

# Sobre este codelab

subject Último mar. 29, 2021 atualizado

account\_circle Escrito por Equipe de treinamento do Google Developers

## 1. Antes de começar

Se você pensar nos apps que costuma usar no smartphone, quase todos têm pelo menos uma lista. A tela do histórico de chamadas, o app Contatos e seu app de mídia social favorito exibem listas de dados. Como mostrado na captura de tela abaixo, alguns desses apps exibem uma lista simples de palavras ou frases, enquanto outros exibem itens mais complexos, como cards que incluem texto e imagens. Seja qual for o conteúdo, exibir uma lista de dados é uma das tarefas mais comuns da IU no Android.

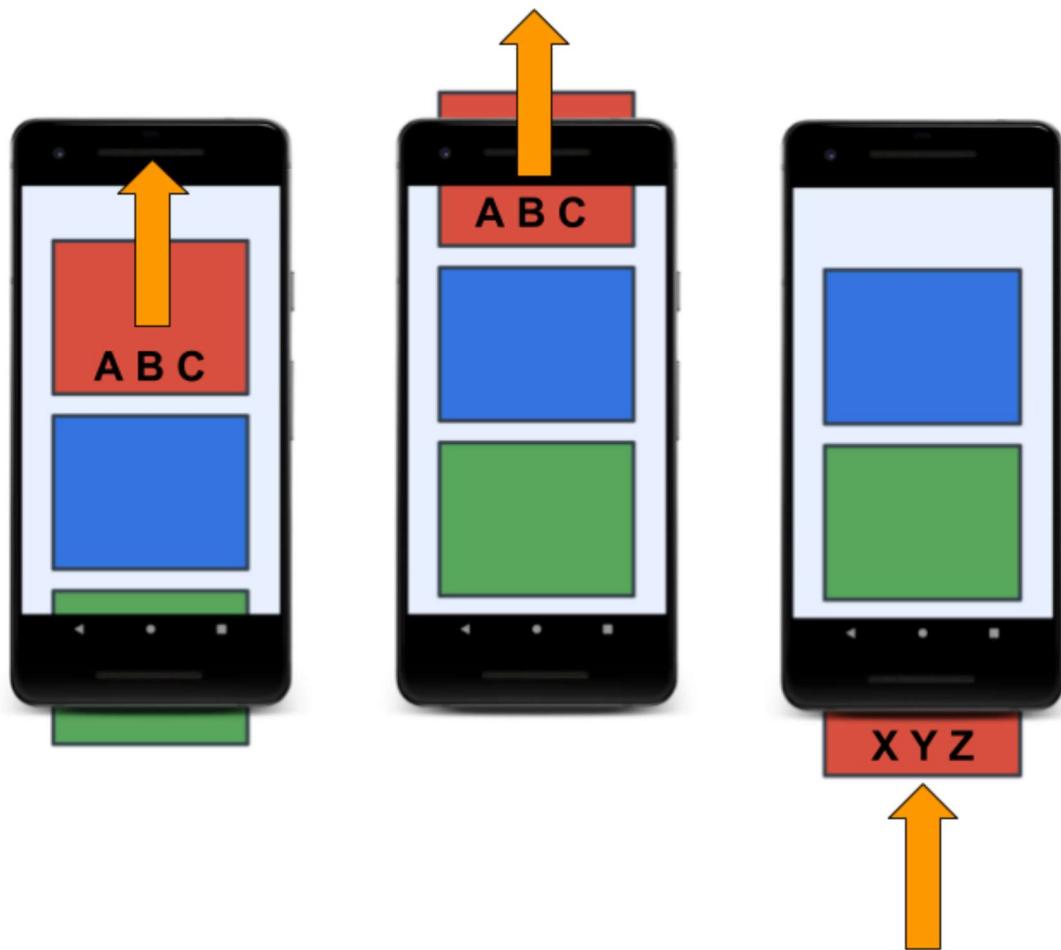


Para ajudar você a criar apps com listas, o Android fornece o `RecyclerView`.

O `RecyclerView` foi projetado para ser eficiente, mesmo com listas grandes, reutilizando ou reciclando as visualizações que rolaram para fora da tela. Quando um item é exibido na

tela, o RecyclerView reutiliza essa visualização para o próximo item da lista prestes a ser exibido. Isso significa que o item é preenchido com um novo conteúdo que aparecerá na tela. Esse comportamento do RecyclerView economiza muito tempo de processamento e ajuda as listas a rolarem de maneira mais suave.

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



Neste codelab, você criará o app Affirmations, um app simples que exibe dez afirmações positivas como texto em uma lista de rolagem. Depois, no codelab de continuação, você analisará as informações, adicionará uma imagem inspiradora a cada uma delas e melhorará a IU do app.

## Pré-requisitos

- Criar um projeto de um modelo no Android Studio.
- Adicionar recursos de string a um app.

- Definir um layout no XML.
- Compreender as classes e a herança no Kotlin (incluindo as classes abstratas).
- Herdar de uma classe existente e substituir os métodos dela.
- Use a documentação em [developer.android.com](https://developer.android.com) para ver as classes fornecidas pelo framework do Android.

## O que você aprenderá

- Como usar um RecyclerView para exibir uma lista de dados.
- Como organizar o código em pacotes.
- Como usar adaptadores com um RecyclerView para personalizar a aparência de um item da lista.

## O que você criará

- Um app que exibe uma lista de strings de afirmações usando uma RecyclerView.

## O que é necessário

- Um computador com a versão 4.1 ou mais recente do Android Studio instalada.

## 2. Como criar o projeto

### Criar um projeto Empty Activity

Antes de criar um novo projeto, verifique se você está usando o Android Studio 4.1 ou mais recente.

1. Inicie um novo projeto Kotlin no Android Studio usando o modelo **Empty Activity**.
2. Insira **Affirmations** no campo **Name** do app, **com.example.affirmations** no campo **Package name** e escolha **API Level 19** no campo **Minimum SDK**.
3. Clique em **Finish** para criar o projeto.

## 3. Como configurar a lista de dados

A próxima etapa na criação do app Affirmations é adicionar recursos. Você adicionará o seguinte ao projeto:

- Recursos de string que serão exibidos como afirmações no app

- Uma fonte de dados para fornecer uma lista de afirmações ao app

**Observação:** na maioria dos projetos de produção do Android, é necessário recuperar os dados de afirmações de um banco de dados ou de um servidor. O uso da rede e de bancos de dados está fora do escopo deste codelab, portanto, você usará uma lista de strings de afirmações definidas no app.

## Adicionar strings de afirmação

1. Na janela **Project**, abra **app > res > values > strings.xml**. No momento, esse arquivo tem um único recurso que é o nome do app.
2. Em **strings.xml**, adicione as seguintes afirmações como recursos de string individuais. Nomeie-os como affirmation1, affirmation2 e assim por diante.

### Texto das afirmações

I am strong.

I believe in myself.

Each day is a new opportunity to grow and be a better version of myself.

Every challenge in my life is an opportunity to learn from.

I have so much to be grateful for.

Good things are always coming into my life.

New opportunities await me at every turn.

I have the courage to follow my heart.

Things will unfold at precisely the right time.

I will be present in all the moments that this day brings.

O arquivo **strings.xml** ficará parecido com este exemplo quando você terminar.

```
<resources>
    <string name="app_name">Affirmations</string>
    <string name="affirmation1">I am strong.</string>
    <string name="affirmation2">I believe in myself.</string>
    <string name="affirmation3">Each day is a new opportunity to grow and be a better
version of myself.</string>
    <string name="affirmation4">Every challenge in my life is an opportunity to learn
from.</string>
    <string name="affirmation5">I have so much to be grateful for.</string>
    <string name="affirmation6">Good things are always coming into my life.</string>
    <string name="affirmation7">New opportunities await me at every turn.</string>
    <string name="affirmation8">I have the courage to follow my heart.</string>
    <string name="affirmation9">Things will unfold at precisely the right time.</string>
    <string name="affirmation10">I will be present in all the moments that this day
brings.</string>
</resources>
```

Agora que você adicionou recursos de string, faça referência a eles no seu código como `R.string.affirmation1` ou `R.string.affirmation2`.

## Criar um novo pacote

Organizar seu código de maneira lógica ajuda você e outros desenvolvedores a entender, manter e a estender o código. Da mesma forma que você organiza a papelada em arquivos e pastas, organize o código em arquivos e pacotes.

### O que é um pacote?

1. No Android Studio, na janela **Project (Android)**, veja os novos arquivos do projeto em **app > java** para o app Affirmations. Eles precisam ser semelhantes à captura de tela abaixo, que mostra três pacotes, um para o código (`com.example.affirmations`) e dois para arquivos de teste (`com.example.affirmations (androidTest)` e `com.example.affirmations (teste)`).



2. Observe que o nome do pacote tem várias palavras separadas por ponto.

Há duas maneiras usar pacotes.

- Criar pacotes diferentes para diferentes partes do código. Por exemplo, muitas vezes os desenvolvedores separam as classes que trabalham com dados e as classes que criam a IU em pacotes diferentes.
- Usar código de outros pacotes no seu código. Para usar as classes de outros pacotes, você precisa defini-las nas dependências do sistema de compilação. Também é uma prática padrão usar `import` no código para que você possa usar os nomes abreviados (por exemplo, `TextView`) em vez dos nomes totalmente qualificados (por exemplo, `android.widget.TextView`). Por exemplo, você já usou instruções `import` para classes como `sqrt` (`import kotlin.math.sqrt`) e `View` (`import android.view.View`).

No app Affirmations, além de importar classes do Android e do Kotlin, você também irá organizar o app em vários pacotes. Mesmo que o app não tenha muitas classes, é uma boa prática usar pacotes para agrupá-las por funcionalidade.

## Como nomear pacotes

Um nome de pacote pode ser qualquer coisa, desde que seja exclusivo no mundo todo: nenhum outro pacote publicado pode ter o mesmo nome. Como há um grande número de pacotes e criar nomes exclusivos aleatórios é difícil, os programadores usam convenções para facilitar a criação e a compreensão dos nomes dos pacotes.

- Geralmente, o nome do pacote é estruturado do mais geral para o mais específico, com cada parte do nome em letras minúsculas e separadas por um ponto.  
Importante: o ponto é apenas parte do nome. Ele não indica uma hierarquia no código nem exige uma estrutura de pastas.
- Como os domínios da Internet são únicos no mundo todo, é uma convenção usar um domínio, geralmente seu ou o domínio da sua empresa, como a primeira parte do nome.
- Você pode escolher os nomes dos pacotes para indicar o que eles contêm e a relação entre eles.
- Em exemplos de código como este, o domínio `com.example` seguido pelo nome do aplicativo é usado com frequência.

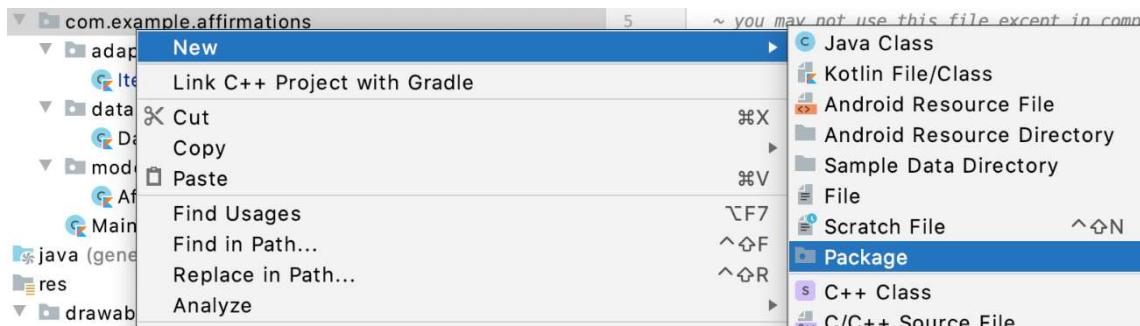
Veja alguns exemplos de nomes de pacote predefinidos e o conteúdo deles:

- `kotlin.math`: funções matemáticas e constantes.
- `android.widget`: visualizações, como `TextView`.

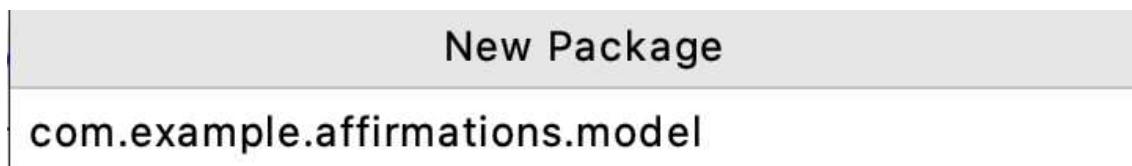
**Observação:** os nomes de pacotes (e a organização deles na janela **Project** do Android Studio como hierarquia de pastas) são exibidos como uma hierarquia. Porém, não há hierarquia real no código executável. Assim como o sistema de numeração de uma biblioteca categoriza e organiza livros, eles ainda estão todos na mesma estante, e você pode remover qualquer um deles.

## Criar um pacote

1. No painel **Project** do Android Studio, clique com o botão direito do mouse em **app > java > com.example.affirmations** e selecione **New > Package**.



2. No pop-up **New Package**, observe o prefixo do nome do pacote sugerido. A primeira parte sugerida do nome do pacote é o nome do pacote em que você clicou com o botão direito do mouse. Os nomes dos pacotes não criam uma hierarquia de pacotes, mas é possível reutilizar partes do nome para indicar a relação e a organização do conteúdo.
3. No pop-up, adicione **model** ao final do nome do pacote sugerido. Os desenvolvedores costumam usar **model** como o nome do pacote para classes que modelam (ou representam) dados.

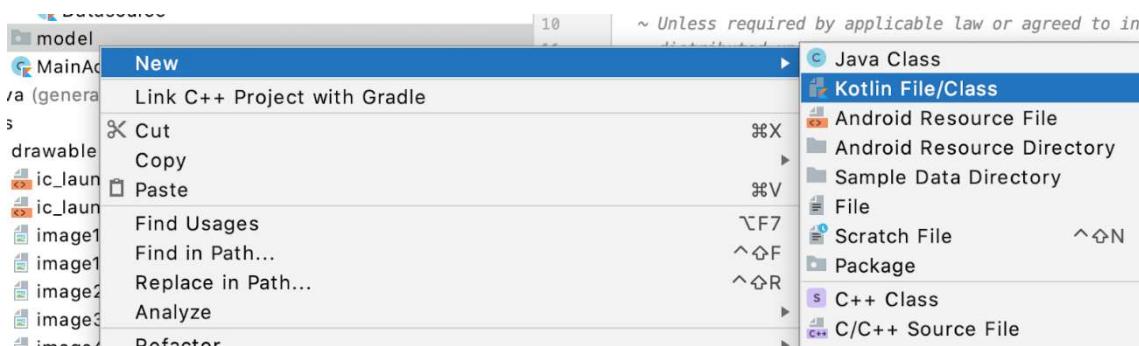


4. Pressione **Enter**. Isso criará um novo pacote no pacote (raiz) **com.example.affirmations**. O novo pacote conterá todas as classes relacionadas a dados definidas no app.

## Criar a classe de dados Affirmation

Nesta tarefa, você criará uma classe com o nome **Affirmation**. Uma instância de objeto de **Affirmation** representa uma afirmação e contém o ID do recurso da string com a afirmação.

5. Clique com o botão direito no pacote **com.example.affirmations.model** e selecione **New > Kotlin File/Class**.



6. No pop-up, selecione **Class** e digite **Affirmation** como o nome da classe. Isso criará um novo arquivo com o nome **Affirmation.kt** no pacote **model**.
7. Transforme a classe **Affirmation** em uma classe de dados adicionando a palavra-chave **data** antes da definição dela. Isso gera um erro, já que as classes de dados precisam ter pelo menos uma propriedade definida.

**Affirmation.kt**

```
package com.example.affirmations.model

data class Affirmation {
```

Ao criar uma instância de **Affirmation**, é necessário transmitir o ID do recurso para a string de afirmação. O ID do recurso é um número inteiro.

8. Adicione um parâmetro **StringResourceID** val inteiro ao construtor da classe **Affirmation**. Isso eliminará o erro.

```
package com.example.affirmations.model
```

```
data class Affirmation(val stringResourceId: Int)
```

## Criar uma classe para ser uma fonte de dados

Os dados exibidos no app podem vir de fontes diferentes (por exemplo, dentro do projeto do app ou de uma fonte externa que exija conexão com a Internet para fazer o download de dados). Como resultado, os dados podem não estar no formato exato que você precisa. O restante do app não precisa se preocupar com o local de origem dos dados ou o formato original deles. Você pode e precisa ocultar essa preparação de dados em uma classe **Datasource** separada que prepare os dados para o app.

Como a preparação de dados é uma questão separada, coloque a classe `Datasource` em um pacote `data` separado.

1. No Android Studio, na janela **Project**, clique com o botão direito do mouse em **app > java > com.example.affirmations** e selecione **New > Package**.
2. Digite `data` como a última parte do nome do pacote.
3. Clique com o botão direito no pacote `data` e selecione **New > Kotlin File/Class**.
4. Digite `Datasource` como o nome da classe.
5. Na classe `Datasource`, crie uma função com o nome `loadAffirmations()`.

A função `loadAffirmations()` precisa retornar uma lista de `Affirmations`. Para fazer isso, crie uma lista e preencha-a com uma instância `Affirmation` para cada string de recurso.

6. Declare `List<Affirmation>` como o tipo de retorno do método `loadAffirmations()`.
7. No corpo de `loadAffirmations()`, adicione uma instrução `return`.
8. Após a palavra-chave `return`, chame o método `listOf<>()` para criar uma `List`.
9. Dentro dos sinais `<>`, especifique o tipo dos itens da lista como `Affirmation`. Se necessário, importe `com.example.affirmations.model.Affirmation`.
10. Dentro dos parênteses, crie uma `Affirmation`, transmitindo `R.string.affirmation1` como o ID do recurso, conforme mostrado abaixo.  
`Affirmation(R.string.affirmation1)`
11. Adicione os outros objetos `Affirmation` à lista de todas as afirmações, separadas por vírgulas. O código finalizado ficará assim.

`Datasource.kt`

```
package com.example.affirmations.data

import com.example.affirmations.R
import com.example.affirmations.model.Affirmation

class Datasource {

    fun loadAffirmations(): List<Affirmation> {
        return listOf<Affirmation>(
            Affirmation(R.string.affirmation1),
            Affirmation(R.string.affirmation2),
            Affirmation(R.string.affirmation3),
        )
    }
}
```

```
        Affirmation(R.string.affirmation4),  
        Affirmation(R.string.affirmation5),  
        Affirmation(R.string.affirmation6),  
        Affirmation(R.string.affirmation7),  
        Affirmation(R.string.affirmation8),  
        Affirmation(R.string.affirmation9),  
        Affirmation(R.string.affirmation10)  
    )  
}  
}
```

## [Opcional] Exibir o tamanho da lista de afirmações em uma TextView

Para verificar se você pode criar uma lista de afirmações, chame `loadAffirmations()` e exiba o tamanho da lista retornada de afirmações na `TextView` que acompanha o modelo do app Empty Activity.

1. Em `layouts/activity_main.xml`, forneça um `id` e uma `textview` à `TextView` que acompanha o modelo.
2. Em `MainActivity`, no método `onCreate()`, depois do código já existente, acesse uma referência à `textview`.

```
val textView: TextView = findViewById(R.id.textview)
```

3. Em seguida, adicione o código para criar e exibir o tamanho da lista de afirmações. Crie uma `Datasource`, chame o método `loadAffirmations()`, acesse o tamanho da lista retornada, converta-o em uma string e atribua-a como o `text` da `textView`.

```
textView.text = Datasource().loadAffirmations().size.toString()
```

4. Execute o app. A tela ficará como o exemplo abaixo.

## Affirmations

10

5. Exclua o código que você acabou de adicionar na `MainActivity`.

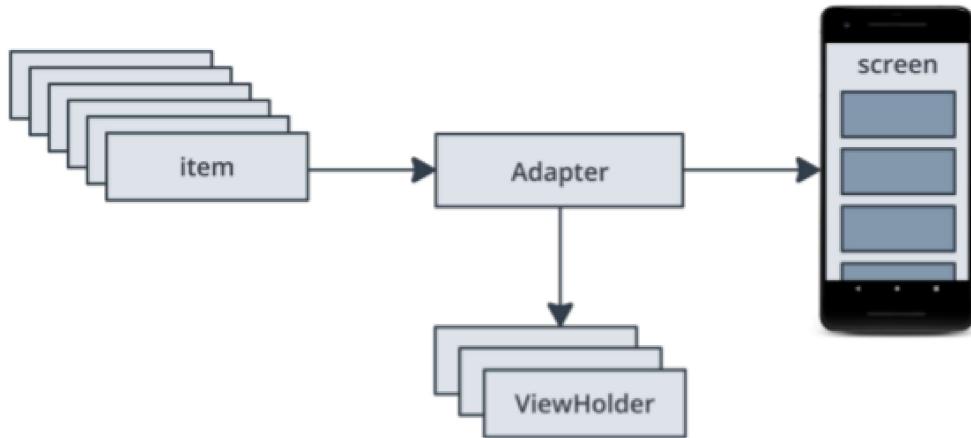
[Voltar](#)

## 4. Como adicionar um RecyclerView ao app

Nesta tarefa, você configurará um `RecyclerView` para exibir a lista de `Affirmations`.

Há várias partes envolvidas na criação e no uso de um `RecyclerView`. Pense neles como uma divisão do trabalho. O diagrama abaixo mostra uma visão geral, e você aprenderá mais sobre cada parte durante a implementação.

- **item**: um item de dados da lista a ser exibida. Representa um objeto `Affirmation` no app.
- **Adapter**: aceita dados e os prepara para que a `RecyclerView` os exiba.
- **ViewHolders**: um conjunto de visualizações para a `RecyclerView` usar e reutilizar para exibir afirmações.
- **RecyclerView**: visualizações na tela.



### Adicionar um RecyclerView ao layout

O app `Affirmations` consiste em uma única atividade com o nome `MainActivity` e o arquivo de layout, `activity_main.xml`. Primeiro, você precisa adicionar o `RecyclerView` ao layout da `MainActivity`.

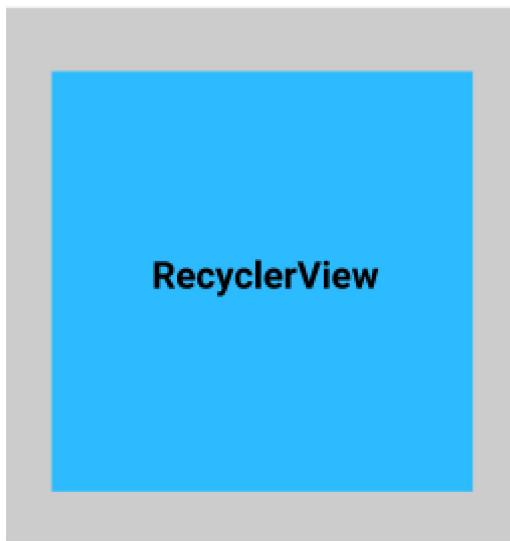
1. Abra `activity_main.xml` (`app > res > layout > activity_main.xml`)
2. Se você ainda não estiver usando a **visualização Split**, mude para ela.

≡ Code   ≡ Split   ▲ Design

3. Exclua a TextView.

O layout atual usa o ConstraintLayout. O ConstraintLayout é ideal e flexível quando você quer posicionar várias visualizações filhas em um layout. Como seu layout tem apenas uma visualização filha, RecyclerView, você pode alternar para um ViewGroup mais simples conhecido como FrameLayout, que precisa ser usado para armazenar uma única visualização filha.

## FrameLayout



4. No XML, substitua ConstraintLayout por FrameLayout. O layout concluído será semelhante ao mostrado abaixo.

activity\_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
</FrameLayout>
```

5. Alternar para a visualização Design.

6. Em **Palette**, selecione **Containers** e localize a **RecyclerView**.
7. Arraste um **RecyclerView** para o layout.
8. Leia o pop-up **Add Project Dependency** se ele for exibido e clique em **OK**. Se o pop-up não for exibido, nenhuma ação será necessária.
9. Aguarde até que o Android Studio termine e o **RecyclerView** seja exibido no layout.
10. Se necessário, mude os atributos `layout_width` e `layout_height` do **RecyclerView** para `match_parent` para que o **RecyclerView** possa preencher toda a tela.
11. Defina o ID do recurso do **RecyclerView** como `recycler_view`.

O **RecyclerView** é compatível com a exibição de itens de maneiras diferentes, como uma lista linear ou uma grade. A organização dos itens é feita por um **LayoutManager**. O framework do Android fornece gerenciadores de layout para layouts de itens básicos. O app **Affirmations** exibe itens como uma lista vertical, portanto, você pode usar o **LinearLayoutManager**.

12. Volte para a visualização **Code**. No código XML, no elemento **RecyclerView**, adicione o **LinearLayoutManager** como o atributo do gerenciador de layout do **RecyclerView**, conforme mostrado abaixo.

```
app:layoutManager="LinearLayoutManager"
```

Para percorrer uma lista vertical de itens maiores do que a tela, é necessário adicionar uma barra de rolagem vertical.

13. Na **RecyclerView**, adicione um atributo `android:scrollbars` definido como `vertical`.

```
android:scrollbars="vertical"
```

O layout XML final ficará assim:

```
activity_main.xml
```

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".MainActivity">
```

```
<androidx.recyclerview.widget.RecyclerView  
    android:id="@+id/recycler_view"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:scrollbars="vertical"  
    app:layoutManager="LinearLayoutManager" />  
  
</FrameLayout>
```

#### 14. Execute o app.

O projeto precisa ser compilado e executado sem problemas. No entanto, apenas um fundo branco será exibido no app porque falta uma parte essencial do código. No momento, você adicionou a origem dos dados e o RecyclerView ao layout, mas o RecyclerView não tem informações sobre como exibir os objetos Affirmation.

### Implementar um adaptador para o RecyclerView

Seu app precisa de uma maneira para capturar os dados da Datasource e formatá-los para que cada Affirmation seja exibida como um item no RecyclerView.

Um adaptador é um padrão de design que adapta os dados para algo que possa ser usado pelo RecyclerView. Nesse caso, você precisa de um adaptador que receba uma instância de Affirmation da lista retornada por loadAffirmations() e a transforme em uma visualização de item de lista para que ela seja exibida no RecyclerView.

Quando você executar o app, o RecyclerView usa o adaptador para descobrir como exibir os dados na tela. O RecyclerView solicita que o adaptador crie uma nova visualização de item de lista para o primeiro item de dados da lista. Quando tiver a visualização, ela solicitará que o adaptador forneça os dados para exibir o item. Esse processo se repete até que o RecyclerView não precise de mais visualizações para preencher a tela. Se apenas 3 visualizações de itens da lista podem ser exibidas na tela ao mesmo tempo, a RecyclerView solicitará ao adaptador para preparar essas 3 visualizações (em vez de todas as 10 visualizações).

Nesta etapa, você criará um adaptador para uma instância de objeto Affirmation para que possa ser exibida no RecyclerView.

### Criar o adaptador

Um adaptador tem várias partes, e você escreverá bastante código mais complexo do que o já feito neste curso até agora. Não tem problema se você não entender completamente

os detalhes no começo. Após concluir todo o app com um RecyclerView, você entenderá melhor como todas as peças se encaixam. Você também poderá reutilizar esse código como base para apps futuros que você criar com um RecyclerView.

## Criar um layout para itens

Cada item no RecyclerView tem seu próprio layout, que você define em um arquivo de layout separado. Como você só exibirá uma string, pode usar um TextView para o layout do item.

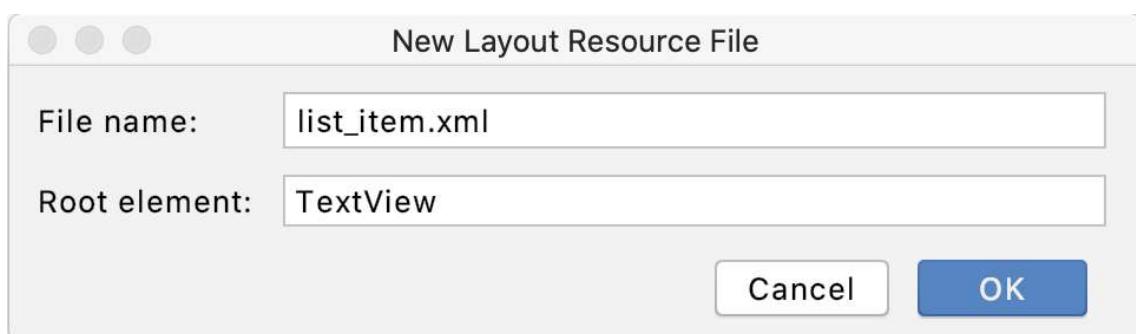
1. Em **res > layout**, crie um novo **Arquivo** vazio com o nome `list_item.xml`.
2. Abra `list_item.xml` na visualização **Code**.
3. Adicione uma `TextView` com o id `item_title`.
4. Adicione `wrap_content` para a `layout_width` e a `layout_height`, conforme mostrado no código abaixo.

Observe que você não precisa de um `ViewGroup` no layout, porque esse layout de item de lista será inflado e adicionado mais tarde como um filho do `RecyclerView` pai.

`list_item.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/item_title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Como alternativa, você pode ter usado **File > New > Layout Resource File**, preenchendo o campo **File name** como `list_item.xml` e o campo **Root element** como `TextView`. Em seguida, atualize o código gerado para corresponder ao código acima.



## Criar uma classe ItemAdapter

1. No painel **Project** do Android Studio, clique com o botão direito do mouse em **app > java > com.example.affirmations** e selecione **New > Package**.
2. Digite **adapter** como a última parte do nome do pacote.
3. Clique com o botão direito do mouse no pacote **adapter** e selecione **New > Kotlin File/Class**.
4. Digite **ItemAdapter** como o nome da classe para terminar e o arquivo **ItemAdapter.kt** será aberto.

É necessário adicionar um parâmetro ao construtor do **ItemAdapter** para que você possa transmitir a lista de afirmações ao adaptador.

5. Adicione um parâmetro ao construtor do **ItemAdapter** que é uma **val** com o nome **dataset** e o tipo **List<Affirmation>**. Importe **Affirmation**, se necessário.
6. Como o **dataset** só será usado nessa classe, torne-o **private**.

**ItemAdapter.kt**

```
import com.example.affirmations.model.Affirmation

class ItemAdapter(private val dataset: List<Affirmation>) {
```

}

O **ItemAdapter** precisa de informações sobre como resolver os recursos de string. Essa e outras informações sobre o app são armazenadas em uma instância de objeto **Context** que pode ser transmitida para uma instância **ItemAdapter**.

7. Adicione um parâmetro ao construtor do **ItemAdapter** que é uma **val** com o nome **context** e o tipo **Context**. Posicione-o como o primeiro parâmetro no construtor.

```
class ItemAdapter(private val context: Context, private val dataset: List<Affirmation>) {
```

}

## Criar um ViewHolder

O **RecyclerView** não interage diretamente com as visualizações de itens, mas lida com **ViewHolders**. Um **ViewHolder** representa uma única visualização de item de lista em um **RecyclerView** e pode ser reutilizado sempre que possível. Uma

instância ViewHolder contém referências às visualizações individuais em um layout de item de lista. Daí o nome "view holder", que significa armazenador de visualizações. Isso facilita a atualização da visualização de itens da lista com novos dados. Os armazenadores de visualização também adicionam informações que o RecyclerView usa para mover as visualizações pela tela de maneira eficiente.

1. Na classe ItemAdapter, antes da chave de fechamento do ItemAdapter, crie uma classe ViewHolder.

```
class ItemAdapter(private val context: Context, private val dataset: List<Affirmation>) {  
  
    class ViewHolder()  
}
```

- Definir uma classe em outra é conhecido como criação de uma **classe aninhada**.
  - Como o ViewHolder é usado apenas pelo ItemAdapter, a criação dele no ItemAdapter mostra essa relação. Isso não é obrigatório, mas ajuda outros desenvolvedores a entenderem a estrutura do programa.
2. Adicione uma private val view do tipo View como um parâmetro ao construtor da classe ViewHolder.
  3. Torne ViewHolder uma subclasse de RecyclerView.ViewHolder e transmita o parâmetro view para o construtor da superclasse.
  4. No ViewHolder, defina uma propriedade val textView do tipo TextView. Atribua a visualização com o ID item\_title definido em list\_item.xml.

```
class ItemAdapter(private val context: Context, private val dataset: List<Affirmation>) {  
  
    class ViewHolder(private val view: View) : RecyclerView.ViewHolder(view) {  
        val textView: TextView = view.findViewById(R.id.item_title)  
    }  
}
```

## Substituir métodos do adaptador

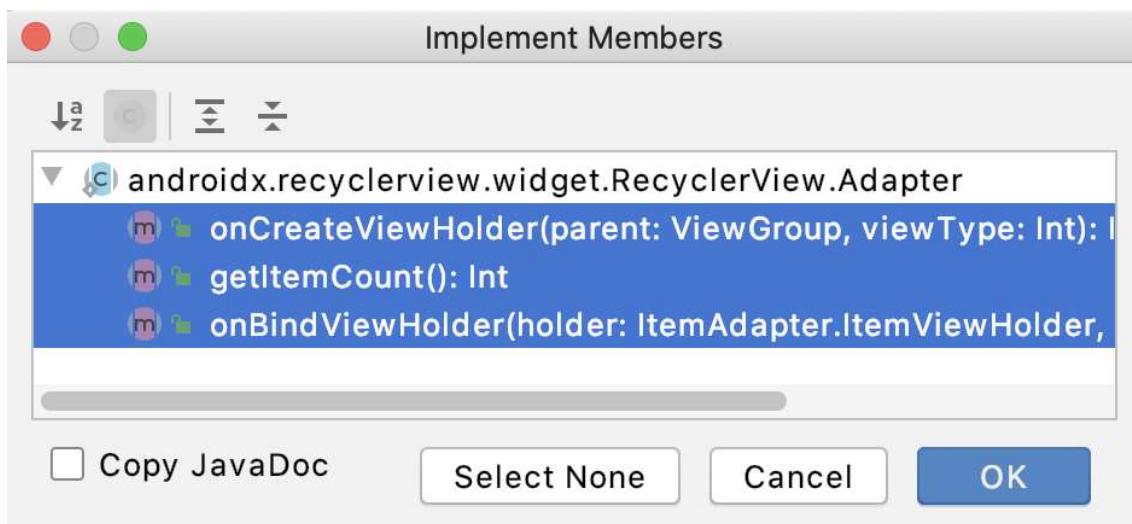
1. Adicione o código para estender o ItemAdapter da classe abstrata RecyclerView.Adapter. Especifique ItemAdapter.ViewHolder como o tipo de armazenador de visualização entre os sinais de "menor que" e "maior que".

```
class ItemAdapter(  
    private val context: Context,  
    private val dataset: List<Affirmation>  
) : RecyclerView.Adapter<ItemAdapter.ViewHolder>() {
```

```
class ItemViewHolder(private val view: View) : RecyclerView.ViewHolder(view) {  
    val textView: TextView = view.findViewById(R.id.item_title)  
}  
}
```

Você verá um erro porque precisa implementar alguns métodos abstratos do `RecyclerView.Adapter`.

2. Coloque o cursor em `ItemAdapter` e pressione **Command+I** (**Control+I** no Windows). Veja a lista de métodos que você precisa implementar: `getItemCount()`, `onCreateViewHolder()` e `onBindViewHolder()`.



3. Selecione as três funções usando **Shift+clique** e clique em **OK**.

Isso criará stubs com os parâmetros corretos para os três métodos, conforme mostrado abaixo.

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {  
    TODO("Not yet implemented")  
}  
  
override fun getItemCount(): Int {  
    TODO("Not yet implemented")  
}  
  
override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {  
    TODO("Not yet implemented")  
}
```

Você não verá mais erros. Em seguida, é necessário implementar esses métodos para que eles executem as ações corretas no app.

## Implementar getItemCount()

O método `getItemCount()` precisa retornar o tamanho do conjunto de dados. Os dados do app estão na propriedade `dataset` que você está transmitindo ao construtor do `ItemAdapter`, e você pode acessar o tamanho usando `size`.

1. Substitua `getItemCount()` por:

```
override fun getItemCount() = dataset.size
```

Isso é uma forma mais concisa de escrever:

```
override fun getItemCount(): Int {  
    return dataset.size  
}
```

## Implementar onCreateViewHolder()

O método `onCreateViewHolder()` é chamado pelo gerenciador de layout para criar novos armazenadores de visualização para o `RecyclerView` (quando não há armazenadores de visualização existentes que possam ser reutilizados). Lembre-se de que um armazenador de visualização representa uma única visualização de item da lista.

O método `onCreateViewHolder()` usa dois parâmetros e retorna um novo `ViewHolder`.

- Um parâmetro `parent`, que é a visualização em grupo a que a nova visualização de item da lista será anexada como filha. O `RecyclerView` é a visualização mãe.
  - Um parâmetro `viewType` que se torna importante quando há vários tipos de visualização de itens no mesmo `RecyclerView`. Se você tiver diferentes layouts de item de lista sendo exibidos no `RecyclerView`, há diferentes tipos de visualização de itens. Só é possível reciclar visualizações com o mesmo tipo de visualização de item. No seu caso, há somente um layout de item de lista e um tipo de visualização de item, portanto, você não precisa se preocupar com esse parâmetro.
1. No método `onCreateViewHolder()`, acesse uma instância do `LayoutInflater` com o contexto fornecido (context do parent). O inflador de layout sabe como inflar um layout XML em uma hierarquia de objetos de visualização.

```
val adapterLayout = LayoutInflater.from(parent.context)
```

- Quando você tiver uma instância de objeto `LayoutInflater`, adicione um ponto seguido por outra chamada de método para inflar a visualização real do item da lista. Transmite o ID do recurso de layout XML `R.layout.list_item` e a visualização em grupo parent. O terceiro argumento booleano será `attachToRoot`. Esse argumento precisa ser `false`, porque o `RecyclerView` adicionará esse item à hierarquia de visualização no momento certo.

```
val adapterLayout = LayoutInflater.from(parent.context)
    .inflate(R.layout.list_item, parent, false)
```

Agora, o `adapterLayout` contém uma referência para a visualização do item da lista (em que poderemos encontrar

visualizações filhas, como a `TextView`, depois).

- No método `onCreateViewHolder()`, retorne uma nova instância do `ItemViewHolder` em que a visualização raiz seja o `adapterLayout`.

```
return ItemViewHolder(adapterLayout)
```

Este é o código do método `onCreateViewHolder()` até agora.

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
    // create a new view
    val adapterLayout = LayoutInflater.from(parent.context)
        .inflate(R.layout.list_item, parent, false)

    return ItemViewHolder(adapterLayout)
}
```

## Implementar `onBindViewHolder()`

O último método que você precisa substituir é `onBindViewHolder()`. Esse método é chamado pelo gerenciador de layout para substituir o conteúdo de uma visualização de item de lista.

O método `onBindViewHolder()` tem dois parâmetros, um `ItemViewHolder` criado anteriormente pelo método `onCreateViewHolder()` e um `int` que representa a `position` do item atual na lista. Neste método, você encontrará o objeto `Affirmation` correto do conjunto de dados com base na posição.

1. No método `onBindViewHolder()`, crie um item `val` e acesse o item na `position` especificada no `dataset`.

```
val item = dataset[position]
```

Por fim, é necessário atualizar todas as visualizações referenciadas pelo titular da visualização para refletir os dados corretos desse item. Nesse caso, há apenas uma visualização: a `TextView` no `ItemViewHolder`. Defina o texto da `TextView` para exibir a string `Affirmation` desse item.

2. Com uma instância de objeto `Affirmation`, é possível encontrar o ID de recurso de string correspondente chamando `item.stringResourceId`. No entanto, ele é um número inteiro e você precisa encontrar o mapeamento para o valor real da string.

No framework do Android, é possível chamar o método `getString()` com um ID de recurso de string e ele retornará o valor da string associado a ele. `getString()` é um método da classe `Resources`, e é possível acessar uma instância da classe `Resources` com o `context`.

Isso significa que é possível chamar `context.resources.getString()` e transmitir um ID de recurso da string. A string resultante pode ser definida como o `text` da `textView` no `holder` do `ItemViewHolder`. Resumidamente, essa linha de código atualiza o armazenador de visualização para mostrar a string de afirmação.

```
holder.textView.text = context.resources.getString(item.stringResourceId)
```

O método `onBindViewHolder()` completo ficará assim.

```
override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
    val item = dataset[position]
    holder.textView.text = context.resources.getString(item.stringResourceId)
}
```

Este é o código do adaptador finalizado.

```
ItemAdapter.kt
```

```
package com.example.affirmations.adapter

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
```

```
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.affirmations.R
import com.example.affirmations.model.Affirmation

/**
 * Adapter for the [RecyclerView] in [MainActivity]. Displays [Affirmation] data object.
 */
class ItemAdapter(
    private val context: Context,
    private val dataset: List<Affirmation>
) : RecyclerView.Adapter<ItemAdapter.ItemViewHolder>() {

    // Provide a reference to the views for each data item
    // Complex data items may need more than one view per item, and
    // you provide access to all the views for a data item in a view holder.
    // Each data item is just an Affirmation object.
    class ItemViewHolder(private val view: View) : RecyclerView.ViewHolder(view) {
        val textView: TextView = view.findViewById(R.id.item_title)
    }

    /**
     * Create new views (invoked by the layout manager)
     */
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
        // create a new view
        val adapterLayout = LayoutInflater.from(parent.context)
            .inflate(R.layout.list_item, parent, false)

        return ItemViewHolder(adapterLayout)
    }

    /**
     * Replace the contents of a view (invoked by the layout manager)
     */
    override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
        val item = dataset[position]
        holder.textView.text = context.resources.getString(item.stringResourceId)
    }

    /**
     * Return the size of your dataset (invoked by the layout manager)
     */
    override fun getItemCount() = dataset.size
}
```

Agora que você implementou o `ItemAdapter`, é necessário instruir o `RecyclerView` a usar esse adaptador.

## Modificar a MainActivity para usar um RecyclerView

Para terminar, é necessário usar suas classes `Datasource` e `ItemAdapter` para criar e exibir itens no `RecyclerView`. Faça isso na `MainActivity`.

1. Abra o `MainActivity.kt`
2. Na `MainActivity`, acesse o método `onCreate()`. Insira o novo código descrito nas etapas a seguir após a chamada para `setContentView(R.layout.activity_main)`.
3. Crie uma instância da `Datasource` e chame o método `loadAffirmations()` nela. Armazene a lista retornada de afirmações em uma `val` com o nome `myDataset`.

```
val myDataset = Datasource().loadAffirmations()
```

4. Crie uma variável com o nome `recyclerView` e use `findViewById()` para encontrar uma referência ao `RecyclerView` no layout.

```
val recyclerView = findViewById<RecyclerView>(R.id.recycler_view)
```

5. Para instruir o `recyclerView` a usar a classe `ItemAdapter` que você criou, crie uma nova instância `ItemAdapter`. O `ItemAdapter` espera dois parâmetros: o contexto (`this`) da atividade e as afirmações em `myDataset`.
6. Atribua o objeto `ItemAdapter` à propriedade `adapter` do `recyclerView`.

```
recyclerView.adapter = ItemAdapter(this, myDataset)
```

7. Como o tamanho do layout do `RecyclerView` é fixo no layout da atividade, é possível definir o parâmetro `setHasFixedSize` do `RecyclerView` como `true`. Essa configuração só é necessária para melhorar o desempenho. Use essa configuração se você souber que as mudanças no conteúdo não mudarão o tamanho do layout do `RecyclerView`.

```
recyclerView.setHasFixedSize(true)
```

8. Quando terminar, o código para o `MainActivity` ficará assim.

```
MainActivity.kt
```

```
package com.example.affirmations
```

```
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.RecyclerView
import com.example.affirmations.adapter.ItemAdapter
import com.example.affirmations.data.Datasource

class MainActivity : AppCompatActivity() {

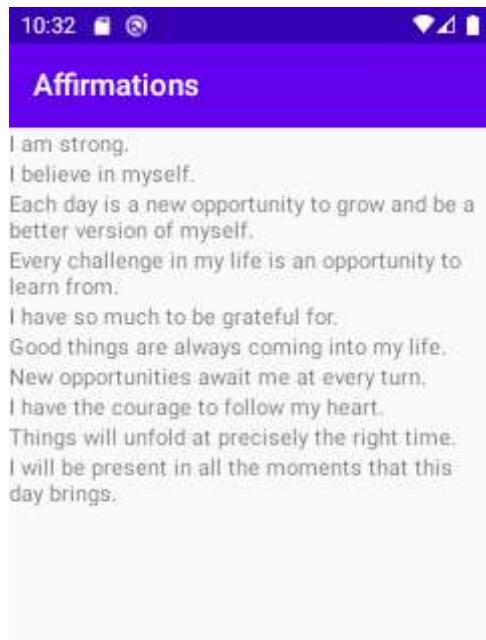
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Initialize data.
        val myDataset = Datasource().loadAffirmations()

        val recyclerView = findViewById<RecyclerView>(R.id.recycler_view)
        recyclerView.adapter = ItemAdapter(this, myDataset)

        // Use this setting to improve performance if you know that changes
        // in content do not change the layout size of the RecyclerView
        recyclerView.setHasFixedSize(true)
    }
}
```

9. Execute o app. Você verá uma lista de strings de afirmações exibidas na tela.



Parabéns! Você acabou de criar um app que exibe uma lista de dados usando um RecyclerView e um adaptador personalizado. Veja o código que você criou e entenda como as diferentes partes funcionam juntas.

Esse app tem todas as partes necessárias para exibir suas afirmações, mas ainda não está pronto para produção. A IU pode melhorar um pouco. No próximo codelab, você melhorará o código, aprenderá a adicionar imagens ao app e deixará a IU mais sofisticada.

[Voltar](#)

## 5. Código da solução

O código da solução para este codelab está no projeto e no módulo mostrados abaixo. Alguns dos arquivos Kotlin estão em pacotes diferentes, conforme indicado pela instrução package no início do arquivo.

res/values/strings.xml

```
<resources>
    <string name="app_name">Affirmations</string>
    <string name="affirmation1">I am strong.</string>
    <string name="affirmation2">I believe in myself.</string>
    <string name="affirmation3">Each day is a new opportunity to grow and be a better
version of myself.</string>
    <string name="affirmation4">Every challenge in my life is an opportunity to learn
from.</string>
    <string name="affirmation5">I have so much to be grateful for.</string>
    <string name="affirmation6">Good things are always coming into my life.</string>
    <string name="affirmation7">New opportunities await me at every turn.</string>
    <string name="affirmation8">I have the courage to follow my heart.</string>
    <string name="affirmation9">Things will unfold at precisely the right time.</string>
    <string name="affirmation10">I will be present in all the moments that this day
brings.</string>
</resources>
```

affirmations/data/Datasource.kt

```
package com.example.affirmations.data

import com.example.affirmations.R
import com.example.affirmations.model.Affirmation

class Datasource {
```

```
fun loadAffirmations(): List<Affirmation> {
    return listOf<Affirmation>(
        Affirmation(R.string.affirmation1),
        Affirmation(R.string.affirmation2),
        Affirmation(R.string.affirmation3),
        Affirmation(R.string.affirmation4),
        Affirmation(R.string.affirmation5),
        Affirmation(R.string.affirmation6),
        Affirmation(R.string.affirmation7),
        Affirmation(R.string.affirmation8),
        Affirmation(R.string.affirmation9),
        Affirmation(R.string.affirmation10)
    )
}
```

affirmations/model/Affirmation.kt

```
package com.example.affirmations.model

data class Affirmation(val stringResourceId: Int)
```

affirmations/MainActivity.kt

```
package com.example.affirmations

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.RecyclerView
import com.example.affirmations.adapter.ItemAdapter
import com.example.affirmations.data.Datasource

class MainActivity : AppCompatActivity() {

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

        // Initialize data.
        val myDataset = Datasource().loadAffirmations()

        val recyclerView = findViewById<RecyclerView>(R.id.recycler_view)
        recyclerView.adapter = ItemAdapter(this, myDataset)

        // Use this setting to improve performance if you know that changes
```

```
// in content do not change the layout size of the RecyclerView
recyclerView.setHasFixedSize(true)
}

}
```

affirmations/adapter/ItemAdapter.kt

```
package com.example.affirmations.adapter

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.affirmations.R
import com.example.affirmations.model.Affirmation

/**
 * Adapter for the [RecyclerView] in [MainActivity]. Displays [Affirmation] data object.
 */
class ItemAdapter(
    private val context: Context,
    private val dataset: List<Affirmation>
) : RecyclerView.Adapter<ItemAdapter.ItemViewHolder>() {

    // Provide a reference to the views for each data item
    // Complex data items may need more than one view per item, and
    // you provide access to all the views for a data item in a view holder.
    // Each data item is just an Affirmation object.
    class ItemViewHolder(private val view: View) : RecyclerView.ViewHolder(view) {
        val textView: TextView = view.findViewById(R.id.item_title)
    }

    /**
     * Create new views (invoked by the layout manager)
     */
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
        // create a new view
        val adapterLayout = LayoutInflater.from(parent.context)
            .inflate(R.layout.list_item, parent, false)

        return ItemViewHolder(adapterLayout)
    }
}
```

```
/**  
 * Replace the contents of a view (invoked by the layout manager)  
 */  
override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {  
    val item = dataset[position]  
    holder.textView.text = context.resources.getString(item.stringResourceId)  
}  
  
/**  
 * Return the size of your dataset (invoked by the layout manager)  
 */  
override fun getItemCount() = dataset.size  
}
```

src/main/res/layout/activity\_main.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".MainActivity">  
  
    <androidx.recyclerview.widget.RecyclerView  
        android:id="@+id/recycler_view"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent"  
        android:scrollbars="vertical"  
        app:layoutManager="LinearLayoutManager" />  
  
</FrameLayout>
```

src/main/res/layout/list\_item.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<TextView xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/item_title"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

[Voltar](#)

## 6. Resumo

- O widget RecyclerView ajuda a exibir uma lista de dados.

- O RecyclerView usa o padrão do adaptador para adaptar e exibir os dados.
- O ViewHolder cria e armazena as visualizações para o RecyclerView.
- O RecyclerView vem com um LayoutManagers integrado. O RecyclerView delega como os itens são dispostos no LayoutManagers.

Para implementar o adaptador:

- Crie uma nova classe para o adaptador, por exemplo, ItemAdapter
- Crie uma classe ViewHolder personalizada que represente uma única visualização do item da lista. Estenda-a da classe RecyclerView.ViewHolder.
- Modifique a classe ItemAdapter para ser estendida da classe RecyclerView.Adapter com a classe ViewHolder personalizada.
- Implemente estes métodos no adaptador: getItemCount(), onCreateViewHolder() e onBindViewHolder().

[Voltar](#)

## 7. Saiba mais

- [Criar uma lista com RecyclerView](#)
- [classe RecyclerView](#)
- [RecyclerView.Adapter](#)
- [RecyclerView.ViewHolder](#)
- [Biblioteca RecyclerView](#)
- [Listas](#) no Material Design (link em inglês)
- [Melhorar a IU com o MaterialCardView e imagens](#)

[Voltar](#)