

SUMÁRIO

1. Informativos.....	2
2. Pré-Requisitos.....	3
3. Objetivo.....	4
4. Versão.....	5
4.1. minSdkVersion.....	5
4.2. targetSdkVersion.....	5
5. RecyclerView.....	6
5.1. Cenário.....	6
6. Novo Projeto.....	7
7. Passo-a-Passo.....	9
7.1. Instalando.....	9
7.2. Definindo o modelo.....	9
7.2.1. Modelo.....	9
7.2.2. DAO.....	13
7.3. Criando o RecyclerView no Layout.....	16
7.4. Criando nossa linha personalizada.....	17
7.5. Criando o RecyclerView.Adapter.....	18
7.5.1. FilmeAdapter.....	20
7.5.2. FilmeViewHolder.....	21
7.6. Vinculando o nosso adapter ao nosso RecyclerView.....	22
8. Conteúdo adicional.....	23
8.1. Selecionando apenas um item da lista.....	23
8.1.1. OnClickListener.....	23
8.1.2. View.LongOnClickListener.....	26
9. Resumo.....	34
9.1. Projetos.....	34
10. Referências.....	35

1. Informativos

Autor(a): Helena Strada

Data de criação: jan/2018

Escola Senai de Informática

CodeXP - Mobile

2. Pré-Requisitos

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

3. 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.

4. Versão

4.1. minSdkVersion

15

4.2. targetSdkVersion

26

5. 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*.

5.1. Cenário

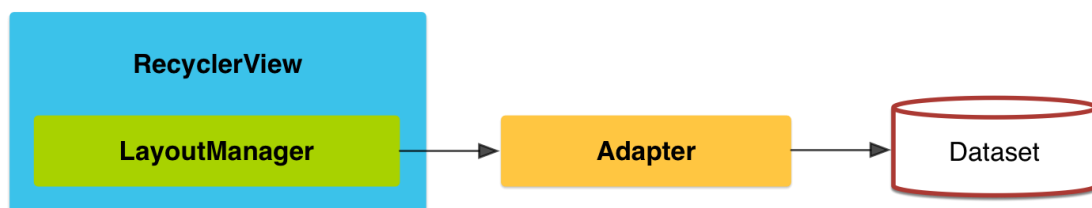
Sendo assim, olhando o cenário, temos os seguintes componentes:

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.

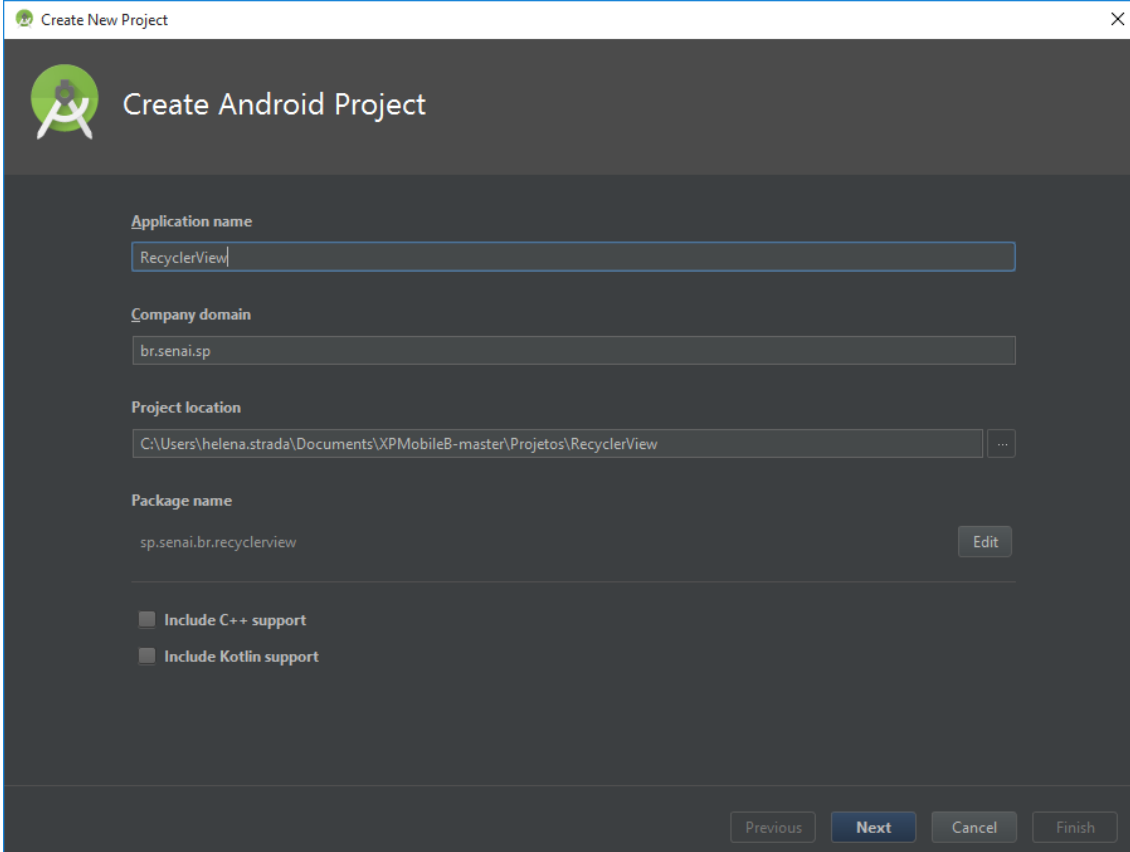
ViewHolder: precisamos mostrar cada item da nossa lista.



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

6. Novo Projeto

Iremos criar um novo projeto chamado RecyclerView.

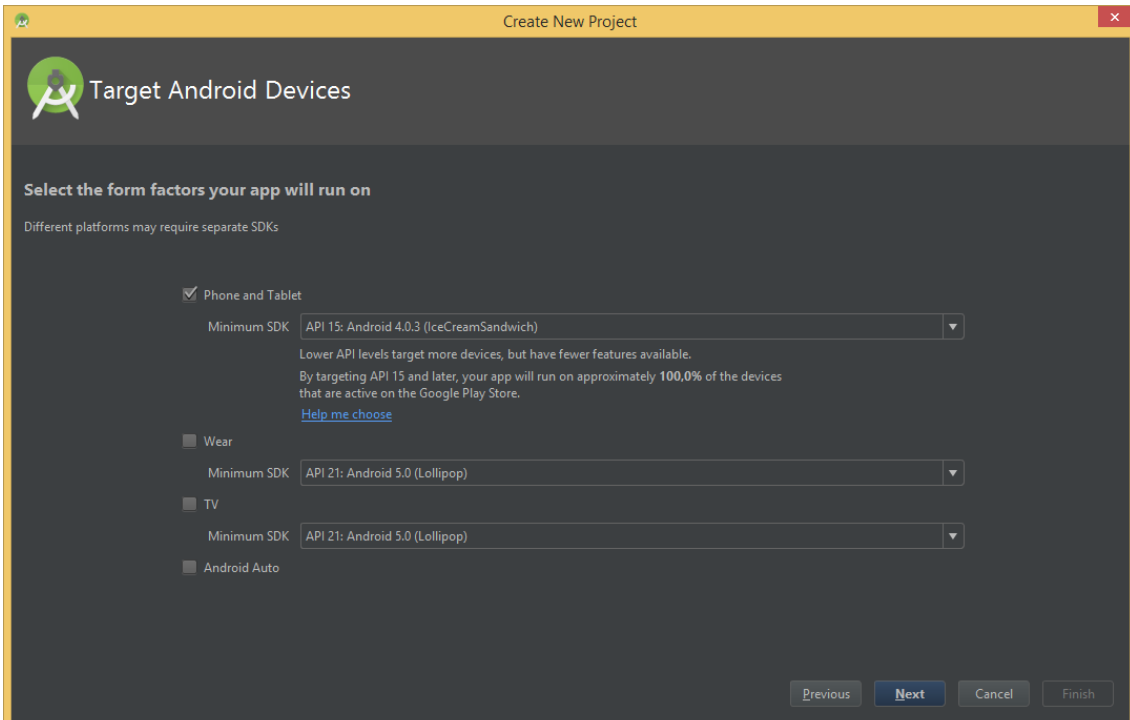


The screenshot shows the 'Create New Project' dialog box in Android Studio. The title bar says 'Create New Project'. The main heading is 'Create Android Project' with the Android logo. The form contains the following fields and options:

- Application name:** A text field containing 'RecyclerView'.
- Company domain:** A text field containing 'br.senai.sp'.
- Project location:** A text field containing 'C:\Users\helena.strada\Documents\XPMobileB-master\Projetos\RecyclerView' with a browse button ('...').
- Package name:** A text field containing 'sp.senai.br.recyclerview' with an 'Edit' button.
- Include C++ support:** An unchecked checkbox.
- Include Kotlin support:** An unchecked checkbox.

At the bottom right, there are four buttons: 'Previous', 'Next' (highlighted in blue), 'Cancel', and 'Finish'.

Clicar em “Next”.



The screenshot shows the 'Target Android Devices' dialog box in Android Studio. The title bar says 'Create New Project'. The main heading is 'Target Android Devices' with the Android logo. The section is titled 'Select the form factors your app will run on' with a subtitle 'Different platforms may require separate SDKs'.

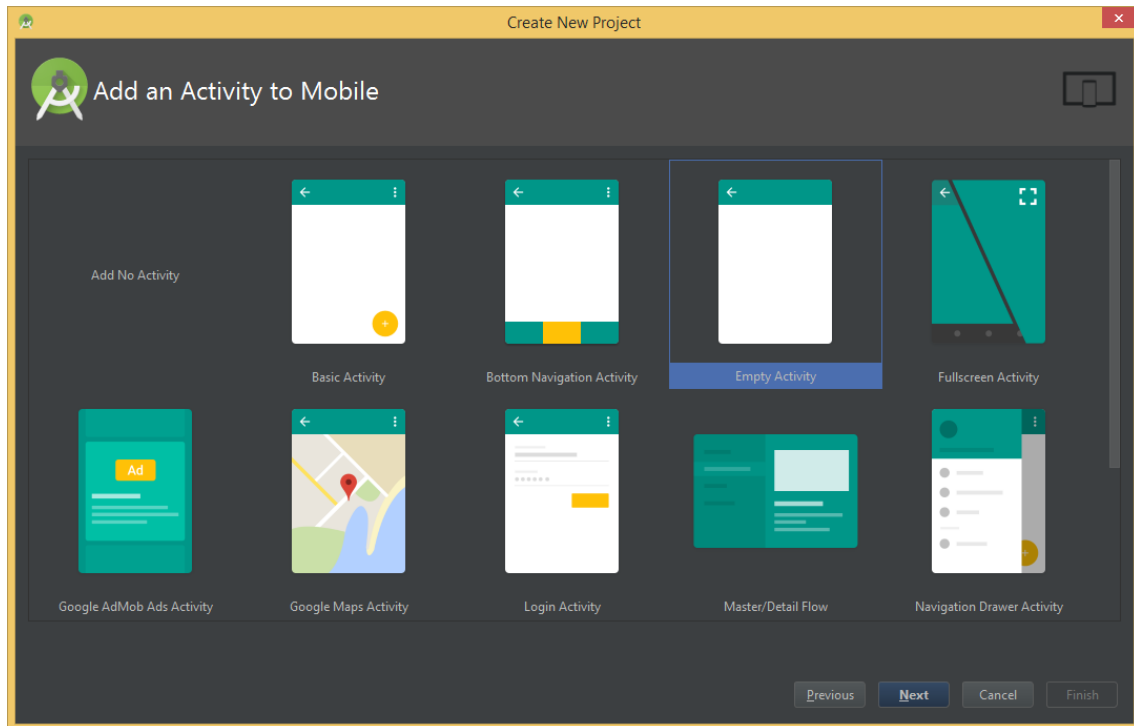
The following options are available:

- Phone and Tablet:** Checked checkbox. Minimum SDK is set to 'API 15: Android 4.0.3 (IceCreamSandwich)'. A note states: 'Lower API levels target more devices, but have fewer features available. By targeting API 15 and later, your app will run on approximately 100,0% of the devices that are active on the Google Play Store.' A link 'Help me choose' is provided.
- Wear:** Unchecked checkbox. Minimum SDK is set to 'API 21: Android 5.0 (Lollipop)'.
- TV:** Unchecked checkbox. Minimum SDK is set to 'API 21: Android 5.0 (Lollipop)'.
- Android Auto:** Unchecked checkbox.

At the bottom right, there are four buttons: 'Previous', 'Next' (highlighted in blue), 'Cancel', and 'Finish'.

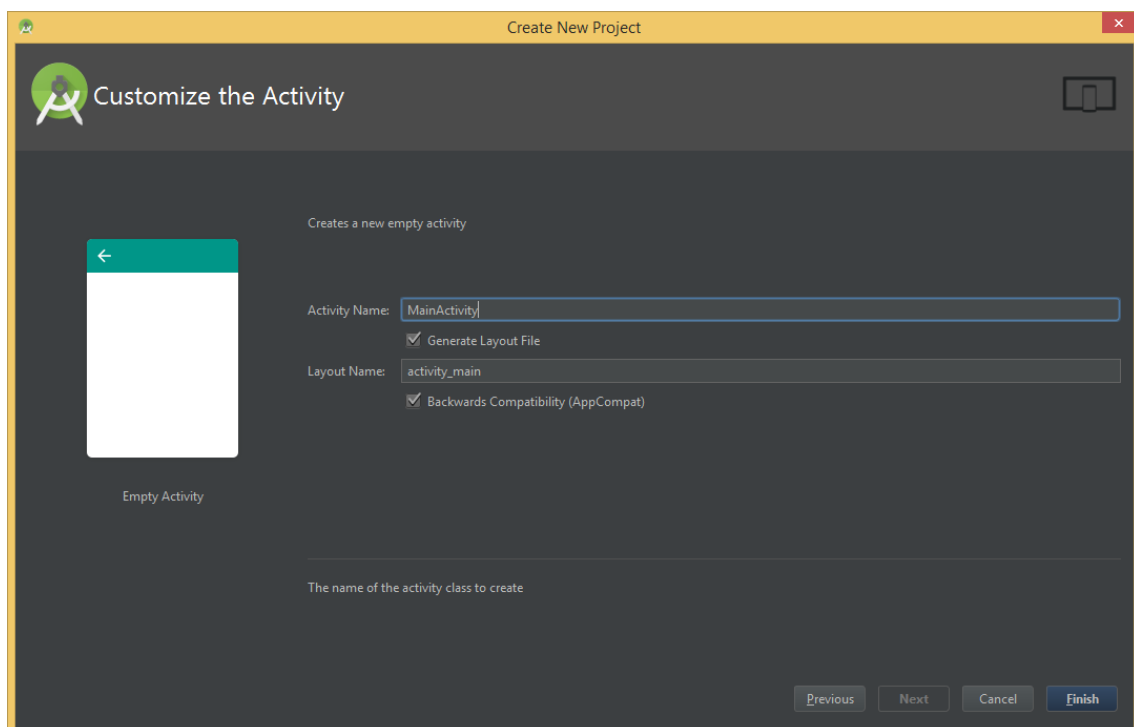
Clicar em “Next”.

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



Clicar em “Next”.

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



Clicar em “Finish”.

7. Passo-a-Passo

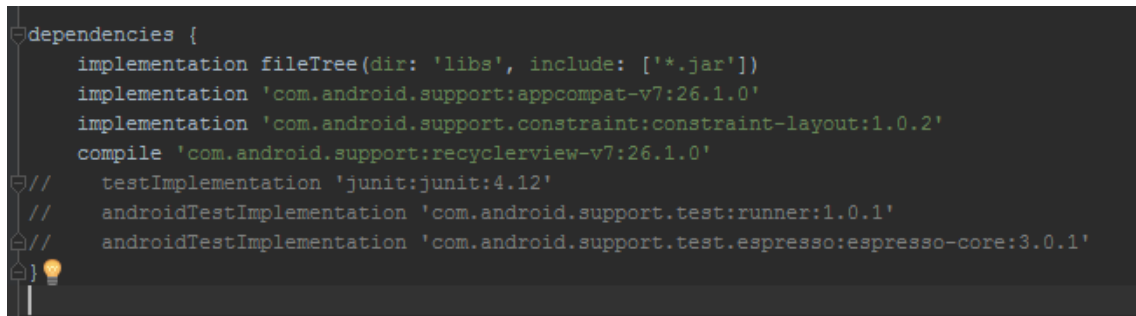
Precisamos de alguns passos chaves para utilizarmos o RecyclerView:

- 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.

7.1. Instalando

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

```
compile 'com.android.support:recyclerview-v7:26.1.0'
```



```
dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    implementation 'com.android.support:appcompat-v7:26.1.0'  
    implementation 'com.android.support.constraint:constraint-layout:1.0.2'  
    compile 'com.android.support:recyclerview-v7:26.1.0'  
    // testImplementation 'junit:junit:4.12'  
    // androidTestImplementation 'com.android.support.test:runner:1.0.1'  
    // androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.1'  
}
```

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

7.2. 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.

7.2.1. Modelo

Vamos criar a nossa classe de modelo:

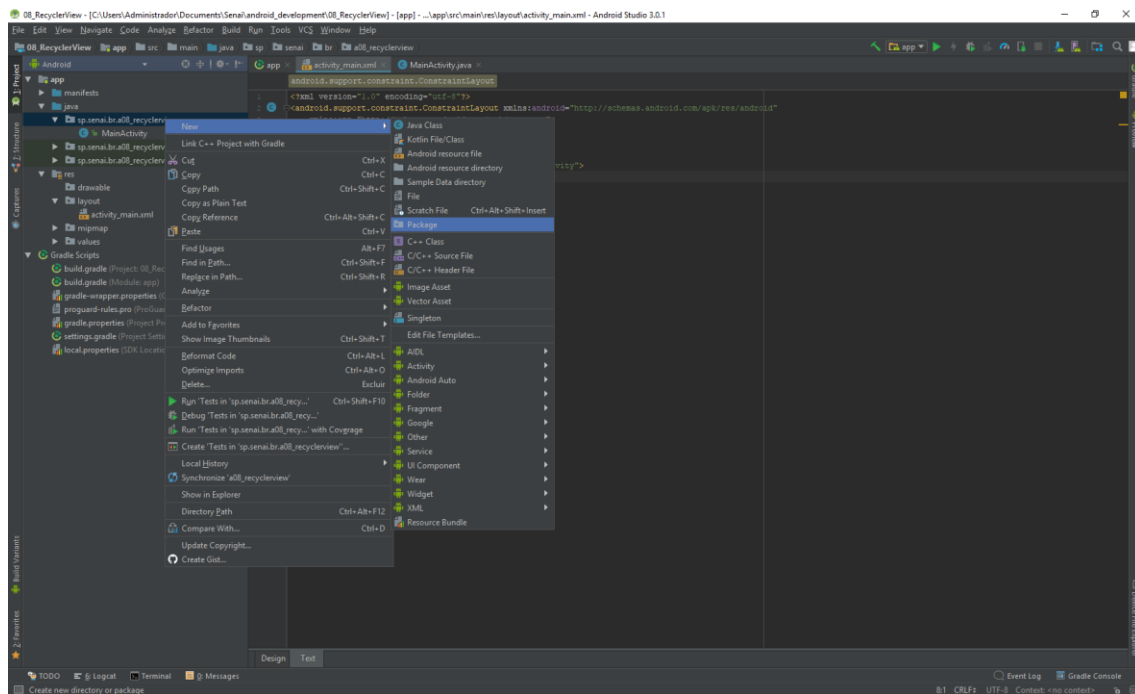


Figura 2 – Criando um novo pacote.

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

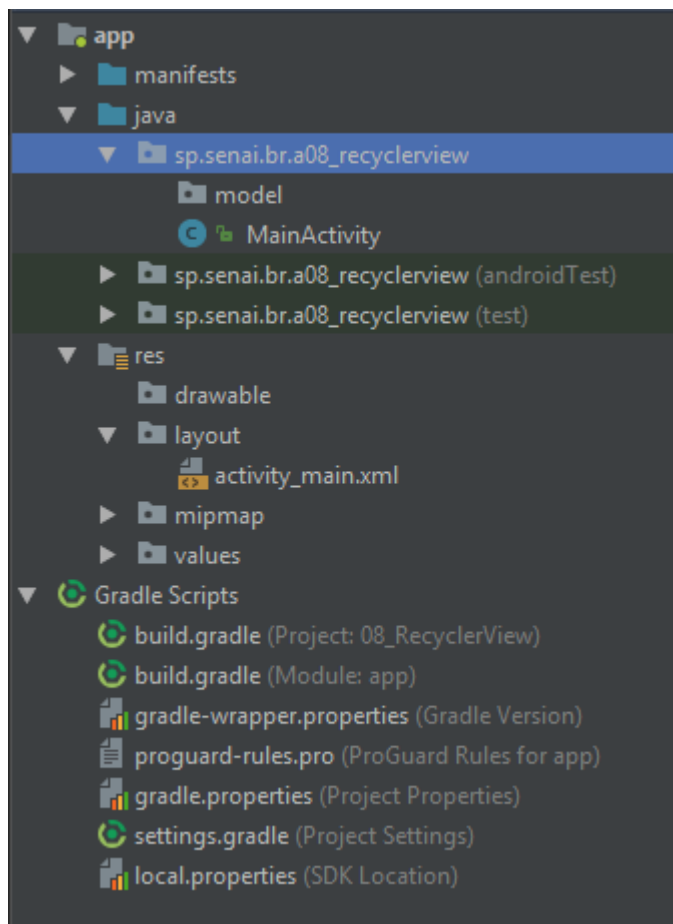


Figura 3 – Estrutura do nosso projeto ao adicionarmos o pacote model.

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

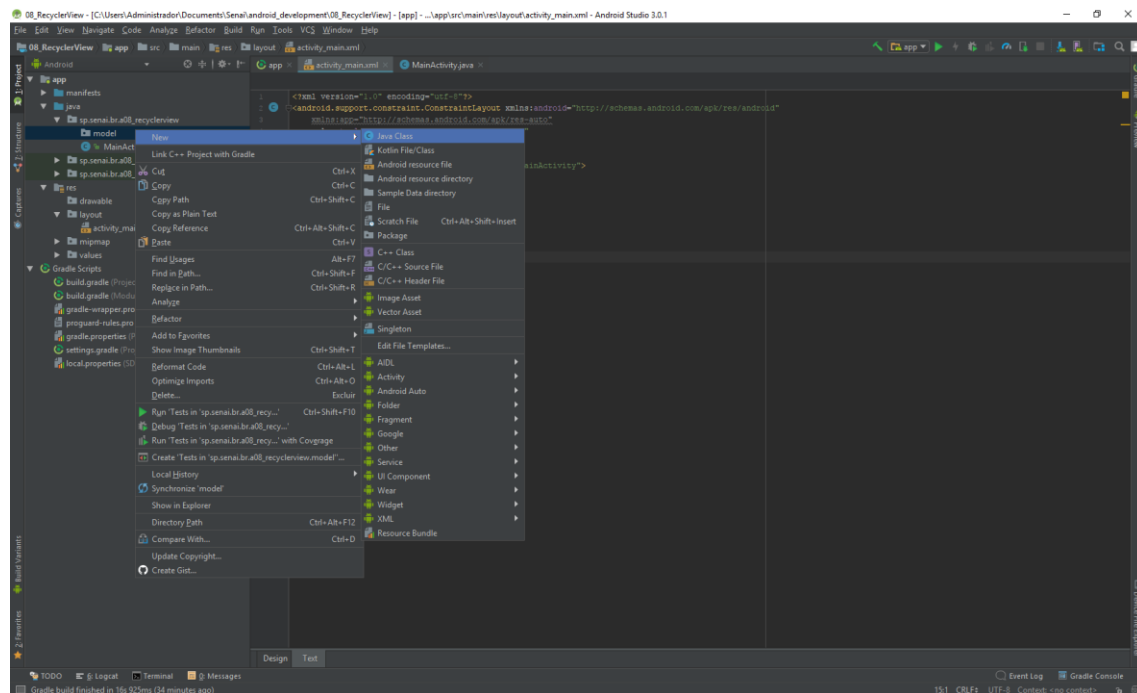


Figura 4 – Criando um novo pacote

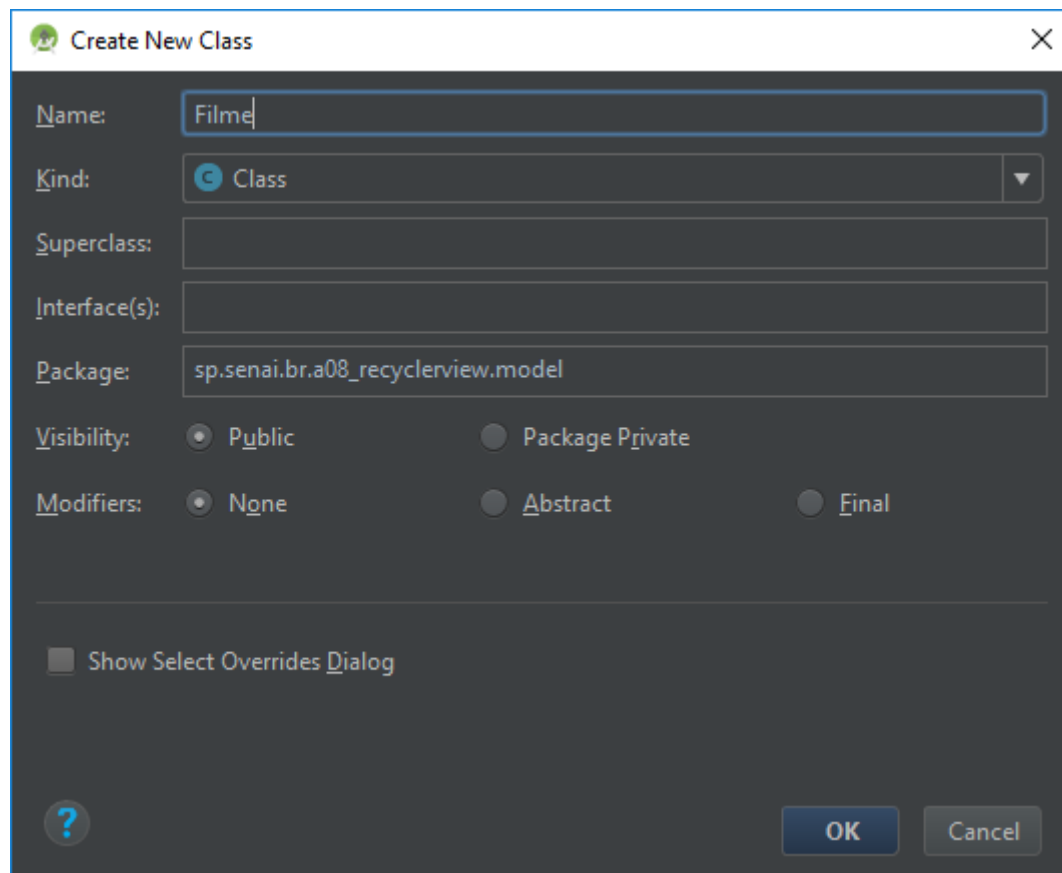


Figura 5 – Inserindo o nome do nosso modelo.

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 + '\n' +
        ", genero=" + genero + '\n' +
        '}';
}

```

7.2.2. DAO

Vamos criar o nosso FilmeDao e os seus respectivos métodos.

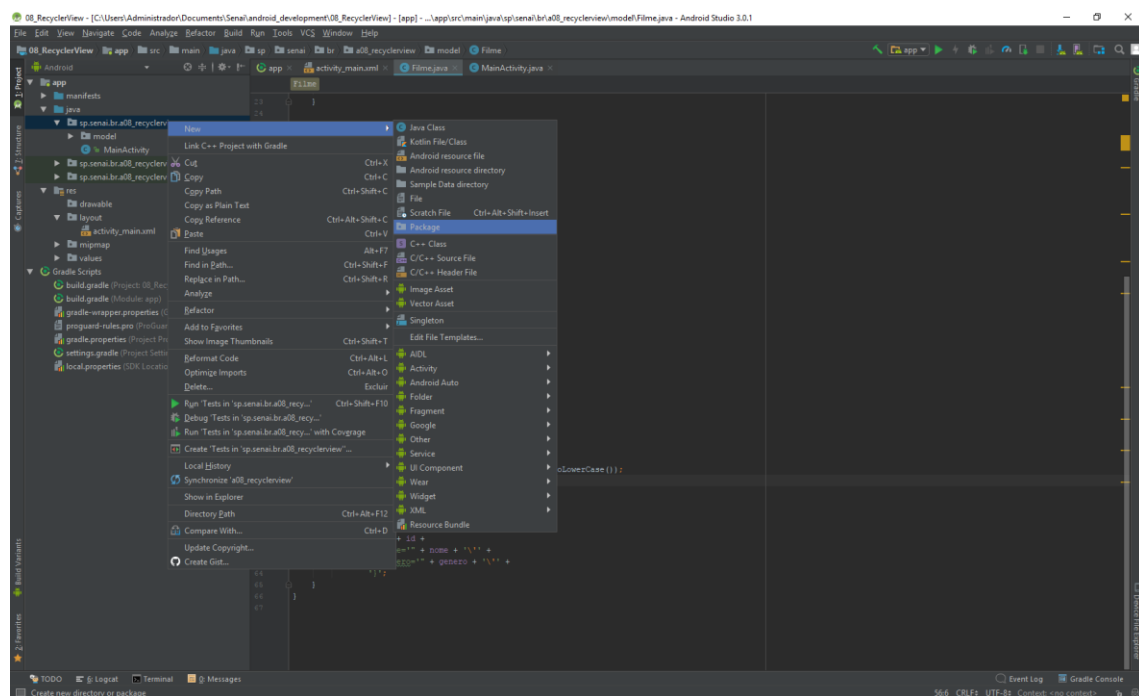


Figura 6 – Criar um novo pacote dao.

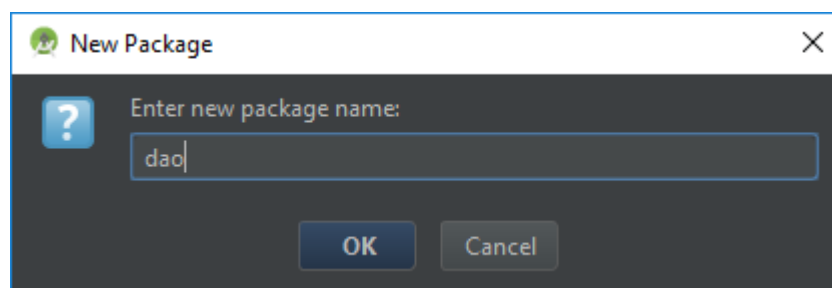


Figura 7 – Colocando o nome do nosso pacote.

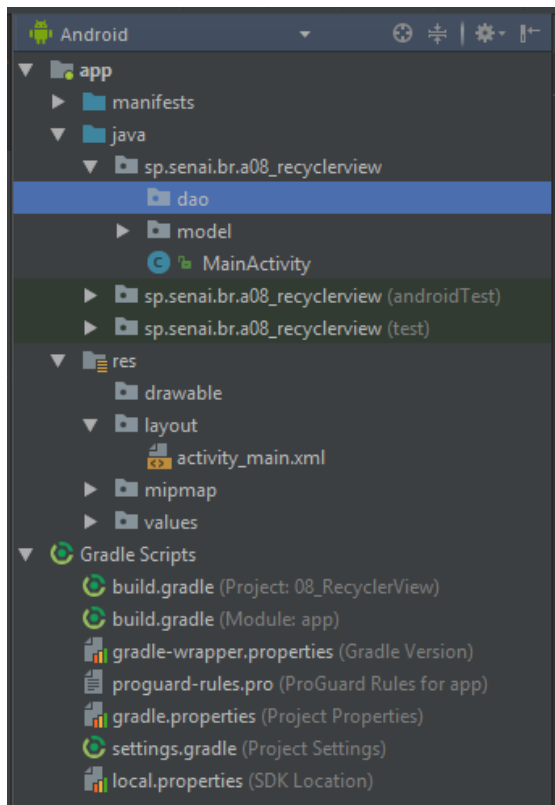


Figura 8 – Estrutura atual do nosso projeto depois de adicionado o pacote dao ao nosso projeto.

Iremos criar o nosso FilmeDao.

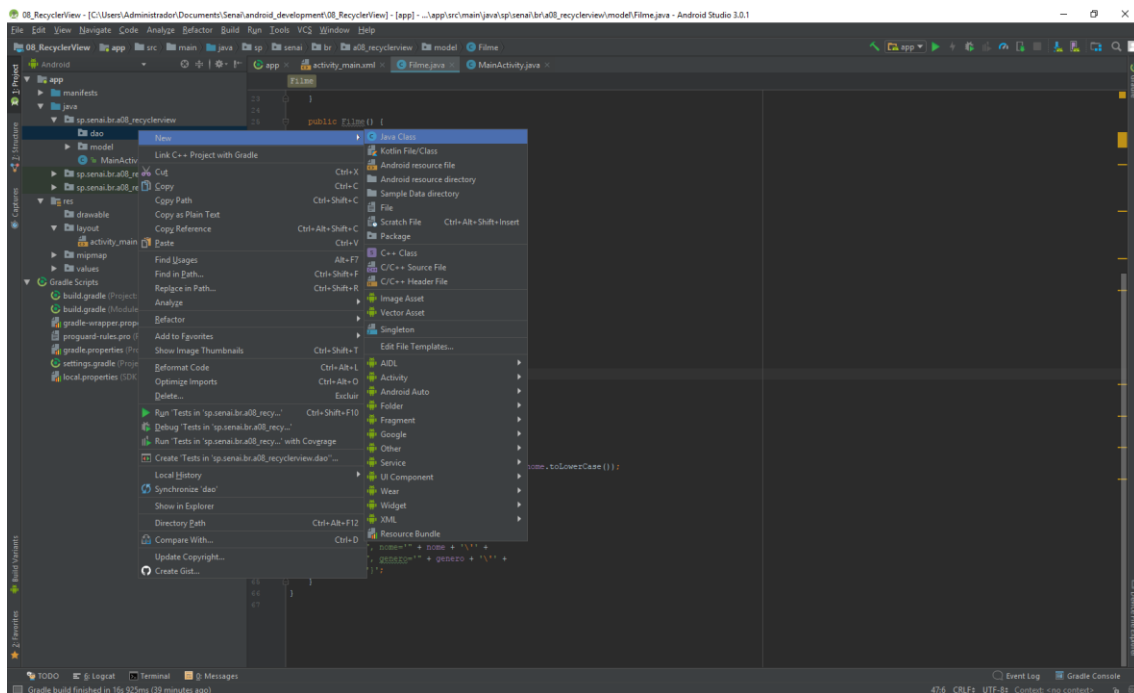
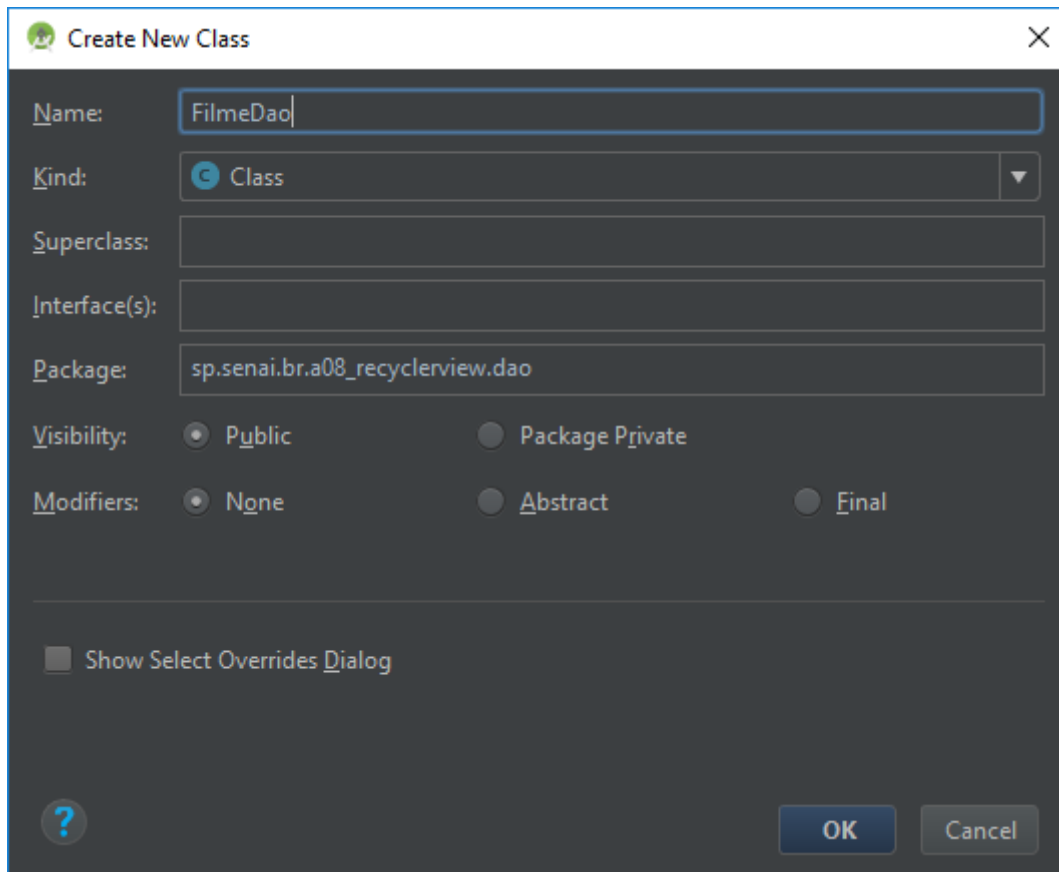


Figura 9 – Inserindo o nosso FilmeDao.



Create New Class

Name:

Kind: ☒ Class

Superclass:

Interface(s):

Package:

Visibility: ☒ Public ☐ Package Private

Modifiers: ☒ None ☐ Abstract ☐ Final

☐ Show Select Overrides Dialog

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);
    }
}
```

```
}
```

7.3. 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.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/rvFilmes"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</RelativeLayout>
```

Figura 11 - Adicionar o nosso RecyclerView dentro da nossa `activity_main.xml`

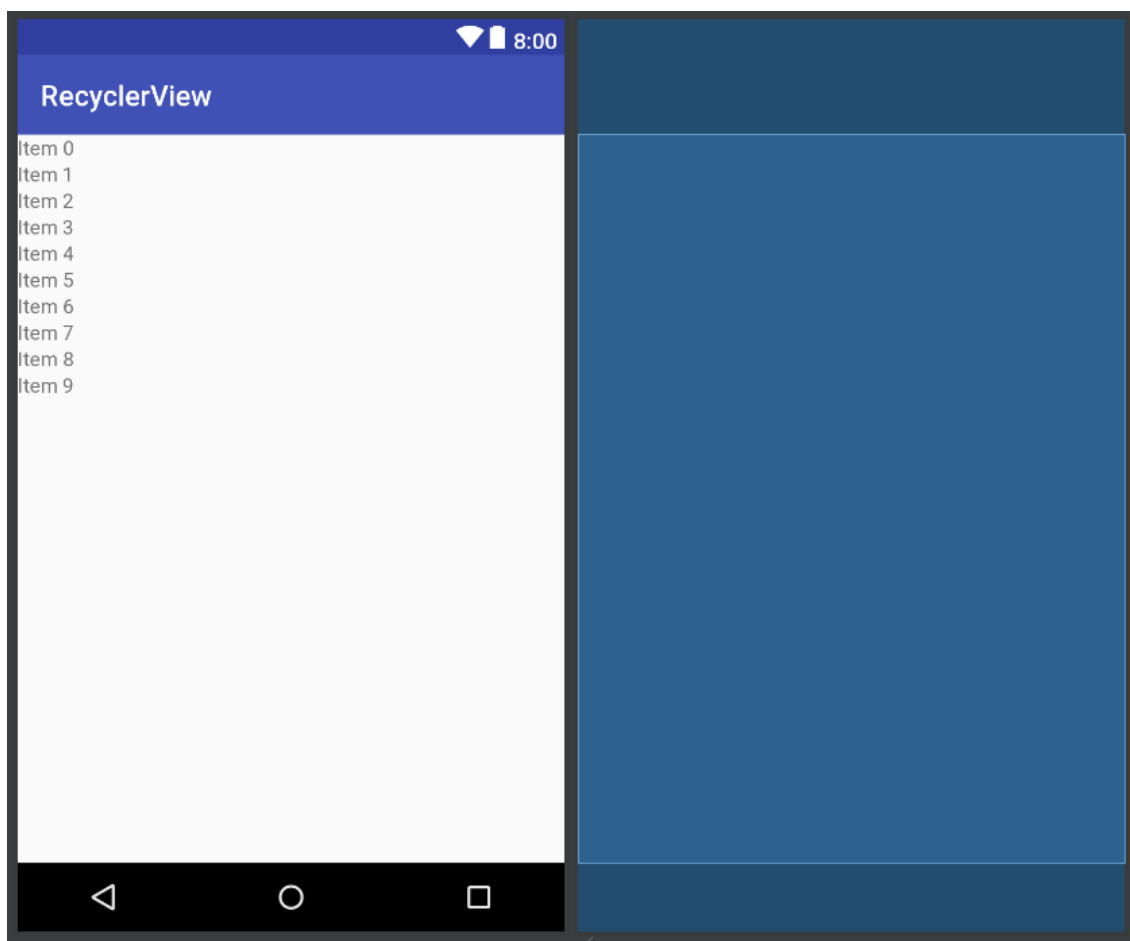


Figura 12 – RecyclerView dentro da nossa activity.

7.4. 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.

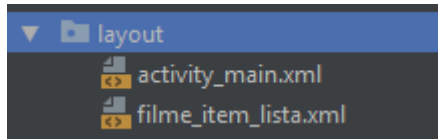


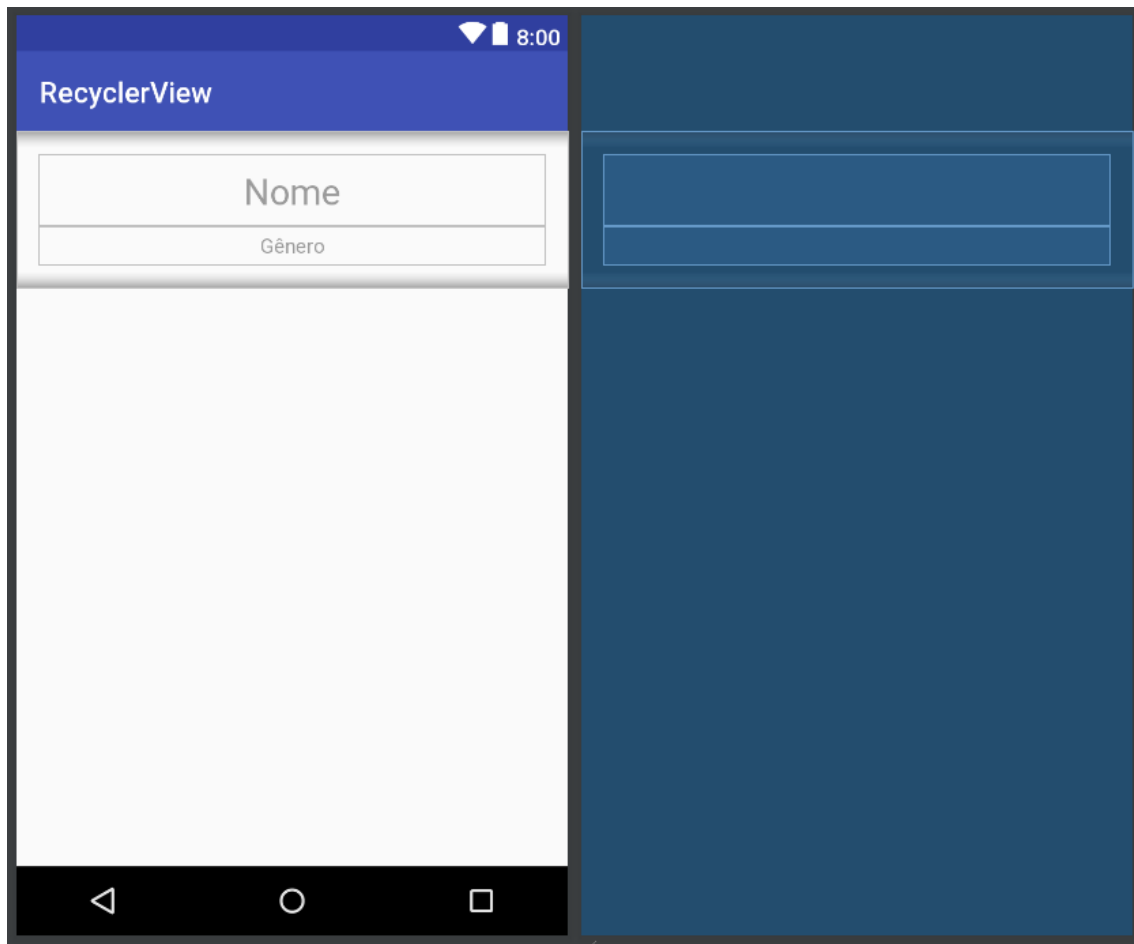
Figura 13 – O layout que será renderizado para cada item da lista.

```
<?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.

7.5. 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*.

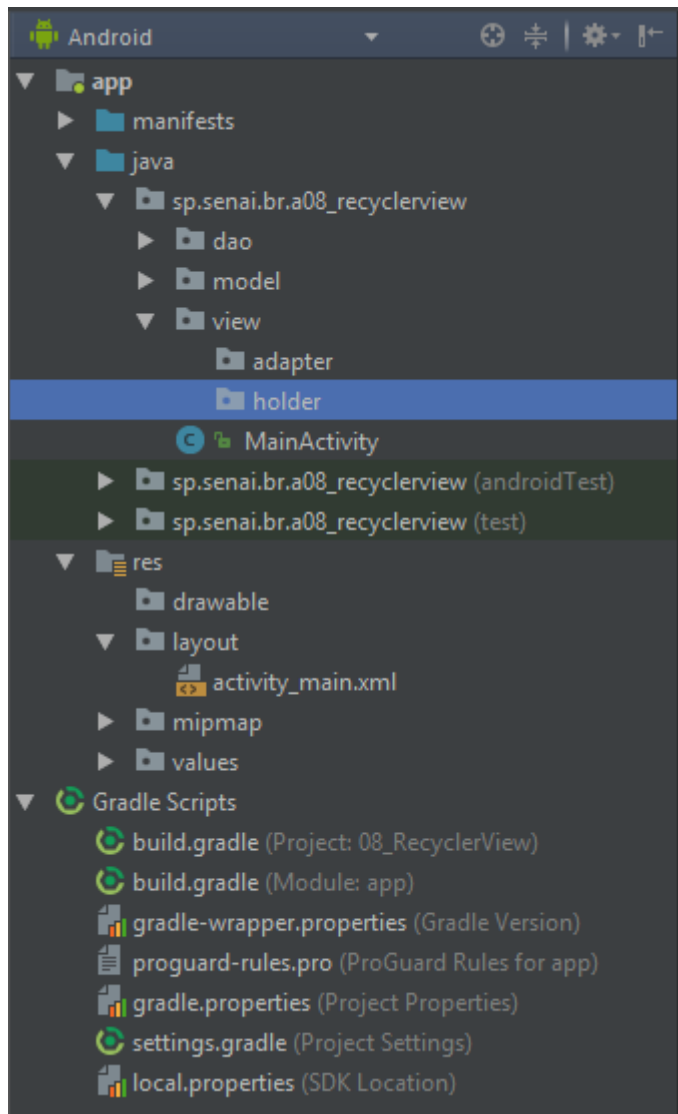


Figura 14 – Estrutura atual do projeto depois de criada as pastas correspondentes: view (adapter, holder).

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

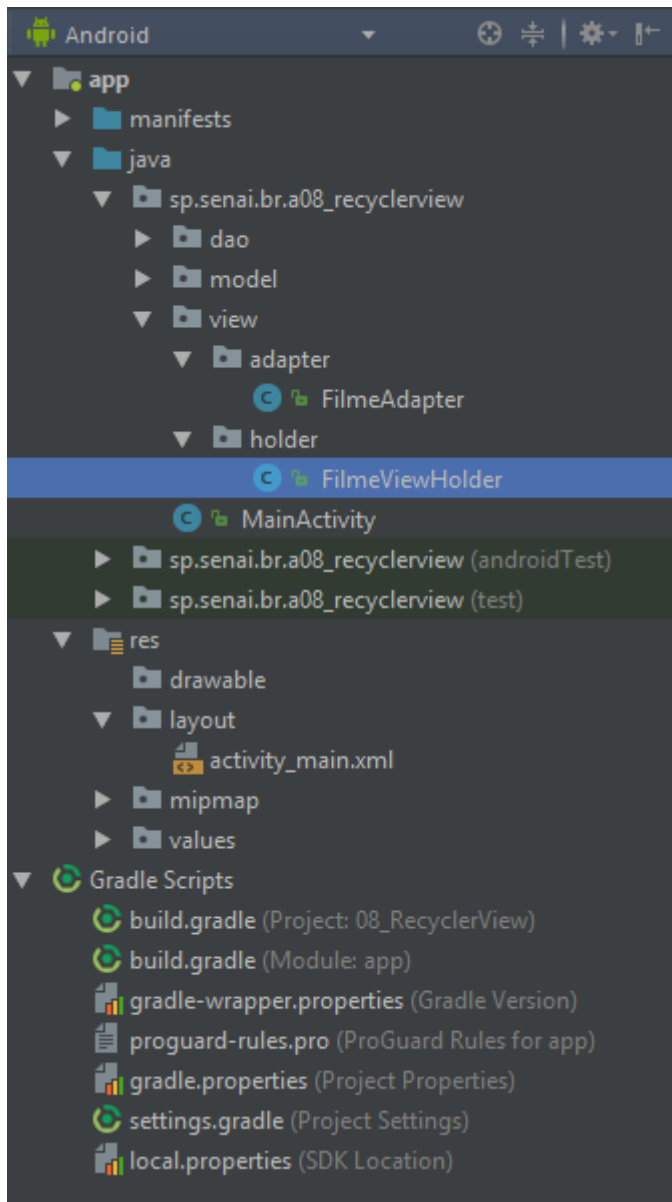


Figura 15 – Estrutura atual do projeto.

7.5.1. 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();
}
}

```

7.5.2. 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());
    }
}

```

```
}
```

7.6. 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.

```
public class MainActivity extends AppCompatActivity {

    // Referência ao nosso dao criado
    private FilmeDao dao = FilmeDao.manager;

    private RecyclerView recyclerView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        List<Filme> filmes = dao.getList();

        recyclerView = findViewById(R.id.rvFilmes);

        recyclerView.setAdapter(new FilmeAdapter(filmes, context: this));

        RecyclerView.LayoutManager layout = new LinearLayoutManager( context: this,
            LinearLayoutManager.VERTICAL, reverseLayout: false);

        recyclerView.setLayoutManager(layout);
    }
}
```

Figura 16 – MainActivity.java com o código completo.

8. Conteúdo adicional

8.1. Selecionando apenas um item da lista

8.1.1. OnClickListener

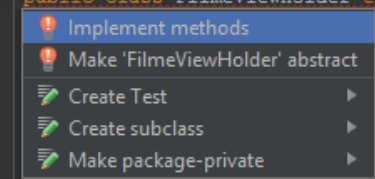
Complementando o nosso projeto, caso quiséssemos selecionar apenas um item da nossa lista, iremos realizar essa ação através do *View.OnClickListener*.

No nosso *FilmeViewHolder* iremos implementar a classe descrita acima.

```
/**
 * Created by helena.strada on 04/01/2018.
 */
public class FilmeViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener{
```

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

```
/**
 * Created by helena.strada on 04/01/2018.
 */
public class FilmeViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener{
```



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

```

/**
 * Created by helena.strada on 04/01/2018.
 */

public class FilmeViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener{

    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;

        view.setOnClickListener(this);

        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());
    }

    // método "obrigatório"
    @Override
    public void onClick(View view) {
        // Realizando um Toast (tela) para mostrar qual ID estamos selecionando
        Toast.makeText(view.getContext(), filmeId.toString(), Toast.LENGTH_SHORT).show();
        // Mostrando o mesmo ID no log
        Log.d( tag: "Filme Clicado", filmeId.toString());
    }
}

```

Figura 17 – Classe final FilmeViewHolder

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

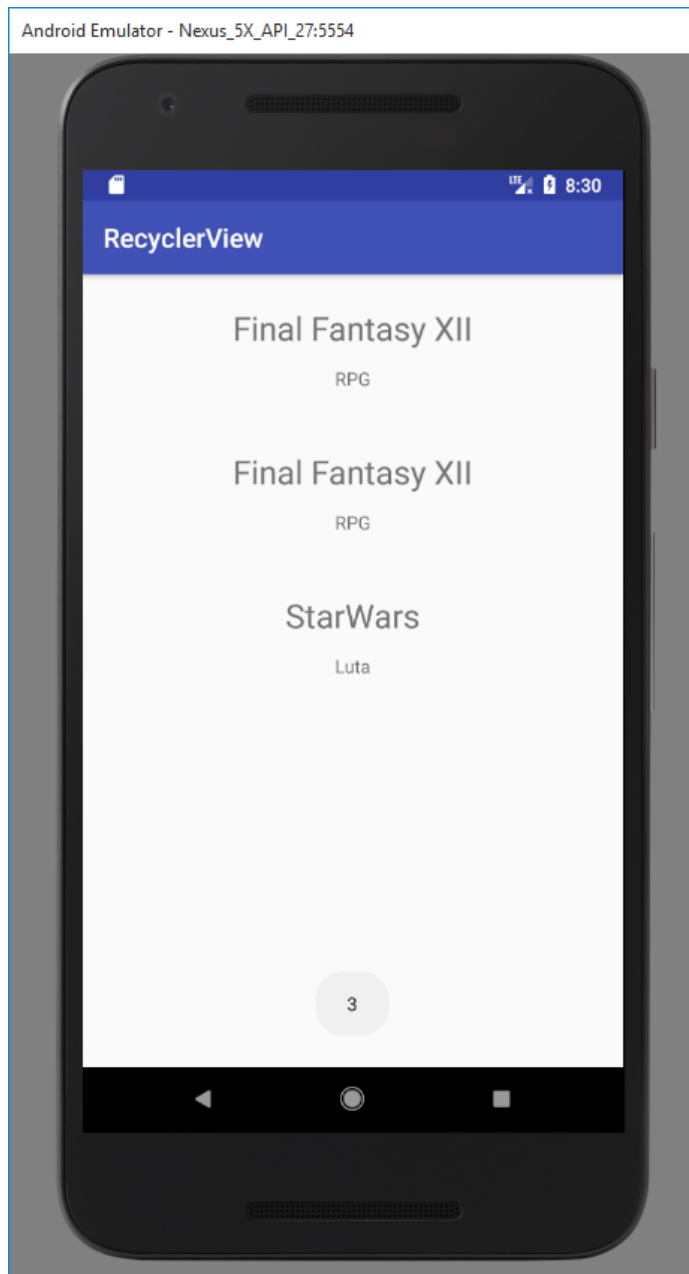
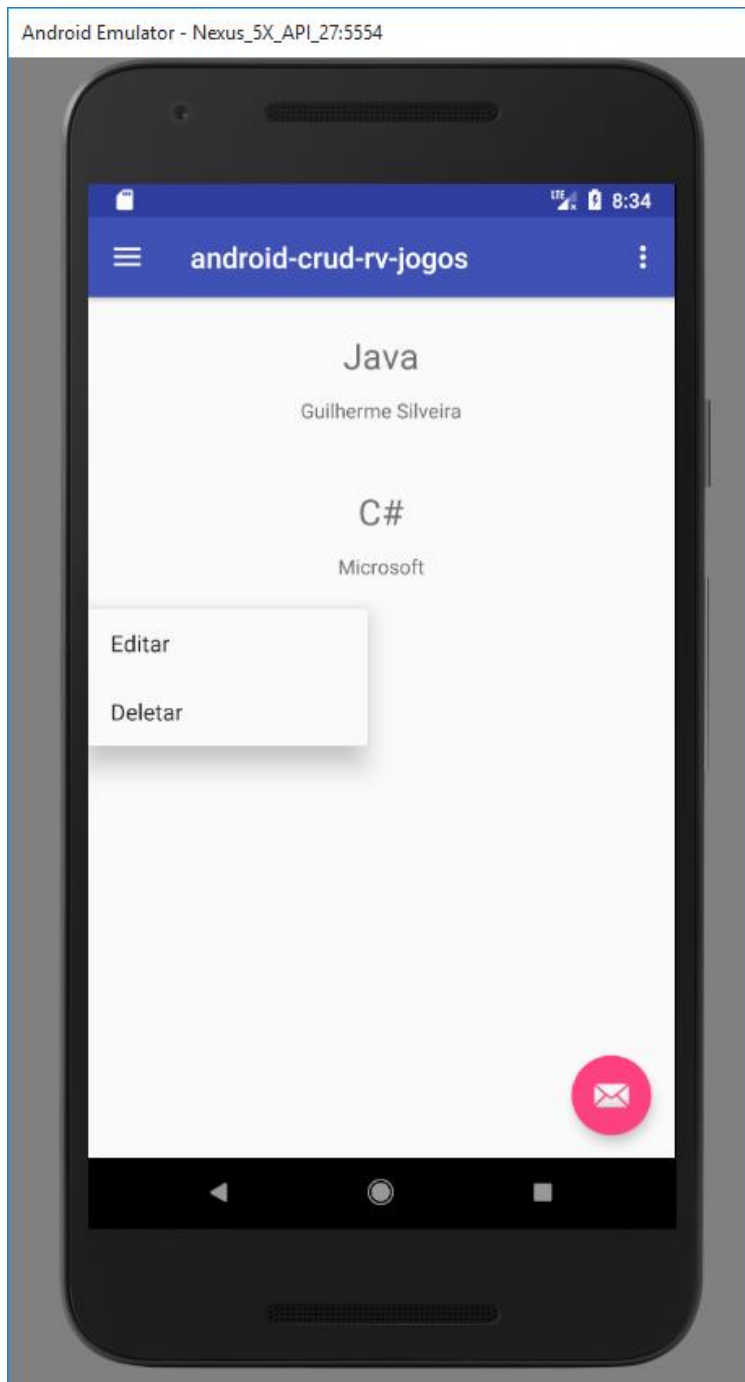


Figura 18 – Item selecionado da nossa lista.

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



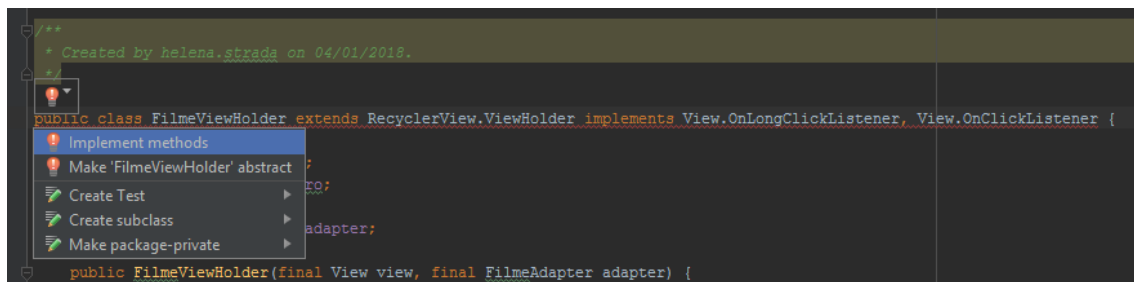
Para isto, utilizamos o *View.LongOnClickListener*.

8.1.2. View.LongOnClickListener

Iremos implementar a interface *View.LongOnClickListener*.

```
/**
 * Created by helena.strada on 04/01/2018.
 */
public class FilmeViewHolder extends RecyclerView.ViewHolder implements View.OnLongClickListener, View.OnClickListener {
```

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



Após implementarmos o método.

```
@Override
public boolean onLongClick(View view) {
    return false;
}
```

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

```
public FilmeViewHolder(final View view, final FilmeAdapter adapter) {
    super(view);
    this.adapter = adapter;

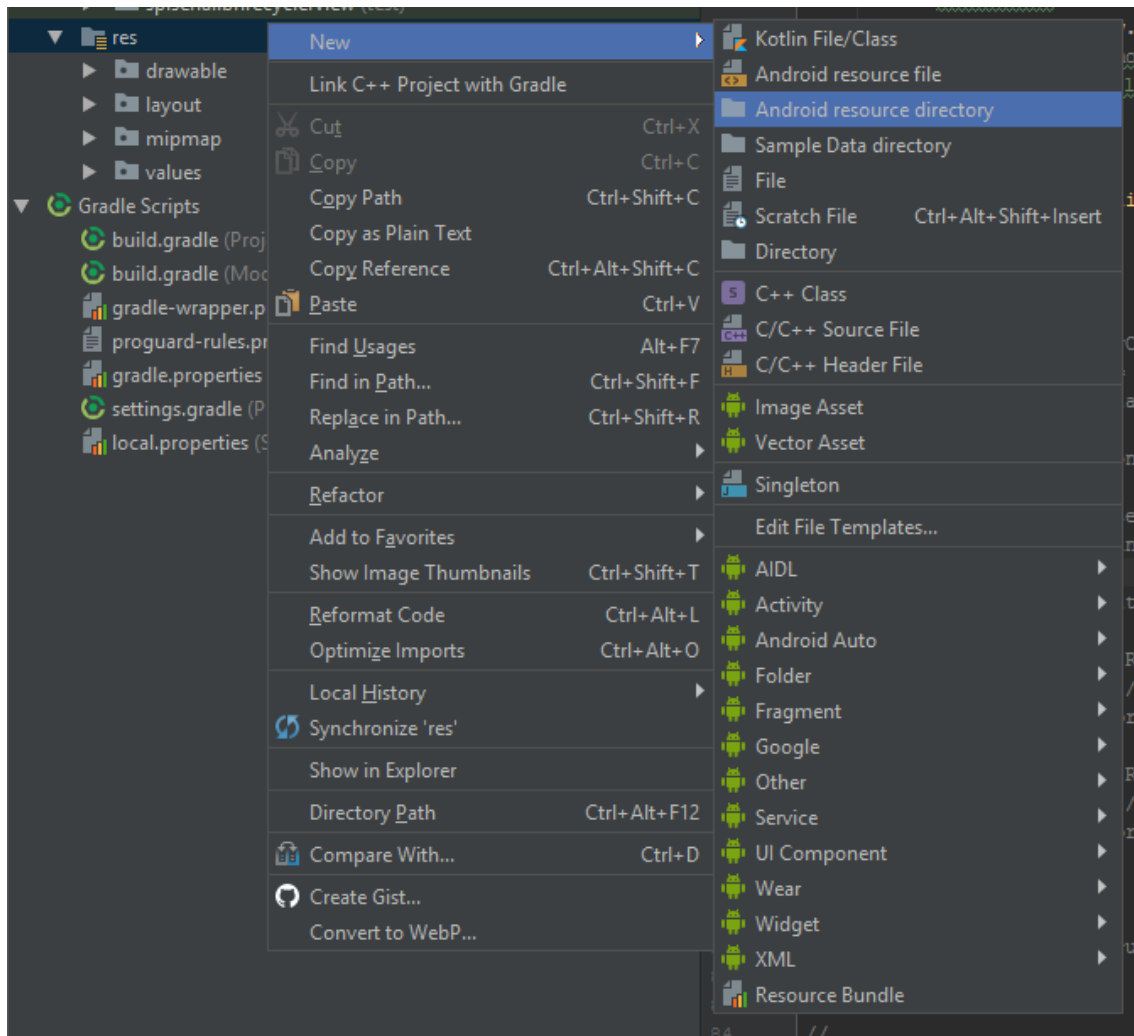
    view.setOnClickListener(this);
    view.setOnLongClickListener(this);

    nome = view.findViewById(R.id.tvNome);
    genero = view.findViewById(R.id.tvGenero);
}
```

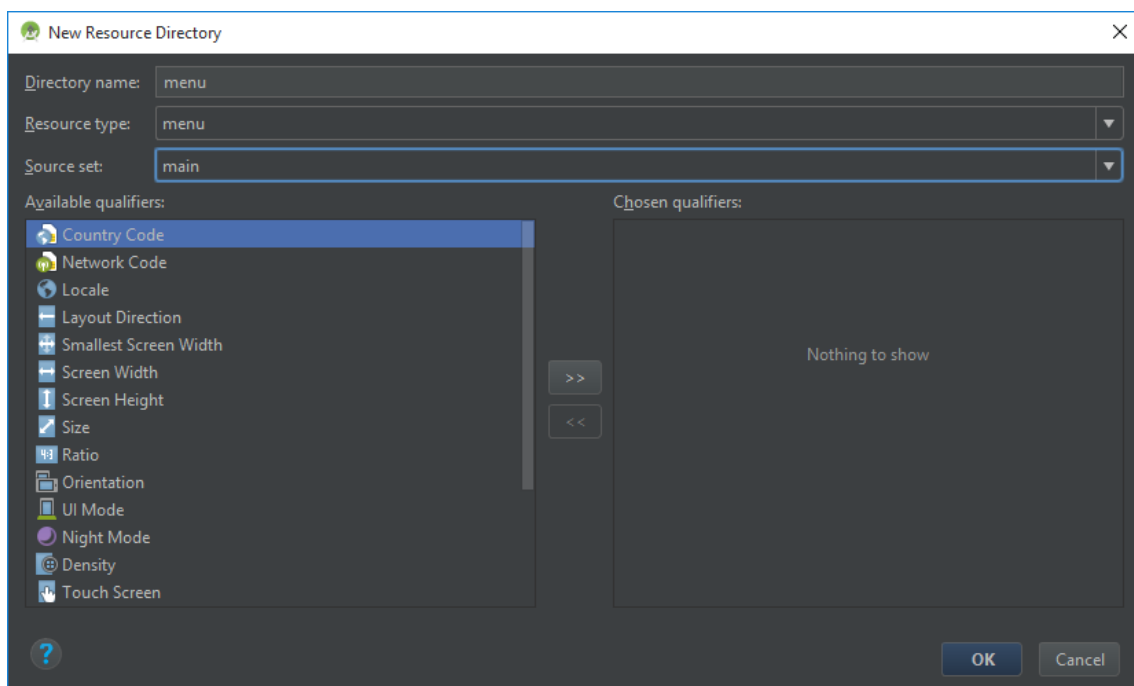
Figura 19 – Inserir o `view.setOnLongClickListener(this)`.

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`.

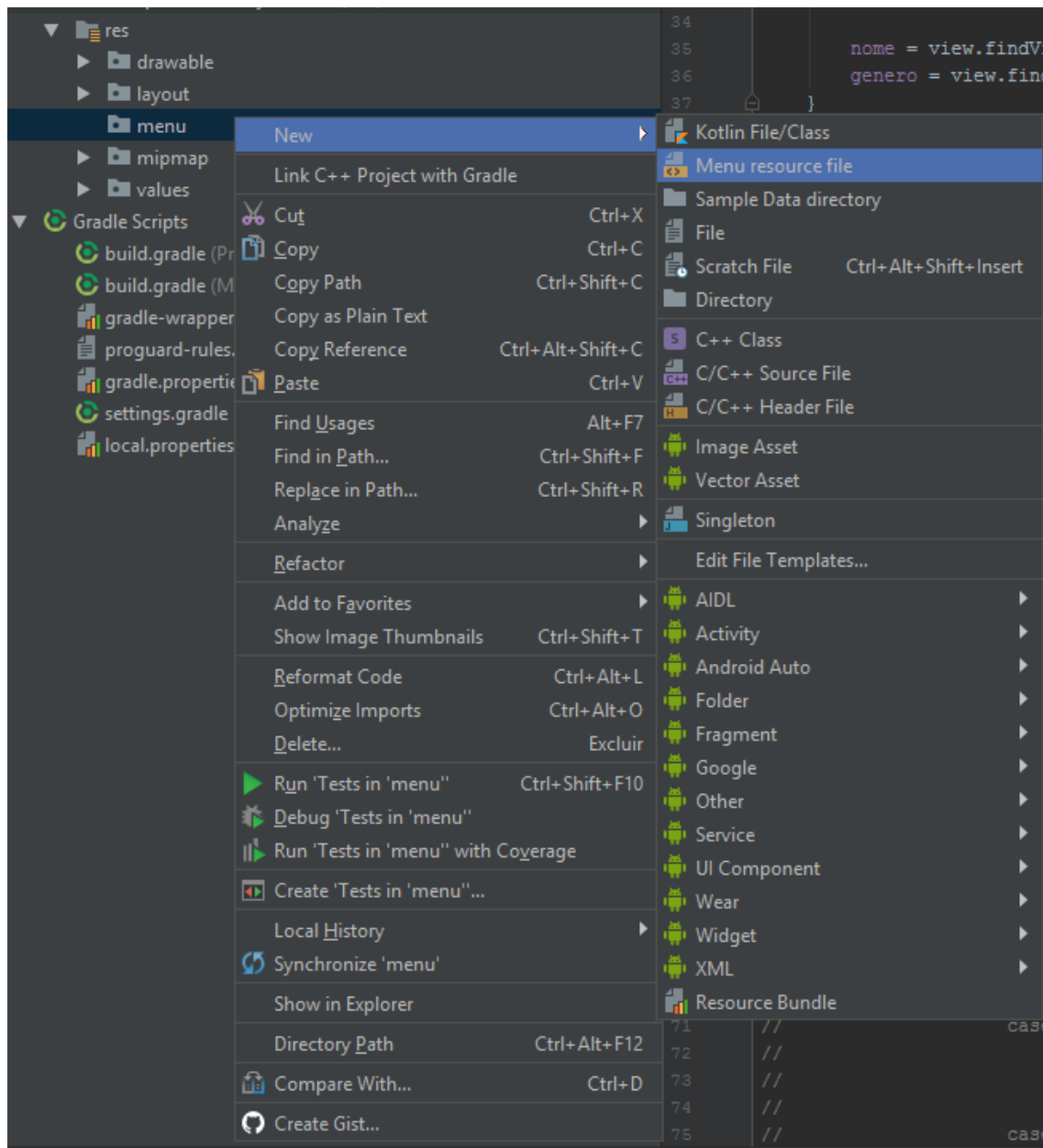


Iremos seleccionar o tipo de diretório que queremos criar.



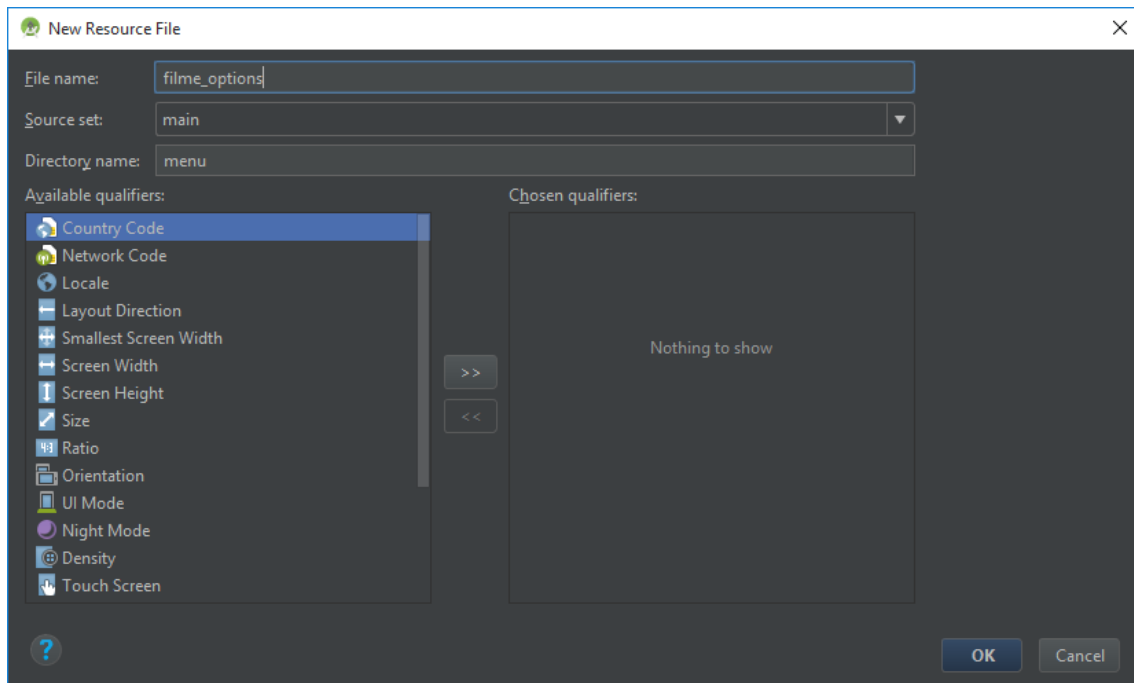
Clicar em "OK".

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.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/menuFilmeEditar"
        android:title="Editar"/>

    <item
        android:id="@+id/menuFilmeDeletar"
        android:title="Deletar"/>

</menu>
```

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.

```

@Override
public boolean onLongClick(View view) {
    PopupMenu popupMenu = new PopupMenu(view.getContext(), view);
    popupMenu.getMenuInflater().inflate(R.menu.filme_options, popupMenu.getMenu());

    final Activity context = (Activity)view.getContext();

    popupMenu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
        public boolean onMenuItemClick(MenuItem item) {

            switch (item.getItemId()) {

                case R.id.menuFilmeEditar:
                    // realiza ação de edição
                    break;

                case R.id.menuFilmeDeletar:
                    // realiza ação de deleção
                    break;

            }

            return true;
        }
    });

    popupMenu.show();
    return false;
}

```

Figura 20 – Ao realizar um LongClick, queremos abrir o nosso menu de opções para o usuário.

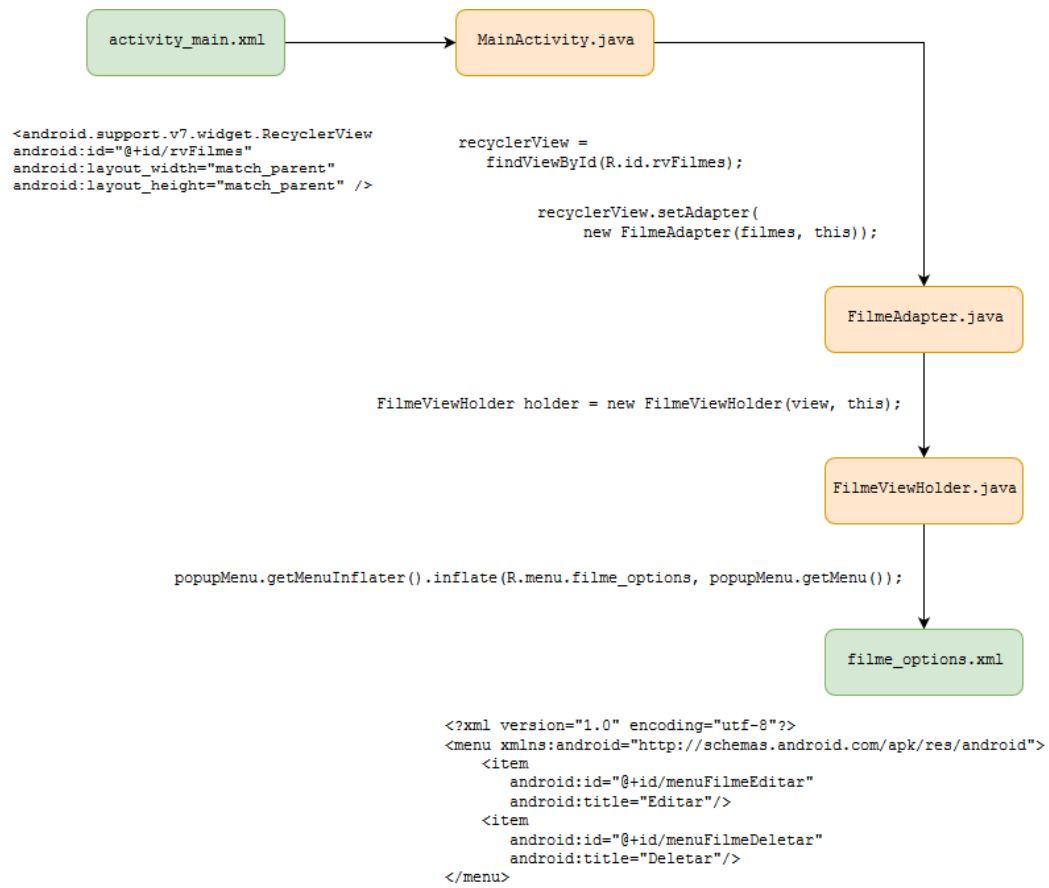


Figura 21 – Esboço da estrutura do Projeto



Figura 22 – Projeto Finalizado

9. 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.

9.1. Projetos

Capítulo 1 até 6 – **android-rv**

Capítulo 7.1.1. – **android-rv-id**

Capítulo 7.1.2 – **android-rv-onclicklistener**

10. 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