07

Adicionando cores nas transações

Transcrição

Com a adaptação da nossa transação para a distinção de seu tipo, vamos começar a implementar o *adapter* para incluirmos algumas cores na app. Antes disto, vejam que há uma sugestão do Android Studio para alterarmos a sintaxe e utilizarmos a *property* em `setAdapter`.

Quando setamos um *adapter*, utilizamos sua *property* também. Com o cursor do mouse sobre `setAdapter`, apertaremos "Alt + Enter" e o texto é alterado para `adapter`, deixando o código assim:

```
lista_transacoes_listview.adapter = ListaTransacoesAdapter(transacoes, context: this)
```

Vamos acessar o *adapter* com o cursor em `ListaTransacoesAdapter` e "Ctrl + B". Já que agora temos o tipo da transação, faremos as estruturas condicionais, os `if`s e `else`s, para indicar que comportamento deverá existir na identificação de uma receita ou despesa.

Significa que, dada uma transação e seu tipo, o `if()` verificará se este tipo é, por exemplo, uma receita - que importaremos a partir de "Alt + Enter". Caso isto seja verdadeiro, algo deverá ser feito. Neste primeiro momento, a cor do valor deverá ser outra.

O valor virá por meio de `viewCriada`, cuja cor, proveniente dos *resources*, mudaremos com a função `setTextColor()`. Para isto precisaremos utilizar a classe do Android `ContextCompat`, com que chamaremos a função `getColor()`.

Para enviarmos uma cor é necessário um contexto, também presente em nossa *property*. Indicaremos a cor desejada com a classe `R` para representar os recursos. Por ser uma transação do tipo `RECEITA`, queremos uma cor de mesmo nome, já definido em nossos recursos.

Repetiremos o processo para as transações de despesa:

```
val transacao = transacoes[posicao]

if(transacao.tipo == Tipo.RECEITA) {
    viewCriada.transacao_valor.setTextColor(ContextCompat.getColor(context, R.color.receita))
} else {
    viewCriada.transacao_valor.setTextColor(ContextCompat.getColor(context, R.color.despesa))
}
```

Vamos verificar se tudo está funcionando direitinho? Com "Alt + Shift + F10" rodaremos a aplicação. No emulador, vê-se a transação "Indefinida" com `20.5` na cor vermelha, como esperado, então já sabemos que se trata de uma despesa.

Falta acrescentarmos os ícones! Para isto, utilizaremos a mesma lógica:

```
if(transacao.tipo == Tipo.RECEITA) {
    viewCriada.transacao_icone.setBackgroundResource(R.drawable.icone_transacao_item_receita)
} else {
    viewCriada.transacao_icone.setBackgroundResource(R.drawable.icone_transacao_item_despesa)
}
```

Em que `transacao_icone` permite colocarmos qualquer ícone que desejarmos, proveniente de um *resource*, e trata-se de uma *ImageView*.

Vamos executar a app e verificar seu funcionamento. Os ícones foram implementados com sucesso! Agora, toda vez que colocarmos uma receita ou uma despesa, ela virá acompanhada do ícone correspondente, o que podemos comprovar ao voltarmos à *Activity* e incluirmos transações: de despesa no valor de `200.0` e de receita, de `500.0` .

Uma das técnicas para organização do código no Android Studio é o atalho "Ctrl + Alt + L":

```
val transacoes = listOf(Transacao(
        tipo = Tipo.DESPESA,
        data = Calendar.getInstance(),
        valor = BigDecimal(20.5)),
        Transacao(valor = BigDecimal(100.0),
                tipo = Tipo.RECEITA,
                categoria = "Economia"),
        Transacao(valor = BigDecimal(200.0),
                tipo = Tipo.DESPESA),
        Transacao(valor = BigDecimal(500.0),
                categoria = "Prêmio",
                tipo = Tipo.RECEITA))
```

Vamos ver o comportamento da nossa lista com esta quantidade maior de transações. Tudo é executado com sucesso, conseguimos atingir nosso objetivo! Fique à vontade para modificar a interface da app conforme seu gosto.