06

Conclusão

Transcrição

Chegamos ao final da segunda parte do curso **Android com Kotlin**!

E se você chegou até aqui, te parabenizamos por tê-lo concluído. Mas o que acabamos aprendendo nesse curso?

A princípio, vimos que era necessário implementar a tela de resumo, que faz com que todas as transações que registramos, sejam computadas como por exemplo, somando as receitas e despesas, e depois aplicando o total.

Vimos que, para colocar essa informação, foi necessário criar a classe `ResumoView`, e a partir dela, pegamos todas as informações que precisávamos, como por exemplo uma lista, a `view` para usar o *synthetic*, e também utilizamos o `context` da *activity*, para poder também utilizar alguns recursos como pegar cores, entre outras coisas que vimos.

Além disso, vimos o `with` do Kotlin, que é justamente o recurso que permite com que façamos diversas chamadas de um objeto, sem ter que referenciá-lo novamente, como podemos ver a seguir:

```
private fun adicionaReceita() {
    val totalReceita = resumo.receita
    with(view.resumo_card_receita) {
        setTextColor(corReceita)
        text = totalReceita.formataParaBrasileiro()
    }
}
```

Nós aplicamos a cor, e não precisamos chamar o objeto novamente. Além disso, também conseguimos criar uma classe `Resumo`, para que ela pudesse calcular tudo para nós. Todos os cálculos feitos, além de serem modificados pelo `ResumoView`, estão sendo calculados por essa classe por meio de *properties*.

Estamos também utilizando a *expressão lambda* para filtrar e realizar cálculos de maneira mais objetiva e expressiva.

Nesse projeto, tivemos que colocar um comportamento um pouco diferente para o usuário. Agora, não temos mais aquela lista fixa. Agora, colocamos as despesas quanto as receitas de modo **dinâmico**.

Colocamos todo esse comportamento e dinamismo na classe `AdicionaTransacaoDialog`. Fizemos diversos processos para que chegássemos em um resultado final. Conseguimos utilizar alguns recursos do Kotlin durante esse curso, como utilizar Lambda, utilizar o `_` para representar um parâmetro que não está sendo usado, como também outros recursos bem legais, como o *try catch expression*.

Aprendemos também um padrão de projeto conhecido como **Delegate**. Esse padrão nos ajudou a chamar um código da *activity* dentro de um *Dialog*. A partir desse *design pattern*, aprendemos uma técnica que não deixa o código acoplado.

Foram diversas coisas que aprendemos nesse curso, e esperamos que você tenha gostado. Deixe um *feedback* bacana para que sempre possamos melhorar o conteúdo.

Até mais! :)