16

Implementando o próprio Adapter - Parte 1

Transcrição

Tendo implementada a nossa lista com `ArrayAdapter`, vamos fazer com que ela tenha o mesmo aspecto visual que vimos na app finalizada. Para personalizamos itens de uma `ListView`, vamos criar um *adapter* com a aparência desejada, e é isto que faremos agora, criando uma classe que o representará.

Acessaremos o projeto em "java > br.com.alura.financask.ui.activity", em que criaremos pacotes para representar todas as classes de *adapter* a serem criadas em nosso projeto.

Usaremos o atalho "Alt + Insert" e escreveremos "Package", e definiremos "adapter" como nome. Observem que o pacote criado está dentro de "ui.activity" sendo que, se analisarmos nosso projeto implementado em Java, temos "activity", "adapter", "dialog" em "ui", ou seja, o "adapter" não fica dentro de "activity".

Então, de que maneira conseguiremos colocar o "adapter" do Kotlin no mesmo nível de "ui"?

Existem diversas técnicas, sendo que uma delas é utilizar um recurso chamado "Move", por meio do atalho F6, ao que se abre uma janela pedindo a confirmação da ação, bastando informar a nova localização, "br.com.alura.financask.ui". Clicaremos em "Refactor" em seguida.

Assim, deixaremos "activity" e "adapter" no mesmo nível. Utilizaremos "Alt + Insert" novamente, e selecionaremos "Kotlin File/Class". Na nova janela, deixaremos "*Kind*" ("tipo") como "Class", e o nome será "ListaTransacoesAdapter".

O programa nos pergunta se queremos colocar o arquivo no Git, já que isto já vinha sendo feito. Marcaremos que não precisaremos que isso seja perguntado novamente, e clicaremos em "No", pois este processo será feito manualmente depois.

A partir da criação do *adapter*, como fazemos com que ele seja capaz de adaptar os itens de uma lista? Faremos uma extensão de uma classe, como visto nos cursos anteriores, usando dois pontos (`:`) e o `BaseAdapter`. Com "Enter", o *import* é feito automaticamente, e o código estará da seguinte forma:

```
package br.com.alura.financask.ui.adapter

import android.widget.BaseAdapter

class ListaTransacoesAdapter : BaseAdapter() {
}
```

Notaremos que o `BaseAdapter` precisará daquela inicialização padrão, sem nenhum argumento, pois não é necessário. Ao fazermos uma extensão do `BaseAdapter`, precisaremos também do `override` das funções abstratas a serem implementadas. Não é qualquer *adapter* que precisaremos implementar.

Para isto, pode-se usar alguns recursos do próprio Android Studio, como "Alt + Enter" e a opção "Implement members", para implementarmos estes membros, ou seja, as funções. Apertaremos "Enter", e tudo aquilo que será implementado é mostrado em uma nova janela.

Selecionaremos todos eles e clicaremos em "OK". Com o atalho "Ctrl + Shift + F12" estendemos a tela para melhor visualização do que foi feito: foram colocadas várias funções, que estão sendo sobrescritas. Começaremos, então, a

implementá-las.

Se repararmos melhor, há chamadas de funções com nome `TODO()`. O que isto significa? Basicamente, trata-se de uma função do Kotlin que, no momento de sua execução, resulta em um *crash* na aplicação, indicando que ele não foi feito, mas que precisará ser feito - por isto, "*to do*", ou "para fazer".

"*not implemented*" é uma mensagem padrão para este tipo de chamada de função, então, se colocarmos um `TODO()`, significa que ainda não implementamos a determinada função e, portanto, quando chegar naquele ponto, ele será quebrado.

Podemos testar para ver se é realmente isto que acontece. Na nossa *Activity*, em vez de colocarmos `arrayAdapter` em `lista_transacoes_listview.setAdapter(arrayAdapter)`, usaremos `ListaTransacoesAdapter()`, instanciando-a. Feito isto, executaremos nossa app com "Alt + Shift + F10".

No emulador, vê-se a mensagem "*Finanças has stopped. Open app again*". Ao acessarmos o Logcat, veremos que ocorreu um erro de exceção `kotlin.NotImplementedError: An operation is not implemented: not implemented`. Isto é, uma operação não foi implementada, justamente a mensagem `TODO()`. Se clicarmos no momento exato em que ocorre o erro, comprovamos isto.

Neste tipo de situação, no momento em que implementamos uma função que ainda não tiver sido implementada, aí sim tiramos o `TODO()`. Isto ajuda a entendermos se a função está realmente finalizada, pois isto será sinalizado.