03

# Adicionando as receitas no resumo

## Transcrição

Atualmente, a *app* contém listas de transações: duas de despesa, e outras duas de receita. Mas, se olharmos bem na parte de cima da *app*, percebemos algumas informações como `Receita, Despesa e Total` , mas ao lado não há nada dizendo o que elas significam para nós.


A imagem a seguir nos mostra como é a nossa *app* base. Veja só:


Como podemos ver, temos a lista de transações na parte debaixo, e temos, ainda, uma *Despesa*, uma *Receita* e um *Total* no painel acima. No entanto, há uma diferença: a "Receita" está com o valor de `R$ 140,00` , a "Despesa" com `R$ 20,50` , e o "Total" com o resultado de `R$ 119,50` .

Em outras palavras, a parte do painel superior é justamente um *resumo* do que está acontecendo na lista abaixo. Mas por que um resumo?

Os `R$ 140,00` mostrados significam a soma da Receita de `R$ 100,00` mais a Receita Indefinida de `R$ 40,00` . Como podemos ver também, em *Despesa* é indicado somente a única transação de despesas de `R$ 20,50` , e no *Total* é feita a subtração.

Inclusive, para checarmos esse resultado, adicionaremos mais uma **despesa**, a fim de realizar outra subtração.

A *despesa* será no valor de `R$ 50,00` , e sua categoria será de **Contas**. Clicamos em "Adicionar". O valor das *Despesas* no painel superior mudou para o valor de `R$ 70,50` , ou seja, foi somado o valor da despesa Comida de `R$ 20,50` mais o valor da despesa Contas de `R$ 50,00` .

A ***app*** **base** é calcula e mostra todas as *despesas* e *receitas*. E é justamente isso o que vamos implementar nessa parte do curso.

## Como podemos indicar essa informação de Resumo?

Já que o **Resumo** está dentro dessa parte do *layout*, trabalharemos com a nossa *activity*.

No **Android Studio**, dentro do projeto, buscaremos pela `ListaTransacoesaActivity` com o comando "Ctrl + N".

Dentro dessa *activity*, como podemos fazer com que essas transações seja somadas? A princípio, precisamos passar por todas as transações e pegar as transações de *Receita*, e então, fazemos uma soma. Vamos começar com esse processo?

A pergunta agora é a seguinte: **Como passaremos por todas as transações no Kotlin?**

Uma das técnicas que podemos utilizar é um recurso muito parecido como o do Java, que é a estrutura do *For Each*. Dentro dele, indicamos que queremos uma **transação** de *transações*, no caso.

```
class ListaTransacoesActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_lista_transacoes)

        val transacoes: List<Transacao> = transacoesDeExemplo()

        // For each no Java
        for(transacao : transacoes){

        }

        configuraLista(transacoes)
    }
}
```

Porém, há uma estrutura no Kotlin muito parecida com o *For Each* do Java, porém ela é conhecida como ***For Loop***.

## Qual é a diferença entre o For Each e o For Loop?

A diferença é muito simples: trocamos `:` pela *keyword* `in` !

```
    // For each no Kotlin
    for(transacao in transacoes){

    }
```

Isso significa que estamos pegando **uma transação** em **transações**. Uma outra parte bacana é que no *For Each* do Java, era necessário colocar o tipo. Mas no Kotlin isso não é necessário, pois já é implícito que a variável `transacao` é do tipo `Transacao` .

Bom, vamos primeiro resolver a parte do *Resumo*. Para conseguirmos filtrar a informação que será mostrada ao lado da *Receita*, precisamos verificar se de fato, a transação em questão é do tipo `Receita` .

Essa verificação será feita com um `If()` .

```
class ListaTransacoesActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_lista_transacoes)
```

```
        val transacoes: List<Transacao> = transacoesDeExemplo()

        for(transacao in transacoes){
            if(transacao.tipo == Tipo.RECEITA) {

            }
        }

        configuraLista(transacoes)
    }
}
```

Agora que já sabemos que uma `transacao` é uma *RECEITA*, podemos fazer o seguinte. Em nossa *App* Base, repare que está sendo somado as transações de "Receita". Então, podemos fazer o mesmo comportamento, o processo de soma!

Para conseguirmos somar todas as transações, precisaremos de uma *variável inicial*, ou seja, uma variável que comece com zero.

```
class ListaTransacoesActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_lista_transacoes)

        val transacoes: List<Transacao> = transacoesDeExemplo()

        var totalReceita = BigDecimal.ZERO
        for(transacao in transacoes){
            if(transacao.tipo == Tipo.RECEITA) {

            }
        }

        configuraLista(transacoes)
    }
}
```

Certo, depois disso podemos pegar o valor de `transacao` e somar com o valor `totalReceita` :

```
    val totalReceita = BigDecimal.ZERO
    for(transacao in transacoes){
        if(transacao.tipo == Tipo.RECEITA) {
            totalReceita.plus(transacao.valor)
        }
    }
```

Só que temos um detalhe a ser discutido. O `BigDecimal` é uma classe mutável, em outras palavras, o valor da receita não está sendo alterado pelo `plus()` .

O que acontece é que o `plus()` está pegando o total da receita e está somando com o valor da transação. E então, ele **devolve** um novo `BigDecimal` que tem a soma desses dois valores.

Precisamos então **reatribuir** esse valor para o `totalReceita`.

```
val totalReceita = BigDecimal.ZERO
for(transacao in transacoes){
    if(transacao.tipo == Tipo.RECEITA) {
        totalReceita = totalReceita.plus(transacao.valor)
    }
}
```

Mas aí, entramos no caso em que estamos alterando a variável `totalReceita`, sendo ela **val**. Agora então, podemos mudar para ***var***.

## O que é preciso fazer depois da soma?

Basicamente, só temos que mostrar o resultado da soma na view. Atualmente nenhum dado está sendo mostrado nela.

Como podemos colocar essa soma na view? Da mesma maneira, podemos utilizar o componente `resumo_card_receita` e chamar a *property* `text`, e depois atribuir essa informação ao `totalReceita`.

```
var totalReceita = BigDecimal.ZERO
for(transacao in transacoes){
    if(transacao.tipo == Tipo.RECEITA) {
        totalReceita = totalReceita.plus(transacao.valor)
    }
}
resumo_card_receita.text = totalReceita.toString()
```

Lembre-se que quando colocamos o `toString()`, estamos pegando o valor sem nenhum tipo de formatação. Então, podemos utilizar a função `formataParaBrasileiro()`.

```
var totalReceita = BigDecimal.ZERO
for(transacao in transacoes){
    if(transacao.tipo == Tipo.RECEITA) {
        totalReceita = totalReceita.plus(transacao.valor)
    }
}
resumo_card_receita.text = totalReceita.formataParaBrasileiro()
```

Vamos ver se funciona? Com o "Alt + Shift + F10" executamos a aplicação.

E olhe só, o valor de `600 reais` apareceu na view na frente de **Receita**. Vamos dar uma olhada em como ficou o código.

Se olharmos o `onCreate`, é difícil de entender que todo esse código esta adicionando uma receita no Resumo. Em outras palavras, faz sentido usar a técnica que vimos anteriormente, fazer a *refatoração* extraindo uma *função* do código.

Selecionando o código parcial acima, e com o "Ctrl + Alt + M", abrimos a janela *Extract Function* do Android Studio, onde informaremos que essa parte do código que foi selecionada, adiciona uma receita no Resumo.

Podemos chamá-la de `adicionaReceitaNoResumo(transacoes)`. E agora, ao olharmos novamente para o código, fica muito mais fácil de saber que o método `adicionaReceitaNoResumo(transacoes)` irá, de fato, adicionar uma receita ao Resumo.

Mas perceba que, nós só fizemos a parte da **adição da Receita**. E agora, começaremos o processo de **adicionar uma despesa** no próximo vídeo. Até mais!