

The background features a complex network of thin grey lines and dots, forming a web-like structure. Scattered throughout are various triangles of different sizes and orientations, some with solid black dots at their vertices. The overall aesthetic is modern and technical.

# **PROGRAMA DEV VENTURE**

---

Desenvolvimento Android

**LISTAS**  
RecyclerView

**01**

**ENUM**  
Classe enum kotlin

**02**

**ADAPTER**  
Padrão de design

**03**

# Aula 12 AGENDA



01

# RECYCLER VIEW

---





# WHAT DID I LEARN

---

Vamos registrar nesse aplicativo os tópicos aprendidos durante o curso.  
Em uma tela, exibiremos todas as lições e num futuro, iremos registrar novas lições em uma tela de cadastro

---





# RECYCLER VIEW

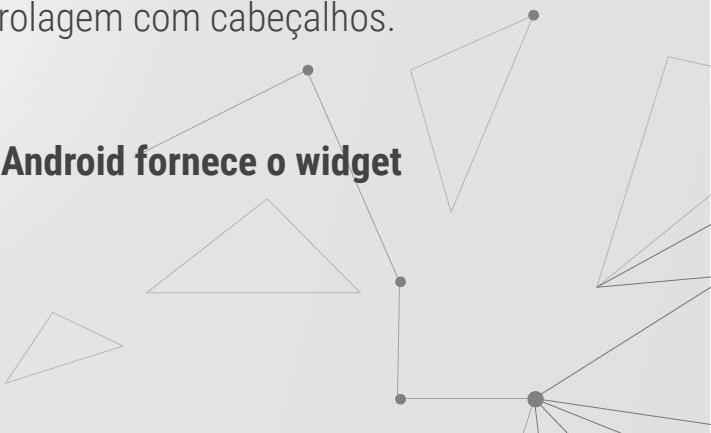
---

Exibir uma lista ou uma tabela de dados é uma das tarefas de UI mais comuns no Android.

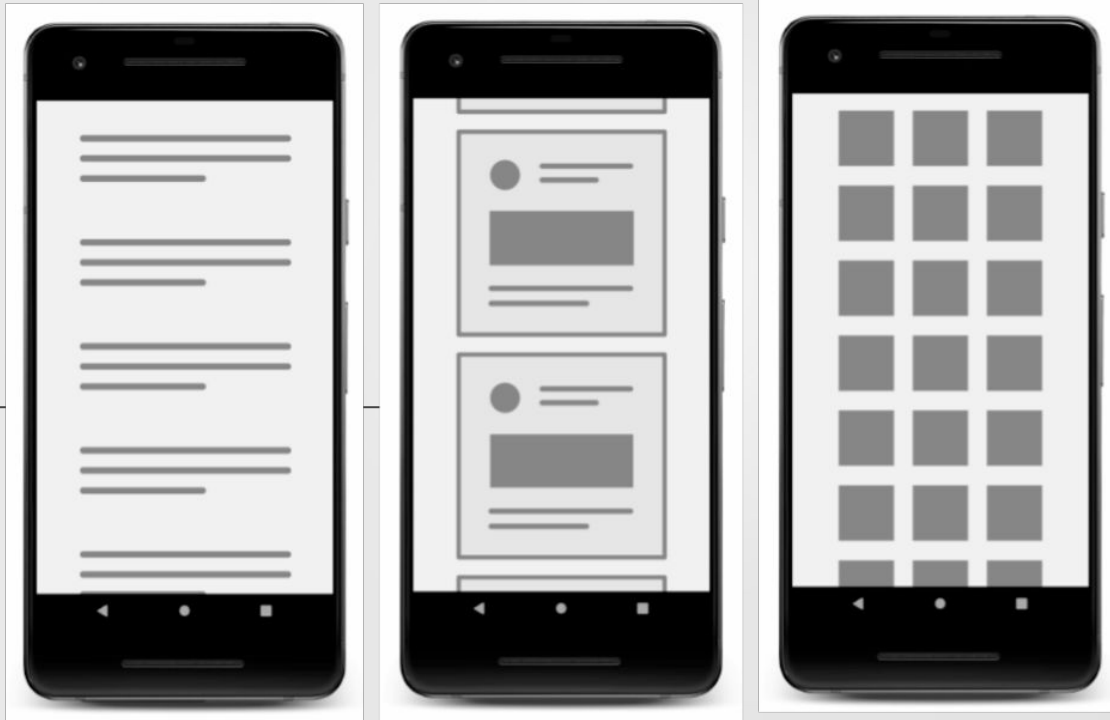
As listas variam de simples a muito complexas. Uma lista de views de texto pode mostrar dados simples, como uma lista de compras. Uma lista complexa, como uma lista anotada de destinos de férias, pode mostrar ao usuário muitos detalhes dentro de uma grade de rolagem com cabeçalhos.

---

**Para dar suporte a todos esses casos de uso, o Android fornece o widget RecyclerView.**



# RECYCLER VIEW





# RECYCLER VIEW

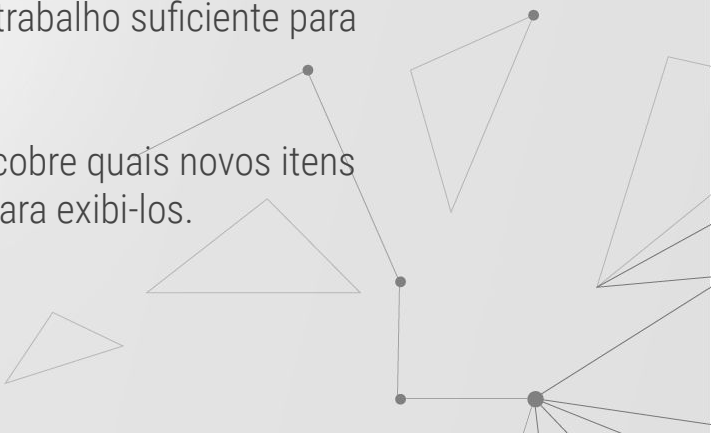
---

O RecyclerView é muito eficiente para listas grandes:  
Por padrão, funciona para processar ou desenhar itens que estão atualmente visíveis na tela.

Por exemplo, se sua lista tiver mil elementos, mas apenas 10 elementos estiverem visíveis, o RecyclerView fará apenas o trabalho suficiente para desenhar 10 itens na tela.

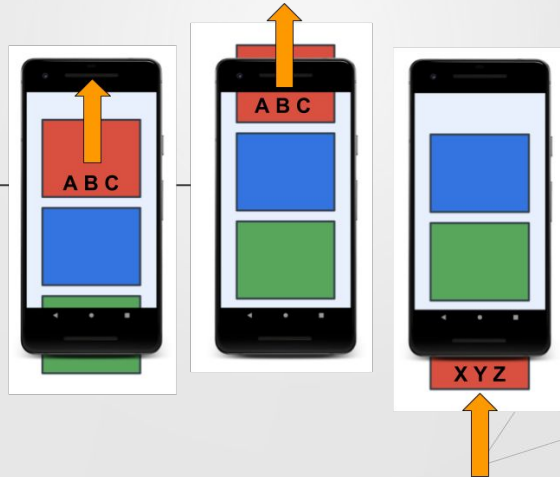
---

Quando o usuário rola a tela, o RecyclerView descobre quais novos itens devem estar na tela e faz o trabalho necessário para exibi-los.



# RECYCLER VIEW

Na sequência mostrada abaixo, você pode ver que uma visualização foi preenchida com dados, ABC. Depois que a exibição rola para fora da tela, o RecyclerView reutiliza a exibição para novos dados, XYZ.





# dojo / CRIANDO PROJETO

## What Do I Learn

### Kotlin - Null safety

O sistema de tipos de Kotlin visa eliminar o perigo de referências nulas do código,

### Layout editor

O Design Editor exibe o layout em vários dispositivos e versões do Android. É possível criar e editar um layout usando apenas componentes visuais.

### Git

É um sistema de controle de versão distribuído. Com ele é possível rastrear mudanças no código-fonte durante o desenvolvimento de software.

### GroupView

É uma view especial que pode conter outras views (chamadas de filhos). É a classe base para layouts e contêineres de views.

- Crie o projeto WhatDidILearn a partir de uma empty activity
- Faça um push inicial do projeto em seu repositório do Github

# 02

## ENUM

---





# ESTRUTURA DO PROJETO

---

- Itens aprendidos
  - Lista de itens
  - Exibir lista de itens aprendidos
- 





# LEARNEDITEM

---

- Nome
  - Descrição
  - Nível de entendimento (low, medium, high)
- 





# *dojo* / ENTIDADES

---

HIGH

MEDIUM

LOW

- Nível de entendimento pode receber apenas três valores:
  - HIGH, MEDIUM E LOW
- Uma ENUM class pode modelar esse comportamento.





# ENUM

---

<https://kotlinlang.org/docs/enum-classes.html>

---



# dojo / DATA

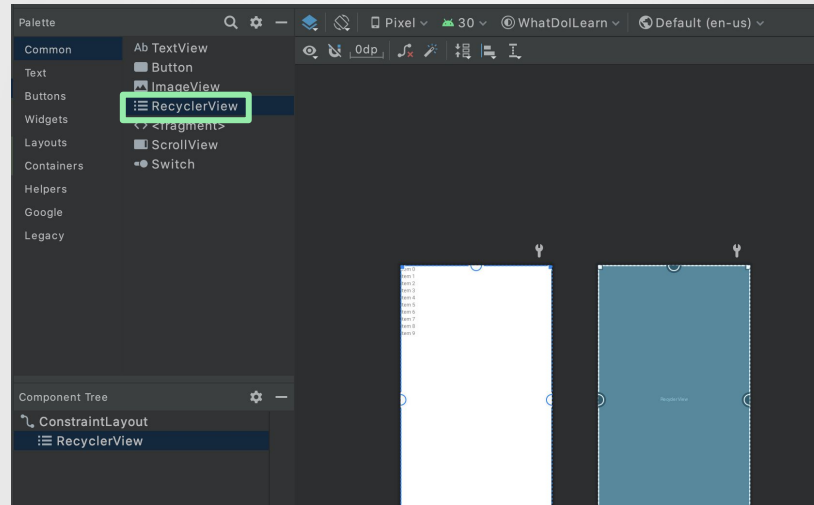
Title	Description	UnderstandingLevel
Kotlin - Null safety	O sistema de tipos de Kotlin visa eliminar o perigo de referências nulas do código,	
Layout editor	O Design Editor exibe o layout em vários dispositivos e versões do Android.É possível criar e editar um layout usando apenas componentes visuais.	
Git	É um sistema de controle de versão distribuído. Com ele é possível rastrear mudanças no código-fonte durante o desenvolvimento de software.	
GroupView	É uma view especial que pode conter outras views (chamadas de filhos).É a classe base para layouts e contêineres de views.	

- Crie um pacote data
- Crie a classe ItemLearnedDatabase que será responsável por armazenar os itens aprendidos e expor todos eles pelo método getAll()
- Você deve povoar essa base de dados, a partir da tabela de conteúdos

OBS: você deve preencher a coluna UnderstandingLevel de acordo com seu entendimento sobre o tópico

# *dojo* / RECYCLER VIEW

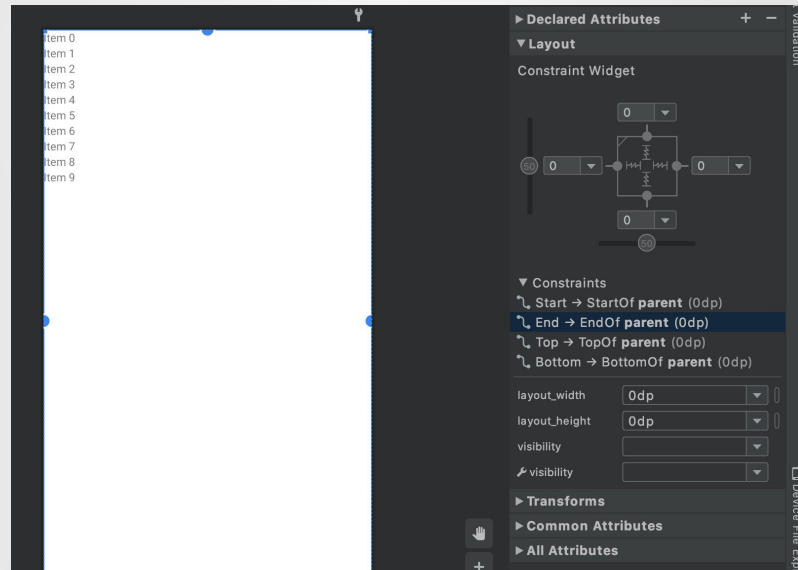
- No arquivo `activity_main.xml` vamos remover o `TextView` e adicionar um `RecyclerView`.
- Adicione ao `RecyclerView` o id `itemsLearnedRecyclerView`





# dojo / RECYCLER VIEW

- Ajustes as constraints do RecyclerView, conectando-o com a tela, e definindo o estilo das constraints como "match\_constraint"






# LAYOUT MANAGER

---

O RecyclerView usa um gerenciador de layout para posicionar os itens individuais na tela e determinar quando reutilizar views de itens que não estão mais visíveis para o usuário.

A Biblioteca de Suporte do Android inclui três gerenciadores de layout padrão, e cada um deles oferece muitas opções de personalização: `LinearLayoutManager`, `GridLayoutManager` e `StaggeredGridLayoutManager`.

---





# LAYOUT MANAGER


---

`LinearLayoutManager`: organiza os itens em uma lista unidimensional. O uso de um `RecyclerView`

`GridLayoutManager`: organiza os itens em uma grade bidimensional, como os quadrados em um quadriculado.

`StaggeredGridLayoutManager`: organiza os itens em uma grade bidimensional, com cada coluna um pouco diferente da anterior, por exemplo, as estrelas da uma bandeira estadunidense.

---



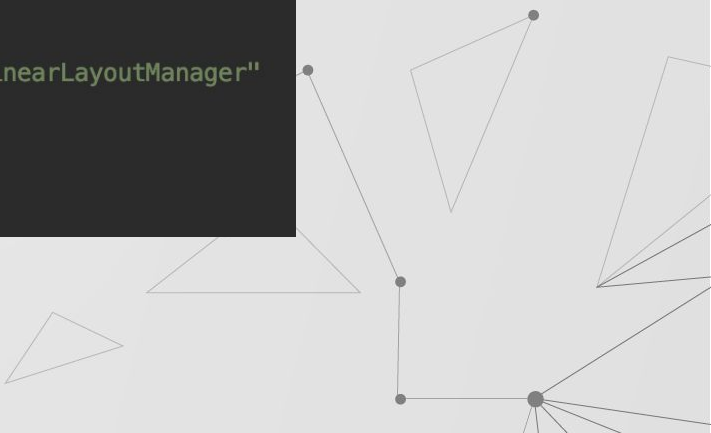


# *dojo* / LAYOUT MANAGER

---

- Adicione ao RecyclerView o LinearLayoutManager
- `app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"`

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/itemsLearnedRecyclerView"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

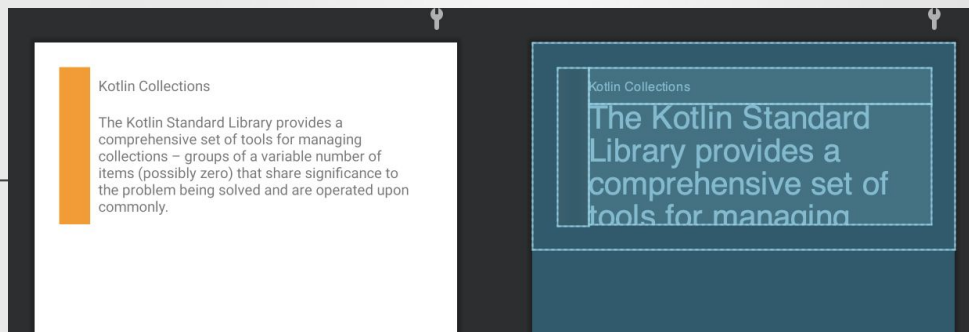


# ITENS DA LISTA

O RecyclerView é apenas um contêiner. Os itens carregados dentro de um RecyclerView vem de um o layout.

Crie o arquivo learned\_item.xml e tente reproduzir o layout a seguir:


OBS: understading level receberá somente uma cor de fundo





## ***dojo / ITEM***

---

- O layout como um todo, tem sua largura definida de acordo com o local onde ele será inserido
  - O container understandingLearning tem largura fixa de 50dp e altura definida de acordo com o tamanho dos textos
  - As caixas de texto tem largura definida de acordo com o local onde ele será inserido, e a altura se ajusta com o conteúdo
- 



## *dojo* / ITEM

---

- Defina o layout criado como listItem do RecyclerView para visualizar no modo design o resultado que teremos:

```
tools:listitem="@layout/learned_item"
```



# 03

## ADAPTER

---





# ADAPTER

---

Vocês já provavelmente usaram um adaptador de tomada para conectar algum aparelho elétrico, certo? O adaptador permite converter um tipo de plugue em outro, ou podemos dizer: converte uma interface em outra.





# ADAPTER

---

O padrão adapter na engenharia de software ajuda um objeto a trabalhar com outra API. O RecyclerView usa um adapter para transformar os dados do aplicativo em algo que o RecyclerView pode exibir, sem alterar a forma como o aplicativo armazena e processa os dados.

---





# *dojo* / ADAPTER

---

A principal tarefa na implementação de um RecyclerView é criar o adapter.

- Crie o pacote view e dentro dele a classe LearnedItemsAdapter





# *dojo* / ADAPTER

---

A classe deverá ter a seguinte estrutura

```
class LearnedItemsAdapter:  
    RecyclerView.Adapter<LearnedItemViewHolder>() {  
        inner class LearnedItemViewHolder(itemView: View) :  
            RecyclerView.ViewHolder(itemView) {  
            }  
        }  
    }
```

---



# ADAPTER

---

A classe `LearnedItemsAdapter` estende `RecyclerView.Adapter`. A classe é chamada `LearnedItemsAdapter` porque adapta um objeto `LearnedItem` a algo que o `RecyclerView` pode usar. O adapter precisa saber qual view holder usar, então definimos o `LearnedItemViewHolder`.

Um `ViewHolder` descreve o layout de item e metadados sobre seu lugar no `RecyclerView`.

---





# *dojo* / ADAPTER

---

No nível superior do adapter, crie uma variável `listOf` `LearnedItem` para armazenar os dados.

```
var data = listOf<LearnedItem>()
```

Ainda no adapter sobreescreva `getItemCount ()` para retornar o tamanho da lista itens. O `RecyclerView` precisa saber quantos itens o adapter tem para exibir e faz isso chamando `getItemCount ()`.

```
override fun getItemCount(): Int = data.size
```





# *dojo* / ADAPTER

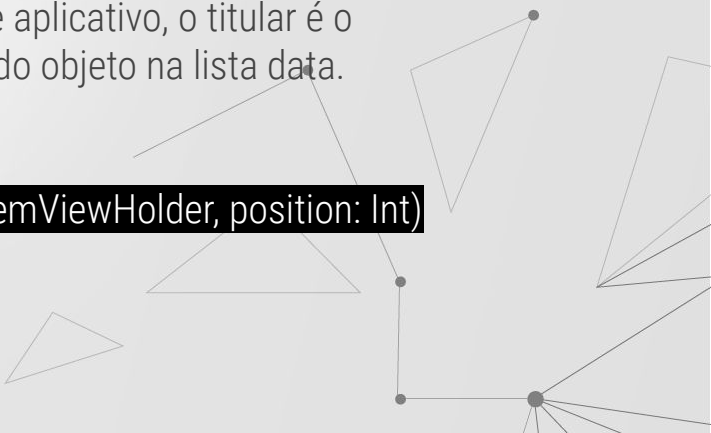
---

Sobrescreva a função `onBindViewHolder ()`. Ela é chamada pelo `RecyclerView` para exibir os dados de um item da lista na posição especificada.

o método possui dois argumentos: um portador de visualização e uma posição dos dados a serem vinculados. Para este aplicativo, o titular é o `LearnedItemViewHolder` e a posição é a posição do objeto na lista `data`.

---

```
override fun onBindViewHolder(holder: LearnedItemViewHolder, position: Int)
```





# *dojo* / ADAPTER

---

Complete o método:

```
override fun onBindViewHolder(holder: LearnedItemViewHolder, position:  
Int){  
    val item = data[position]  
    holder.bind(item.title, item.description, item.understandingLevel.color)  
}
```

---





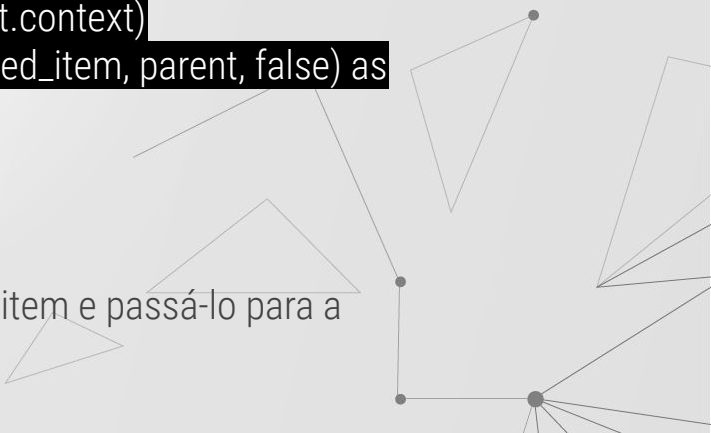
# *dojo* / ADAPTER

---

Sobrescreva o método onCreateViewHolder()

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):  
    LearnedItemViewHolder {  
        val inflater = LayoutInflater.from(parent.context)  
        val view = inflater.inflate(R.layout.learned_item, parent, false) as  
        View  
        return LearnedItemViewHolder(view)  
    }
```

Ele é responsável por inflar o layout do seu item e passá-lo para a  
ViewHolder





# ADAPTER

---

O adapter precisa informar o RecyclerView quando os dados foram alterados, porque o RecyclerView não sabe nada sobre os dados.

Para informar ao RecyclerView que os dados que ele está exibindo foram alterados, adicione um setter personalizado à variável de dados que está no topo da classe.

---






# *dojo* / ADAPTER

---

No setter, dê aos dados um novo valor e, em seguida, chame `notifyDataSetChanged()` para disparar o redesenho da lista com os novos dados

```
var data = listOf<LearnedItem>()  
set(value) {  
    field = value  
    notifyDataSetChanged()  
}
```

Nota: Quando `notifyDataSetChanged()` é chamado, o `RecyclerView` redesenha a lista inteira, não apenas os itens alterados. É possível melhorar esse comportamento.





## *dojo* / ACTIVITY


---

Agora que nossas estruturas estão definidas, vamos uni-las na nossa activity:

```
val recyclerView = findViewById<RecyclerView>(R.id.itemsLearnedRecyclerView)
val learnedItems = LearnedItemsDatabase().getAll()
val adapter = LearnedItemsAdapter()
```

---

```
recyclerView.adapter = adapter
adapter.data = learnedItems
```



# *dojo* / ACTIVITY





# ***EXERCÍCIOS***

---

Como o RecyclerView exibe os itens? Selecione tudo que se aplica.

- ☐ Exibe itens em uma lista ou grade.
  - ☐ Rola verticalmente ou horizontalmente.
  - ☐ Rola diagonalmente em dispositivos maiores, como tablets.
  - ☐ Permite layouts personalizados quando uma lista ou grade não é suficiente para o caso de uso.
- 



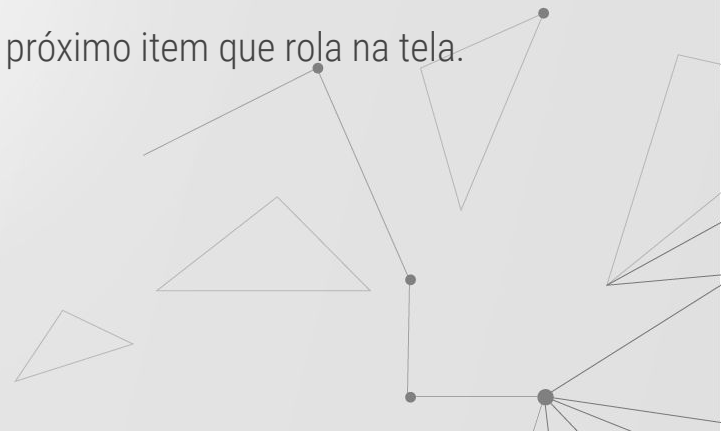


# ***EXERCÍCIOS***

---

Quais são os benefícios de usar o RecyclerView? Selecione tudo que se aplica.

- ☐ Exibe listas grandes com eficiência.
  - ☐ Atualiza automaticamente os dados.
  - ☐ Minimiza a necessidade de atualizações quando um item é atualizado, excluído ou adicionado à lista.
  - ☐ Reutiliza a exibição que rola para fora da tela para exibir o próximo item que rola na tela.
- 





# ***EXERCÍCIOS***

---

Quais são algumas das razões para usar adaptadores? Selecione tudo que se aplica.

- ☐ A separação de interesses torna mais fácil alterar e testar o código.
  - ☐ RecyclerView é independente dos dados que estão sendo exibidos.
  - ☐ As camadas de processamento de dados não precisam se preocupar com a forma como os dados serão exibidos.
  - ☐ O aplicativo será executado mais rápido.
- 







# ***EXERCÍCIOS - Desafio***

---

Quais das seguintes afirmações são verdadeiras para o ViewHolder? Selecione tudo que se aplica.

- ☐ O layout ViewHolder é definido em arquivos de layout XML.
  - ☐ Há um ViewHolder para cada unidade de dados no conjunto de dados.
  - ☐ Você pode ter mais de um ViewHolder em um RecyclerView.
  - ☐ O adaptador vincula os dados ao ViewHolder.
- 



# Palavras chave da aula de hoje:

RECYCLER VIEW  
PADRÃO DE DESIGN  
ADAPTER  
VIEW HOLDER  
ENUM  
CLASSE ANÔNIMA

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**.

**Please keep this slide for attribution.**