01

Realizando o processo de refatoração

Transcrição

Chegamos à mesma representação visual da app base! Ao desenvolvermos apps e softwares, precisamos nos atentar ao processo de refatoração do código. Significa que quando terminamos uma *feature*, é recomendado visualizar o que foi feito, verificando se há pontos a serem melhorados, alguma manutenção ou inserção de uma nova funcionalidade a ser realizada.

Faremos isso a partir deste ponto do curso! Para começarmos o processo de refatoração, acessaremos "br.com.alura.financask" e, em cada um dos pacotes, veremos o que foi feito com as classes e arquivos em Kotlin.

Começaremos pelo diretório "extension", com as extensões de `BigDecimal`, `Calendar` e `String`. Em `BigDecimal` fizemos a formatação do valor para o padrão da moeda brasileira, criamos o formatador e a variável `moedaFormatada`, que está devolvendo-a, sendo que é possível fazê-lo no momento do `replace`:

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

Esta é uma das maneiras de refatorarmos o código. É claro que deixar da maneira como estava anteriormente não está errado, inclusive há mais semântica para entender o que está sendo retornado. No entanto, já que é bem claro para nós que está sendo retornada `moedaFormatada`, nada mais justo do que a possibilidade de se retornar o `replace`, deixando o código mais objetivo.

Abriremos em seguida `CalendarExtension`, cujo código parece muito bem resolvido, não há nem a variável que informa que a data está formatada, pois retornamos com `format`. Vamos simplesmente deletar o comentário do arquivo.

Veremos em `StringExtension` um código bem resumido, em que utilizamos a *string* que é o próprio objeto, fazendo um *early return*. A única alteração que podemos fazer é indicar o significado de `0` pois, da maneira como está, apenas se indica que é um número mágico. Sabemos que se trata da primeira letra, mas não sabemos seu significado apenas olhando o código pela primeira vez.

Para dar maior significado ao valor, poderemos fazer com que ele seja uma variável local a partir do atalho "Ctrl + Alt + V", com que indicaremos que o `0` significa `primeiraLetra` ou `primeiroCaracter`, assim:

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

Deste modo, fica bem mais claro que pegaremos uma `substring` utilizando o `primeiroCaracter` até a quantidade limite `caracteres`. Vamos fechar todas as abas com "Alt + Shift + X", e voltar ao diretório "model", que contém "Tipo" e "Transacao".

Deletaremos os comentários de "Tipo" para deixar o código mais limpo. Não existe mais nada a ser acrescentado, então passaremos à "Transacao", em que também apagaremos o comentário desnecessário, que não agrega em nada no código.

As *properties* já estão muito bem definidas, pois pedimos para que o acesso a elas seja público, e fazemos com que elas não sejam alteradas (com `val`), já que isto não foi necessário em nenhum momento. E também definimos valores padrão para data e categoria, com `String = "Indefinida"`. Não colocamos sequer corpo nesta classe; tudo está bem resumido.

Nossos modelos também estão ok! Agora, acessaremos "ui > activity > ListaTransacoesActivity" e, em `onCreate`, há bastante código para ser interpretado, dificultando até sua leitura em um primeiro momento.

Sabemos que inicialmente pegamos transações de exemplo e depois configuramos a nossa lista, incluindo um *adapter* nela. Refatoraremos o código extraindo o `listOf`, que devolve transações de exemplo, para uma função que faz isto, por meio do atalho "Ctrl + Alt + M".

Seu nome será determinado como "transacoesDeExemplo", e devolveremos `transacoes . lista_transacoes_listview` configura nossa lista, em que incluímos um *adapter*, porém, para entendermos isto, é preciso ler toda a linha de código. Não está claro, de imediato, que é isto que está acontecendo.

Portanto, repetiremos o processo anterior usando o atalho "Ctrl + Alt + M" mantendo-se a linha selecionada, e nomearemos a função com "configuraLista":

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

    val transacoes = transacoesDeExemplo()

    configuraLista(transacoes)
}
```

O `transacoesDeExemplo` pode não ser muito claro ao informar o que está sendo retornado, então, ao lidarmos com este tipo de retorno de uma função, é necessário informaram que trata-se de uma lista de transações, por exemplo:

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

    val transacoes: List<Transacao> = transacoesDeExemplo()

    configuraLista(transacoes)
}
```

A ideia, ao trabalharmos com o que é implícito ou explícito, é tentarmos usar a forma implícita sempre que a atribuição é clara em relação ao que está sendo devolvido.

Feito isto, terminamos a *activity*. Vamos voltar ao nosso projeto, e o que ficou faltando é o *adapter*, que possui muito o que ser refatorado! Veremos isto tudo em seguida.