01

Formatando a moeda da transação

Transcrição

Atualmente, as nossas transações possuem as mesmas representações visuais da app base feita em Java. Se repararmos nela, saberemos que a primeira transação é uma receita, e a segunda, uma despesa.

Isto porque na receita a cor do valor aparece em azul, e a despesa é representada pelo amarelo. Da mesma forma, se abrirmos nossa app no emulador, saberemos que a primeira transação é uma despesa pelos mesmos motivos e, a segunda, uma receita.

No entanto, se analisarmos melhor, o valor da transação em si não diz muito, pois seu significado não é especificado. Isto é, não se sabe se lidamos com um valor do tipo Real, Dólar, Euro, entre outras moedas existentes.

Lidamos com a moeda brasileira na app base, perceptível por conta do `R$` que acompanha o valor. Para deixarmos esta informação bem mais descritiva facilitar sua compreensão, alteraremos o código!

Acessaremos `ListaTransacoesAdapter` em que há uma representação de *string* em `transacao.valor.toString()`. Uma das abordagens possíveis é fazer diversas concatenações e formatações para se alcançar o mesmo resultado.

Porém, como estamos lidando com a API do `BigDecimal`, existe uma forma mais fácil e objetiva de fazê-lo, por meio da classe `DecimalFormat`, a partir da qual pegaremos uma instância de moeda com que definiremos nosso formato.

Solicitarmos uma instância de moeda para o `DecimalFormat` usando a função `getCurrencyInstance()`. No entanto, ao fazermos isto, não há garantia de que moeda solicitamos, então usa-se a moeda do local em que ele está. Por exemplo, se estivéssemos nos EUA, o programa poderia usar o dólar americano.

Precisaremos especificar esta informação usando a classe `Locale()`, determinando a língua e localização a serem utilizadas. O código ficará assim:

```
DecimalFormat.getCurrencyInstance(Locale(language: "pt", country: "br"))
```

Feita a instância da moeda, devolveremos para uma variável com `.val`. Devolvemos um formatador de moeda, sendo possível até especificarmos explicitamente o seu tipo, marcando o checkbox correspondente, porém não o faremos neste momento pois isso já está bem claro para nós.

Com `formatoBrasileiro` identificaremos o significado deste formatador, o qual utilizaremos para que ele formate o valor da transação. Seu retorno é a própria `String`, ou seja, a moeda formatada. Portanto, teremos:

```
val formatoBrasileiro = DecimalFormat.getCurrencyInstance(Locale(language: "pt", country: "br"))
val moedaFormatada = formatoBrasileiro.format(transacao.valor)
```

Feito isto, precisaremos apenas colocar a `moedaFormatada` no `TextView`, substituindo o `transacao.valor.toString()`:

```
viewCriada.transacao_valor.text = moedaFormatada
viewCriada.transacao_categoria.text = transacao.categoria
```

```
viewCriada.transacao_data.text = transacao.data.formataParaBrasileiro()
```

Vamos ver como isso funciona? O Android Studio executou a app sem problemas e, abrindo-a no emulador, veremos que o `R$` foi incluído como gostaríamos, trocando também o `.` (ponto) por `,` (vírgula) de acordo com o padrão da moeda nacional.

Porém, o `R$` está muito próximo do valor e, para corrigir isto, uma vez que o `format` devolve uma *String*, utilizarmos suas funções, pedindo para que se substitua o padrão `"R$"` para `"R$ "` (com espaçamento).

```
val formatoBrasileiro = DecimalFormat
        .getCurrencyInstance(Locale(language: "pt", country: "br"))
val moedaFormatada = formatoBrasileiro
        .format(transacao.valor)
        .replace(oldValue: "R$", newValue: "R$ ")
```

Com "Alt + Shft + F10" verificaremos que as alterações foram implementadas sem problemas. Agora, estamos no mesmo cenário de quando formatamos a data, isto é, responsabilizamos o *adapter*, cuja função é de nos devolver uma moeda formatada baseando-se no valor. Esta responsabilidade, portanto, deveria ser de alguma classe específica para tal.

Poderemos fazer uma extensão da classe `BigDecimal`, porque faz sentido ela formatar o valor para nós. Para isto, faremos algo que foi visto com a classe `Calendar`.

Em `ListaTransacoesAdapter`, criaremos uma função denominada `formataParaBrasileiro()`, extensão para uma classe `BigDecimal`. Copiaremos e colaremos em seu corpo todo o código escrito anteriormente.

Quando estamos no escopo de uma extensão, temos acesso ao seu objeto a partir de `this`, que neste caso substituirá `transacao.valor`. E para devolvermos este valor formatado, nossa `moedaFormatada`, faremos um retorno e indicaremos que ele é uma *string*.

```
fun BigDecimal.formataParaBrasileiro() : String {
    val formatoBrasileiro = DecimalFormat
        .getCurrencyInstance(Locale(language: "pt", country: "br"))
    val moedaFormatada = formatoBrasileiro
        .format(obj: this)
        .replace(oldValue: "R$", newValue: "R$ ")

    return moedaFormatada
}
```

Deste modo, fazemos com que o `BigDecimal` tenha esta extensão e consiga formatá-la. Na `viewCriada`, trocaremos `moedaFormatada` por `transacao.valor.formataParaBrasileiro()`:

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

return viewCriada
```

Vamos executar o código e ver se tudo funciona direitinho!

Este tipo de função não é um comportamento esperado para um *adapter*, sendo necessário isolá-lo. Acessaremos "app > java > br.com.alura.financask > extension" e criaremos um arquivo com extensões do `BigDecimal` por meio de "Alt + Insert", selecionando "New Kotlin File/Class". O nome será "BigDecimalExtension".

Tendo gerado o arquivo, basta voltarmos à função criada no *adapter* (a extensão do `BigDecimal`), copiá-la e colá-la no novo arquivo, importando-a. Notem que o Android Studio faz a importação de todo o `util`, sendo que precisaremos apenas do `Locale`.

Nosso código ficará desta forma:

```
import java.math.BigDecimal
import java.text.DecimalFormat
import java.util.Locale

fun BigDecimal.formataParaBrasileiro() : String {
    val formatoBrasileiro = DecimalFormat
        .getCurrencyInstance(Locale(language: "pt", country: "br"))
    val moedaFormatada = formatoBrasileiro
        .format(obj: this)
        .replace(oldValue: "R$", newValue: "R$ ")

    return moedaFormatada
}
```

Por fim, é preciso fazermos o *import* de `formataParaBrasileiro`, mas veremos que isto já foi feito, lá no começo do código, em que inclusive há importações que se tornaram desnecessárias, identificadas pela cor cinza. Um atalho que serve para deletá-las é "Ctrl + Alt + O".

Executaremos a app e verificaremos que tudo funciona como esperado. No entanto, percebam que a extensão do `formataParaBrasileiro`, possui o mesmo nome que o `BigDecimal` e o `Calendar`, mas o *import* foi de apenas um arquivo (`import br.com.alura.financask.extension.formataParaBrasileiro`).

O Kotlin permite este tipo de importação, pois é capaz de identificar que são diferentes por conta dos respectivos formatos, então, não é necessário importá-los um a um, por mais que isto pareça estranho.

Conseguimos fazer com que nossas transações tenham o aspecto visual de moeda no padrão brasileiro!