08

Criando a primeira tela da App - Parte 2

Transcrição

De que forma é possível criarmos uma *Activity*, ou melhor, um arquivo ou classe, no Kotlin? Pode-se acessar "java > br.com.alura.financask", que é o pacote base, e criar uma *Activity* ali. Como vimos anteriormente, na linguagem Java ela foi criada em "java > br.com.alura.financas > ui > activity", portanto, em Kotlin, manteremos o padrão.

Para criar um pacote usaremos o atalho "Alt + Insert", escolheremos "*Package*" e definiremos como nome "ui.activity". Feito isso, criaremos o arquivo em Kotlin com "Alt + Insert" novamente, a partir do qual clicaremos em "Kotlin File/Class" ou simplesmente apertaremos "Enter".

Será solicitado o nome para o arquivo, digitaremos "ListaTransacoesActivity". Com "Enter", notaremos que foi criado um arquivo de extensão `.kt` no pacote previamente definido, incluindo apenas o formato padrão do Android Studio, um comentário indicando o momento de criação do mesmo. Vamos apagar isto, pois não será necessário.

Cadê nossa classe, nossa *Activity*? O que houve aqui?

Conforme comentado anteriormente, **a linguagem Kotlin é multiparadigma**, o que quer dizer que atende tanto o paradigma orientado a objetos e classes, quanto o paradigma funcional, que não exige o conhecimento de classes para que o código funcione.

Pensando neste detalhe, por padrão, o Kotlin não cria uma classe com o mesmo nome do arquivo, sendo necessário indicarmos isso, e é por este motivo que quando utilizamos "Alt + Insert" e criamos um arquivo existe a opção de alterarmos seu tipo ("*File*", "*Class*", "*Interface*", e por aí vai), algo que também é indicado pelo "*File/Class*".

Solicitaremos ao Kotlin para que se crie uma classe para verificarmos o que acontece. Deletaremos o arquivo recém criado e usaremos o atalho "Alt + Insert", e selecionaremos "Kotlin File/Class", nomeando este com "ListaTransacoesActivity" de novo. Na opção que aparece logo abaixo na janela, colocaremos como tipo "Class", e não "File". Apertaremos "OK" em seguida.

Removeremos o comentário novamente, e o que veremos é o seguinte:

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

class ListaTransacoesActivity {
}
```

Agora que temos uma classe, que no caso queremos que seja uma *Activity*, o que faremos para que ela seja configurada como nos projetos em Java? Basicamente, é necessário informá-la de que ela precisa herdar de uma *Activity* do Android, disponível, acrescentando `extends` após `ListaTransacoesActivity` ...

Mas o programa não entende o que é `extends`, isto é, não sabe qual é seu significado. Como poderemos herdar de alguém a partir de uma classe?

No Kotlin, basta utilizarmos o operador `:` (dois pontos), após o qual indicamos a fonte da herança, no caso, `AppCompatActivity`, responsável por dar suporte às APIs mínimas do nosso projeto. Teremos, então:

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

import android.support.v7.app.AppCompatActivity

class ListaTransacoesActivity : AppCompatActivity {
}
```

Porém, o programa indica que precisaremos **inicializar o construtor padrão** a ser utilizado no momento desta herança, algo que é exigido quando se usa Kotlin. Já que não passaremos nenhum argumento para esta classe, poderemos utilizar o construtor padrão sem argumento:

```
class ListaTransacoesActivity : AppCompatActivity() {
}
```

Caso quiséssemos usar o construtor enviando algum parâmetro, isto poderia ser feito sem problemas. Significa que, agora, a `ListaTransacoesActivity` possui todos os comportamentos e membros herdados de `AppCompatActivity`.

Neste momento, usaremos o `onCreate` para acrescentar nossa tela! Escolheremos um dentre os vários disponíveis, com menos opções em relação a parâmetros, já que não os utilizaremos por ora. Com "Enter", o programa completa o código incluindo o membro da *Activity*:

```
class ListaTransacoesActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
    }

}
```

Percebam que há algumas diferenças em relação ao Java em que, para colocarmos um `override`, por exemplo, existe uma notação, e um retorno explícito das funções, que não aparece no Kotlin. Vamos tentar entender melhor o que houve quando este membro entrou em nossa classe.

O `override` é bem similar e equivalente ao que temos na notação do Java. A única diferença é que **no Kotlin seu uso é obrigatório** - isto é, ao fazermos uma sobrescrita, somos obrigados a manter o `override`.

Em todas as versões do Java, quando criávamos algum bloco de código em alguma estrutura, chamávamos de **método**. No caso, `onCreate` seria método da *Activity*. No Kotlin, a abordagem é outra: o `fun`, como podemos ver, reflete em um conceito denominado "**funções**", que de forma prática age como se fosse um método da nossa classe.

Isto tem a ver com o fato da linguagem ser multiparadigma e, por atender às duas perspectivas, a ideia é que usemos funções. Toda vez que formos usar algum código do Java como "método", estaremos usando uma função.

Você pode se perguntar qual a diferença real disto, já que estamos apenas no campo da nomenclatura. Inicialmente, é isso mesmo, e quando vamos fazer a chamada de uma função, usaremos `fun`. Ou seja, não colocamos simplesmente o nome da função e seu retorno, devemos colocar explicitamente que trata-se de uma função.

Notem que não há nenhum retorno no código, nenhum `void` ou `string`, comuns no Java. **Quando não retornamos nada no Kotlin, há um retorno padrão** chamado ***Unit***. E onde ele se localiza quando não fazemos nada?

```
class ListaTransacoesActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) : Unit {
        super.onCreate(savedInstanceState)
    }

}
```

Por debaixo dos panos, quando não declaramos nenhum tipo de retorno, teremos a mesma chamada de retorno da função, com os dois pontos. O `Unit` é uma classe equivalente ao famoso `void` do Java. Este tipo de retorno no Kotlin corresponde ao uso de um `void` e um método no Java, só que no caso das funções, o retorno sempre será `Unit`.

Não tem problema deixarmos assim, e o próprio Kotlin já retorna um `Unit` por padrão, sendo desnecessário colocá-lo manualmente. Isto é, quando tivermos uma função e não queremos retornar nada, não precisaremos indicar explicitamente que o retorno é `Unit`.

Vamos ver o que ocorre com nossos parâmetros. No Java, colocamos primeiro o tipo (neste caso o `Bundle`) e depois o nome. Aqui, usaremos outra notação, parecida com a Pascal, com o nome da variável primeiro, dois pontos (`:`) e seu tipo.

A maneira como o ponto de interrogação está impactando no código não será vista neste primeiro momento, pois não utilizaremos isso agora.

Vamos dar continuidade ao `onCreate`: há `super`, como já fazemos com o Java, chamando o `super` da nossa *Activity* que estamos herdando. Como colocamos um layout na *Activity*? Usamos o `setContentView` passando `R.layout` e importando-o.

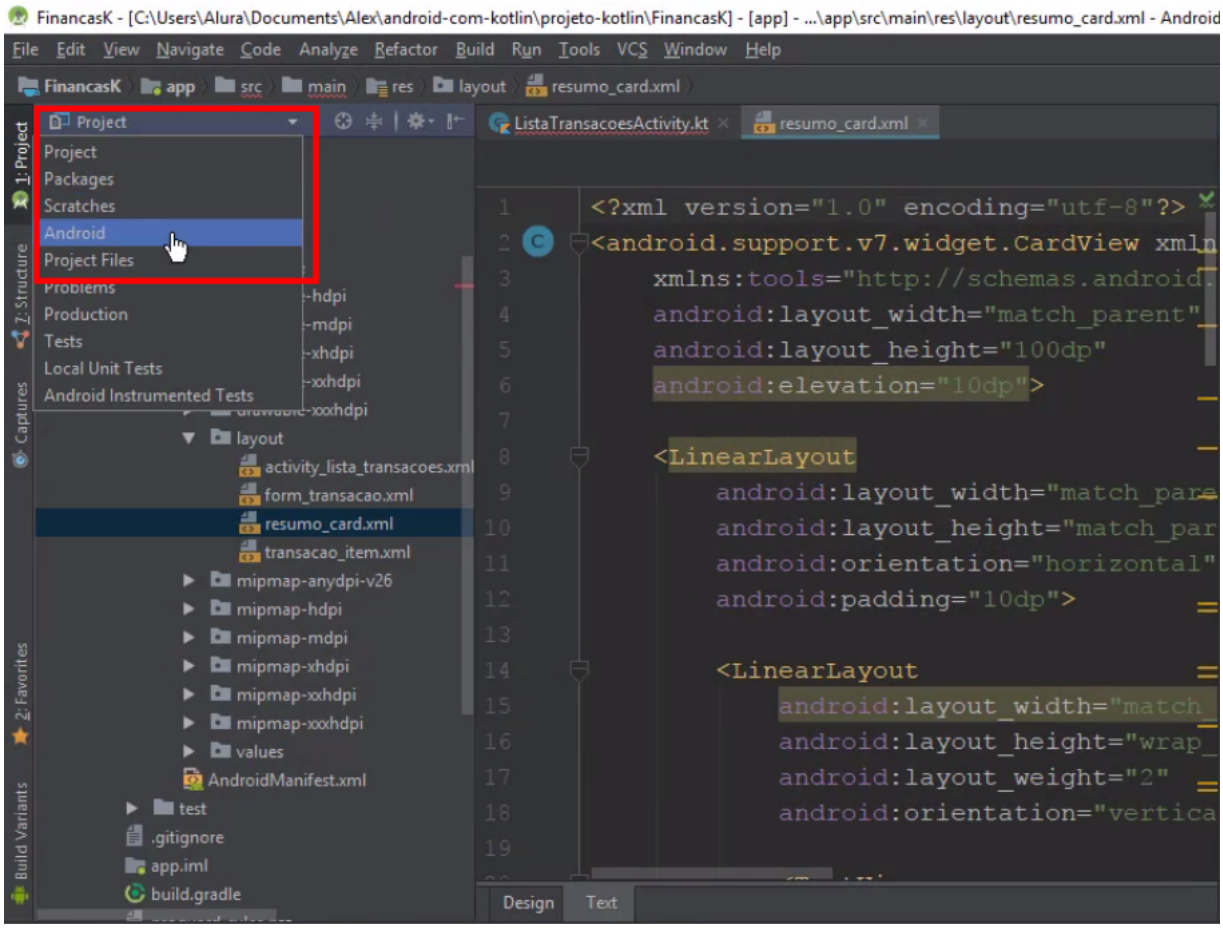
No entanto, nenhum layout foi criado ainda, e esta nem é a ideia do curso! Usaremos os layouts disponíveis no projeto feito em Java. Poderíamos simplesmente copiá-los diretamente do projeto Java, mas esta perspectiva do Android Studio não permite a cópia dos recursos.

Iremos, então, acessar "Project > financas > app > src > main" no menu lateral, clicar em "res" e copiar usando o atalho "Ctrl + C", ou clicar com o lado direito do mouse em "Copy". A partir daí, voltaremos ao "FinancasK" e, com "Ctrl + V", colaremos em "app".

Abre-se uma janela solicitando novo nome para o diretório, ao que clicaremos em "OK". Ocorrerá um erro, porque ele já existe e, nesta perspectiva Android, não é possível colar. Mudaremos-na para "Project" e colaremos, desta vez em "FinancasK > app > src > main".

Percebam, portanto, que a outra perspectiva tende a proteger nosso código, mas neste caso não é o que queremos, e sim realmente sobrescrever e mudar o que temos atualmente. Na janela de confirmação da ação, selecionaremos "Overwrite for all", para que tudo seja alterado, pois queremos utilizar exatamente o mesmo recurso existente em Java.

Voltaremos as perspectivas de ambos os projetos, em Java e em Kotlin, para "Android":



Tendo os *resources*, ou seja, todos os recursos necessários, veremos como implementá-los na tela da aplicação. Se digitarmos `activity` após `R.layout.` , veremos que aparece como sugestão `activity_lista_transacoes` , justamente o layout que faz a tela para nós.

Mantendo o cursor sobre ele e apertando "Ctrl + B", verificaremos o layout indo diretamente à ele - o que também pode ser feito acessando-se "app > java > res > layout > activity_lista_transacoes.xml".

No entanto, nada é mostrado no *Preview*! Por que isto ocorre?

Alguns componentes utilizados vêm de certas dependências do nosso projeto em Java, o qual também precisaremos incluir no projeto em Kotlin. Abrindo o projeto em Java, no arquivo `build.gradle` , existem dependências relacionadas à visualização.

Basta copiá-las, voltar ao "FinancasK", acessar o `build.gradle (Module: app)` e colar no lugar de `implementation 'com.android.support:appcompat-v7:26.1.0'` , para podermos usar a mesma versão utilizada nos outros, garantindo a compatibilidade. Obteremos:

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:26.0.1'
    implementation 'com.android.support:design:26.0.1'
    implementation 'com.android.support:cardview-v7:26.0.1'
    implementation 'com.github.clans:fab:1.6.4'
    // ...
}
```

Clicaremos em "Sync Now", que aparece no topo da parte do código, e esperaremos o Gradle realizar todo o processo. Feito isto, ao voltarmos ao *preview* do layout, veremos que ele já aparece corretamente, e não precisaremos mais nos preocupar com isso, pois iremos reutilizar o que está pronto.

Conseguimos criar nossa *Activity*, entender o que significa `override`, e setar o layout. Para executarmos a aplicação e mostrar a tela inicial, informaremos o "manifests" de que agora temos uma *Activity*, pois a princípio não solicitamos que ela fosse criada, sendo necessário fazermos isto manualmente também.

Clicando em "manifests" no menu lateral, deixaremos o código assim:

```
<application
    //...>
    <activity android:name=".ui.activity.ListaTransacoesActivity">
        <intent-filter>
            <category android:name="android.intent.category.LAUNCHER" />
            <action android:name="android.intent.action.MAIN" />
        </intent-filter>
    </activity>
</application>
```

Quando digitamos `activity`, sugere-se um nome, o que já criamos antes. O `<intent-filter>` serve para incluirmos a categoria, e também sua ação. Assim, conseguimos fazer com que a aplicação rode normalmente. Vamos executá-la para confirmar isso?

Pode-se clicar no ícone de "*Play*", o triângulo verde virado para o lado, ou utilizar o atalho "Alt + Shift + F10". Selecionaremos como modo "app", apertaremos "Enter" novamente no emulador, e o programa, a partir daí, faz o processo de `build` para verificar e rodar nossa app.

Após aguardarmos um tempinho, nossa tela inicial está pronta! É uma tela crua, sem muitas informações, e quando clicamos no botão referente a "Adiciona receita", nada acontece. Durante o decorrer do curso, veremos como melhorar isto. Até já!