacaoDialog". Vamos dar uma olhada aqui, Ctrl+N, "AdicionaTransacaoDialog", Ctrl+N, "AlteraTransacaoDialog", deixa eu aumentar o código para vocês, Ctrl+Shift+F12. Se a gente reparar aqui no código deles, a gente percebe que, além de ele estar um código grande.

[00:26] Todo esse código que a gente implementou, eles podem ser reutilizados. Como a gente vê, as properties são basicamente as mesmas, essa função chama, também, é muito similar, como as outras funções que a gente está vendo, que é a configura formulário, a título por, converte campo valor, elas são bem similares, ou seja, a gente tem capacidade de reutiliza-las.

[00:46] Considerando essa questão, de que a gente pode reutilizar esses mesmos comportamentos de membros que a gente tem dentro das classes, o que a gente pode fazer, considerando o mundo de OB, ou seja, orientação objetos, que permite a reutilização desses mesmos membros e fazer com que eles sejam extensíveis? Que tanto a "AdicionaTransacaoDialog" consiga usar, como também, a "AlteraTransacaoDialog" consiga usar, mas que eles consigam fazer a implementação deles.

[01:12] Basicamente, o que a gente pode estar utilizando aqui é, justamente aquele recurso de herança, que é por meio de classes mães, classes pais, que fazem com que os seus filhos herdem delas, ou seja, a gente pode fazer o seguinte, a gente pode criar uma classe pai, uma classe mãe, que também é considerada como SuperClass, que vai conter todas essas funções que podem ser reutilizadas, tanto no "AdicionaTransacaoDialog", como, também, no "AlteraTransacaoDialog", a gente pode fazer isso.

[01:40] Como que a gente pode estar, por exemplo, criando uma classe que vai ser a classe mãe, a partir de todos os membros esses membros que a gente está criando aqui. A princípio, a gente pensa, é só a gente criar uma classe, e começar a copiar e colar todos esses membros aqui, que podem ser reutilizados, a princípio é uma abordagem bem válida fazer isso.

[01:58] Só que, já que a gente está utilizando uma IDE, no caso do Android Studio, a gente tem uma capacidade de criar SuperClass, a partir de classes que a gente tem aqui, de uma maneira muito mais objetiva, e é justamente isso que a gente vai ver agora e, também, a gente vai entender como é que a peculiaridade de SuperClass dentro do Kotlin.

[02:15] Vamos lá, o primeiro passo que a gente pode fazer é, utilizando um recurso do Android Studio que é o Refactor This, como que a gente o utiliza? A gente pode utilizar por meio do atalho Alt+Shift+Ctrl ou Ctrl+Shift+Alt e a gente vai lá e coloca o "T", então Ctrl+Shift+Alt+T, a gente abre essa possibilidade aqui, que é o Refactor This. Desse Refactor This a gente até consegue filtrar as opções, e se a gente escrever, por exemplo, "su", nem precisa escrever o restante, a gente percebe que tem essa opção de SuperClass.

[02:48] Que é justamente para criar uma classe mãe, uma classe pai, baseando-se em uma classe que a gente está utilizando essa opção, que no caso é a "AdicionaTransacaoDialog". A gente vai dar um "Enter" aqui, reparem que, nesse momento, o que acontece, quando a gente pega nessa opção de SuperClass, ele nos dá duas opções de arquivos, que são alvos de arquivos que a gente quer colocar a SuperClass. A princípio, nos indica em colocar essa SuperClass, dentro do próprio arquivo do "AdicionaTransacaoDialog.kt", que é de Kotlin.

[03:16] Só que, essa classe, a gente quer deixar em um arquivo separado, um arquivo exclusivo para isso, portanto, a gente vai nessa segunda opção que é, o "Extract to separate file", a partir dessa opção, a gente vai tocar nela aqui com o

"Enter", a gente vai ter capacidade, agora, de criar uma nova classe, que vai ser a SuperClass, dessa "AdicionaTransaçãoDialog", só que ela vai ser um arquivo separado, mesmo no caso que a gente vai colocar o nome dela, ele vai criar um arquivo para gente, essa que é a opção para extrair para um outro arquivo.

[03:44] Reparem que, depois que a gente colocou essa opção, ele abriu essa janelinha aqui para gente, que é, justamente, para começar o processo, para criar essa nova classe. Qual seria o nome de uma classe bem genérica, que vai representar uma classe mãe, uma classe pai, tanto da "Adiciona", como da "AlteraTransaçãoDialog" e que fica bem claro, que é uma classe genérica que pode ser usada pelas duas?

[04:04] Basicamente, o nosso dialog das transações cria um formulário, portanto, de modo genérico, ele trata-se de um formulário de transação Dialog, ele é um dialog para mostrar esse formulário das transações. Logo, a gente pode escrever o nome dessa classe como "FormularioTransacaoDialog", essa vai ser a representação genérica, onde vai conter todos os membros que podem ser reutilizados pelos nossos Dialogs que a gente implementou até o momento.

[04:34] Reparem, que a gente tem o nome dele, a gente tem aqui, também, o pacote onde ele vai ser armazenado, a gente vai deixar exatamente o dialog, porque ele se trata de um dialog também e ele está mostrando, também, o arquivo. Por mais que aqui esteja a classe, já marcou o arquivo também, por padrão, se caso quiser mudar para um outro nome, fique à vontade. É claro, faz muito sentido a gente deixar o mesmo nome representando a classe, considerando o que a gente utilizava em Java, para poder manter esse mesmo padrão.

[04:58] Agora que a gente definiu essas primeiras informações, reparem que aqui embaixo, ele tem essa outra opção aqui, que é mais bacana, no caso, que é justamente enviar os membros que a gente quer enviar para essa SuperClass, o que a gente vai fazer agora? Agora a gente vai fazer uma análise, na qual, a gente vai verificar, quais membros daqui de dentro, faz todo sentido ir para a SuperClass, ou seja, que pode ser reutilizado, tanto na "Altera", como na "AdicionaTransacaoDialog".

[05:25] Como a gente viu, a princípio, a gente pode mandar, por exemplo, todos esses carinhas aqui, todas essas properties que a gente viu, porque essas properties são iguais, tanto no "Adiciona", como também no "AlteraTransacaoDialog", então a gente pode mandar elas. Agora, a gente tem aqui também, essa função chama.

[04:42] Reparem que essa função chama também pode ser reutilizada aqui, porque, tanto no "Altera", quanto no "Adiciona", a gente está chamado ela, o "configuraFormulario" também, o "tituloPor" também o "converteCampoValor" também, esse daqui "configuraCampoCategoria" também, esse "configuraCampoData" também e o "criaLayout" também, por mais que a gente já ia marcar todos, a gente percebe que em nenhum deles, tem uma certa restrição.

[06:08] É claro, por exemplo, o "tituloPor", vai precisar ser uma implementação pouquinho diferente e a gente vai ver como a gente pode fazer isso aos poucos. A gente já poderia estar marcando essa opção agora, mas vamos fazer isso aos poucos para a gente entender as peculiaridades que a gente tem no meio do caminho, em relação a essa parte da herança.

[06:24] Por padrão, eu vou marcar todos, vamos mandar todos e vamos ver qual é comportamento, o que acontece, ao fazer isso. Não precisa se preocupar nesse momento. A gente está enviando todos os membros "AdicionaTransacaoDialog", e vamos ver o que acontece, agora a gente clica em "Refactor", no momento que a gente fez esse Refactor, olha o que aconteceu, olha o tamanho que ficou a nossa classe "AdicionaTransacaoDialog", porque ela ficou desse tamanho?

[06:46] Porque agora, ela não tem nenhum membro, todos os membros que ela utiliza, estão vindo aqui do "FormularioTransacaoDialog". A gente está reutilizando dessa classe padrão, dessa classe base, que a gente tem aqui, que é o "FormularioTransacaoDialog", a gente tá pegando todas as informações de lá, a gente está reutilizando elas, o que isso significa? Significa que agora a gente nem precisa de corpo, dessa classe.

[07:09] A gente não precisa mais o corpo dela, porque a gente já tá fazendo com que essa "FormularioTransacaoDialog ", consiga, no caso, manter todos os membros que a gente precisa utilizar do "AdicionaTransacaoDialog". Reparem que tem até os importes aqui, a gente pode se desfazer deles com o Ctrl+Alt+O. Agora veja como ficou bem mais simples, aqui, o nosso "AdicionaTransaçãoDialog ", e agora que a gente conseguiu fazer essa refaturação, agora que a gente adicionou essa superclasse, vamos ver como ficou a implementação dela.

[07:36] Vamos lá, aqui nessa "FormularioTransacaoDialog ",eu vou dar um Ctrl+B. Veja só, nesse momento que fez esse processo de refaturação, ele ainda falhou nesse ponto de fazer os importes dos membros que a gente mandou. A princípio, a gente vai ter que fazer esses importes aqui, vamos começar fazendo eles, vou dar um Alt+Enter aqui, nessa parte, ele importar o Synthetic, ele importou normalmente, eu já vou fazer o seguinte, eu vou pulando as linhas para ficar fácil de compreender.

[07:59] São todas as properties, que a gente está utilizando no nosso "FormularioTransacaoDialog ", aqui tá tudo certo. Reparem que o "AlertDialog" não importou, vamos importar aqui, importei, aqui, o "context", reparem que ele não conseguiu pegar a referência do contexto. Por mais que a gente tenha mandado, ele não conseguiu pegar essa referência, portanto, a gente precisa pedir ele, como que a gente pedia ele?

[08:20] Veja que, no "AdicionaTransacaoDialog", a gente estava recebendo via parâmetro, da mesma maneira a gente vai fazer aqui, também. Temos até um atalho para isso, Alt+Enter, aqui embaixo, quando a gente pede para criar uma variável local, ou criar um objeto, entre outras coisas, ou criar até uma property, termos o criar uma property "context", como construtor, ou como parâmetro do construtor, é esse "as Constructor parameter".

[08:41] A gente pode utilizar essa opção, para ele vir aqui em cima e falar, eu tô criando aqui para você, e olha onde vou deixar, aqui no construtor primário. A gente tem essa capacidade, a gente está mandando agora o context para cá, a gente resolveu essa referência, vamos voltar aqui embaixo. Reparem que aqui a gente tem que importar essa Extension Function, importar a transação também, agora a gente tem a classe "R", também, para importar, e bastante coisa, esse Toast, aqui, também, vamos importa-lo.

[09:09] Temos o Array Adapter, agora aqui embaixo a gente tem o Calendar, importamos aqui, o FormataParaBrasileiro", que é uma outra extensão, o "DatePickerDialog", o "LayoutInflater" e, aqui, a gente tem esse "ViewGroup", que ele representa, justamente, a "view" que tá sendo recebida via Construtor também, da mesma maneira que "context", Alt+Enter, "Create property ViewGroup as constructor parameter", para a gente ter esse parâmetro, para ter a nossa property, via construtor primário, vamos ver como que ficou.

[09:40] A gente tá mandando, só para poder colocar aquela boa prática de pular uma linha aqui, para cada uma das properties, e agora ficou bem mais bonito. Veja que a gente conseguiu fazer com que o nosso código, pelo menos aqui da nossa classe base, classe mãe, classe Pai ou SuperClass, no caso, do jeito que você preferir chamar, ele já foi resolvido, agora ele está compilando, vamos voltar aqui "AdicionaTransacaoDialog ".

[10:01] Reparem quem no momento que a gente fez esses ajustes, a gente teve um probleminha aqui, porque isso aconteceu? Já que agora a gente está utilizando esse conceito de classe Mãe, classe Pai, SuperClass, a gente precisa tomar alguns cuidados, nessa questão de acesso das nossas variáveis, das nossas funções. Porque o que acontece, o acesso que tá tendo para essas properties, que é a "ViewGroup" e que é a "context", ele tá sendo um acesso público.

[10:27] Quando a gente não coloca nenhum valor aqui em cima, o acesso, por padrão, é o "public". Então, "public" e colocar nada, dá na mesma, tá vendo, ele fala que é até redundante, quando a gente coloca "public" aqui no Kotlin. Então tudo que a gente coloca no Kotlin, por padrão, é "public". Essa é uma das abordagens que estão acontecendo aqui, a gente precisa fazer com que, esses acessos sejam compatíveis, e como a gente viu, essas properties não precisam ser acessadas por outras pessoas.

[10:55] Essas properties precisam ser acessadas apenas internamente aqui, pelo "FormulárioTransacaoDialog", portanto, faz todo sentido a gente diminuir o nível de acesso delas, e deixar compatível com essa parte do "private". Por mais que a gente utilizou o Android Studio para fazer isso para gente, ele deixou como público, por padrão, mas para a gente não faz sentido deixar padrão público, faz sentido a gente proteger e deixar privado, porque a gente não quer que alguém de fora tenha acesso a isso, ou saiba que a gente tenha acesso a isso.

[11:19] Novamente aqui, volta a compilar, agora, tem outro detalhe, reparem que aqui, deixa eu até pular uma linha aqui, também, para ficar mais visível, considerando aquela boa prática. Reparem que aqui também, quando a gente chama aqui essa herança, quando a gente está herdando de "FormularioTransacao", a gente precisa agora mandar os parâmetros, a gente precisa mandar, tanto o "context", como a "ViewGroup", a gente chega aqui e manda o "context", tem até essas duas opções.

[11:40] Usando o Named parameter, como a gente viu anteriormente, ou mandando diretamente os dois, vamos mandar diretamente os dois, são só dois parâmetros, eu não vejo tanta necessidade de colocar o Named parameter, reparem que volta a compilar. Se a gente voltar aqui, tudo funcionando normalmente. Considerando essa abordagem que a gente tem agora uma SuperClass, a gente teve uma mudança aqui, que eu acabei não comentando com vocês, e é justamente isso que a gente vai entender agora.

[12:03] Reparem que aqui, no momento que a gente criou essa classe, essa SuperClass, olha o que apareceu aqui, apareceu esse tal de Open, porque apareceu esse Open. Basicamente, tudo aqui no Kotlin, são membros, são entidades e mutáveis, o que isso significa? As classes que a gente tem aqui são imutáveis, portanto, a gente não consegue fazer herança de qualquer classe, o que significa?

[12:26] Se a gente apagar esse open, por exemplo, olha o que acontece, quando a gente tenta fazer uma herança, ele tá falando que o tipo dessa classe é "final", ou seja, a gente não consegue fazer a herança dela. Novamente, o Kotlin tem uma abordagem, na qual, tudo que ele tem por patrão aqui, seja classe, seja função, são imutáveis. A gente não consegue sobrescreve-las, caso a gente não abra elas, por meio daquela Key Word "open".

[12:51] O open é justamente, para a gente permitir que elas sejas sobrescritas, sejam modificáveis. Por isso, que há a necessidade chamar essa KeyWord "open", por isso que, quando a gente fez o processo automático, de fazer uma SuperClass, o próprio Android Studio colocou isso gente. Vamos voltar lá e, novamente, a gente coloca o "open", essa é uma abordagem bem bacana, porque ela nos protege bastante, de alguém de fora querer modificar o nosso código, de alguém de fora querer manipular o nosso código.

[13:17] De tal maneira que a gente corra risco, que a gente nem sabe que a gente está correndo. Por padrão, o Kotlin nos protege disso, aquela questão de segurança do código. Agora que a gente fez essa refaturação, que a gente está fazendo com que todos esses membros estão sendo implementados, pela SuperClass, a gente pode agora até testar, inclusive, antes de fazer esse teste, reparem que que a gente nem precisa mais usar essas properties.

[13:38] Eu até modifiquei ,só para a gente ver esse caso, de que o nível de acesso precisa ser o mesmo, tem que ser compatível com quem a gente tem lá, agora a gente pode até testar nossa App, para ver o que tá acontecendo, para ver se essa mudança, ela impactou ou não no nosso aplicativo.

[13:53] Alt+Shift+F10, e vamos ver o que acontece, veja que o Android Studio conseguiu executar sem nenhum problema, vamos ver aqui. Vamos tentar adicionar alguma transação de receita, mantendo as informações, adicionar valor, vamos colocar uma data só para ver se nada impactou, uma outra categoria, "pagamento", por exemplo, tudo funcionando normalmente, vamos ver a despesa?

[14:16] Adicionar despesa, coloca o valor aqui, vamos ver aqui, agora uma outra categoria "lazer", por exemplo, ele está adicionando normalmente, essa refaturação que a gente fez, mandando todos os membros para SuperClass, funcionou sem nenhum problema. Só que agora entra alguns detalhes, porque a gente fez isso, essa herança, para

"AdicionaTransacaoDialog", só que, se a gente vem aqui em "AlteraTransacaoDialog", ainda está mantendo aquele código que a gente viu anteriormente.

[14:42] Ele está pegando toda a responsabilidade de manter esses membros, e a gente nem está utilizando esses carinhas aqui. Então, a nossa proposta logo a seguir é, justamente, fazer com que essa nossa "AlteraTransacaoDialog", também faça essa herança, até mais!