SUMÁRIO

[1. Pré-Requisitos 2](#_Toc503270670)

[2. Objetivo 3](#_Toc503270671)

[3. Versão 4](#_Toc503270672)

[3.1. minSdkVersion 4](#_Toc503270673)

[3.2. targetSdkVersion 4](#_Toc503270674)

[4. RecyclerView 5](#_Toc503270675)

[4.1. Cenário 5](#_Toc503270676)

[5. Novo Projeto 6](#_Toc503270677)

[6. Passo-a-Passo 8](#_Toc503270678)

[6.1. Instalando 8](#_Toc503270679)

[6.2. Definindo o modelo 8](#_Toc503270680)

[6.2.1. Modelo 8](#_Toc503270681)

[6.2.2. DAO 12](#_Toc503270682)

[6.3. Criando o RecyclerView no Layout 15](#_Toc503270683)

[6.4. Criando nossa linha personalizada 16](#_Toc503270684)

[6.5. Criando o RecyclerView.Adapter 17](#_Toc503270685)

[6.5.1. FilmeAdapter 19](#_Toc503270686)

[6.5.2. FilmeViewHolder 20](#_Toc503270687)

[6.6. Vinculando o nosso adapter ao nosso RecyclerView 21](#_Toc503270688)

[7. Conteúdo adicional 22](#_Toc503270689)

[7.1. Selecionando apenas um item da lista 22](#_Toc503270690)

[7.1.1. OnClickListener 22](#_Toc503270691)

[7.1.2. View.LongOnClickListener 25](#_Toc503270692)

[8. Resumo 33](#_Toc503270693)

[9. Referências 34](#_Toc503270694)

# Pré-Requisitos

* Java (Orientação a Objetos, APIs e Bibliotecas – Comparable, List, ArrayList);
* Básico de Android (Activity, View);
* Recomendação:
  + ListView.

# Objetivo

É comum que aplicativos desejem mostrar listas para o usuário de maneira eficiente. Para isto, iremos utilizar e mostrar as funcionalidades do RecyclerView. Ele é uma evolução do ListView.

# Versão

## minSdkVersion

15

## targetSdkVersion

26

# RecyclerView

O *RecyclerView* é uma “evolução” das listas *ListView* e *GridView*.

Uma das vantagens do RecyclerView é que ele é constantemente atualizado. Porém, como ela não está presente diretamente na SDK do Android (como no caso da ListView, por exemplo), nós precisamos incluir essa biblioteca no nosso *gradle*.

Como o nome do componente sugere (*Recycler* é Reciclar em inglês) quando o usuário descer/subir a lista o componente identifica as *views* que não estão mais visíveis para o usuário e as **reutiliza** colocando novos valores. Fazendo isso evita-se criar novas *views* para cada novo conteúdo. O objetivo é reduzir o tempo e custo, reaproveitando objetos.

O usuário pode também definir sua própria animação ao remover ou adicionar um novo conteúdo na lista. Além disso, podemos definir nossa lista horizontal, vertical, grade sem a necessidade de recriar todo o nosso RecyclerView.

## Cenário

Sendo assim, olhando o cenário, temos os seguintes componentes:

RecyclerView: irá posicionar a lista na tela (interface) para o usuário.

LayoutManager: Identificamos se os itens serão mostrados horizontalmente, verticalmente.

Adapter: associar o conteúdo à view. Exibição dos nossos itens na lista.

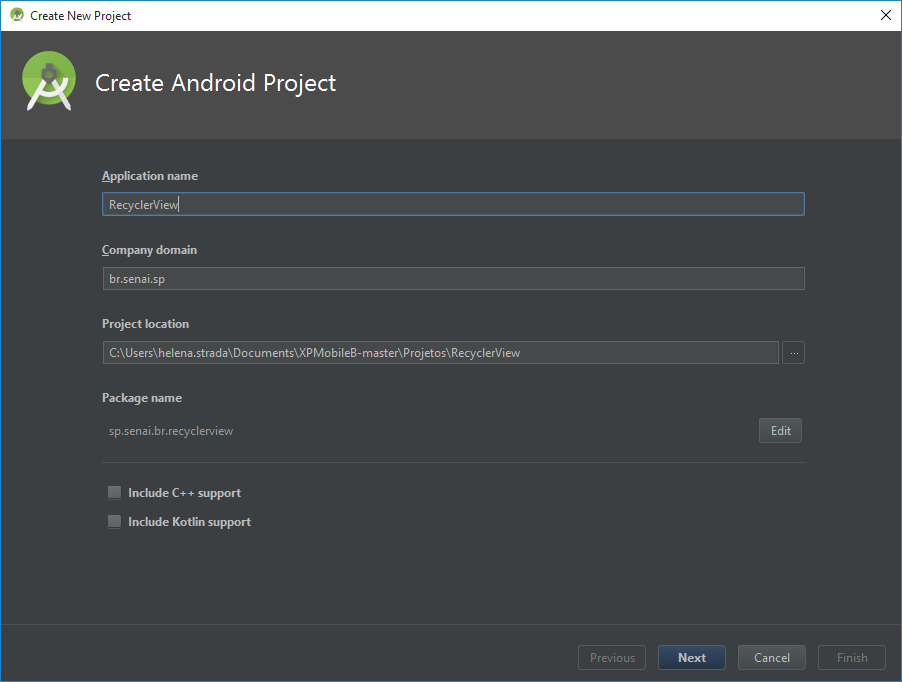
ViewHolder: precisamos mostrar cada item da nossa lista.



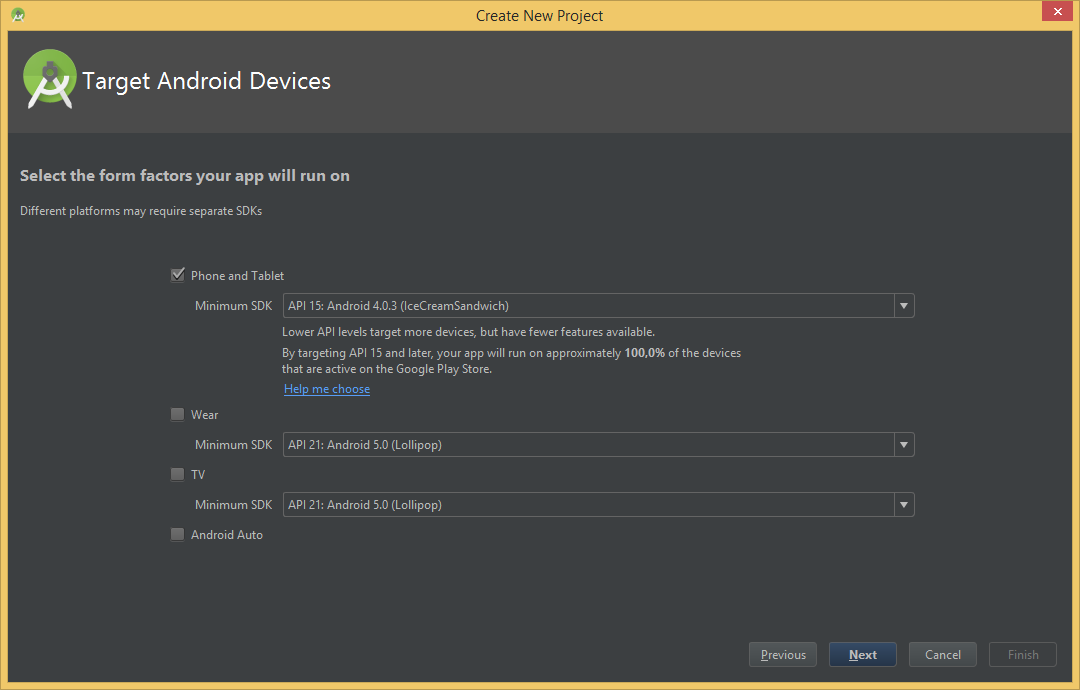
<https://developer.android.com/training/material/lists-cards.html>

# Novo Projeto

Iremos criar um novo projeto chamado RecyclerView.

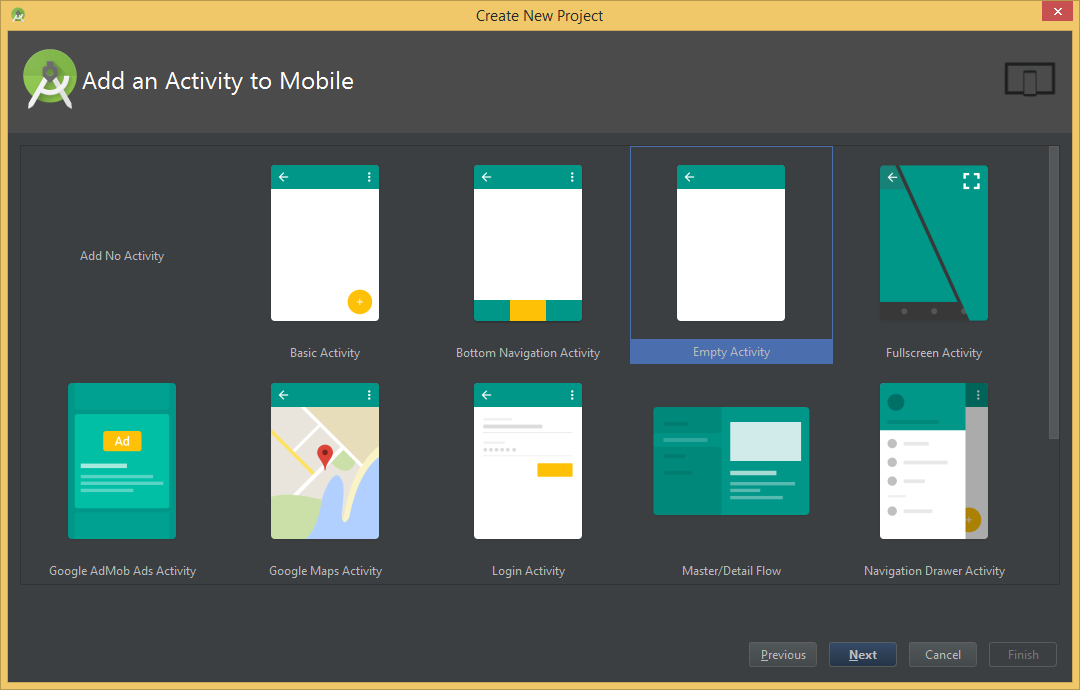


Clicar em “*Next*”.



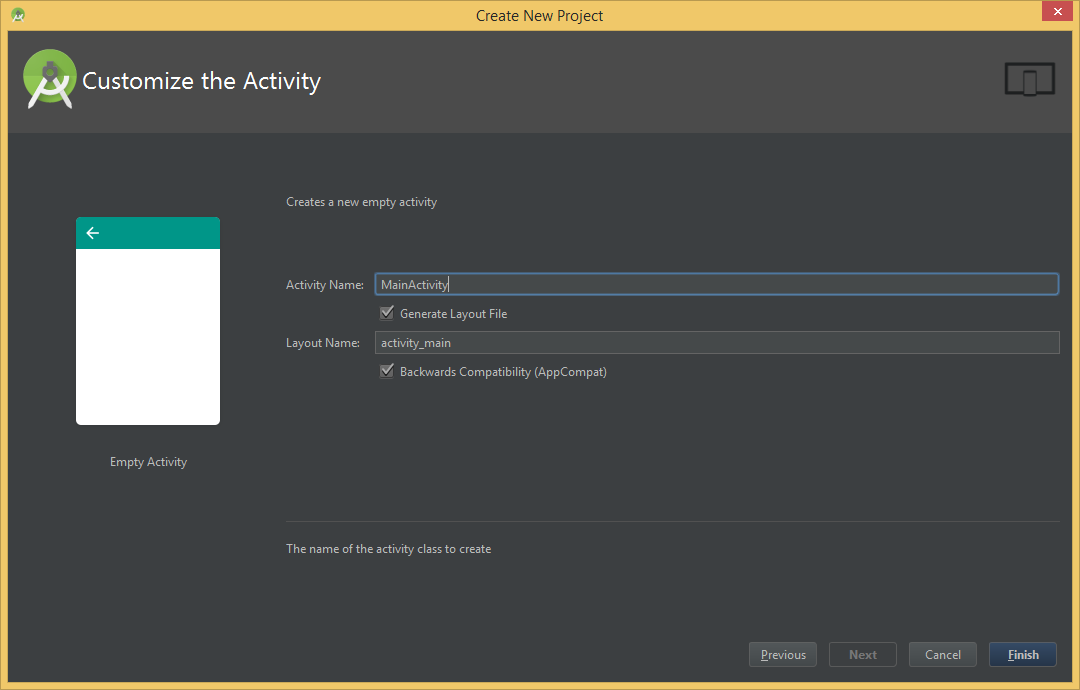
Clicar em “*Next*”.

Utilizaremos uma *Empty Activity* apenas para mostrarmos como *RecyclerView* funciona.



Clicar em *“Next”*.

O nome da activity será o padrão: *MainActivity*.



Clicar em “*Finish*”.

# Passo-a-Passo

Precisamos de alguns passos chaves para utilizarmos o RecyclerView:

* Adicionar o suporte da biblioteca do RecyclerView em nosso projeto (arquivo do gradle);
* Definirmos as nossas classes de modelo (model, dao, db - listas);
* Adicionar o nosso RecyclerView em nossa activity para mostrarmos os itens da lista anterior;
* Criarmos uma linha personalizada no nosso layout XML para visualizarmos o item da lista;
* Criarmos o nosso RecyclerView.Adapter e o nosso ViewHolder para renderizarmos o item;
* Vincularmos o nosso adapter a nossa fonte de dados para popularmos o RecyclerView.

## Instalando

Como vimos anteriormente, o RecyclerView não é nativo do Android. Sendo assim, precisamos incluir sua biblioteca no nosso *gradle.*



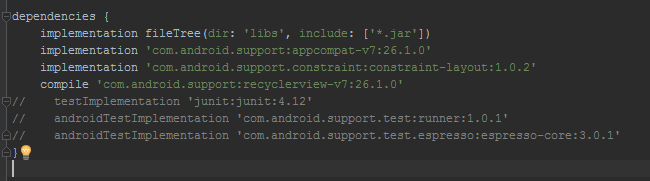


Figura 1 – Dependência do RecyclerView sendo adicionado ao nosso projeto (build.gradle).

## Definindo o modelo

Vamos agora definir o modelo da nossa aplicação, ou seja, vamos definir a fonte de dados da nossa aplicação. No nosso exemplo, estaremos utilizando o modelo de Filme com as propriedades de nome, filme e gênero.

### Modelo

Vamos criar a nossa classe de modelo:

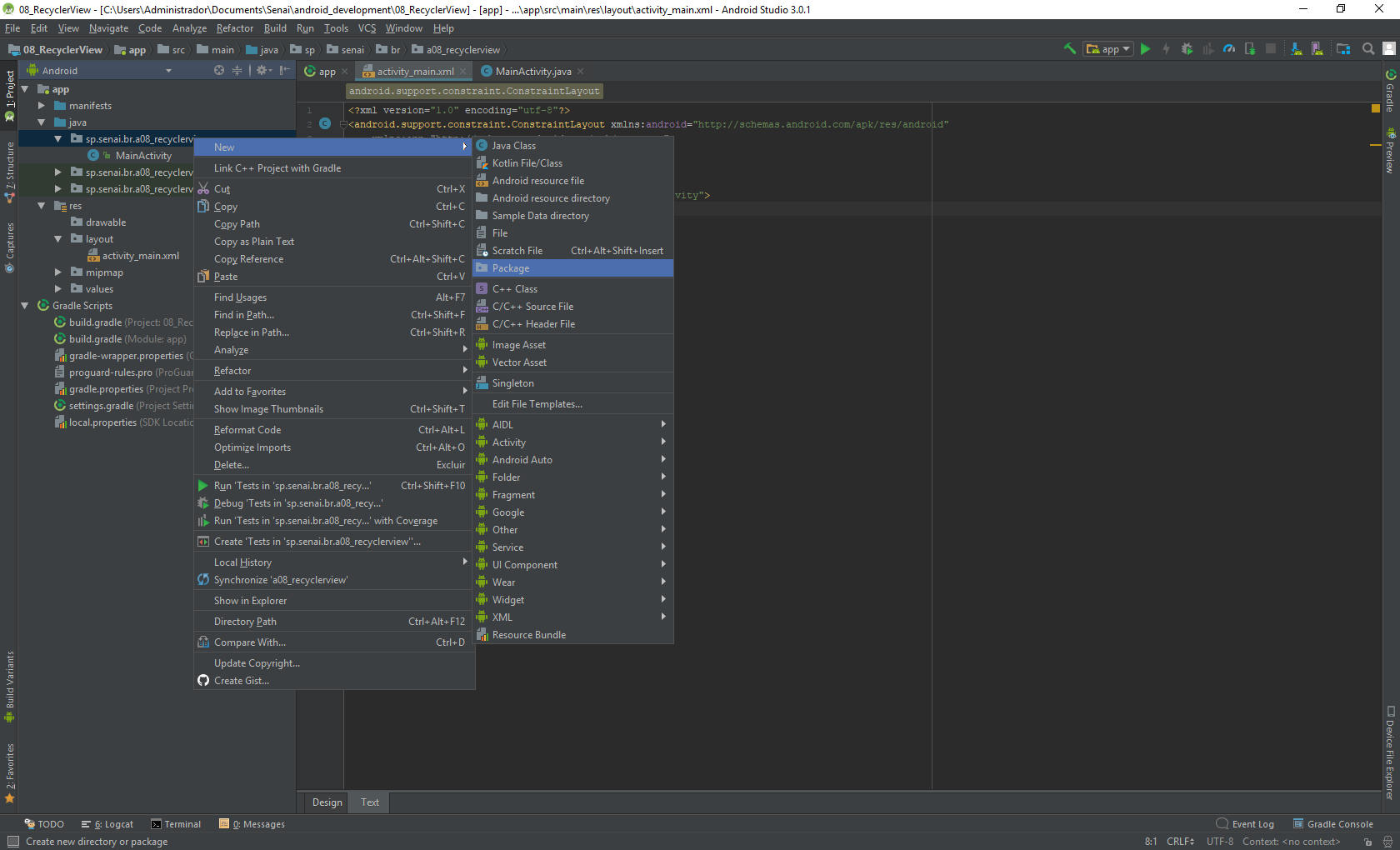


Figura 2 – Criando um novo pacote.

Vamos criar o nosso primeiro pacote chamado “model” e incluir o nosso modelo Filme.

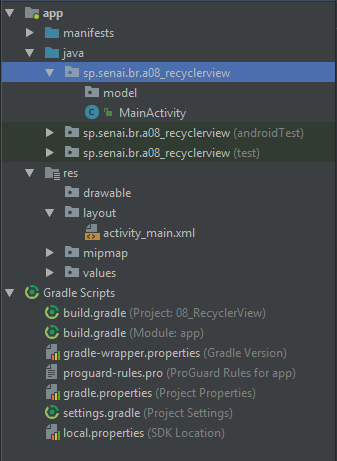


Figura 3 – Estrutura do nosso projeto ao adicionarmos o pacote model.

Iremos criar agora o nosso modelo Filme dentro da pasta model que acabamos de criar.

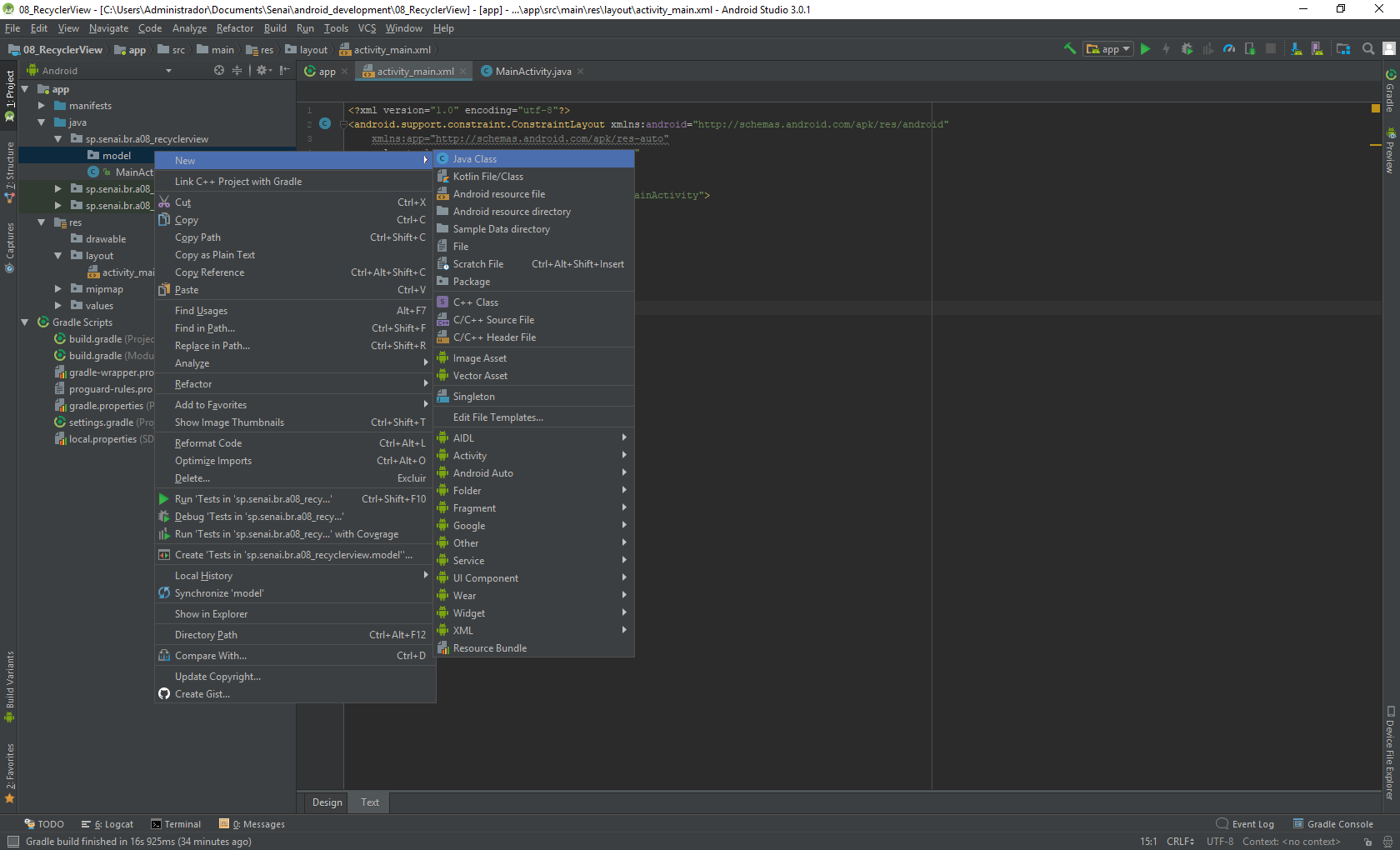


Figura 4 – Criando um novo pacote

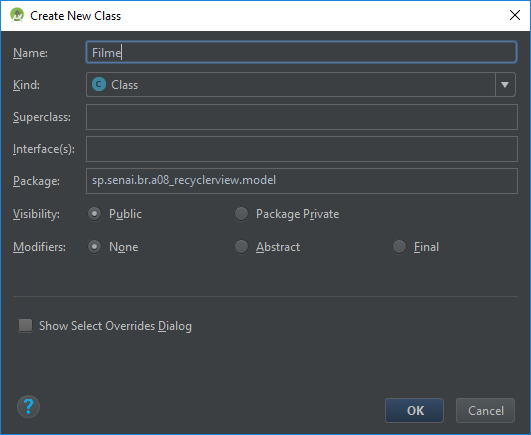


Figura 5 – Inserindo o nome do nosso modelo.

O nosso código para o nosso modelo Filme:

package sp.senai.br.a08\_recyclerview.model;  
  
import android.support.annotation.NonNull;  
  
*/\*\*  
 \* Created by helena.strada on 21/12/2017.  
 \*/*public class Filme implements Comparable<Filme> {  
  
 private Long id;  
 private String nome;  
 private String genero;  
  
 public Filme(Long id, String nome, String genero) {  
 this.id = id;  
 this.nome = nome;  
 this.genero = genero;  
 }  
  
 public Filme(Long id) {  
 this.id = id;  
 }  
  
 public Filme() {  
  
 }  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public String getNome() {  
 return nome;  
 }  
  
 public void setNome(String nome) {  
 this.nome = nome;  
 }  
  
 public String getGenero() {  
 return genero;  
 }  
  
 public void setGenero(String genero) {  
 this.genero = genero;  
 }  
  
 @Override  
 public int compareTo(@NonNull Filme filme) {  
 return nome.toLowerCase().compareTo(filme.nome.toLowerCase());  
 }  
  
 @Override  
 public String toString() {  
 return "Filme{" +  
 "id=" + id +  
 ", nome='" + nome + '\'' +  
 ", genero='" + genero + '\'' +  
 '}';  
 }  
}

### DAO

Vamos criar o nosso FilmeDao e os seus respectivos métodos.

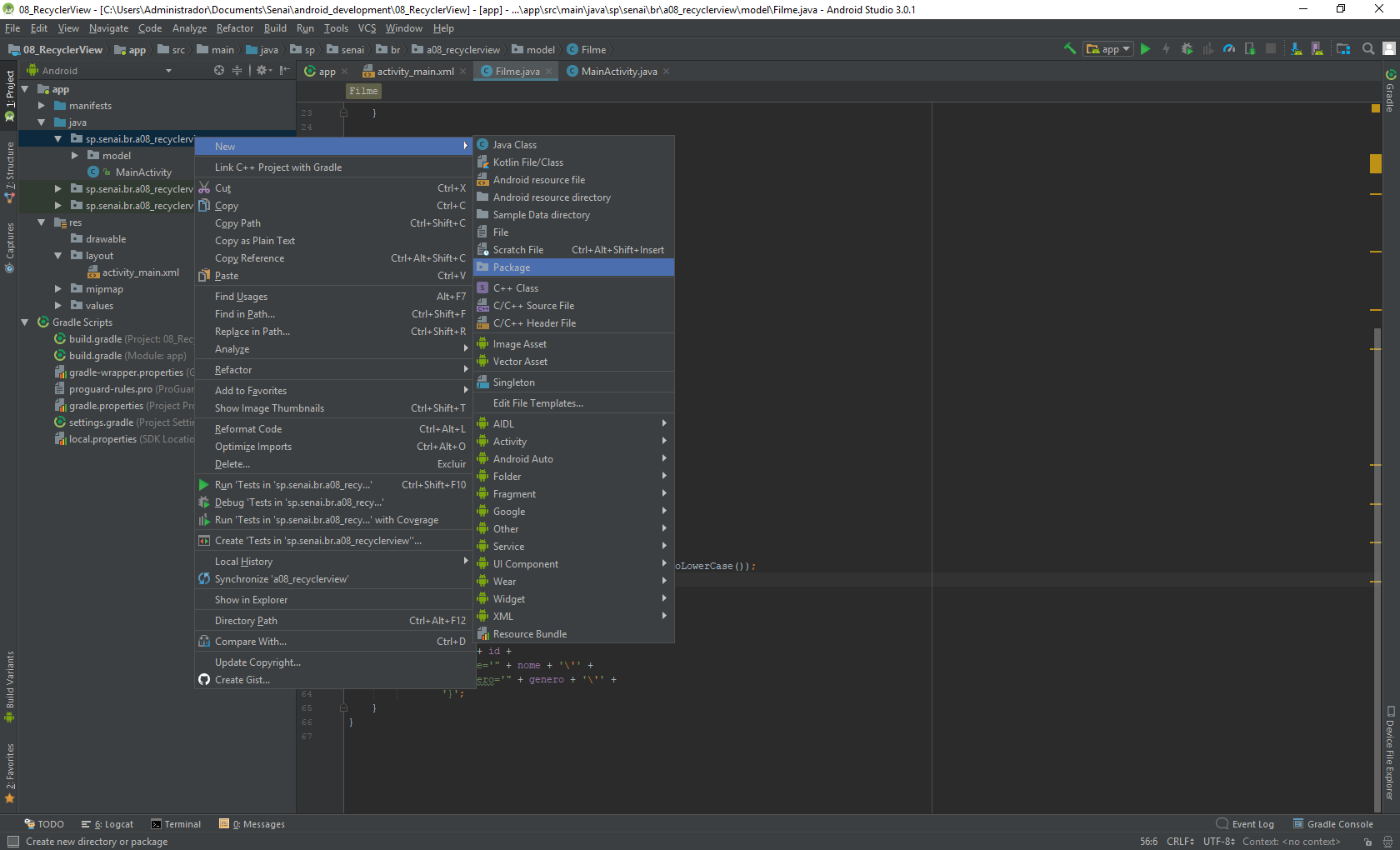


Figura 6 – Criar um novo pacote dao.

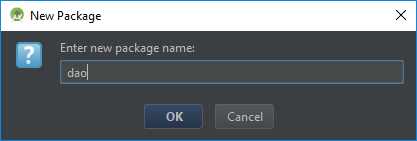


Figura 7 – Colocando o nome do nosso pacote.

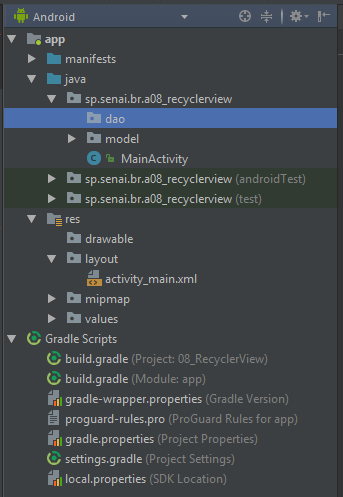


Figura 8 – Estrutura atual do nosso projeto depois de adicionado o pacote dao ao nosso projeto.

Iremos criar o nosso FilmeDao.

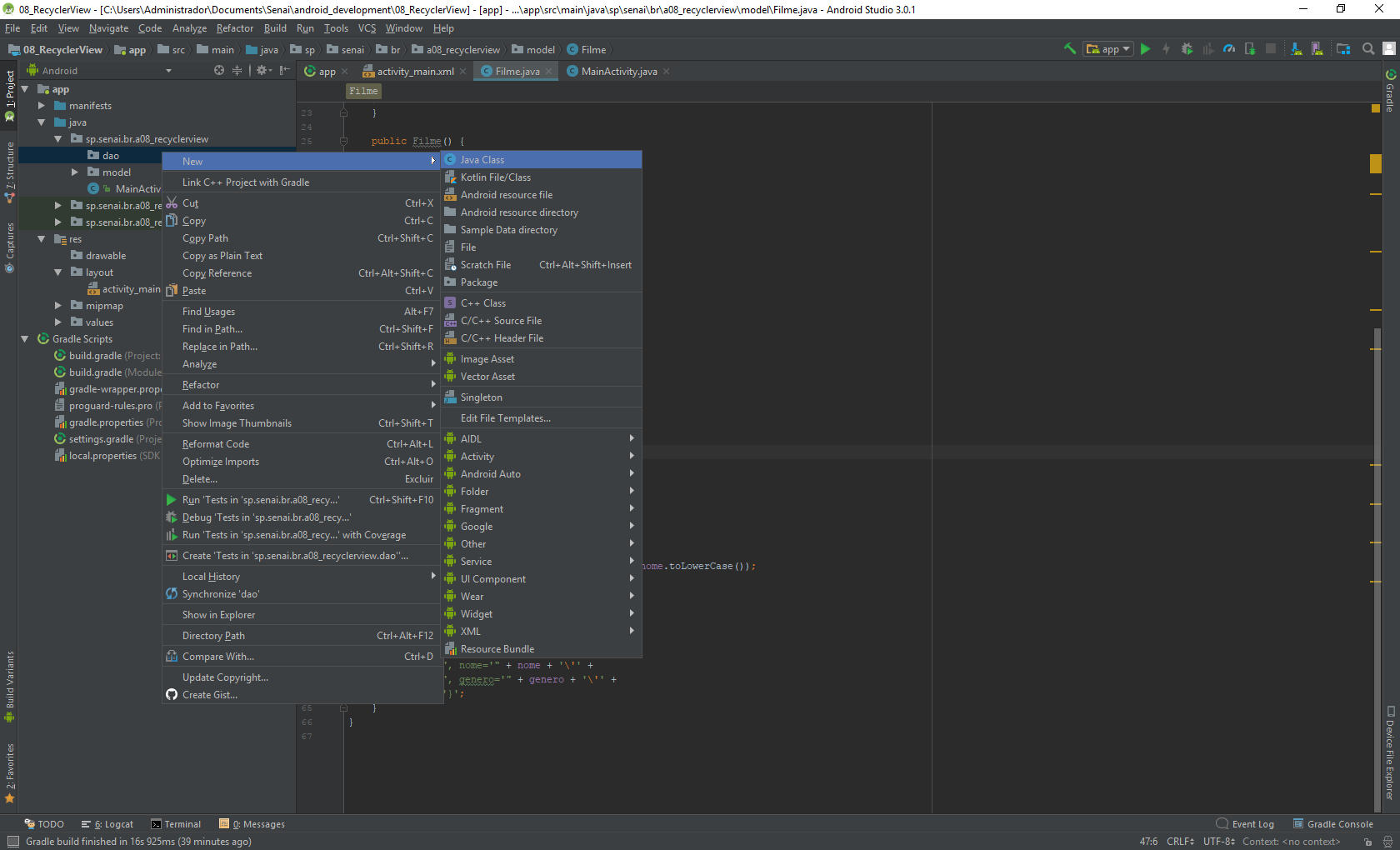


Figura 9 – Inserindo o nosso FIlmeDao.

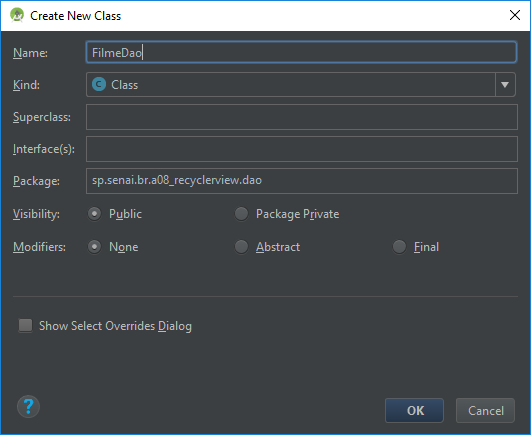


Figura 10 – Escolhendo o nome da nossa classe.

Nosso código da nossa classe FilmeDao ficará assim:

*/\*\*  
 \* Created by helena.strada on 21/12/2017.  
 \*/*public class FilmeDao {  
  
 public static FilmeDao *manager* = new FilmeDao();  
  
 // Lista aonde serão armazenados os filmes  
 private List<Filme> lista;  
  
 // Geração do id para cada novo filme. No nosso caso, teremos somente uma lista para exemplo.  
 private long id = 0;  
  
 private FilmeDao() {  
 lista = new ArrayList<>();  
 lista.add(new Filme(id++, "StarWars", "Luta"));  
 lista.add(new Filme(id++, "Final Fantasy XII", "RPG"));  
 }  
  
 public List<Filme> getLista() {  
  
 Collections.*sort*(lista);  
 return Collections.*unmodifiableList*(lista);  
  
 }  
  
}

## Criando o RecyclerView no Layout

Dentro da activity que desejamos incluir o nosso RecyclerView, no nosso caso a res/layout/activity\_main.xml, vamos adicionar o RecyclerView que agora temos suporte da nossa biblioteca.

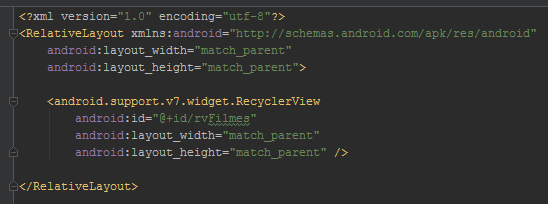


Figura 11 - Adicionar o nosso RecyclerView dentro da nossa activity\_main.xml

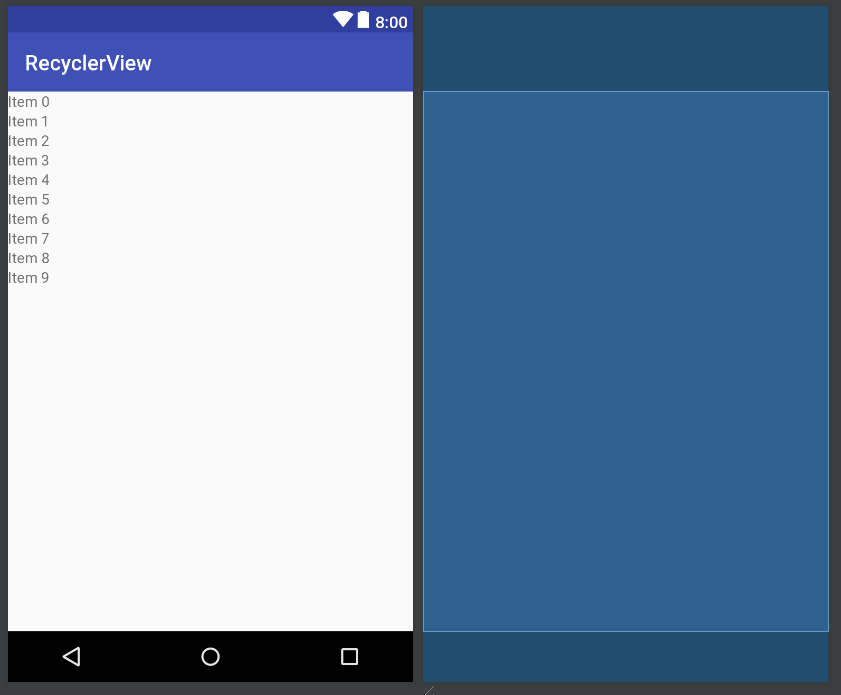


Figura 12 – RecyclerView dentro da nossa activity.

## Criando nossa linha personalizada

Antes de criarmos o adapter, vamos definir o nosso arquivo XML que será utilizado para cada item da nossa lista. Colocaremos para item da nossa lista, o nome do filme e o gênero.

Vamos criar esse layout com o nome de “*filme\_item\_lista.xml*” dentro da pasta res/layout. Ele será um LinearLayout.

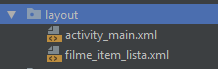
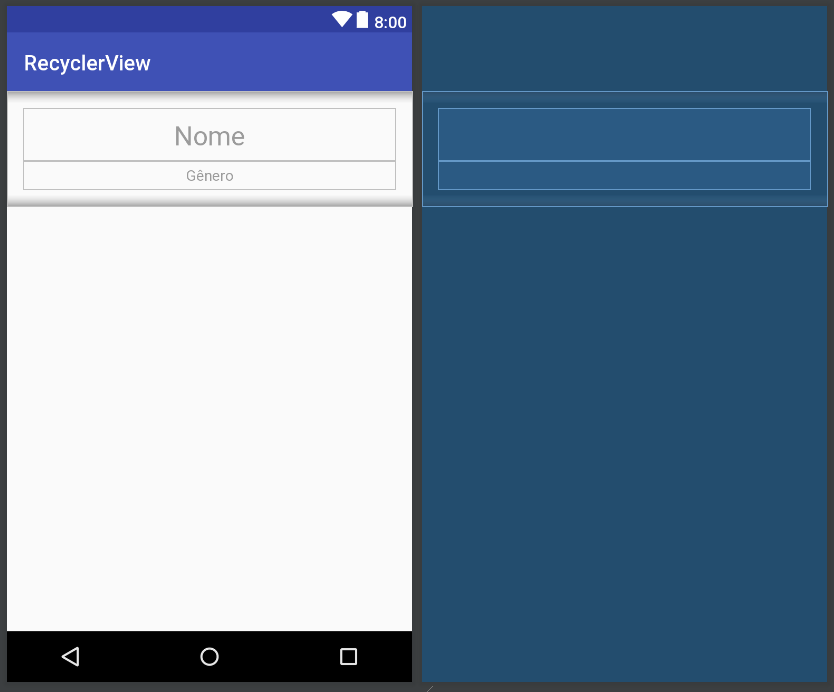


Figura 13 – O layout que será renderizado para cada item da lista.

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:background="?android:selectableItemBackground"  
 android:clickable="true"  
 android:focusable="true"  
 android:foreground="?android:attr/selectableItemBackground"  
 android:orientation="vertical"  
 android:padding="16dp">  
  
 <TextView  
 android:id="@+id/tvNome"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Nome"  
 android:padding="8dp"  
 android:textAlignment="center"  
 android:textSize="25dp" />  
  
 <TextView  
 android:id="@+id/tvGenero"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Gênero"  
 android:padding="4dp"  
 android:textAlignment="center" />  
  
</LinearLayout>



Agora que criamos o nosso layout personalizado para os itens da nossa lista, vamos criar o adapter para popular com os dados e preenchermos no nosso RecyclerView.

## Criando o RecyclerView.Adapter

Precisamos criar o nosso adapter que irá de fato popular os dados para o RecyclerView. No entanto, ele requer a existência de um objeto “*ViewHolder*” que irá descrever e fornecer o acesso a cada item da nossa lista.

Vamos criar uma pasta chamada view dentro de java/nome\_do\_pacote. E adicionar mais duas pastas dentro desse pacote, *adapter* e *holder*.

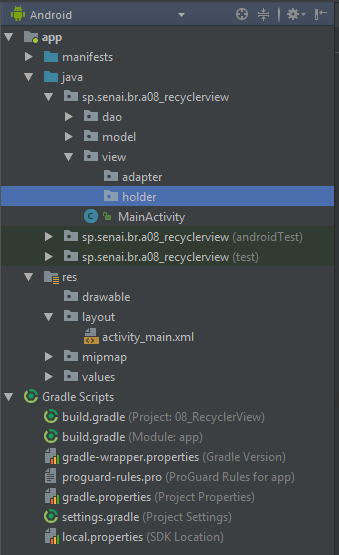


Figura – Estrutura atual do projeto depois de criada as pastas correspondentes: view (adapter, holder).

Além disso, precisamos criar agora as nossas duas classes. FilmeAdapter dentro da pasta *adapter* e FilmeViewHolder dentro da pasta *Holder*.

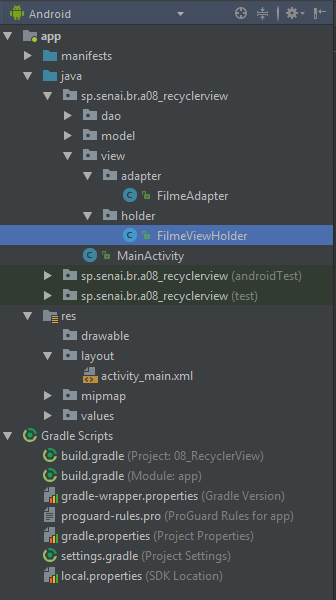


Figura – Estrutura atual do projeto.

### FilmeAdapter

Todo adapter possui três métodos principais: o *onCreateViewHolder* para “inflar” o item que desejamos. *onBindViewHolder* para “settar” os atributos da view baseado nos dados de cada filme. E o *getItemCount* para determinar o número de itens.

*/\*\*  
 \* Created by helena.strada on 22/12/2017.  
 \*/*public class FilmeAdapter extends RecyclerView.Adapter {  
  
 private List<Filme> filmes;  
 private Context context;  
  
 public FilmeAdapter(List<Filme> filmes, Context context) {  
 this.filmes = filmes;  
 this.context = context;  
 }  
  
 @Override  
 public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {  
 View view = LayoutInflater.*from*(context)  
 .inflate(R.layout.*filme\_item\_lista*, parent, false);  
 FilmeViewHolder holder = new FilmeViewHolder(view, this);  
 return holder;  
 }  
  
 @Override  
 public void onBindViewHolder(RecyclerView.ViewHolder holder, int position) {  
  
 FilmeViewHolder viewHolder = (FilmeViewHolder) holder;  
  
 Filme filme = filmes.get(position);  
  
 ((FilmeViewHolder) holder).preencher(filme);  
  
 }  
  
 @Override  
 public int getItemCount() {  
 return filmes.size();  
 }  
}

### FilmeViewHolder

Nosso código do ViewHolder.

*/\*\*  
 \* Created by helena.strada on 22/12/2017.  
 \*/*public class FilmeViewHolder extends RecyclerView.ViewHolder {  
  
 public final TextView nome;  
 public final TextView genero;  
 private Long filmeId;  
 public final FilmeAdapter adapter;  
  
 public FilmeViewHolder(final View view, final FilmeAdapter adapter) {  
 super(view);  
 this.adapter = adapter;  
 nome = view.findViewById(R.id.*tvNome*);  
 genero = view.findViewById(R.id.*tvGenero*);  
 }  
  
 public void preencher(Filme filme) {  
 filmeId = filme.getId();  
 nome.setText(filme.getNome());  
 genero.setText(filme.getGenero());  
 }  
}

## Vinculando o nosso adapter ao nosso RecyclerView

Na nossa classe principal, precisamos apenas inicializar a nossa lista de filmes e vincular o nosso adapter ao nosso RecyclerView.

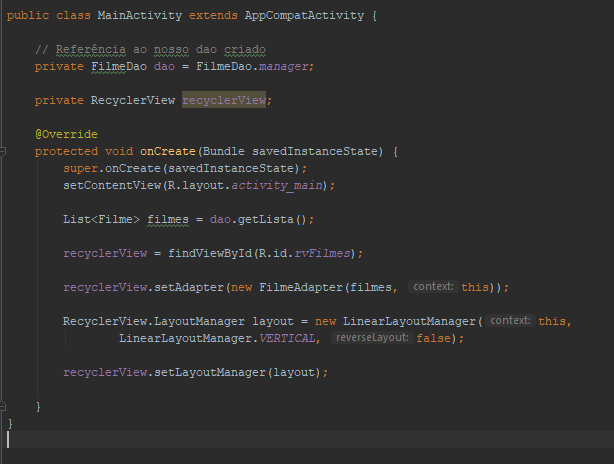


Figura 18 – MainActivity.java com o código completo.

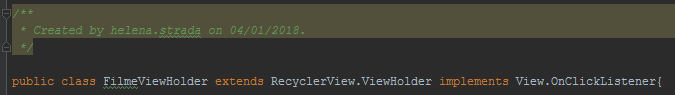
# Conteúdo adicional

## Selecionando apenas um item da lista

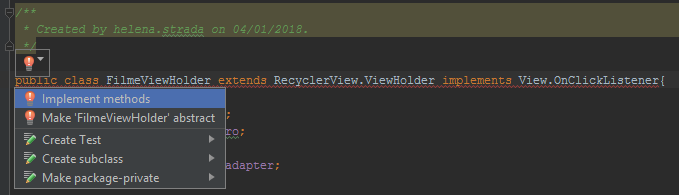
### OnClickListener

Complementando o nosso projeto, caso quiséssemos selecionar apenas um item da nossa lista, iremos realizar essa ação através do *View.OnClickListener*.

No nosso FilmeViewHolder iremos implementar a classe descrita acima.



Ele irá pedir para implementarmos os métodos, uma vez que estamos implementando uma interface.



Além de colocarmos a ação que desejamos no nosso método, iremos também no nosso construtor, colocarmos a chamada ao nosso método.

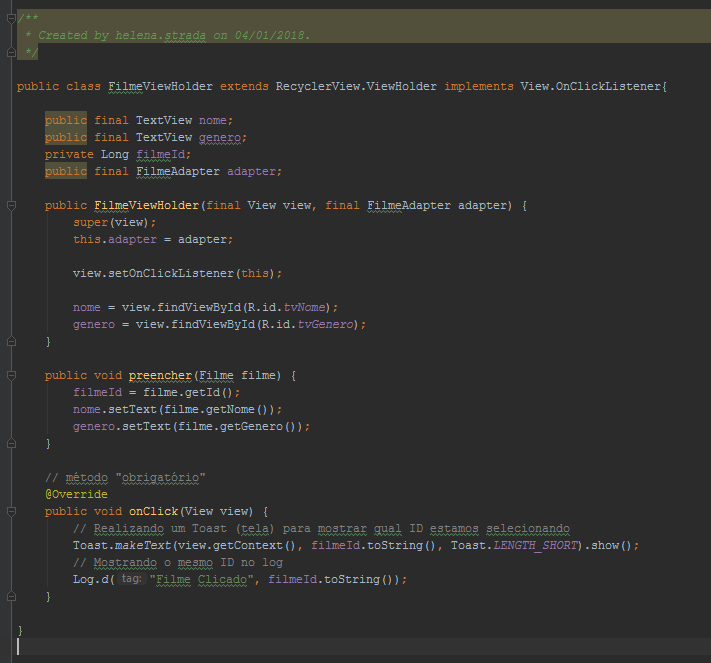


Figura 19 – Classe final FilmeViewHolder

Este método serve apenas para clicarmos no item e ele mostrar o id do item selecionado.

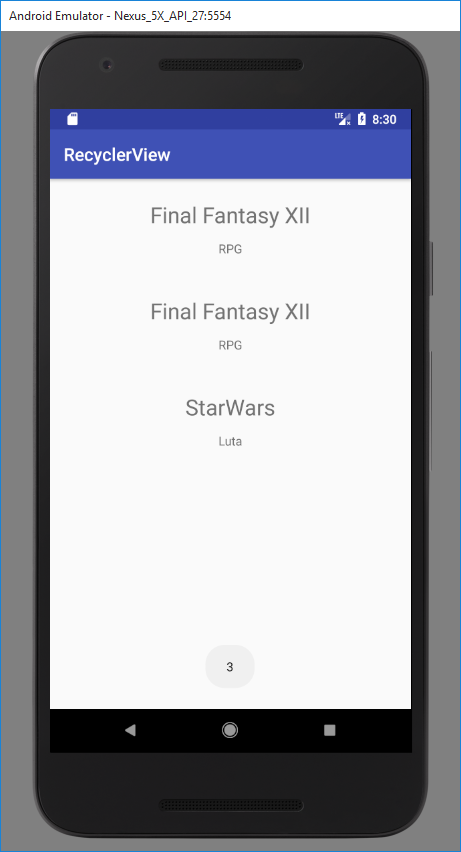
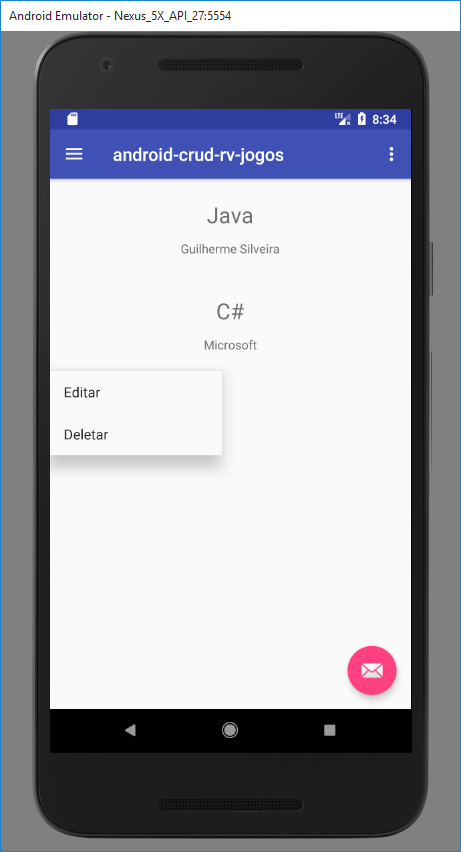


Figura 20 – Item selecionado da nossa lista.

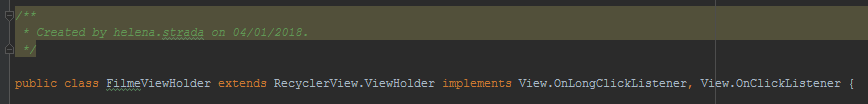
E se, além de dermos um clique no nosso item e apenas mostrar o ID, por exemplo, nós quiséssemos trabalhar com um clique longo no item?



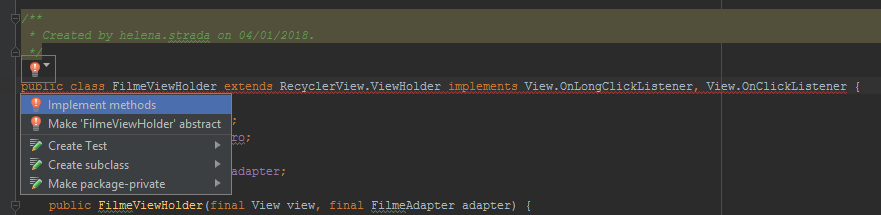
Para isto, utilizamos o *View.LongOnClickListener*.

### View.LongOnClickListener

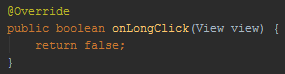
Iremos implementar a interface View.LongOnClickListener.



Assim como o exemplo anterior, ele irá solicitar para implementar os métodos obrigatórios.



Após implementarmos o método.



Além disso, precisamos incluir no nosso construtor, o clique dele.

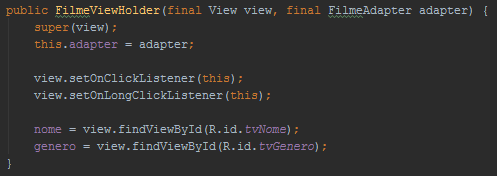
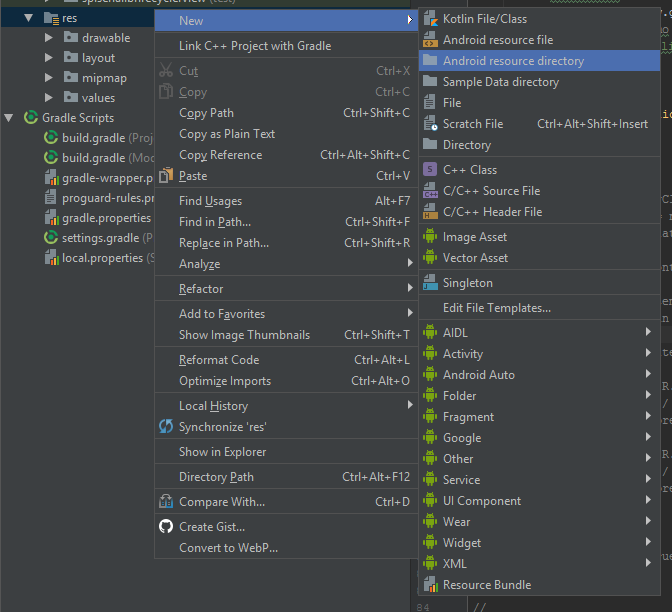


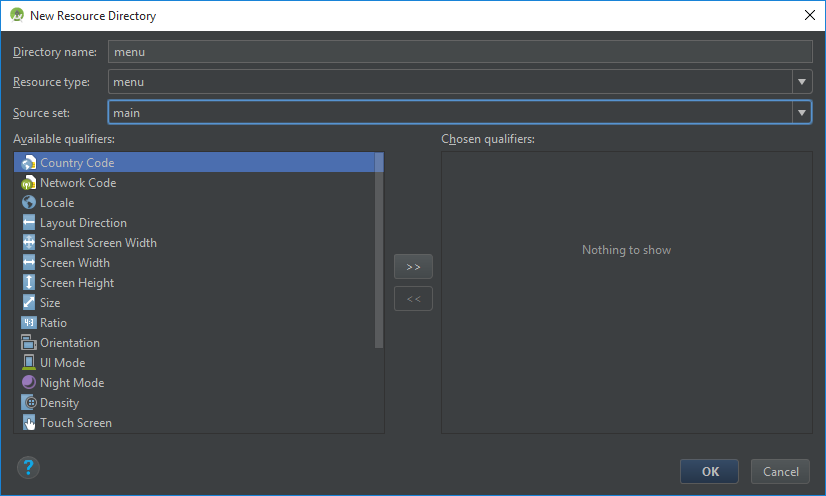
Figura 21 – Inserir o view.setOnLongClickListener(this).

Mas além de apenas utilizarmos o OnLongClickListener, precisamos também criar um menu correspondente para o nosso item.

Para isto, iremos em res -> New -> Android resource directory.

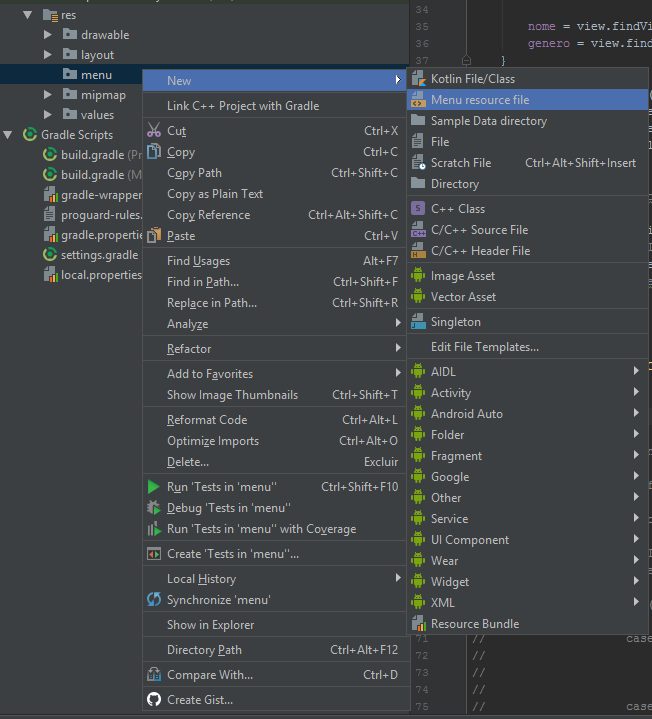


Iremos selecionar o tipo de diretório que queremos criar.



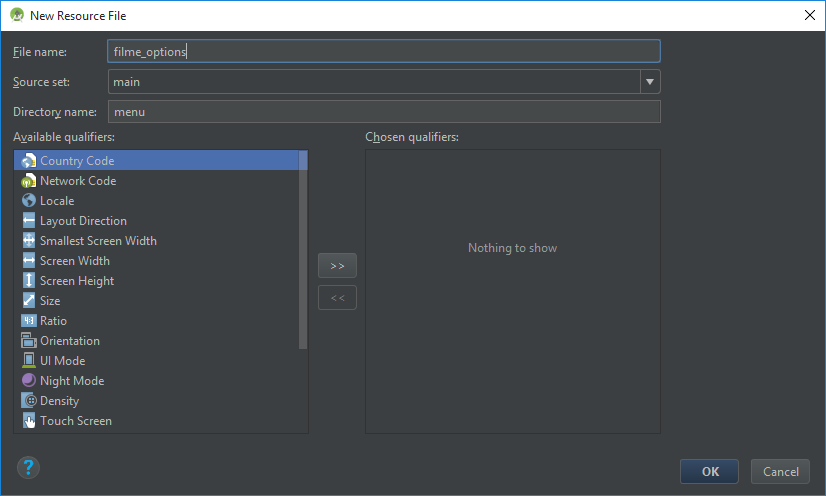
Clicar em “*Ok*”.

Iremos agora criar um novo menu para o nosso exemplo.



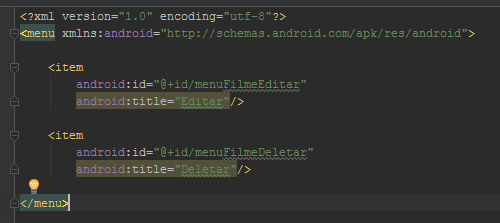
Res -> menu -> New -> Menu resource file

Iremos definir o nome de “*filme\_options*”.



Clicar em “*Ok*”.

No novo arquivo que foi criado (filme\_options.xml) dentro da pasta menu, iremos editar este arquivo e colocar as informações que seguem.



Estaremos criando um menu com 2 (dois) itens. O primeiro item que será o de edição e o segundo que será de exclusão.

Precisamos vincular o nosso menu ao nosso clique longo.

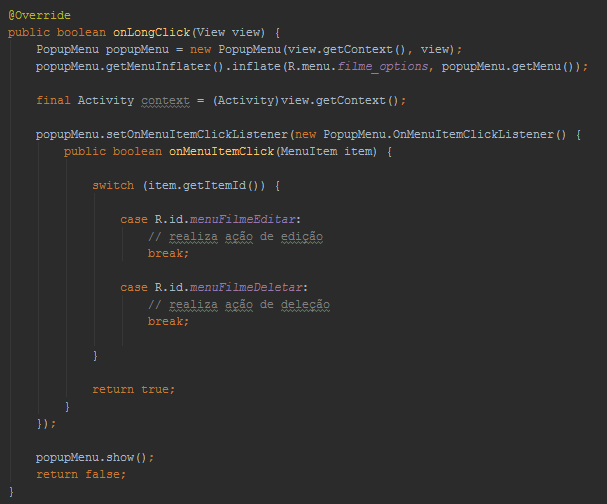


Figura 22 – Ao realizar um LongClick, queremos abrir o nosso menu de opções para o usuário.

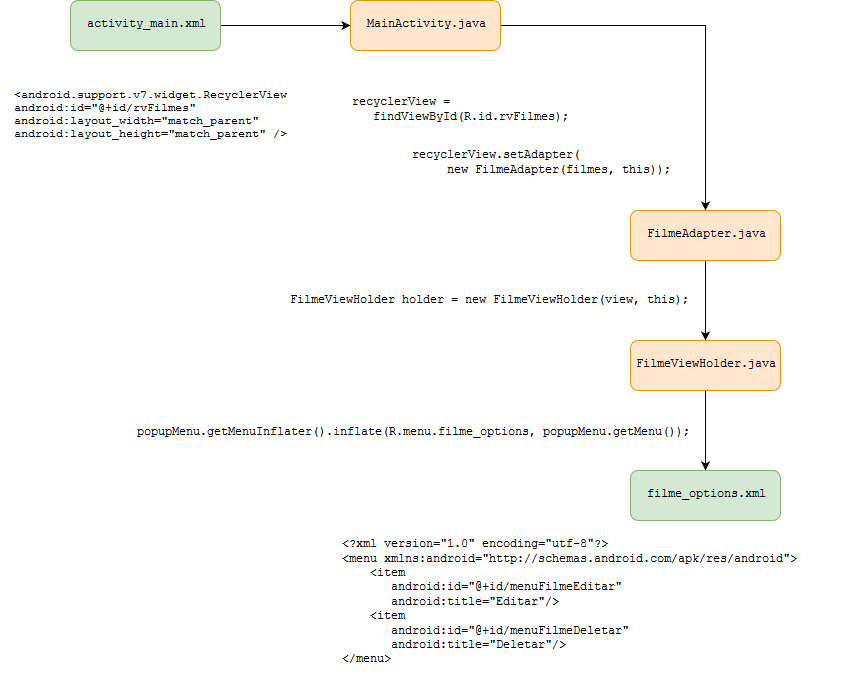


Figura 23 – Esboço da estrutura do Projeto

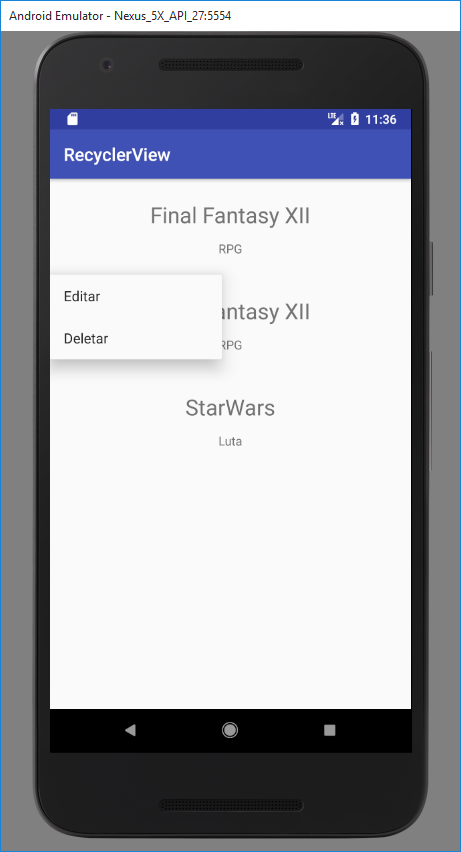


Figura 24 – Projeto Finalizado

# Resumo

Nessa apostila vimos como criar um novo RecyclerView, a utilização de um adapter e como associarmos os dados do nosso modelo ao nosso ViewHolder. Além disso, aprendemos como selecionar um item da nossa lista e ao realizar um clique longo no item, aprendemos como abrir um menu de opções para o usuário.

# Referências

<https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html>

<http://blog.alura.com.br/criando-listas-com-recyclerview/>

<https://developer.android.com/training/material/lists-cards.html?hl=pt-br>

<https://medium.com/android-dev-br/listas-com-recyclerview-d3f41e0d653c>

<https://github.com/codepath/android_guides/wiki/Using-the-RecyclerView>