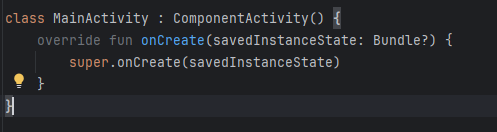
No projeto do Android Studio temos sempre uma “Main Activity”, essa Activity é normalmente responsável por apresentar uma tela de interface de usuário. Nesse caso estamos falando da “atividade principal”, apresentará a funcionalidade principal do nosso app. Sendo assim ele será o ponto de entrada principal para um app e tem como seu papel a inicialização e coordenação da interface do usuário e da lógica do app.



Nesse print temos a classe MainActivity que herda de ComponentAcitivity. Essa que por sua vez tem tudo que é necessário para que possamos desenvolver uma activity.

onCreate é um método do ComponentActivity e é chamado quando uma activity está sendo criada. O override fará começar a valer o código que estamos colocando dentro do método onCreate. O parâmetro “savedInstanceState” é uma instância de “Bundle”, este que é reponsável por conter dados previamente salvos que podemos ajudar a manter o estado da Activity após um reset, por exemplo.

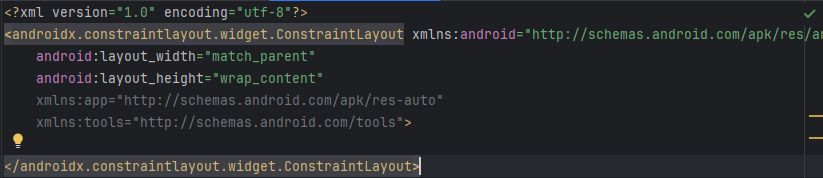
Depois chamamos a implementação do método onCreate(), garantindo que a funcionalidade padrão de criação de atividade seja feita.

Dentro do build.gradle colocamos uma dependência:



Ele será usado no arquivo que criaremos agora:

Criamos na pasta “res” uma pasta chamada “layout” e um arquivo de layout XML chamado produto\_item:



Essa tag define o início do layout ConstraintLayout, que faz parte da biblioteca androidx. Ele fornece uma maneira flexível de criar layouts complexos com hierarquia plana de visualização.

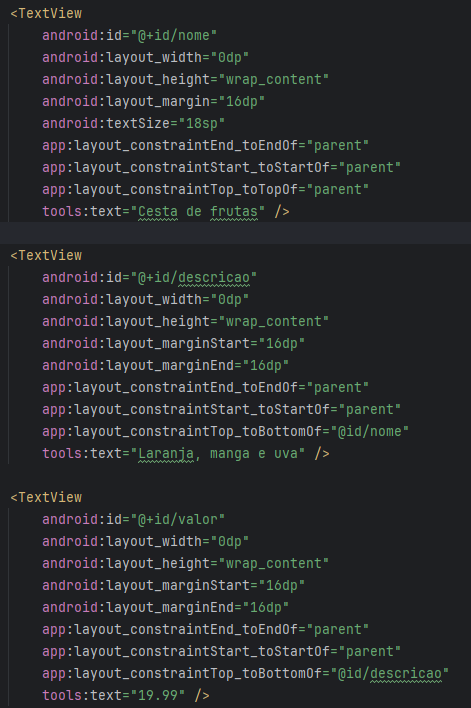
A URL serve para que possamos usar atributos Android padrão dentro do layout.

O atributo layout\_width define a largura do nosso Constraint Layout, preenchendo a largura disponível de seu container pai por causa do “match\_parent”. O pai é o layout.

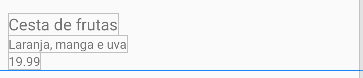
O atributo layout\_height define a altura do nosso Constraint Layout, se ajustando ao conteúdo dentro dele. Sendo assim ele terá a altura necessária para guardar todo seu conteúdo, sem “cortá-lo”.

Os dois em cinza associam prefixos com uma URL do esquema XML do Android. No caso do “app” ele é comumente usado para acessar atributos personalizados. Agora no “tools” ele é utilizado para definir atributos que são usados apenas em tempo de design.

Adicionamos entre a marcação do ConstraintLayout:



E assim temos a seguinte imagem sendo gerada:



Cada uma dessas linhas é uma TextView que criamos.

“id” vai definir um id para o elemento de interface de usuário. Sempre antes deve conter um: “@+id/” para depois colocarmos o id que queremos colocar nesse elemento. Isso serve para referenciarmos ele de forma única caso necessário.

O width como sabemos é para definir a largura, mas nele utilizamos “0dp”. “dp” é density-independent pixels, essa é uma unidade de medida que utiliza pixels que não vão variar independentemente do tamanho do dispositivo, por isso não usamos “px”. Selecionamos “0dp” para que ele possa se expandir para preencher o espaço disponível (isso juntamente às restrições de layout)

Na parte de “margin”, quando não há nada junto dela será colocado uma margem em todos os 4 lados, agora o Start é a esquerda e o End é a direita.

Os “constraint...\_to...of” que possuem “app” no começo são para definir uma restrição de layout. Por exemplo, estamos definindo que o começo da TextView é o mesmo que o começo de seu pai, que é o ConstraintLayout. Em alguns casos definimos que o topo de um elemento é o fundo do outro elemento referenciando-o pelo seu ID, fazendo assim com que um fique em cima do outro.

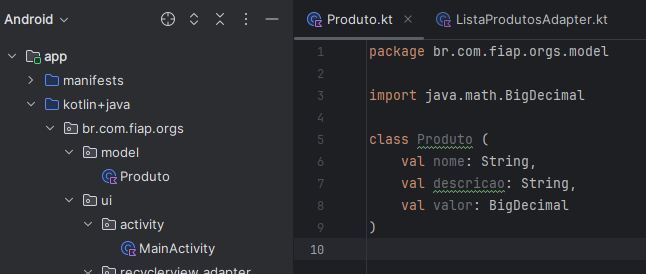
No fim usamos o “tools” para definir o texto do elemento.



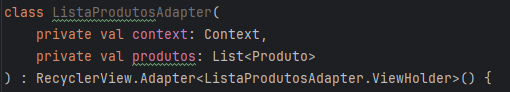
Adicionamos mais uma dependência ao build.gradle

Dentro do arquivo AndroidManifest, alterado o “name” para .ui.activity.MainActivity pois mudamos o arquivo MainActivity para essa pasta.

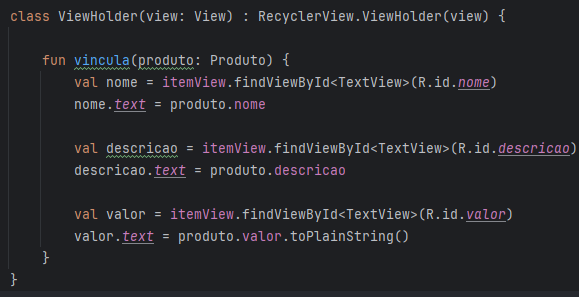
Criamos o package model e criamos uma Kotlin class chamada Produto:



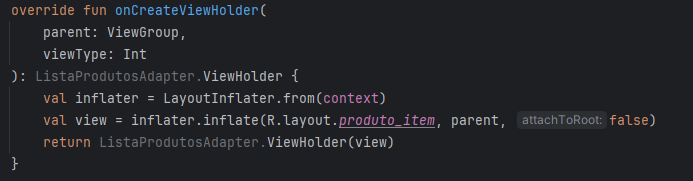
Criamos um package chamado recyclerview.adapter no package “ui” e criamos uma Kotlin class chamada “ListaProdutosAdapter”:



Seu construtor recebe um contexto e uma lista do tipo Produto, onde nele vamos inserir os textos que serão apresentados. O contexto é utilizado para inflar layouts e outras operações relacionadas.



A classe viewHolder vai vincular cada componente do nosso layout a uma variável e setar seu valor com o valor passado no objeto produto.

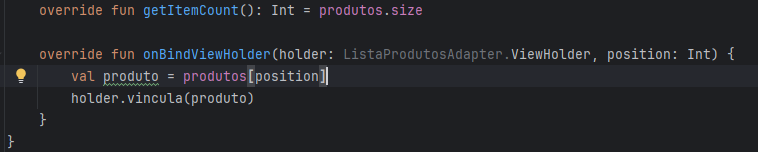


O parent é o grupo de views, o tipo de view é caso o RecyclerView tiver vários tipos de itens.

LayoutInflater utiliza o método from(context) para obter uma instância do LayoutInflater associada ao contexto passado no construtor do adaptador. Desta forma ele é utilizado para inflar o layout do item da lista a partir do arquivo XML.

O méotod inflate() é chamado passando o ID do layout do item da lista, o parent “RecyclerView” e false para indicar que não queremos que a view inflada seja automaticamente adicionada ao “parent”, nos permitindo configurar as propriedades da view antes de retorna-la.

Um novo objeto ViewHolder é criado passando a view inflada como parâmetro, assim sendo retornado para ser utilizado pelo RecyclerView.



“fun” é utilizado para criar uma função. Com esse método “getItemCount” é possível retornar o número de elementos totais na lista de produtos.

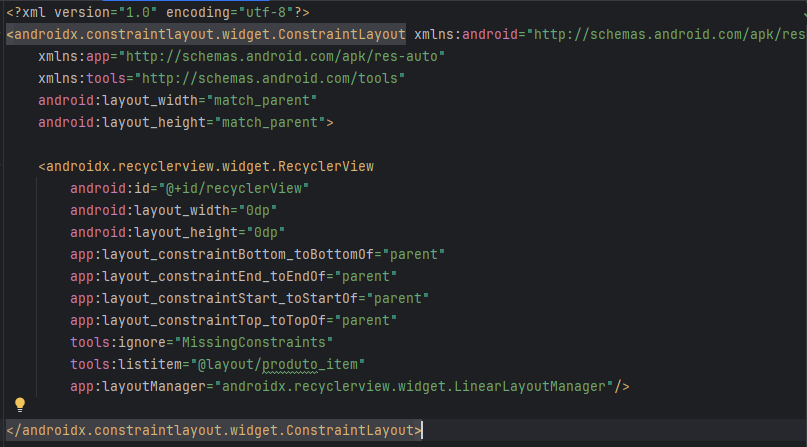
Já o método “onBindViewHolder” é responsável por atualizar a ViewHolder existente com dados do item da lista na posição que for fornecida a ele.

Ele recebe como parâmetro o ViewHolder que será atualizado e a posição do item da lista.

Dentro dele é recuperado o objeto produto dentro da lista na posição fornecida e o método vincula é chamado para vincular os dados do produto aos elementos do layout dentro do item da lista.

O “R” utilizado é usado para referenciar recursos estáticos, como layouts, IDs, etc.

Criado na pasta resources uma “activity\_main”:



Nela temos o ignore que é para ignorar avisos de falta de restrições em elementos de layout.

O listitem é utilizado para servir de conteúdo para ser repetido pelo RecyclerView.

A linha do layoutManager é para definir o LayoutManager do RecyclerView. Ele é responsável por determinar como os itens da lista são organizados e exibidos. Nesse nosso caso o LinearLayoutManager é utilizado para mostrar itens de uma lista vertical, onde cada item é exibido um em cima do outro.

Assim temos o seguinte resultado, um “loop”:



No nosso MainActivity o que foi feito:



A primeira linha é para definir o layout que será exibido na tela.

Colocamos em uma variável a nossa View que tem o ID de recyclerView.

Depois disso utilizamos o objeto ListaProdutosAdapter e atribuindo-o para a recyclerView. O adaptador recebe o parâmetro de contexto e a lista de produtos que já comentamos anteriormente.

Assim temos o seguinte resultado:

