04

Inserindo layout no Dialog

Transcrição

Daremos o início a implementação do *Dialog*. O primeiro passo é tirar o *Toast*, pois não iremos mais utilizá-lo. Bom, como vamos implementar um *Dialog* no Android?

A princípio, podemos considerar o uso da classe `Dialog`, uma classe base para criar todos os tipos de *Dialog* que temos no Android. Porém a própria documentação do *Google* diz o seguinte: `Por se tratar de uma classe base, é recomendado que utilizemos uma classe específica para montar um Dialog que desejamos.`

Na *app* base, temos o *Dialog* que é uma caixa onde podemos colocar as informações referentes à receita ou despesa. Esse *Dialog* é chamado de `AlertDialog`.

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

Quando realizamos uma instância de `AlertDialog()`, vemos que houve alguns problemas. E por que isso acontece?

A princípio, o `AlertDialog()` não tem o construtor público sem parâmetros, da mesma maneira que vemos em diversas classes. Isso significa que, se caso nós quiséssemos construir um `AlertDialog()`, teríamos que utilizar uma classe específica dentro dele, chamada de `Builder`. O `Builder` será responsável por construir um `AlertDialog()` para nós.

Acessamos o `Builder` como se fosse um membro da classe, e então criaremos uma instância dele, dessa forma:

```
lista_transacoes_adiciona_receita
    .setOnClickListener {
        AlertDialog.Builder(context: this)
    }
```

Agora, já podemos dar início as configurações vistas na *app* base. A primeira configuração que podemos fazer, é colocar o **título**. Mas como podemos colocar um título no `AlertDialog`? Podemos chamar funções a partir desse *builder*, e no caso, chamaremos a função `setTitle()`

```
lista_transacoes_adiciona_receita
    .setOnClickListener {
        AlertDialog.Builder(context: this)
            .setTitle()
    }
```

Dentro de `setTitle()`, podemos colocar desta forma:

```
AlertDialog.Builder(context: this)
    .setTitle("Adicionar receita")
```

Entretanto, já que temos os recursos da *app* base, é interessante utilizar os recursos que fazem referência aos valores de `String`. Então, ao invés de colocar a informação parada em uma "string solta", podemos *pedir um recurso* de `String` chamado de `adiciona_receita`.

```
lista_transacoes_adiciona_receita
    .setOnClickListener {
        AlertDialog.Builder(context: this)
            .setTitle(R.string.adiciona_receita)
    }
```

Como vamos ter a certeza de que está funcionando? Ao tentarmos executar a *app*, essa alteração não será mostrada, pois precisamos pedir para que o `AlertDialog` apareça, da mesma forma como o `Toast`. Portanto, usaremos o método `show()`:

```
lista_transacoes_adiciona_receita
    .setOnClickListener {
        AlertDialog.Builder(context: this)
            .setTitle(R.string.adiciona_receita)
            .show()
    }
```

Executaremos a aplicação.

Ao clicar no botão verde, e depois no botão azul para adicionar a receita, aparece a primeira amostra do *Dialog*. Feito isso, falta implementar as outras partes do *Dialog*. Para isso, utilizaremos um *layout* que já está pronto.

Indicaremos para o `builder` que utilizaremos tal *layout*, a partir da função `setView()`:

```
lista_transacoes_adiciona_receita
    .setOnClickListener {
        AlertDialog.Builder(context: this)
            .setTitle(R.string.adiciona_receita)
            .setView()
            .show()
    }
```

A `setView()` irá receber uma `View` que contém o *layout* visto na *app* base. Isso significa que iremos criar esse *layout*, retornar a view, e setar para o `AlertDialogBuilder.

Para `criar um layout`, utilizaremos a classe `LayoutInflater` a partir de um context:

```
lista_transacoes_adiciona_receita
    .setOnClickListener {

        LayoutInflater.from(context: this).inflate(R.layout.form_trasacao)

        AlertDialog.Builder(context: this)
            .setTitle(R.string.adiciona_receita)
            .setView()
```

```
                .show()
        }
```

A função `inflate()` é responsável por criar a view, e o `form_transacao` representa o *layout* da *app* base composto por "Valor", "Data" e "Categoria".

Como próximo parâmetro, indicaremos o *ViewGroup*, ou o *parent*. Mas não temos acesso a ele neste momento. Como que podemos pegar esse *ViewGroup*? A partir da view da *activity*! Pois é ela que será o *parent* do *Dialog*.

```
lista_transacoes_adiciona_receita
    .setOnClickListener {
        val view: View = window.decorView
        LayoutInflater.from(context: this)
            .inflate(R.layout.form_trasacao)

        AlertDialog.Builder(context: this)
            .setTitle(R.string.adiciona_receita)
            .setView()
            .show()
    }
```

Já temos a view, basta mandarmos a informação dela para o `inflate()`:

```
LayoutInflater.from(context: this)
    .inflate(R.layout.form_trasacao, view)
```

O que acontece se mandarmos a `view` como parâmetro dessa forma? O `inflate()` entende que esse parâmetro tem que ser uma **ViewGroup**, ao invés de uma `View`. Mas sabemos que a `view` passada como parâmetro indica um `ViewGroup`, pois ela é o *parent* do layout!

Então, usaremos o **casting** para indicar ao `inflate()` que estamos mandando uma `ViewGroup`. No Kotlin, usamos a *keyword* **as**, que pode ser traduzida pela frase "*como se fosse*":

```
LayoutInflater.from(context: this)
    .inflate(R.layout.form_trasacao, view as ViewGroup)
```

Dessa forma, estamos nos comprometendo que mandamos uma `view` que é na verdade uma `ViewGroup`. Um outro parâmetro muito importante é o *attachToRoot*. Esse parâmetro pode ser `true` ou `false`, permitindo ou não que a view seja criada agora. No caso, escolhemos enviar como `false`, pois a view só será criada quando aparecer o `AlertDialog`:

```
LayoutInflater.from(context: this)
    .inflate(R.layout.form_trasacao,
        view as ViewGroup,
        attachToRoot: false)
```

Conseguimos realizar o processo de criar a view, nos resta devolver essa informação. Vamos inserir `.val` logo após o parêntese do `false`, e dar "Enter". Teremos uma nova view e a chamaremos de `viewCriada`. O código será refatorado, e ficará assim:

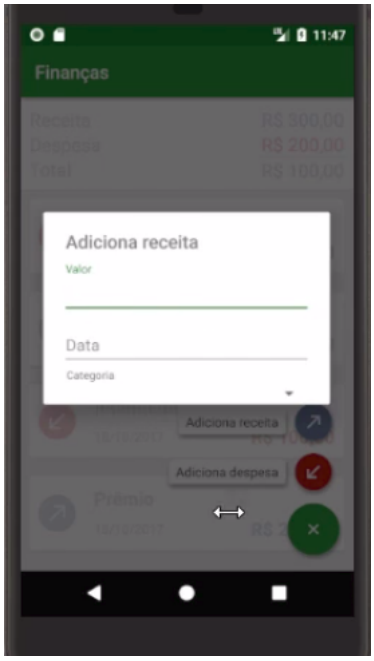
```
lista_transacoes_adiciona_receita
    .setOnClickListener {
        val view: View = window.decorView
        val viewCriada = LayoutInflater.from(context: this)
            .inflate(R.layout.form_trasacao,
                view as ViewGroup,
                attachToRoot: false)

        AlertDialog.Builder(context: this)
            .setTitle(R.string.adiciona_receita)
            .setView(viewCriada)
            .show()
    }
```

Não se esqueça de mandar a `viewCriada` para o *Dialog* em `setView()`.

Então nós criamos o *layout,* e mandamos o *layout* criado para o `AlertDialog`. Vamos executar a aplicação para ver o resultado.

Temos a primeira amostra do nosso *Dialog*:



Ao compararmos com a *app* base, percebemos que já estamos bem perto das informações. Entretanto, há detalhes que necessitam de melhorias como o campo de data, e de categoria.

Para o primeiro passo, conseguimos colocar o *layout,* e então adiante iremos tratar desses detalhes que restam.