01

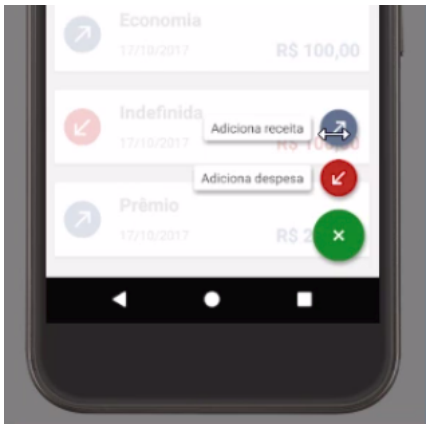
Implementação de interfaces com Object Expression e expressão lambda

Transcrição

Finalizamos o processo de refatoração, vamos dar continuidade em nosso projeto.

Podemos ver que conseguimos implementar o *Resumo*, como também a *Lista de Transações*. Mas se quisermos adicionar uma transação nova, o que pode acontecer?

Clicaremos no botão inferior direito da cor verde da nossa aplicação, que abrirá duas opções: "Adiciona Receita" e "Adiciona Despesa". Cada uma dessas opções possui um botão:



Mas, ao clicarmos no botão para a transação de receita, nada acontece, ou seja, não temos a capacidade de inserir uma nova transação de receita dentro da nossa *app*.

De acordo com a *app base*, ao clicarmos no botão para inserir uma nova receita ou despesa, é aberto uma **caixa de diálogo** conhecida como ***Dialog*** no Android. Com esse *Dialog*, conseguimos preencher informações para indicar algum valor da transação. E depois, conseguimos adicioná-las! Quantas quisermos.

Precisamos então, colocar esse aspecto na nossa *app*, para que ela também tenha essa funcionalidade. O primeiro passo, é conseguir pegar informações de cliques dos botões. Acessaremos a *ListaTransacoesActivity* primeiramente, pois é ela que mantêm o *layout*.

Na função `onCreate()`, faremos com que ela seja capaz de pegar o clique do **botão flutuante**, ou seja ***Floating Action Button***. Vamos dar uma olhada no *layout*, a fim de ver como ele foi implementado. Posicionaremos o cursor em cima do *layout* em `setContentView(R.layout.activity_lista_transacoes)` e usaremos o atalho "Ctrl + B".

Para visualizarmos melhor a hierarquia dos IDs, clicaremos na aba `Text` no canto inferior esquerdo. Se você preferir usar a aba `Design`, fique a vontade.

Na metade do código, temos o `FloatingActionMenu`, e ele é justamente o botão **verde**, que está compondo a mais outros dois componentes: os `FloatingActionButton`. Os `FloatingActionButton` representam os dois botões que aparecem após clicarmos no botão verde. Cada um deles possui um ID para a receita e para a despesa, e é a partir desses IDs que seremos capazes de pegar esse componente.

Voltando na `ListaTransacoesActivity`, chamaremos o ID:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_lista_transacoes)

    val transacoes: List<Transacao> = transacoesDeExemplo()

    configuraResumo(transacoes)

    configuraLista(transacoes)


    lista_transacoes_adiciona_receita
}
```

A partir dessa última linha, teremos o acesso ao ID, pois estamos utilizando o *synthetic*. E como podemos indicar que vamos pegar um clique dele? A partir de **Listeners**. Vamos chamar a função `setOnClickListener()`:

```
lista_transacoes_adiciona_receita.setOnClickListener()
```

Implementamos os *Listeners* para pegar eventos de cliques, ou qualquer outro tipo de evento do Android.

Como vimos anteriormente, é necessário implementar *interfaces*. No nosso caso, a implementação que precisamos fazer é da `View.OnClickListener`. Costumamos fazer esse tipo de implementação por meio do recurso de *classe anônima*. Entretanto, usaremos uma técnica chamada de ***Object Expression***, que também nos permite fazer esse tipo de implementação.

Colocamos a palavra `object` e em seguida `:`, como mostra o exemplo a seguir:

```
lista_transacoes_adiciona_receita.setOnClickListener(object : View.OnClickListener {

})
```

No momento em realizamos essa chamada, indicamos que queremos fazer uma implementação de uma interface, utilizando *Object Expression*.

Observe que algo no `object` não está certo. Isso está acontecendo porque, já que fazendo a implementação de classe anônima, é necessário implementar as suas funções. Com o cursor em cima de `object`, através do atalho "Alt + Enter", implementaremos os seus membros.

```
lista_transacoes_adiciona_receita
    .setOnClickListener(object : View.OnClickListener {
        override fun onClick(p0: View?) {

        }
    })
```

Faremos um teste simples, somente para ver se está tudo funcionando. Basicamente, ao clicar no botão "Adiciona receita", esperaremos um *toast* para dizer que foi clicado nesse botão.

```
lista_transacoes_adiciona_receita
    .setOnClickListener(object : View.OnClickListener {
        override fun onClick(p0: View?) {
            Toast.makeText()
        }
    })
```

Agora, falta mandarmos o contexto. Ao digitarmos o `this`, aparecerá duas opções. A primeira opção se refere ao objeto da interface `View.OnClickListener`. Mas, para referenciar o `this` da *activity*, ou então de uma outra classe que não seja da interface que estamos implementando, podemos utilizar uma "label". A segunda opção oferece essa label para nós.

```
lista_transacoes_adiciona_receita
    .setOnClickListener(object : View.OnClickListener {
        override fun onClick(p0: View?) {
            Toast.makeText(context: this@ListaTransacoesActivity)
        }
    })
```

Assim feito, basta colocarmos o texto para saber que é referente à receita, e logo após, informaremos o tempo do *Toast*, utilizando o `LENGTH_LONG`, e depois `show()`:

```
lista_transacoes_adiciona_receita
    .setOnClickListener(object : View.OnClickListener {
        override fun onClick(p0: View?) {
            Toast.makeText(context: this@ListaTransacoesActivity,
                "clique de receita", Toast.LENGTH_LONG).show()
        }
    })
```

Faremos o teste para saber se o `Listener` que colocamos no *FloatingActionButton*, realmente está funcionando.

Após executar a aplicação com "Alt + Shift + F10", clicamos no botão verde. Se clicarmos no *botão vermelho*, que se refere à despesa, não irá acontecer nada. Mas, se clicarmos no *botão azul*, que é o "Adiciona receita", o *toast* aparece!

Com isso, temos a certeza que conseguimos capturar o *Listener*. Agora podemos dar continuidade ao próximo passo. Mas antes, para finalizar a parte do *Listener*, podemos refatorar esse código, de modo que ele fique mais simples, como fizemos anteriormente.

Quando estamos lidando com essas interfaces do Java, temos a capacidade de converter um código para a expressão lambda.

Com o cursor em `object`, e com o "Alt + Enter", temos a opção *Convert to lambda*, da mesma maneira como vimos, utilizando o `filter()` e o `sumByDouble()`. Após o "Enter" nessa opção, o nosso código ficará assim:

```
lista_transacoes_adiciona_receita
    .setOnClickListener {
        Toast.makeText(context: this@ListaTransacoesActivity,
                text: "clique de receita", Toast.LENGTH_LONG).show()
    }
```

Fizemos com que toda a chamada ficasse omitida. Sabemos que estamos implementando o `View.OnClickListener`, porém não é necessário colocar todo aquele código aqui, pois o Kotlin já faz isso para nós utilizando a expressão lambda. Bem mais simples, não é mesmo?

Inclusive, quando fazemos esse tipo de chamada, o escopo dentro da expressão lambda é justamente **da classe que está chamando ela**, ou seja, não precisamos mais usar essas *Labels*, e sim o `this` diretamente.

```
lista_transacoes_adiciona_receita
    .setOnClickListener {
        Toast.makeText(context: this,
                text: "clique de receita", Toast.LENGTH_LONG).show()
    }
```

E ao testarmos, vemos que o *toast* ainda está lá!

A seguir, começaremos a implementar a parte do *Dialog*.