Android com Kotlin

Flávio Augusto de Freitas

Lista de Compras

Neste projeto vamos implementar uma lista de compras! Você provavelmente já passou pela situação de ir ao supermercado para fazer compras e teve que levar uma listinha para se lembrar de quais produtos você precisava comprar. A ideia desse projeto é criar um aplicativo para criar uma lista de compras que você poderia usar em um supermercado ou em qualquer outro lugar.

Para implementação desse projeto, vamos trabalhar com um dos principais objetos no Android, que são as listas. Se você reparar nos aplicativos que você utiliza no dia a dia, você perceberá que a grande maioria utiliza listas em algum momento, por exemplo: os aplicativos de mensagem usam uma lista para mostrar seus contatos, uma rede social usa uma lista para mostrar as últimas postagens, o aplicativo de e-mail usa uma lista para mostrar seus e-mails. Se você pensar bem, quase todos os aplicativos que você utiliza no dia a dia utilizam listas para mostrar informações ao usuário, então é superimportante saber trabalhar com essa estrutura no Android.

Funcionalidades do app

Vamos começar a pensar no funcionamento do nosso App.

Começaremos pelo layout da tela: que informações o usuário precisará informar para o aplicativo? E que informações o aplicativo deve mostrar na tela?

Como estamos pensando em uma lista de compras, seria interessante o usuário poder inserir o nome do produto que deseja adicionar à lista. Uma coisa já sabemos, precisaremos de uma caixa de texto no nosso aplicativo para que o usuário possa inserir o nome do produto. Se esse produto será inserido em uma lista posteriormente, então precisaremos também de um botão para o usuário clicar e inserir o item digitado na lista! Por fim, o aplicativo deve mostrar uma lista com todos os produtos que o usuário inseriu, então devemos ter na tela um componente de lista! Agora que já temos em mente como nosso aplicativo deve funcionar, vamos começar nosso projeto. Abra o Android Studio e crie um projeto com as seguintes características:

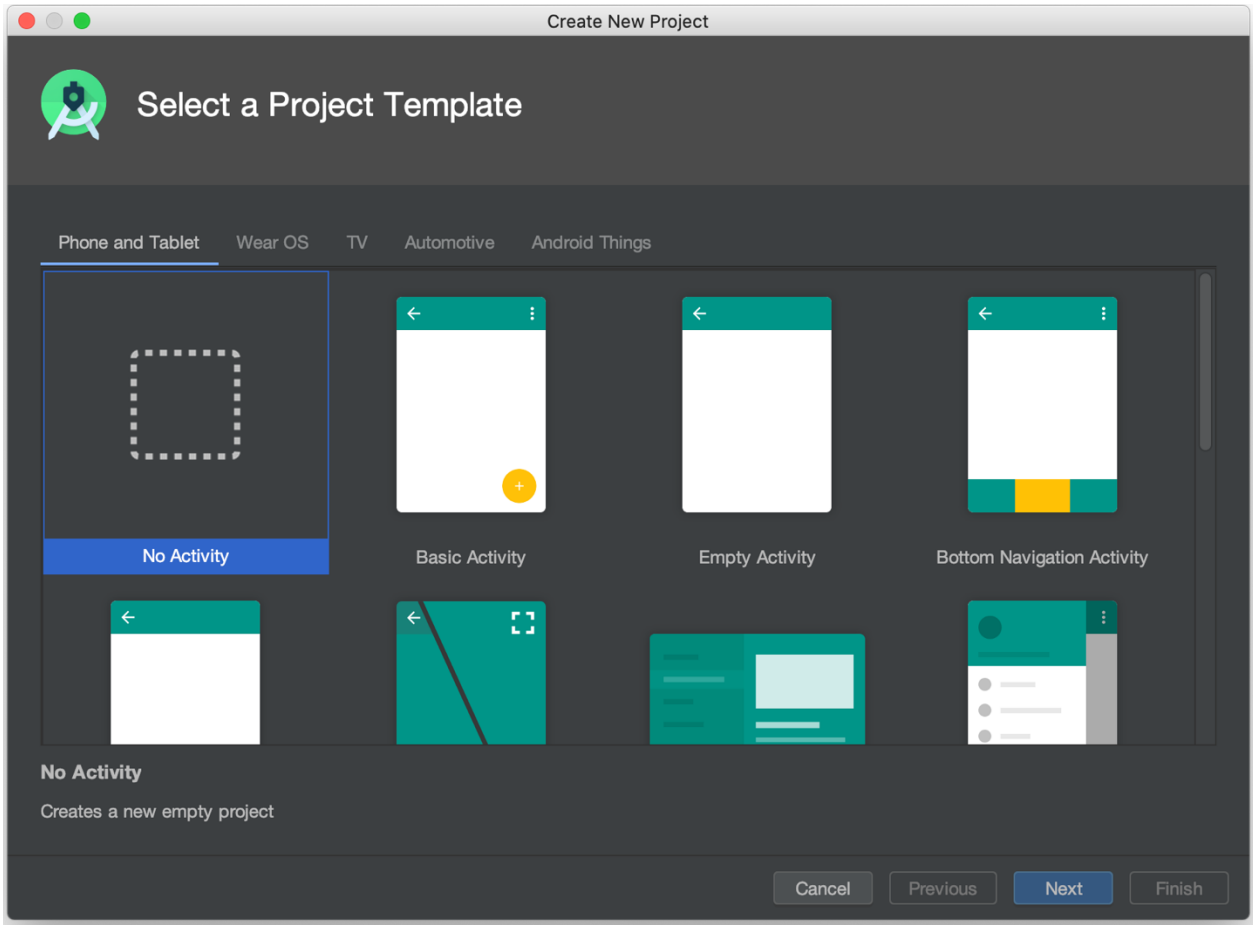

Figura 1: Project template.

- Nome do aplicativo: Lista de compras
- Nome do pacote: br.com.androidkotlin
- API Mínima: 21 - Android 5.0
- Modelo de Activity: Add No Activity

Não se esqueça de escolher a linguagem Kotlin.

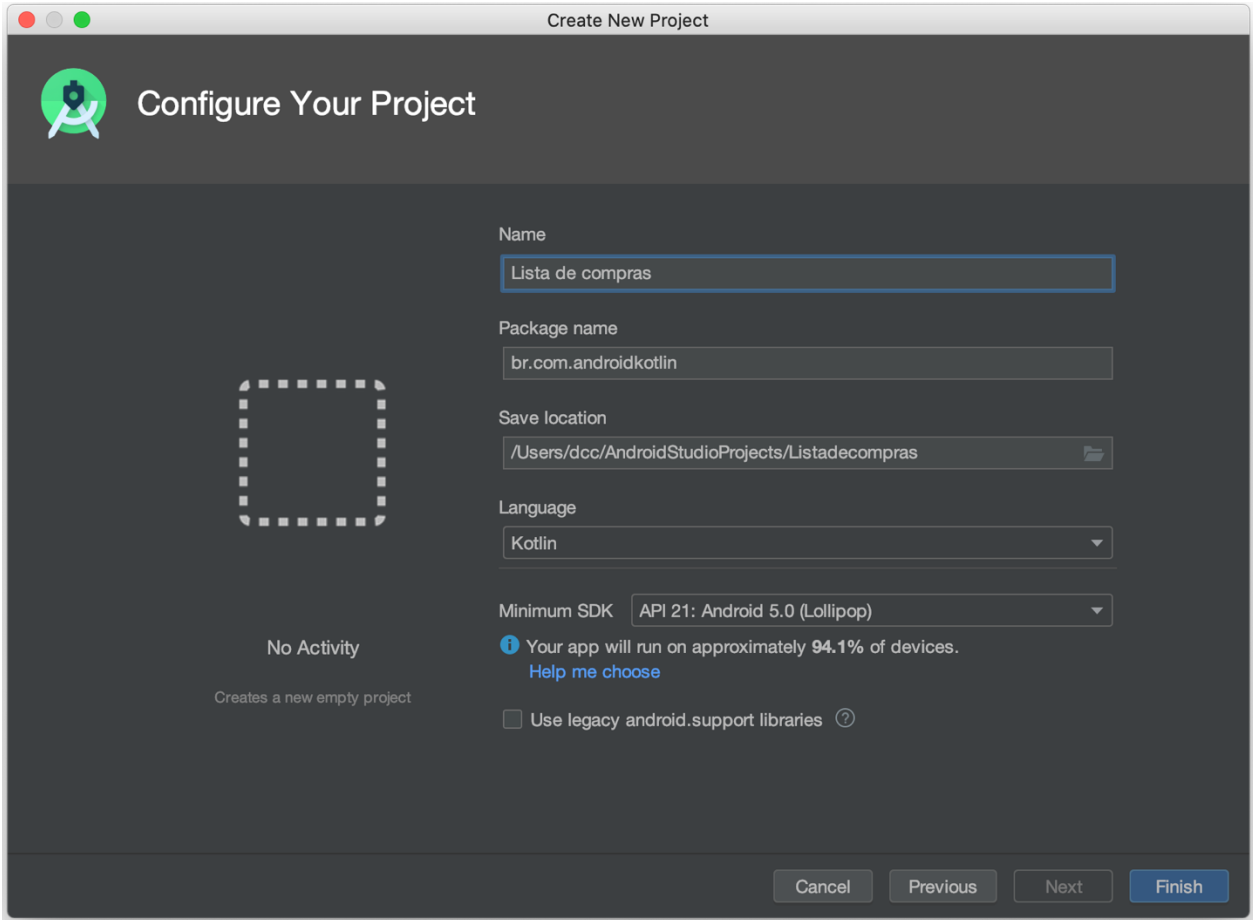

Figura 2: Configuração do projeto.

Após a criação do projeto, precisamos criar a Activity principal do nosso App. Diferente do projeto do capítulo passado em que criamos a Activity do zero (criando seu arquivo xml, depois o arquivo kotlin e, por fim, declarando-a no arquivo Android Manifest), desta vez utilizaremos o menu de criação de Activity da IDE. Desta forma, a criação da tela como um todo é agilizada uma vez que a IDE cria os arquivos necessários e já registra a Activity no manifesto do aplicativo!

Em comparação com a forma anterior, há diversas vantagens de se criar todas as Activities dessa maneira, primeiro porque o processo fica agilizado e o programador ganha tempo, e segundo porque nos dá a segurança de que o programador não vai se esquecer de nenhum passo. No entanto, é importante conhecer bem esse processo porque se houver algum problema o programador saberá onde procurar.

Vamos então criar nossa Activity. Para isso, selecione o item app no navegador da estrutura do projeto e clique no menu File > New > Activity > Empty Activity.

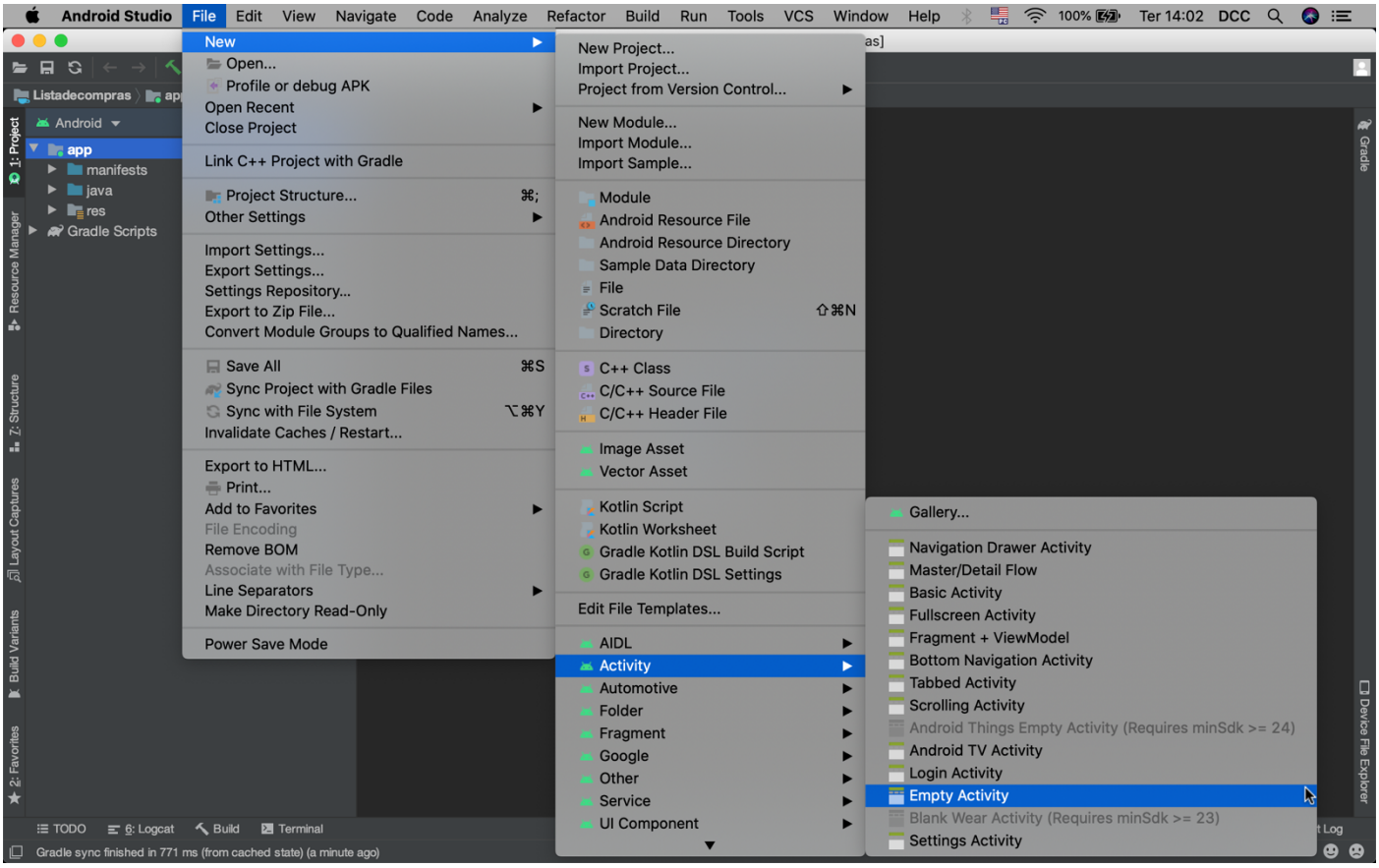

Figura 3: Criar Activity.

Na próxima tela, mantenha o nome MainActivity e activity_main que a IDE sugere, marque a opção LauncherActivity (atividade de inicialização), que indica que queremos tornar essa tela como principal no nosso aplicativo. Sendo assim, a IDE vai definir as configurações necessárias no arquivo de manifesto do aplicativo.

Deixe marcada a opção Generate Layout File (gerar arquivo de layout). Essa opção fará com que a IDE já crie para nós um arquivo XML de layout.

Se houver, deixe marcada também a opção Backwards Compatibility (AppCompat) (compatibilidade retroativa), que criará uma Activity do tipo AppCompatActivity, e não uma Activity. Um pouco mais adiante falarei sobre as diferenças entre a classe Activity e a AppCompatActivity.

Em Package name (nome de pacote) mantenha com br.com.androidkotlin. Aqui você poderia escolher um outro pacote para esse arquivo se seu projeto tiver mais de um pacote.

Em Source Language (linguagem do código-fonte) escolha Kotlin. Aqui ainda temos a opção de criar arquivos em Java e, como o Kotlin e o Java têm total interoperabilidade, conseguiríamos trabalhar com as duas linguagens no mesmo projeto.

E por fim, clique em Finish (finalizar).

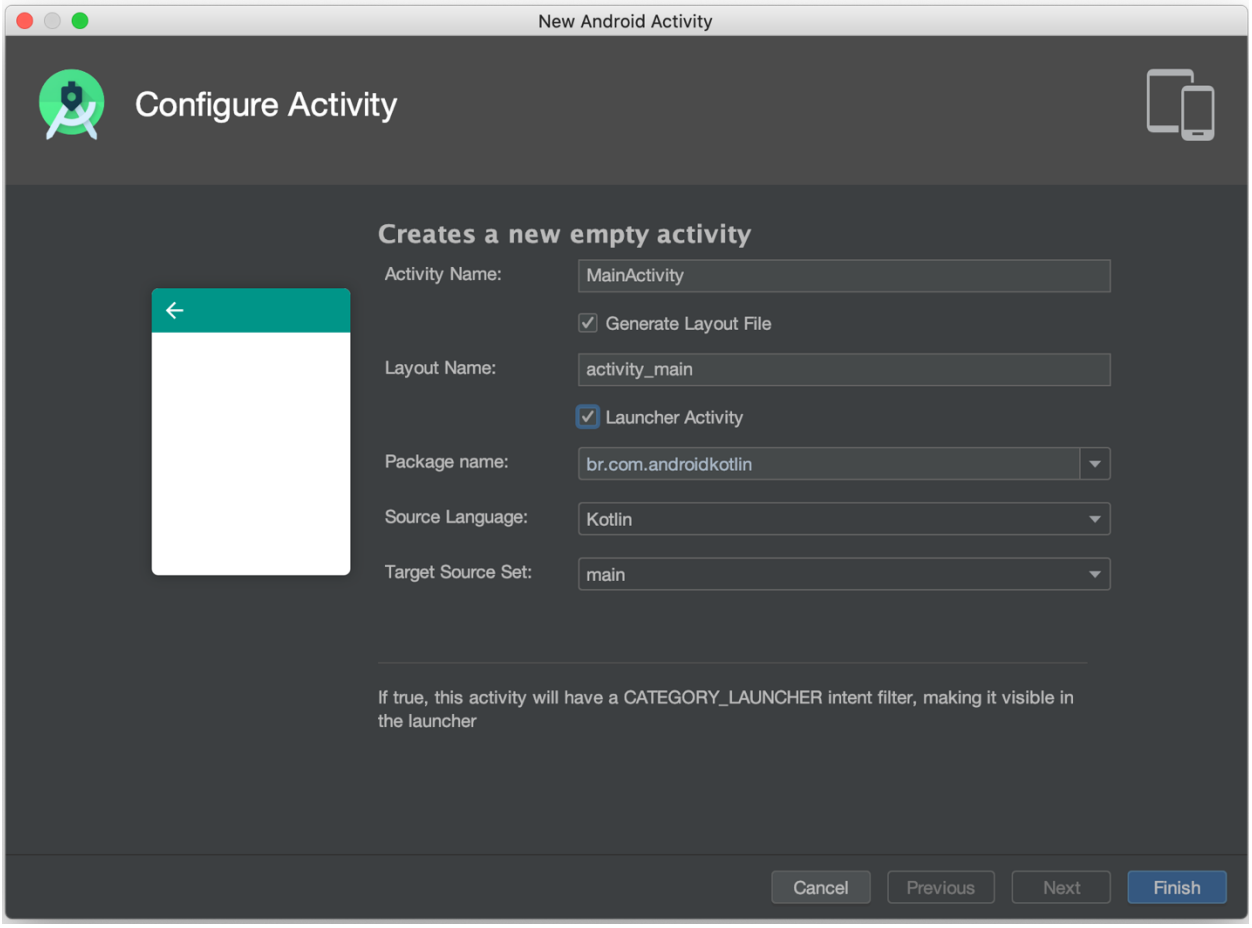

Figura 4: Configuração da Activity.

> *Vamos manter os nomes MainActivity e activity_main porque essa será nossa tela principal, então o nome juntando Main (principal) e Activity é bom, pois facilita a localização do arquivo posteriormente, logo a sugestão da IDE é bem-vinda.*

A opção Empty Activity (atividade vazia) cria todo o esqueleto de uma tela vazia para trabalharmos: o arquivo Kotlinc om o nome MainActivity, o arquivo de layout com o nome activity_main e também já registra a Activity no manifesto do Android!

Com isso, já temos os arquivos necessários para começar a trabalhar!

Criando o layout

Agora vamos trabalhar no layout do nosso aplicativo. Para criá-lo, vamos usar os seguintes componentes visuais:

- 1 EditText (caixa de edição de texto) para o usuário digitar o nome do produto;
- 1 Button (botão) para chamar a ação de inserir o produto na lista;
- 1 ListView (visualização em lista) para mostrar todos os produtos cadastrados.

Desses componentes, o único que não abordamos ainda é a ListView. Vamos então entender um pouco mais dela para podermos usá-la. Uma ListView é o componente visual do Android que exibe uma lista no

aplicativo. Essa lista pode exibir um texto simples ou pode ser totalmente personalizada pelo programador, isto é, podemos exibir imagens, textos e qualquer informação em uma lista.

A definição da ListView no arquivo é muito simples, basta adicionar a tag <ListView />, definir a altura, largura e o id. Veja um exemplo:

```
<ListView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/list_view"/>
```

A definição dos itens dessa lista e também a definição de como ficará o layout de cada item será feita no código em Kotlin, isso porque precisaremos de um objeto para gerenciar os itens da lista, chamado de Adapter (adaptador), veremos como criar um adaptador para a lista um pouco mais adiante.

Agora que temos uma noção de todos os componentes visuais que usaremos, vamos começar a montar o layout. Abra o arquivo activity_main localizado na pasta res > layout. Neste arquivo, vamos criar um layout linear com orientação vertical para que os componentes fiquem um embaixo do outro. Também já incluirei a propriedade padding com valor de 16dp para que os componentes não fiquem grudados nas laterais da tela.

Clique no botão de Split para ver o código e a tela do dispositivo móvel enquanto edita o código.

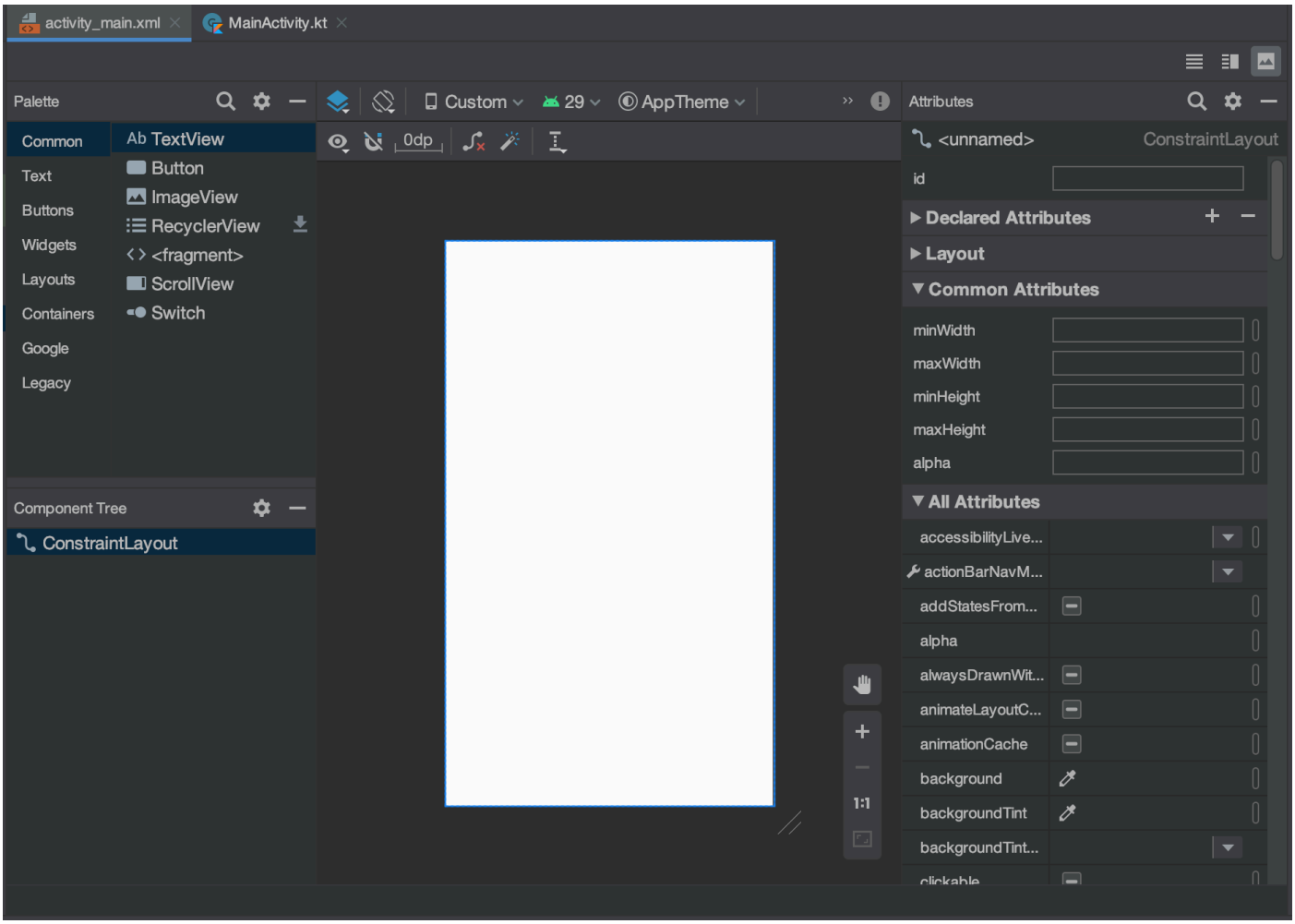

Figura 5: O botão de Split é o botão do meio em cima, à direita.

O código ficará assim:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="br.com.androidkotlin.MainActivity"
  android:orientation="vertical"
  android:padding="16dp">
</LinearLayout>
```

Em seguida, vamos montar a tela dentro desse layout linear. Criaremos um componente de EditText, em seguida, um Button e, por último, uma ListView. O código ficará assim:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="br.com.androidkotlin.MainActivity"
  android:orientation="vertical"
  android:padding="16dp">

<EditText
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:id="@+id/txt_produto"
  android:hint="Nome do produto"
  />

<Button
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="inserir"
  android:id="@+id/btn_inserir"
  />

<ListView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:id="@+id/list_view_produtos"></ListView>
</LinearLayout>
```

Podemos executar o aplicativo para ver como está ficando.

Figura 6: Execução do aplicativo.

Programando o aplicativo

Agora vamos partir para a programação do aplicativo! Abra o arquivo MainActivity localizado na pasta java. Você encontrará uma parte do código da Activity já pronta, que é o mínimo que uma Activity precisa para funcionar, então a IDE já nos traz pronta uma classe com a herança implementada: classMainActivity: AppCompatActivity() {... }. Também nos traz o método onCreate implementado dentro da classe: override fun onCreate(savedInstanceState: Bundle?) {... }, e dentro do método a chamada para o conteúdo da tela:setContentView(R.layout.activity_main). Veja o código completo:

```
package br.com.androidkotlin.listadecompras

import android.os.Bundle
import android.support.v7.app.AppCompatActivity

class MainActivity: AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
  }
}
```

Repare que a classe MainActivity faz uma herança da classe AppCompatActivity e não da classe Activity como estávamos usando no capítulo anterior. A classe AppCompatActivity é uma classe derivada da classe Activity, ou seja, ela ainda possui todas as características de uma Activity, mas com algumas diferenças. A AppCompatActivity é uma classe de suporte que nos permite utilizar o componente ActionBar (barra de ações) mesmo em versões mais antigas do Android. É uma boa ideia utilizar o AppCompatActivity porque

garantimos que nosso App se comportará da mesma maneira em uma quantidade maior de aparelhos e versões de sistema operacional.

Esse código então basicamente cria a Activity: MainActivity: AppCompatActivity(), implementa o método de criação da tela: override fun onCreate(savedInstanceState:Bundle?) e define o layout utilizado pela Activity: setContentView(R.layout.activity_main).

Nossa tarefa agora é programar a lógica do nosso aplicativo. Não será uma lógica complexa, muito pelo contrário, nós simplesmente queremos que, após o usuário preencher o nome do produto e clicar no botão inserir, que este produto seja inserido na lista! Para isso, precisamos entender como as listas funcionam e como podemos inserir novos itens nelas!

A ideia geral de uma lista é bem simples, ela é um componente que exibirá dados em forma de lista. No entanto, as listas são dinâmicas, podendo exibir qualquer estrutura de dados pré-programada pelo desenvolvedor. No nosso caso, queremos que alista seja simplesmente uma exibição de um texto, o nome do produto que foi previamente inserido pelo usuário. Mas é possível exibir imagens, botões e qualquer componente dentro de uma lista.

Criando um Adaptador

Para a estrutura de ListView funcionar, precisamos de umobjeto que fará a ligação entre o código e a lista definida na tela, que é o Adapter (adaptador). O Adapter é responsável por gerenciar a criação da lista, a exibição dos itens e também por definir o layout que será utilizado nos itens da lista. Sua função é abstrair toda a criação da lista, sendo assim, o objeto ListView não precisa saber como seus itens internos estão sendo montados e nem precisa saber do layout que está sendo implementado, ela só precisa ter um adaptador, que fará todo o resto.

Esse modelo de funcionamento é muito interessante pois é por conta dos adaptadores que conseguimos personalizar qualquer lista, então se precisarmos de uma lista diferente não mexeremos no objeto de ListView, mas sim implementaremos um adaptador diferente. Pense em um adaptador do Android como um adaptador na vida real. Imagine que você tenha um computador com uma tomada de 3 pinos, mas você precisa ligá-lo em uma tomada de 2, como você resolveria isso? Você mudaria a tomada do seu computador, mudaria a rede elétrica ou utilizaria um adaptador de tomadas? Provavelmente, você utilizaria um adaptador. E se a rede elétrica fosse de 3 pinos, mas no padrão americano? Bastaria utilizar um adaptador diferente!

Para nossa lista precisaremos de um adaptador também. O Android tem uma classe chamada ArrayAdapter, que implementa um adaptador genérico, isto é, um adaptador multiuso. Para criar um ArrayAdapter precisaremos de algumas informações: primeiro, precisaremos definir qual informação esse adaptador vai representar, isto é, com qual tipo de dado esse adaptador trabalhará. Faremos isso inferindo o tipo do adaptador. Parece algo complicado, mas na prática é muito simples, veja o seguinte exemplo: ArrayAdapter<String>. Nesse código, estamos dizendo que esse adaptador representa o tipo String, por isso passamos String entre o sinal < e >. Desta forma, o adaptador sabe que internamente os dados com que ele vai trabalhar serão do tipo String. Seguindo essa lógica, podemos criar um adaptador de qualquer tipo. Por

exemplo, se eu criar uma classe nova chamada Pessoa eu posso criar um adaptador que trabalhe com esse tipo: ArrayAdapter<Pessoa>.

Para a nossa lista, vamos utilizar um adaptador de String porque a informação que ele vai representar é um texto simples, o nome do produto.

Para criação do objeto, precisamos ainda passar alguns parâmetros pelo construtor. O primeiro parâmetro obrigatório é o contexto, que nesse caso é a própria tela, a própria Activity, então passaremos this, que se refere à própria classe. Ainda precisamos passar qual o layout que vamos utilizar para a montagem de cada item da lista. Aqui podemos utilizar algum modelo que o Android já possui ou criar nosso próprio modelo baseado em um arquivo xml.

Por enquanto, vamos utilizar um modelo pronto do Android, mas um pouco mais adiante vamos personalizar nossa lista e criar um layout personalizado! Para acessar esse modelo de layout já disponível no Android, podemos utilizar o código:android.R.layout.simple_list_item_1. Este é um modelo simples para exibição de um único texto, exatamente o que precisamos no momento!

> *O Android possui diversos layouts já definidos que podemos utilizar. Para uma lista completa consulte a documentação oficial no endereço: https://developer.android.com/reference/android/R.layout.html.*

A declaração completa do adaptador é a seguinte:

```
val adapter = ArrayAdapter<String>(this, android.R.layout.simple_list_item_1)
```

Ainda precisamos saber como inserir e remover itens deste adaptador. Para isso, a classe ArrayAdapter implementa dois métodos: add e remove. O primeiro é usado para adicionar itensna lista e o segundo para remover, sendo que ambos recebem o parâmetro do item que você deseja remover ou adicionar. Veja um exemplo:

```
val item = "Feijão"
adapter.add(item)
```

E para remover um item existente:

```
adapter.remove(item)
```

Existem alguns casos em que você vai precisar criar um adaptador já com itens inseridos e trazer uma lista na tela já com informações. Imagine, por exemplo, que você tenha uma lista de contatos, que estão guardados em um banco de dados, e você deseja exibi-los em uma ListView. Para isso, você lerá o banco de dados para buscar os contatos e criará um adaptador já comesses contatos preenchidos, assim quando o usuário abrir a tela já aparecerá uma lista exibindo informações. Para esses casos, podemos passar um terceiro parâmetro na criação do adaptador, que deve ser uma lista com as informações. Veja o exemplo a seguir da criação de um Adapter sendo criado com uma lista de nomes de pessoas:

```
//lista com contatos
var contatos = mutableListOf("Mariana", "João", "Francisco", "Vitória")

//criação do adaptador passando a lista como o terceiro parâmetro
val produtosAdapter = ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
contatos )
```

Agora que já criamos nosso adaptador, falta ligá-lo na ListView. O processo é muito simples. A ListView possui um atributo chamado adapter, que é justamente para definição do adaptador da lista. Pensando no exemplo de que eu falei acima, é como se o atributo adapter da ListView fosse o plug datomada em que ligaremos o adaptador.

Veja como é feita a ligação do adaptador na lista:

```
list_view_produtos.adapter = produtosAdapter
```

Agora, o código completo do método onCreate ficará assim:

```
package br.com.androidkotlin.listadecompras

import android.os.Bundle
import android.support.v7.app.AppCompatActivity
import android.widget.ArrayAdapter

import kotlinx.android.synthetic.main.activity_main.*

class MainActivity: AppCompatActivity() {

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main) //Implementação do adaptador
    val produtosAdapter = ArrayAdapter<String>(this, android.R.layout.simple_list_item_1)
    //definindo o adaptador na lista
    list_view_produtos.adapter = produtosAdapter
  }
}
```

FindViewById, para quê?

Perceba que aqui eu não utilizei o método findViewById para acessar a list_view_produtos. Essa mágica pode ser feita graças ao plugin kotlin-android-extensions, que já vem configurado no projeto quando estamos trabalhando com o Kotlin. Você pode conferir se ele está implementado no arquivo build.gradle do módulo, que deve conter a seguinte linha logo abaixo da configuração do plugin do Kotlin:

```
apply plugin: 'kotlin-android-extensions'
```

Confira também o arquivo build.gradle do projeto. Na seção de dependencies deste arquivo deve ter a seguinte linha:

```
classpath "org.jetbrains.kotlin:kotlin-android-extensions:$kotlin_version"
```

Esse plugin faz um mapeamento dos componentes declarados no xml e seus respectivos Ids e nos dá acesso a eles como se fossem propriedades da Activity. E você deve ter percebido também um novo import: kotlinx.android.synthetic.main.activity_main.*. É através desse import que todas as propriedades serão mapeadas. Geralmente, a IDE fará esse import automaticamente. Com isso, não nos preocuparemos mais com métodos findViewById nas Activities.

Adicionando itens na lista

Agora que temos o adaptador criado e já devidamente definido na lista, precisamos agora tratar o clique do botão para adicionar novos itens na lista. De fato, não adicionamos os itens diretamente na ListView, como é o adaptador que gerencia os itens dentro de uma ListView vamos adicioná-los no adaptador e ele será responsável por notificar a lista de que existem novos itens a serem exibidos. No nosso caso, queremos que os itens sejam adicionados somente quando o usuário clica no botão e o item que será adicionado deverá ser o texto que o usuário digitou na caixa txt_produto. Vamos primeiro tratar o clique do botão, e para isso vamos implementar um listener (ouvinte) no botão. Já usamos este método no projeto do capítulo anterior, mas vamos relembrar seu funcionamento: Os listeners (ou métodos ouvintes) são utilizados quando queremos tratar alguma ação de algum componente. O exemplo mais clássico é do próprio botão, a ação mais comum em um botão é o clique, então conseguimos monitorá-lo por meio de um listener de click do botão!

> *Para conhecer mais sobre os eventos de entrada do Android, consulte a documentação oficial através do link: https://developer.android.com/guide/topics/ui/ui-events.html?hl=pt-br.*

Como queremos que o item seja adicionado somente quando o usuário clica no botão, devemos então monitorar este evento, e para isso podemos utilizar o método setOnClickListener do botão. Este método é acionado toda vez que ocorre uma ação de clique no botão. Veja como essa implementação pode ser feita:

```
btn_inserir.setOnClickListener {

}
```

Dentro das chaves { } é que colocaremos o código que será executado a cada vez que esse botão for clicado. Precisamos primeiro pegar o nome do produto que o usuário digitou na caixa de texto e precisaremos de uma variável para guardar esse texto. Podemos fazer assim:

```
val produto = txt_produto.text.toString()
```

Em seguida, usaremos o método add para inserir a variável produto no adaptador:

```
adapter.add(produto)
```

Inserimos a variável produto no adaptador porque ela guarda o nome do produto inserido pelo usuário! Veja como ficará o código completo do botão:

```
btn_inserir.setOnClickListener {
  val produto = txt_produto.text.toString()
  adapter.add(produto)
}
```

Agora temos todas as partes do aplicativo programada, fizemos o adaptador da lista, tratamos o clique do botão e adicionamos novos itens na lista! Vamos ver como ficará o código completo dessa Activity:

```
package br.com.androidkotlin.listadecompras

import android.os.Bundle
import android.support.v7.app.AppCompatActivity
import android.widget.ArrayAdapter
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity: AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    //Implementação do adaptador
    val produtosAdapter = ArrayAdapter<String>(this, android.R.layout.simple_list_item_1)

    //definindo o adaptador na lista
    list_view_produtos.adapter = produtosAdapter

    //definição do ouvinte do botão
    btn_inserir.setOnClickListener {

      //pegando o valor digitado pelo usuário
      val produto = txt_produto.text.toString()

      //envia o item para a lista
      produtosAdapter.add(produto)
    }
  }
}
```

Chegou a hora de testar nosso App e ver como ele ficou! Execute o aplicativo e veja o resultado final:

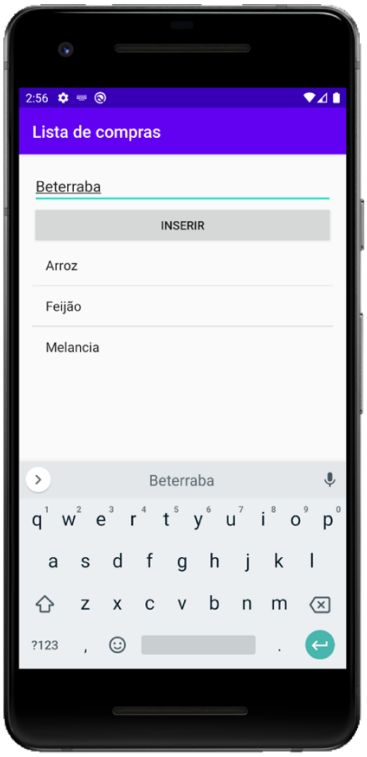

Figura 7: Aplicativo em execução.

Faça alguns testes e veja se a lista é atualizada assim que um novo item é inserido!

Melhoramentos

A base do nosso aplicativo já está funcionando, mas existem alguns pequenos ajustes que podemos fazer para tornar a experiência do usuário ainda melhor. A primeira melhoria que podemos fazer é verificar se a caixa não está vazia antes de inserir na lista. Pense na possibilidade de o usuário não digitar nada na caixinha e mesmo assim clicar no botão, nesse caso seria inserido um item vazio na lista. Para fazer essa verificação, podemos usar o método isNotEmpty (não está vazio), que é um método booleano, ou seja, seu retorno será ou true (verdadeiro) ou false (falso). Caso o método nos retorne um valor verdadeiro, significa que a caixa não está vazia, ou seja, o usuário digitou alguma coisa então podemos adicionar o item na lista; caso contrário, não adicionaremos nada na lista e o avisaremos de que deve preencher um valor. Vamos implementar a verificação no código do botão:

```
//verificando se o usuário digitou algum valor
if (produto.isNotEmpty()) {
  //envia o item para a lista
  produtosAdapter.add(produto)
}
```

Essa verificação evita que seja inserido um item vazio na lista, mas ainda não avisa ao usuário de que ele deve digitar um valor. Para isso, incluiremos o comando else ao if para tratar essa situação. Precisamos também avisar que ele precisa preencher um valor naquele campo, para isso vamos utilizar a propriedade error da EditText. Ela exibe uma mensagem de erro na caixa de texto, e será personalizada pelo valor passado a ela em formato de texto. No nosso caso, avisaremos ao usuário de que este campo precisa ser

preenchido com algum valor, então poderíamos fazer o seguinte código: txt_produto.error = "Preencha um valor". Desta forma, o usuário saberá qual ação ele deve tomar. Veja como fica a implementação no código:

```
if (produto.isNotEmpty()) {
  //enviado o item para a lista
  produtosAdapter.add(produto)
} else {
  txt_produto.error = "Preencha um valor"
}
```

O resultado final ficará assim:

Figura 8: Exibição de mensagem de erro. (Precisa clicar no ícone)

Outra melhoria que poderíamos fazer é limpar o campo após a inserção do produto. Do jeito que está, quando inserimos um novo item na lista, o campo com o nome do produto continua com o nome lá e, se o usuário quiser inserir outro produto, ele tem que apagar o valor anterior e inserir um novo valor. Esse comportamento pode ser evitado se, quando o item fosse inserido na lista, nós apagarmos o texto da caixinha através do código. Para isso, podemos usar o método clear (limpar), cuja função é simplesmente limpar um conteúdo de texto. Poderíamos também simplesmente definir uma String vazia a caixa de texto, no entanto, o método clear é um pouco mais elegante no código. Podemos usá-lo da seguinte maneira: txt_produto.text.clear() e pronto, ele vai limpar todo o conteúdo da caixa de texto deixando-a pronta para receber um novo valor. Veja o código completo:

```
if (produto.isNotEmpty()) {
  //enviado o item para a lista
  produtosAdapter.add(produto)

  //limpando a caixa de texto
  txt_produto.text.clear()
} else {
  txt_produto.error = "Preencha um valor"
}
```

Removendo itens da lista

Nosso App já está bem legal, o usuário consegue inserir vários itens na lista, já colocamos algumas restrições e melhorias para a experiência do usuário, mas ainda não pensamos em uma questão: e se o usuário quiser remover um item da lista? É possível?

Do jeito que o aplicativo está, atualmente não é possível remover um item da lista, somente adicionar novos itens! O que precisamos para implementar mais essa funcionalidade?

Precisamos de duas coisas: primeiro, temos que saber como se remove um item da lista, qual comando faz isso? Segundo, devemos pensar em que ação o usuário fará para remover um item da lista.

Remover um item da lista não é nada complicado, basta usarmos o método remove do adaptador passando como parâmetro o objeto que desejamos remover da lista. Lembre-se, não manipulamos os itens da lista diretamente, manipulamos o adaptador e ele refletirá as alterações na lista!

O próximo ponto que devemos pensar é na ação que o usuário executará para remover o item da lista, que pode ser a de longclick (clique longo). Essa ação é executada quando o usuário clica em um item da lista e segura por alguns segundos. Podemos definir que, quando o usuário clicar em um item da lista e segurar por alguns segundos, esse item será removido.

Para utilizar essa ação devemos implementar mais um método ouvinte, dessa vez na ListView. O método ouvinte que utilizaremos é o setOnItemLongClickListener, acionado quando um item da lista recebe um clique longo.

Na implementação desse método, é obrigatório que sejam declaradas 4 variáveis: adapterView, view, position e id. Elas já vão estar preenchidas toda vez que um clique longo ocorrer. Através delas, o programador saberá em qual posição da lista exatamente ocorreu o clique (position), qual o adaptador da lista (adapterView), qual o id da linha (id) e qual a visualizaçãodo item clicado (view). Veja a seguir a função de cada uma dessas variáveis:

- adapterView: a lista na qual o item foi clicado;
- view: a view do item que foi clicado;
- position: a posição desse item na lista;
- id: Um número que corresponde ao ID da linha que item se encontra

E, por fim, o método deve retornar um booleano indicando se o clique longo foi realizado ou não. Sua implementação no código será assim:

```
list_view_produtos.setOnItemLongClickListener{ adapterView: AdapterView<*>, view: View,
position: Int, id: Long ->
  //retorno indicando que o click foi realizado com sucesso
  return@setOnItemLongClickListener true

}
```

A notação -> faz parte da sintaxe do Kotlin e é usada em alguns contextos diferentes. Esta sintaxe é similar à de expressões lambda do Java. Aqui no Kotlin você encontrará essa notação em expressões lambdas e também na expressão when.

Em expressões when utilizamos a notação -> para separar a condição do bloco de resultado, veja um exemplo:

```
when (numero) {
  1 -> println("número é 1")
  2 -> println("número é 2")
  else -> println("número não é 1 nem 2")
}
```

Em expressões lambdas, utilizamos a notação -> para separar os parâmetros do corpo da função. O método setOnItemLongClickListener está recebendo uma expressão lambda que recebe os parâmetros: adapterView, view,position e id, e logo após os parâmetros utilizamos a notação-> para indicar o corpo da função.

Ainda precisamos resolver um problema para poder implementar essa lógica: para remover um item da lista vamos utilizar o método remove do adaptador, no entanto esse método precisa do objeto que ele vai remover, ou seja, como parâmetro, precisamos passar exatamente o item da lista em que o usuário clicou!

Para resolver isso, podemos utilizar outro método do adaptador, o método getItem. Ele retorna um item específico da lista de acordo com a posição passada.

Por exemplo, se quiser retornar o primeiro item da lista, basta passar a posição 0: getItem(0) ; para o segundo item, a posição 1:getItem(1) e assim por diante. Então, precisaremos saber em qual item exato o usuário clicou, mas isso não será um problema, porque o método setOnItemLongClickListener já nos passa essa informação através da variável position!

Tendo essas informações em mãos fica fácil. Primeiro vamos acessar exatamente o item clicado: adapter.getItem(position). Vamos também guardar essa informação em uma variável pois precisaremos dela para utilizar no método remove, então o código ficaria: val item = adapter.getItem(i). Na sequência, chamaremos o método remove do adaptador: adapter.remove(item). Veja a implementação completa do código:

```
list_view_produtos.setOnItemLongClickListener{ adapterView: AdapterView<*>, view: View,
position: Int, id: Long ->

  //buscando o item clicado
  val item = produtosAdapter.getItem(position)

  //removendo o item clicado da lista
  produtosAdapter.remove(item)

  //retorno indicando que o click foi realizado com sucesso
  return@setOnItemLongClickListener true
true
}
```

Não se esqueça de que a definição desse método estará logo após a definição do método ouvinte do botão. Teste o aplicativo e veja se é possível adicionar e remover itens da lista!

Veja o código completo da MainActivity:

```
package br.com.androidkotlin.listadecompras

import android.os.Bundle
import android.view.View
import android.widget.AdapterView
import android.support.v7.app.AppCompatActivity
import android.widget.ArrayAdapter
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity: AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    //Implementação do adaptador
    val produtosAdapter = ArrayAdapter<String>(this, android.R.layout.simple_list_item_1)

    //definindo o adaptador na lista
    list_view_produtos.adapter = produtosAdapter

    //definição do ouvinte do botão
    btn_inserir.setOnClickListener {

      //pegando o valor digitado pelo usuário
      val produto = txt_produto.text.toString()
      if (produto.isNotEmpty()) {

        //envia o item para a lista
        produtosAdapter.add(produto)

        //limpando a caixa de texto
        txt_produto.text.clear()
      } else {
        txt_produto.error = "Preencha um valor"
      }
    }

    list_view_produtos.setOnItemLongClickListener {
      adapterView: AdapterView<*>, view: View, position: Int, id: Long ->

      //buscando o item clicado
      val item = produtosAdapter.getItem(position)

      //removendo o item clicado da lista
      produtosAdapter.remove(item)
```

```
      //retorno indicando que o click foi realizado com sucesso
      return@setOnItemLongClickListener true

    }
  }
}
```

Resumo

Neste capítulo, aprendemos o básico de como se trabalhar com listas. Esse conteúdo é muito importante porque a grande maioria dos aplicativos utiliza listas em algum momento. Por enquanto, essa lista de compras está bem simples, mas no próximo capítulo vamos fazer a lista de compras 2.0!

Vamos trabalhar ainda em cima desse mesmo projeto e vamos implementar modificações significativas. Primeiro, vamos personalizar o layout dos itens da lista e, em seguida, guardaremos os dados em um banco de dados. Da forma como está agora, os dados estão salvos somente na memória RAM, isso quer dizer que assim que o aplicativo for fechado todos os dados serão perdidos.

Para resolver isso, devemos começar a guardar as informações em um banco de dados. Espero você no próximo capítulo!