09

Inserindo os demais valores nas transações

Transcrição

Agora que entendemos todas as peculiaridades ao lidarmos com propriedades, vamos modificar nossa classe `Transacao` para trabalharmos com elas de maneira adequada.

Já vimos que não é preciso colocar as *properties* no corpo da classe quando estamos usando apenas os seus *gets* e *sets* de forma padrão. Assim sendo, é possível deixar o código da seguinte maneira:

```
class Transacao (val valor: BigDecimal,
                 val categoria: String,
                 val data: Calendar)
```

E caso alguém queira modificar seu valor, utilizaremos o `var`, se isto for realmente necessário. Voltando ao nosso `ListaTransacoesAdapter`, vejam que estamos mantendo o mesmo esquema anterior, e o mesmo ocorre na *Activity*, isto é, nada foi quebrado.

Conseguimos acessar nossa *property* `valor`, e colocá-la na lista de transação. Precisaremos preencher o restante das informações, indicando à `viewCriada` quanto à categoria, que é de fato uma *string*, e setando seu valor:

```
val transacao = transacoes[posicao]

viewCriada.transacao_valor.setText(transacao.valor.toString())
viewCriada.transacao_categoria.setText(transacao.categoria)

return viewCriada
```

Percebam como `setText(transacao.valor.toString())` está marcado como tendo uma sugestão. O que isso significa? Apertando "Alt + Enter" com o cursor sobre `toString()`, o Android Studio nos sugere alterar o uso da sintaxe como uma *property*.

Ou seja, se apertarmos "Enter", tenta-se utilizar `text` do `TextView` como uma *property*, porque quando fazemos um `setText` e um `getText`, o mesmo valor é inserido e retornado.

O Kotlin permite que se faça uso de uma *property*, essa parte do atributo `text`. Esta é outra maneira como o Kotlin resume o código para nós, em vez de termos que colocar uma chamada de função que colocará o valor via parâmetro. Isto é feito de maneira bem mais objetiva.

Por fim, é necessário acessarmos a `viewCriada` e chamar `transacao_data`, referente a um `Calendar`, chamando também a *property* diretamente, e não o `setText`, e solicitando sua impressão com `toString()`. O código ficará assim:

```
val transacao = transacoes[posicao]

viewCriada.transacao_valor.text = transacao.valor.toString()
viewCriada.transacao_categoria.text = transacao.categoria
viewCriada.transacao_data.text = transacao.data.toString()
```

```
return viewCriada
```

Feito isto, vamos executar a app com "Alt + Shift + F10" e verificar o que acontece! Aguardaremos o `Gradle` do Android Studio ser executado. Se abrirmos o emulador, teremos as categorias implementadas corretamente.

Porém, na data, imprimiu-se o `toString()` padrão da classe `Calendar`, que mostra uma informação gigantesca. Na verdade, o que esperamos é que se mostre a data formatada conforme conhecemos, em português brasileiro, com dia, mês e ano separados por barra.

Para isso, pode-se utilizar o mesmo processo do que fazemos em Java, indicando este formato e extraindo-o para uma variável (`formatoBrasileiro`). Mas como fazemos para que este formato seja convertido para a leitura do `Calendar` ?

Existe uma classe específica do Java para isto, a `SimpleDateFormat`, de que faremos uma instância, durante a qual se vê que é esperado um *pattern*, um padrão. Devolveremos uma variável, um `format`, responsável por usar este padrão como o formato esperado quando formos converter um `Calendar`.

No `Calendar`, há o `time`, que acessaríamos com `getTime()`, já que se trata de um modificador de acesso, junto com `setTime()`. No entanto, com Kotlin poderemos usar a *property* `time`, o qual devolve um *date* de acordo com o `SimpleDateFormat`.

Se extrairmos isto, teremos `dataFormatada` ! Assim:

```
viewCriada.transacao_valor.text = transacao.valor.toString()
viewCriada.transacao_categoria.text = transacao.categoria

val formatoBrasileiro = "dd/MM/yyyy"
val format = SimpleDateFormat(formatoBrasileiro)
val dataFormatada = format.format(transacao.data.time)
viewCriada.transacao_data.text = dataFormatada


return viewCriada
```

Vamos executar a aplicação e ver o que acontece?

Agora, sim, tudo aparece conforme esperado. Uma das vantagens de se usar o Kotlin é que não precisamos "dispensar" tudo que foi aprendido em Java, famosa característica conhecida por **interoperabilidade**.

Conseguimos finalizar o primeiro processo de envio das informações à nossa lista de transações. Na próxima aula avançaremos um pouco mais e veremos como incluir outras informações e melhorar o código, uma vez que se repararmos, o *adapter* está se responsabilizando pela formatação da nossa data, algo que não deveria ocorrer.