04

Utilizando o padrão Delegate

Transcrição

Estamos em uma situação, na qual temos um código que está dentro da classe `AdicionaTransacaoDialog`, que é uma classe distinta da *activity*, porém esse código precisa ser executado dentro da *activity*. Como vamos resolver esse problema?

Pensando nesse detalhe, veremos algumas alternativas que podemos fazer para realizar essa execução, e entenderemos quais são os impactos de se utilizar essas alternativas.

A princípio, podemos pensar da seguinte maneira: já que `atualizaTransacoes(transacaoCriada)` precisa ser executada pela *activity*, para isso, no momento em que o `configuraDialog()` for chamado, podemos mandar uma *activity*, pois é ela que tem a função que precisa ser executada. Da mesma forma, podemos mandar esse parâmetro para o `configuraFormulario()`:

```
class AdicionaTransacaoDialog(private val viewGroup: ViewGroup, private val context: Context) {

    private val viewCriada = criaLayout()

    private fun configuraDialog(activity: ListaTransacoesActivity) {
        configuraCampoData()
        configuraCampoCategoria()
        configuraFormulario(activity)
    }

    private fun configuraFormulario() {...}
}
```

Ao mandar a *activity* via parâmetro do `configuraFormulario()`, é necessário também adicionar o parâmetro automaticamente na própria função. Podemos utilizar o atalho "Alt + Enter" para nos ajudar.

```
    private fun configuraFormulario(activity: ListaTransacoesActivity) {...}
```

Agora que temos esse parâmetro da *activity*, podemos chamar o objeto e chamar a `atualizaTransacoes(transacaoCriada)`:

```
    activity.atualizaTransacoes(transacaoCriada)
```

Entretanto, repare que o método ficou sublinhado. Isso quer dizer que temos algumas complicações. A primeira delas envolve questões de segurança do nosso código, pois essa é uma **função privada** da *activity*. Será preciso executá-la em outro ponto do código, fazendo com o que ele fique inseguro. O outro caso é a chamada `lista_transacoes_adiciona_menu.close(true)`. Nessa chamada, estamos utilizando o *Synthetic*, e da maneira que está sendo chamado, é como se `AdicionaTransacaoDialog` fosse uma *activity*, e com isso, será necessário ter acesso ao objeto `view`.

Em outras palavras, por mais que a gente consiga fazer a primeira chamada do `atualizaTransacoes()` a partir do objeto *activity*, temos que lidar com esses problemas, algo que não é apropriado, como também, fazemos com que isso seja acoplado a lista de `ListaTransacoesActivity`. Se quisermos reutilizar o `Dialog` em uma outra entidade do Android, não teremos essa capacidade, pois vinculamos demais com a *activity*.

Visando esse detalhe de tirar o acoplamento e fazer com que não tenhamos que lidar com essas diversas peculiaridades, como faremos para indicar que essas duas linhas de código precisam ser executadas pela pessoa que está chamando o `AdicionaTransacaoDialog`? Uma das técnicas muito comuns no mundo Java, é enviar uma interface que será responsável por **delegar a responsabilidade** para quem chamou. Usaremos o padrão chamado ***Delegate***, um *design pattern* que visa delegar a responsabilidade para quem estiver executando o código e quiser pegar uma resposta dele.

Para implementar o *Delegate*, acessaremos o projeto e criaremos um novo pacote chamado de "delegate". Nesse pacote, criaremos a interface `TransacaoDelegate` para delegar responsabilidades para quem o chamou.

Vamos criar uma assinatura que fará com que a gente mande a `transacaoCriada` para quem utilizar essa interface.

```
interface TransacaoDelegate {
    fun delegate(transacao: Transacao)
}
```

Para conseguirmos utilizar esse `delegate`, vamos em `AdicionaTransacaoDialog` e, ao invés do `configuraDialog()` receber uma *activity*, ele receberá `transacaoDelegate`:

```
    private fun configuraDialog(transacaoDelegate: TransacaoDelegate) {
        configuraCampoData()
        configuraCampoCategoria()
        configuraFormulario(transacaoDelegate)
    }
```

Vamos modificar também o `configuraFormulario()`:

```
    private fun configuraFormulario(transacaoDelegate: TransacaoDelegate) {...}
```

Então, já estamos recebendo a interface que irá delegar a responsabilidade no momento em que criar a transação, para quem chamou o `configuraDialog()`.

Vamos comentar o código:

```
    //activity.atualizaTransacoes(transacaoCriada)
    //lista_transacoes_adiciona_menu.close(true)
```

Então, ao invés de chamar todo esse código, simplesmente chamaremos o `transacaoDelegate`, e pedir para ele **delegar** a `transacaoCriada`.

```
    transacaoDelegate.delegate(transacaoCriada)

    //activity.atualizaTransacoes(transacaoCriada)
    //lista_transacoes_adiciona_menu.close(true)
```

Agora que modificamos todo o código para a `AdicionaTransacaoDialog`, podemos chamar la na *activity*. Primeiro, é necessário fazer a instância do `AdicionaTransacaoDialog()`, e mandar os parâmetros:

```
override fun onCreate(savedInstanceState: Bundle?) {

    // chamadas de métodos

    lista_transacoes_adiciona_receita
        .setOnClickListener {
            AdicionaTransacaoDialog(window.decorView as ViewGroup, context: this)
        }
}
```

A responsabilidade de indicar que esse parâmetro passado é um **ViewGroup** é da própria *activity*, e não do `AdicionaTransacaoDialog`.

> Não vamos nos esquecer de retirar o modificador de acesso do `configuraDialog()` pois no momento ele está **privado**, e ao retirá-lo, ele se torna **público**.

Depois de ter setado os parâmetros, podemos chamar o `configuraDialog()` e mandar a interface utilizando **object expression** para implementar a `TransacaoDelegate`:

```
    lista_transacoes_adiciona_receita
        .setOnClickListener {
            AdicionaTransacaoDialog(window.decorView as ViewGroup, context: this)
                .configuraDialog(object : TransacaoDelegate {

                })
        }
```

No momento em que estamos fazendo a implementação do `object expression`, basta apenas realizar a implementação do `Delegate`. Com o cursor em cima de `object`, utilizamos o atalho "Alt + Enter" e escolhemos a opção "Implement members", e depois clicamos em "OK".

```
    lista_transacoes_adiciona_receita
        .setOnClickListener {
            AdicionaTransacaoDialog(window.decorView as ViewGroup, context: this)
                .configuraDialog(object : TransacaoDelegate {
                    override fun delegate(transacao: Transacao) {

                    }
                })
        }
```

Quando estamos implementando a função `delegate()`, estamos implementando justamente a transação que foi criada pelo usuário. Todo o código que podemos executar dentro do escopo do `delegate()`, é visando em atualizar todos os pontos do código que precisam ser atualizados de acordo com a transação que foi criada. Da mesma maneira, para fechar o componente, podemos adicionar o `lista_transacoes_adiciona_menu` para fechar.

```
lista_transacoes_adiciona_receita
    .setOnClickListener {
        AdicionaTransacaoDialog(window.decorView as ViewGroup, context: this)
            .configuraDialog(object : TransacaoDelegate {
                override fun delegate(transacao: Transacao) {
                    atualizaTransacoes(transacao)
                    lista_transacoes_adiciona_menu.close(animated: true)
                }
            })
    }
```

Vamos testar o código para ver se realmente tudo está funcionando.

Está tudo funcionando! Em outras palavras, conseguimos modificar e refatorar o código de tal forma que todo o `Dialog` não é mais uma responsabilidade da *activity*, e sim da `AdicionaTransacaoDialog`. Então, é essa classe que irá se responsabilizar por fazer todo o comportamento, e a única coisa que está fazendo para nós é delegando a responsabilidade de lidar com a transação que ela está criando para nós, por meio da interface `TransacaoDelegate`.

Agora que deixamos o código bem mais simples de se utilizar na *activity* para poder **adicionar uma receita**, o nosso próximo passo será ***adicionar uma despesa***! Até mais!