13

Utilizando o synthetic para buscar as views

Transcrição

Após a criação da *Activity* e, agora que conseguimos rodar nossa aplicação chegando na tela inicial crua, vamos avançar em nosso projeto! Se repararmos, há uma lista de transações na app. Vamos focar em implementá-la.

Voltando à *Activity*, usaremos o atalho "Ctrl + N" para buscar uma classe, `ListaTransacoesActivity`, a qual abriremos e, com o cursor sobre `activity_lista_transacoes`, usaremos "Ctrl + B" para vasculhá-lo.

No caso do layout, então, veremos que há vários componentes, dentre os quais está a `ListView`, que representa a lista de transações de fato.

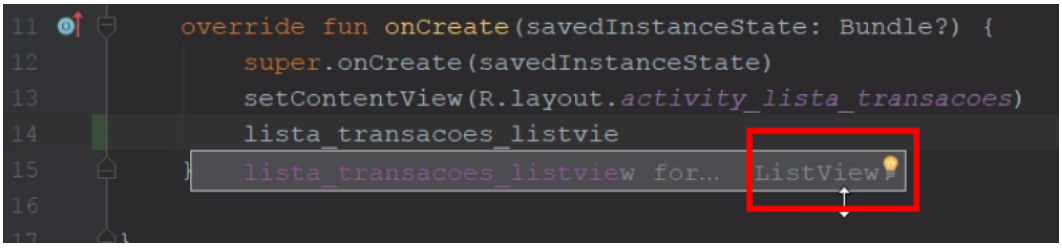
Vamos retornar à `ListaTransacoesActivity`, onde informaremos que queremos uma `ListView`. Para isto, uma opção válida seria acrescentar a linha `findViewById<>(R.id.lista_transacoes_listview)` após a linha de `setContentView`.

No Android com Java, não tínhamos a opção de *Generics*, nem como pré definir a *view* que queríamos, sendo necessário um *cast*, por exemplo. Aqui, já poderemos colocar `ListView`, assim:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_lista_transacoes)
    findViewById<ListView>(R.id.lista_transacoes_listview)
}
```

Esta é uma abordagem bem bacana do Kotlin, pois evitamos correr riscos. Além disso, estamos mantendo o uso de `findViewById`. Felizmente, há ainda outra opção, desenvolvida pela própria equipe da JetBrains: o plugin conhecido como ***Kotlin Android Extensions***.

Ele nos auxilia a fazer um `import` no *id* do componente, e seu uso está "liberado". Quando tentamos autocompletar, por exemplo, veremos que ele é a própria `ListView`:



O plugin, então, permite que peguemos os componentes apenas pelos seus *ids*. Inclusive, poderemos deletar a linha `import android.widget.ListView`, uma vez que ela não é mais necessária - isto pode ser feito pelo atalho "Ctrl + Alt + O".

Quando entramos neste tipo de abordagem, há diversos questionamentos sobre como isto é possível. Será que isso ocorre por padrão da linguagem? O que acontece realmente?

Para entendermos o que está acontecendo, no momento em que incluímos `lista_transacoes_listview`, percebam que o `import` correspondente aparece automaticamente (`import kotlinx.android.synthetic.main.activity_lista_transacoes.*`).

O `import` representa o `synthetic`, um módulo que permite com que, a partir de algum layout, consigamos fazer com que todos os seus componentes sejam extensíveis à nossa *Activity*. Isto não vem necessariamente por padrão nesta linguagem, mas acontece por termos configurado-o via Android `3.0`. Mas onde ele se localiza exatamente?

Clicaremos em `build.grandle(Module: app)` e veremos os plugins `kotlin-android` e `kotlin-android-extensions`, a partir do qual habilitamos o `synthetic`, que permite que chamemos nossas *views* pelo *id*.

Agora precisaremos incluir informações na `ListView` para verificar se tudo funciona corretamente. Neste primeiro momento, faremos uma lista básica, de *strings*, e colocaremos seu *adapter* como um `ArrayAdapter` para deixar o mais simples possível.

No Android, costumamos implementar um `ArrayList` por exemplo, e então voltamos a alguma lista, e por aí vai. No Kotlin, temos uma abordagem um tanto mais sucinta, com funções pré definidas para este tipo de estrutura, como é o caso da `listOf()`.

Ela permite que passemos objetos para dentro dela e, a partir do processamento deles, é devolvida uma lista. Vamos testar deixando o código assim:

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

    listOf("Comida - R$ 20,50", "Economia - R$ 100,00")

    lista_transacoes_listview
}
```

Acima, há uma lista contendo estas duas *Strings*, a qual precisa ser devolvida. Porém, como se cria uma variável em Kotlin? Podemos pedir para que ele devolve uma lista, deste modo:

```
val lista = listOf("Comida - R$ 20,50", "Economia - R$ 100,00")
```

No Kotlin, há duas maneiras de se declarar variáveis: o `val` e o `var`. Quando utilizamos `var`, indicamos que **a variável pode ser mutável**, isto é, ter seu valor alterado posteriormente.

Já quando usamos `val`, seu valor **não** pode mudar, então, se você acrescentar uma segunda linha com outra variável abaixo da primeira, ocorrerá um erro de compilação, pois a variável não poderá ser reassinada ou ter o valor reatribuído.

Portanto, no momento de declaração de variáveis, é preciso estar consciente acerca da possibilidade de mudança. Se não queremos mudar a variável, protegeremos com `val` e deixaremos o código íntegro. Caso contrário, usaremos `var`.

Apesar de termos devolvido a lista, não fizemos algo comum no Java, que é declarar o tipo de retorno desta variável. O que fica implícito é que a variável não possui um tipo, mas isto não é necessariamente verdade no Kotlin.

O que acontece é que no momento em que colocamos o retorno de uma variável, não precisaremos ser obrigados a acrescentar seu tipo. Isto é, ela possui tipo, tanto é que se colocarmos um valor não compatível com uma lista, o programa não deixará que isto aconteça.

De que forma informaremos que o código está devolvendo uma lista de *Strings*? Basta usarmos `List<String>`, de forma explícita:

```
val lista: List<String> = listOf("Comida - R$ 20,50", "Economia - R$ 100,00")
```

Se tentarmos colocar números nesta lista, por exemplo, veremos que não será possível, resultando em erro por conta do `<String>`. Este tipo determinado será mantido durante o código todo e, por mais que esta tipagem esteja implícita, o valor não é alterado durante o código, mantendo-se íntegro.

Já que queremos de fato uma lista de *Strings* e estamos devolvendo uma lista de mesmo tipo, não é necessário deixar isto explícito. E já que não pretendemos modificar a lista, a boa prática é que deixemos `val`.

Agora precisaremos colocar o `ArrayAdapter` mas, como vimos nos cursos anteriores, é necessário fazermos uma instância disto antes. Pode-se usar `ArrayAdapter()` direto, sem o `new`, seguido da classe e da instância.

Neste momento, simplesmente mandaremos os parâmetros necessários, sendo que o primeiro é o contexto, o layout a ser utilizado (próprio do Android, apenas para fins de teste) e, por fim, nossa lista em si.

Alteraremos o nome `lista` com "Shift + F6" para melhorar seu entendimento, e para usarmos atribuições diretamente no nosso IDE, poderemos optar por alguns templates disponíveis no Android Studio, e por isto acrescentaremos `val` para usarmos uma variável do tipo leitura, cujo valor não será alterado.

Feito isto, ainda setaremos o *Adapter*:

```
val transacoes: List<String> = listOf("Comida - R$ 20,50", "Economia - R$ 100,00")

val arrayAdapter = ArrayAdapter(context: this, android.R.layout.simple_list_item_1, transacoes)

lista_transacoes_listview.setAdapter(arrayAdapter)
```

Com isto, conseguimos setar nosso `ArrayAdapter` em nossa lista!

Recapitulando o que fizemos: colocamos uma variável do tipo apenas leitura ou, do modo técnico, *Read Only*, que não muda de valor. Estaremos retornando a ela uma lista de *strings* representando as transações, criamos o `ArrayAdapter` com um layout bem simples, e colocamos o `Adapter` na `ListView`.

Vamos ver se tudo está funcionando?

Testaremos executando a aplicação no emulador. Observaremos que a lista foi colocada com sucesso, e que conseguimos fazer tudo aquilo que estávamos prevendo: usar `lista_transacoes_listview` a partir de seu *id* sem o `findViewById`, criar uma lista de *Strings* sem `ArrayList`, ou seja, de forma bem sucinta!

Também notamos que não precisaremos mais de `new` para fazermos uma instância. Basta chamarmos uma classe, informando o construtor que está sendo utilizado para ela.

A app não está tão bonita quanto o que foi mostrado no início do curso, no entanto veremos a seguir como fazer um `Adapter` que deixará a tela mais atraente.