03

Inserindo a cor do total

Transcrição

Nesse vídeo, colocaremos a cor do **total**.

Para isso, na função `adicionaTotal()`, faremos uma *comparação* pra entender qual é o valor que representa a variável `total`. Como vimos nas aulas, em nossa *app* base temos diversos momentos em que pegamos uma cor para representar o **total**.

Quando o total for *negativo*, ele pegará a cor da **despesa**, que é o vermelho. Quando o total for *positivo* ou *neutro*, ele manterá a cor da **receita**.

Para começar, logo depois da variável `total` dentro do método `adicionaTotal()`, faremos uma comparação utilizando o **if**.

Como estamos lidando com o *BigDecimal*, não tem como fazer comparações iguais as que fazemos com os **tipos primitivos**, por exemplo utilizando o `=`, `<=` ou `>=` entre outros. Mas, como conseguimos utilizar *funções Java* dentro do Kotlin, então temos uma função que permite **comparar objetos**: `compareTo()`!

```
fun adicionaTotal() {
    var total = resumo().total()
    if(total.compareTo(BigDecimal.ZERO))
    view.resumo_card_total.text = total.formataParaBrasileiro()
}
```

A partir da função `compareTo()`, somos capazes de pedir que a variável `total` seja comparada com um outro objeto, no caso, o **BigDecimal**.

A princípio, vamos comparar com um **valor zerado**, pois baseando-se nele que faremos a lógica.

Se ele for maior ou igual a zero, queremos deixar uma **receita**.

Se ele for menor do que zero, queremos deixar uma **despesa**.

E por que esse código não está compilando? Pois o `compareTo()` nos devolve um *valor inteiro*, podendo este ser de 3 tipos.

Quando esse método nos devolver o valor inteiro `1`, significa que o valor comparado é **maior** que o zero.

Quando o método nos devolver o valor `0`, significa que o valor que estamos comparando é **igual** a zero.

E por último, quando esse método nos devolver o valor inteiro `-1`, significa que o valor que estamos comparando é **menor** que zero.

Precisamos nos basear nos três retornos que a função `compareTo()` pode nos devolver. Com isso, o código ficará assim:

```
fun adicionaTotal() {
    var total = resumo().total()
    if(total.compareTo(BigDecimal.ZERO) >= 0){

    }
    view.resumo_card_total.text = total.formataParaBrasileiro()
}
```

De início, dentro do escopo do `if()`, pegaremos o componente `view.resumo_card_total` e *setaremos* a cor.

```
fun adicionaTotal() {
    var total = resumo().total()
    if(total.compareTo(BigDecimal.ZERO) >= 0){
        view.resumo_card_total.setTextColor(ContextCompat.getColor(context, R.color.receita)
    }
    view.resumo_card_total.text = total.formataParaBrasileiro()
}
```

Quando essa condição for **falsa**, ou seja, quando essa condição der `-1`, então atribuiremos a cor de *despesa*.

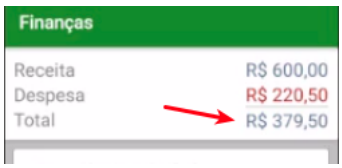
```
fun adicionaTotal() {
    var total = resumo().total()
    if(total.compareTo(BigDecimal.ZERO) >= 0){
        view.resumo_card_total.setTextColor(ContextCompat.getColor(context, R.color.receita)
    } else {
        view.resumo_card_total.setTextColor(ContextCompat.getColor(context, R.color.despesa)
    }
    view.resumo_card_total.text = total.formataParaBrasileiro()
}
```

É dessa maneira que conseguimos fazer com que o nosso objeto seja comparado e verificado. Testaremos em seguida, utilizando o atalho "Alt + Shift + F10".

Como o total é negativo, ele pegou a cor atribuída para a **despesa**.

Para testar o outro retorno da *comparação de objetos*, vamos voltar na *activity* `ListaTransacoesActivity`, e na função `transacoesDeExemplo()`, *setaremos* um novo valor manualmente de `700.0` para `200.0`. Assim, faremos com que o total seja **positivo**, e então, ele terá que atribuir a mesma cor da despesa.

Vamos executar novamente.



Legal! Temos um valor positivo, e que está com a mesma cor da receita.

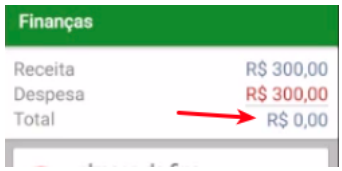
E o que acontece quando colocamos um valor **neutro**, ou "zerado"?

Vamos alterar o código para que ele fique assim:

```
private fun transacoesDeExemplo(): List<Transacao> {
    return listOf(Transacao(
        tipo = Tipo.DESPESA,
        categoria = "almoço de final de semana",
        data = Calendar.getInstance(),
        valor = BigDecimal(val: 100.0)),
        Transacao(valor = BigDecimal(val:100.0), tipo = Tipo.RECEITA, categoria = "Economia"),
        Transacao(valor = BigDecimal(val:200.0), tipo = Tipo.DESPESA),
        Transacao(valor = BigDecimal(val:200.0), categoria = "Premio", tipo = Tipo.RECEITA))
}
```

Dessa forma, os valores são equivalentes.

Ao executar novamente o programa, obtemos esse resultado:



O valor **total** está zerado, e está pegando a cor da *receita*.

Para dar uma reforçada no conteúdo, vamos falar um pouco mais a fundo o que aconteceu por debaixo dos panos durante a execução do programa. A função `compareTo()` faz parte de uma interface `Comparable`. A partir dessa interface, depois de implementada, conseguimos indicar se uma classe é *maior*, *menor* ou *igual* a algum objeto que queremos comparar.

O código ficou extenso. Será que não há como resumi-lo um pouco mais? Nas aulas passadas, reforçamos bastante a refatoração do código para poder melhorá-lo quando necessário, e na próxima aula continuaremos com esse mesmo conceito.