04

Formatando a categoria da transação

Transcrição

Os valores das transações se encontram no padrão da moeda brasileira, então já temos esta informação de forma bem descritiva para o usuário final. Há, porém, um detalhe que ainda não testamos: as categorias estão setadas como tendo mais ou menos a mesma quantidade de caracteres.

Vamos experimentar um cenário no qual a categoria tenha muitos caracteres. Acessaremos a *Activity* e incluiremos uma categoria para a primeira transação, denominada "almoço de final de semana":

```
val transacoes = listOf(Transacao(
    tipo = Tipo.DESPESA,
    categoria = "almoço de final de semana",
    data = Calendar.getInstance(),
    valor = BigDecimal(val: 20.5)),
    //...)
```

Com "Alt + Shift + F10", executaremos a app. A nova categoria foi implementada sem problemas, mas no momento em que isto foi feito, a data da transação sumiu. Ou seja, ao colocarmos uma categoria que ultrapassa o limite de caracteres, acabamos perdendo informações.

O que ocorre em nossa app base neste tipo de situação? Vamos dar uma olhada adicionando uma nova transação clicando em "Adiciona receita", definindo "40" como valor, mantendo a data atual e selecionando a maior categoria, neste caso, "Receita indefinida".

O que se vê no emulador é que a categoria é cortada a partir de um determinado momento, seguido de reticências ("Receita indefi..."). Isto é, quando se ultrapassa um limite de caracteres, a própria aplicação faz uma abreviação e acrescenta os três pontos (" `...` ").

Para evitarmos a perda de informações, implementaremos isto em nossa aplicação também. Nosso primeiro passo é, no *adapter*, quando formos utilizar a categoria, primeiramente a formataremos, e a incluiremos em `transacao.categoria`, da mesma forma como fizemos na parte de `valor` e `data`.

Pegaremos uma variável da `transacao.categoria`, uma `val`, que será `categoriaFormatada`, a própria categoria, verificando seu tamanho em seguida. Caso ele ultrapasse um certo limite, faremos algo a respeito. Mas qual seria este limite?

Vamos utilizar o mesmo parâmetro do projeto `financas`, acessando `ListaTransacaoAdapter`, em que há uma constante, `LIMITE_DE_CATEGORIA = 14`, que poderemos usar. Voltando à `financask`, incluiremos este mesmo limite:

```
val categoriaFormatada = transacao.categoria
if(categoriaFormatada.length > 14){
    categoriaFormatada = categoriaFormatada.substring(0, 14)
}
```

Com isto, definiremos que nós é que queremos modificar o valor da variável e, portanto, alteraremos `val` para `var`. Agora que conseguimos formatar a categoria, basta enviá-la!

```
var categoriaFormatada = transacao.categoria
if(categoriaFormatada.length > 14){
    categoriaFormatada = categoriaFormatada.substring(0, 14)
}

viewCriada.transacao_valor.text = transacao.valor.formataParaBrasileiro()
viewCriada.transacao_categoria.text = categoriaFormatada
viewCriada.transacao_data.text = transacao.data.formataParaBrasileiro()
```

Usaremos o "Alt + Shift + F10" para verificar se a app funciona como esperado. Rodando-a no emulador, veremos que a primeira transação possui categoria "almoço de fina", ou seja, apenas um pedaço do texto é mostrado, como gostaríamos.

Na aplicação base, aparecem os três pontinhos (`...`) indicando que há uma continuação oculta do texto, então precisaremos incluí-los em nossa versão também, para melhorar o entendimento da categoria.

Uma das maneiras de fazer isto é com a concatenação:

```
var categoriaFormatada = transacao.categoria
if(categoriaFormatada.length > 14){
    categoriaFormatada = categoriaFormatada.substring(0, 14) + "..."
}
```

Ao executarmos o código, veremos que as reticências foram implementadas com sucesso.

No entanto, o Kotlin possui uma maneira mais elegante de lidar com concatenações, uma *feature* conhecida como *String Templates*. Ela permite que usemos a *string* em todo o nosso código, e o uso de variáveis e expressões por meio do `$` , com que indicaremos a impressão do valor de uma variável, neste caso a `categoriaFormatada` :

```
var categoriaFormatada = transacao.categoria
if(categoriaFormatada.length > 14){
    categoriaFormatada = "$categoriaFormatada.substring(0, 14)"
}
```

Se rodarmos nossa aplicação agora, é mostrada toda a categoria que colocamos no código, então, "almoço de final de semana.substring(0, 14)", em que "almoço de final de semana" representa a `categoriaFormatada` , concatenando-se a `substring` , que não é nem considerada uma função.

Sendo assim, lidamos com algo bem específico, pois não queremos simplesmente imprimir o valor da variável, e sim o retorno da função de sua `substring` . Como faremos para que, em uma *String Template*, consigamos executar `"$categoriaFormatada.substring(0, 14)"` ?

Poderemos abrir o corpo desta *String Template* e, a partir do símbolo `$` , toda a expressão será considerada e executada dentro de uma *string*. E para a concatenação com as reticências, basta acrescentá-las assim:

```
var categoriaFormatada = transacao.categoria
if(categoriaFormatada.length > 14){
    categoriaFormatada = "${categoriaFormatada.substring(0, 14)}..."
}
```

Através da execução da app vê-se que categoria foi abreviada conforme desejávamos!

Como vimos anteriormente com `formataParaBrasileiro`, com `valor` e `data`, também lidamos com a formatação cuja responsabilidade não é do *adapter*. Faz sentido que a *string* seja a responsável pela limitação de caracteres.

Por conta disto, repetiremos o procedimento de criação de uma função que determinará a limitação da *string*, sendo este um comportamento comum.

Criaremos a função `limitaEmAte()`, a receber o parâmetro `caracteres`. Em seu corpo, indicaremos que este tipo de função será chamada pela *string*. Neste momento, é necessário verificarmos se o tamanho da *string* que for chamá-la é maior do que a quantidade de caracteres que estamos enviando.

Se isto for verdade, queremos devolver `"${categoriaFormatada.substring(0, 14)}..."`, a partir do qual faremos seu retorno, pois não estamos mais lidando com `categoriaFormatada`, e sim com a própria *string* que chama a si mesma.

Por fim, quando não entrarmos neste caso em que ultrapassamos o limite de caracteres, devolveremos a própria *string*. O código ficará da seguinte forma:

```
fun String.limitaEmAte(caracteres: Int) : String{
    if(this.length > caracteres){
        return "${this.substring(0, 14)}..."
    }
    return this
}
```

Em seguida, deletaremos o código referente à variável `categoriaFormatada`:

```
var categoriaFormatada = transacao.categoria
if(categoriaFormatada.length > 14){
    categoriaFormatada = "${categoriaFormatada.substring(0, 14)}..."
}
```

Feito isto, substituiremos `categoriaFormatada` da `viewCriada` por `transacao.categoria.limitaEmAte(caracteres: 14)`, em que `14` será uma *property*, para agregar mais significado. Faremos isto por meio do atalho "Ctrl + Alt + F", selecionando `14`, e a classe, não o *adapter*.

Qual será o nome desta *property*? Apertando-se "Enter", é mostrado o nome dado pelo Android Studio, `i`. É possível modificá-lo, refletindo no `i` mais abaixo no código, uma vez que ele não traz muito significado.

Com o cursor do mouse sobre `i`, clicaremos em "Shift + F6", renomeando-o com `limiteDaCategoria`, a partir do qual o programa pede que se faça uma refatoração.

```
private val limiteDaCategoria = 14
```

A `viewCriada` após a refatoração ficará assim:

```
viewCriada.transacao_categoria.text = transacao.categoria.limitaEmAte(limiteDaCategoria)
```

É claro que poderemos melhorar a visualização do código alterando a identação e não deixando tudo na mesma linha:

```
viewCriada.transacao_valor.text = transacao.valor
        .formataParaBrasileiro()
viewCriada.transacao_categoria.text = transacao.categoria
        .limitaEmAte(limiteDaCategoria)
viewCriada.transacao_data.text = transacao.data
        .formataParaBrasileiro()

return viewCriada
```

Agora, no momento em que solicitamos a categoria `transacao`, pedimos para que seja levado em conta o `limiteDaCategoria`, que é `14`. Depois, chama-se a extensão desta `String`.

Vamos executar a app e ver seu funcionamento. No emulador, mostra-se que tudo permanece funcionando corretamente! Legal!

Porém, o `limitaEmAte()` não é responsabilidade do *adapter* e, da mesma maneira como fizemos para `formataParaBrasileiro`, tanto para o `BigDecimal` quanto para o `Calendar`, isolaremos o `limitaEmAte()` em uma classe específica para isto.

Acessaremos nosso projeto no menu lateral e criaremos um arquivo em Kotlin que se chamará "StringExtension", uma extensão da *string*.

Apagaremos o comentário deste novo arquivo, para o qual colaremos o código relativo à função `limitaEmAte()`, localizado em `ListaTransacoesAdapter`.

Quando fizemos a formatação, o valor do `limiteDaCategoria` foi alterado, isto é, manteve-se a constante que tínhamos no nosso *adapter*, portanto o alteraremos para `caracteres`, que é de fato a variável que pegará o limite da nossa categoria. O código será:

```
fun String.limitaEmAte(caracteres: Int) : String{
    if(this.length > caracteres){
        return "${this.substring(0, caracteres)}..."
    }
    return this
}
```

Voltando à `ListaTransacoesAdapter`, o Android Studio pede para que se importe `limitaEmAte()`, o que faremos com "Enter". Executando-se a aplicação com "Alt + Shift + F10", vê-se que tudo se mantém como esperado.

Estamos conseguindo fazer com que este tipo de comportamento de limitação de caracteres seja próprio da *string*, algo bem comum, pois poderemos utilizar `limitaEmAte()` em outros locais sem que o *adapter* tenha que se responsabilizar por isto.

Além disso, o aspecto visual da nossa aplicação atualmente está bem próximo ao da app base em Java.