Desenvolvimento de Aplicações para Dispositivos Móveis

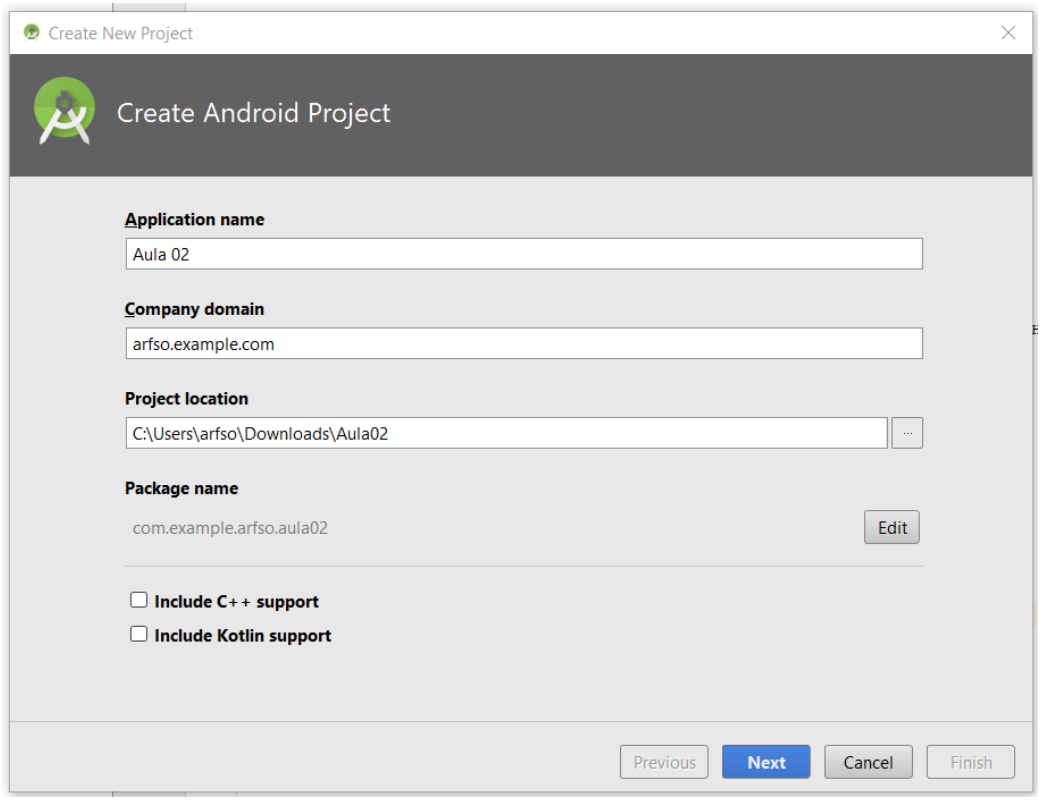
Aula 02 – Trabalhando Telas

- Criando um Novo Projeto
- Tela Principal
- Transição de tela

Prof. Fernando Gonçalves Abadia

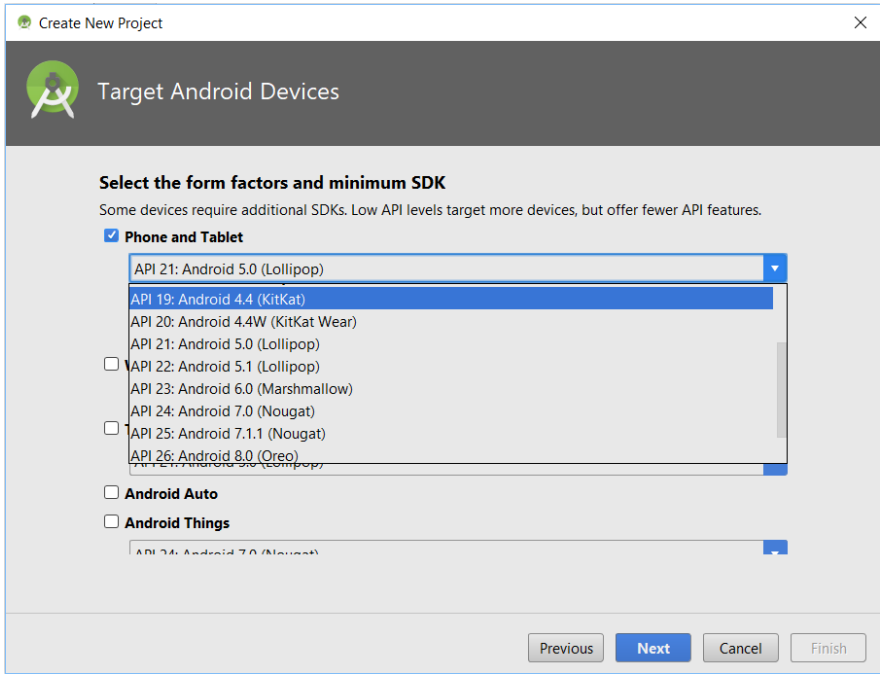
Criando um Novo Projeto

- Ao iniciar um novo projeto é importante verificar o nome, pois o mesmo será o título do app.



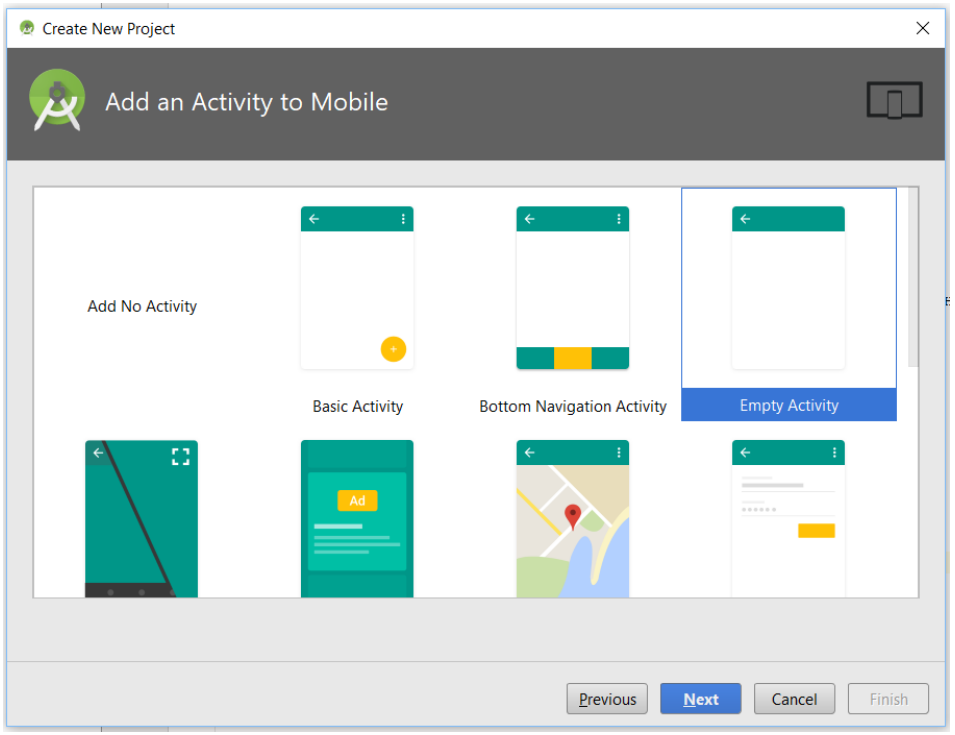
Criando um Novo Projeto

- Escolha uma ***API*** alvo (*Application Programming Interface* ou, em português, Interface de Programação de Aplicativos).
- Estas ***APIs*** correspondem ao conjunto de padrões de programação. Permite a construção de aplicativos e a sua utilização de maneira não tão evidente para os usuários.



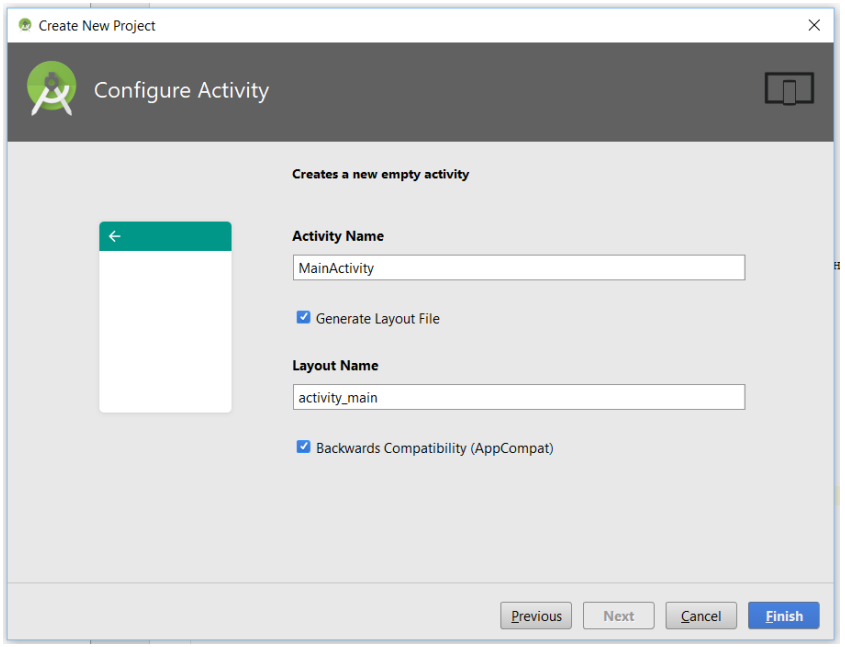
Criando um Novo Projeto

- Uma **Activity** é um módulo único e independente que normalmente está relacionada diretamente com uma tela de interface de usuário e suas funcionalidades correspondentes.
- Iniciar uma Empty Activity para uma tela vazia.



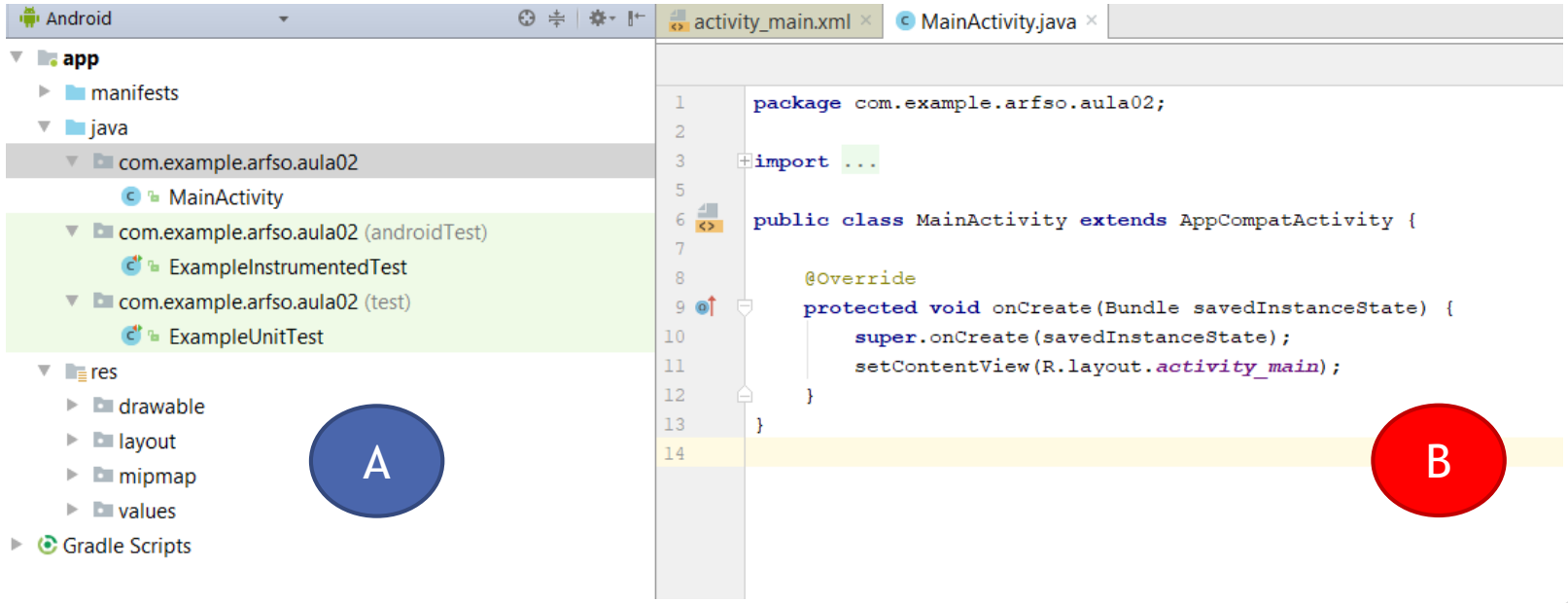
Criando um Novo Projeto

- O nome de uma atividade é sempre importante para identificação. A sugestão que o Android Studio oferece é “MainActivity”.
- É necessário usar as **Activities** de forma correta, como seu ciclo de vida, gerenciamento na memória, seus métodos e etc. Tudo influencia em como seu aplicativo vai ser desenvolvido e na qualidade dele também.



Criando um Novo Projeto

- Ao abrir o novo projeto, iremos deparar com a Janela de Ferramenta de Projeto (A) e a Janela de Edição (B).

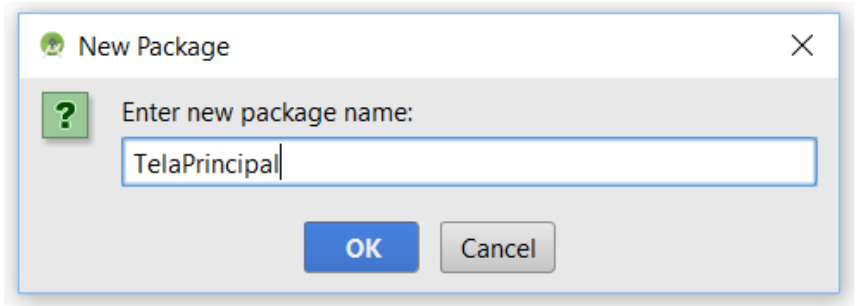


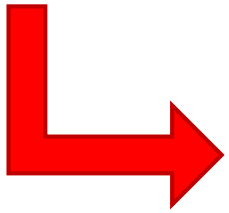
Trabalhando a Tela Principal: Views

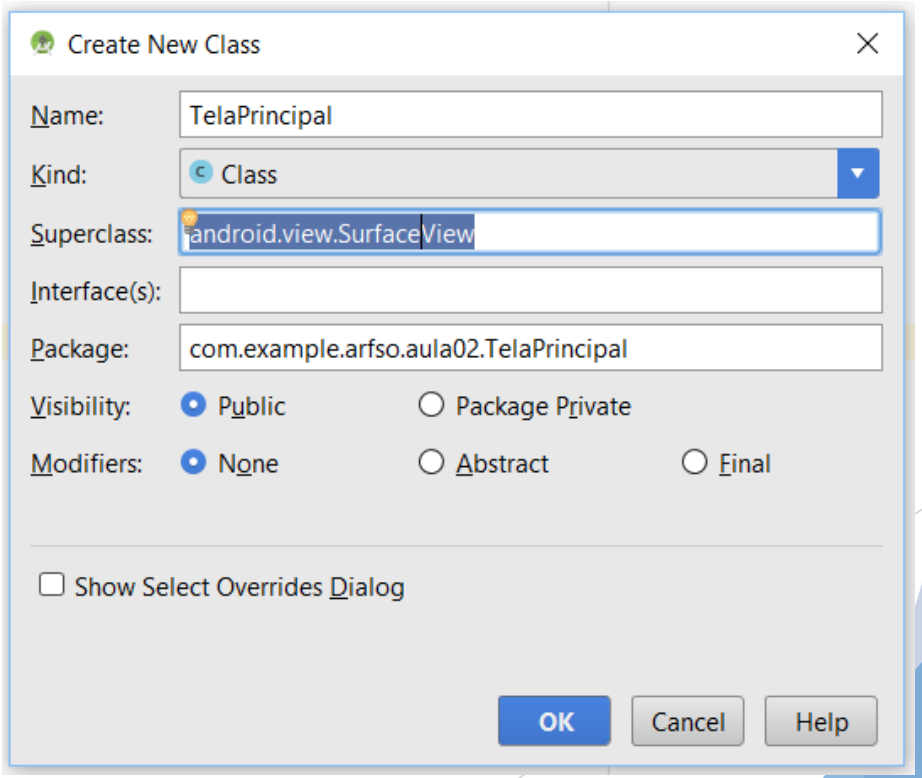
- Ao se trabalhar com aplicativos, será necessário entender sobre as Android **Views:** são os principais componentes que usamos para desenvolver as interfaces dos aplicativos Android.
- Responsáveis pela maior parte da interação entre o usuário e o aplicativo, elas têm uma grande responsabilidade em definir o design da interface para que seja amigável com uma alta usabilidade.
- Com exceção de um alguns casos, todos os aplicativos têm alguma forma de interface de usuário. No Android, isso é feito através do uso de **Views** e **ViewGroups**.

Trabalhando a Tela Principal: Views

- Para criar um componente de view próprio, deve-se iniciar uma nova classe dentro do pacote a ser trabalhado. Neste caso foi criado um novo pacote em: com.example.arfso.aula02 e uma nova classe herdando uma **View** ou **SurfaceView** em: com.example.arfso.aula02.TelaPrincipal

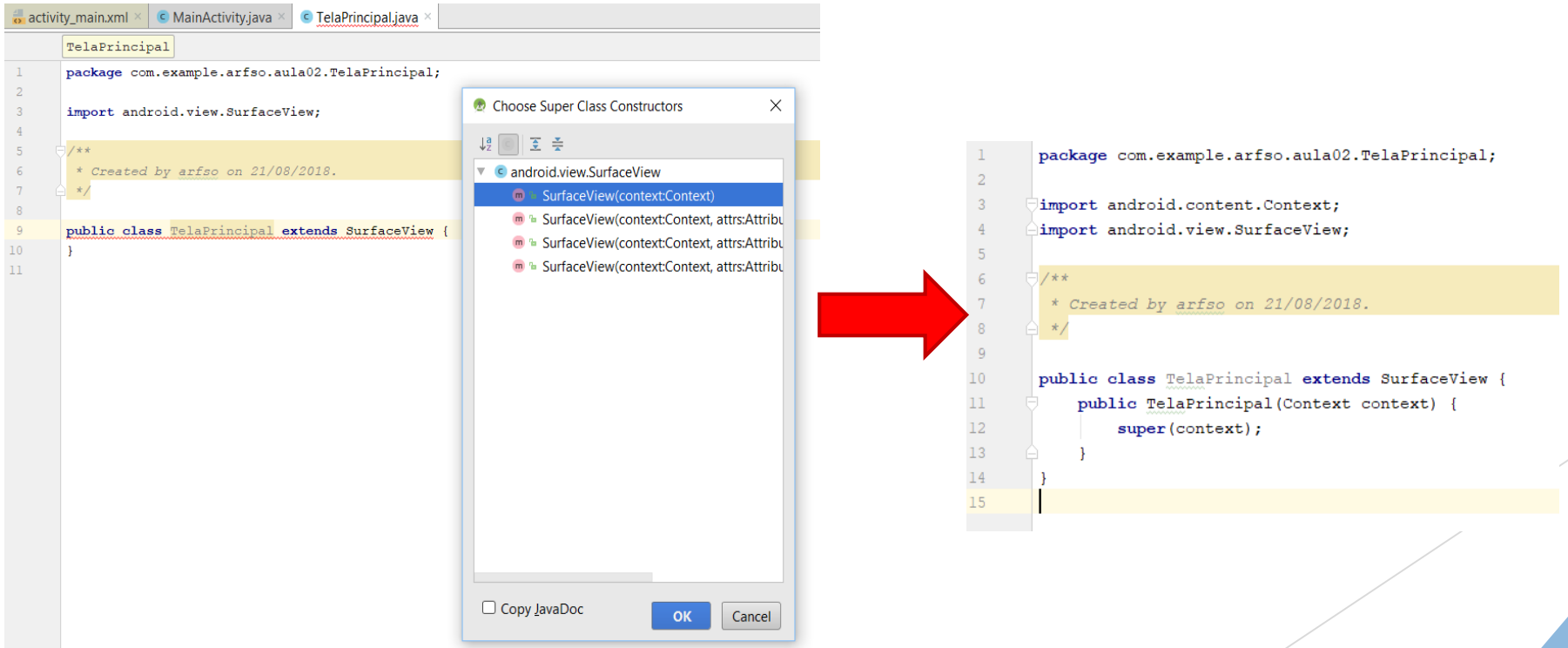






Trabalhando a Tela Principal: Views

- Após a criação da classe, será necessário criar um construtor que trabalhará o contexto da View .
- Simplificando, esta View será um retângulo na tela que mostrará algum conteúdo.



Trabalhando a Tela Principal: Views

- Existem diferentes tipos de **Views**:
  - Uma **View** que mostra algum texto é chamada de **TextView**
  - Uma **View** que mostra uma imagem é chamado de **ImageView**
  - Uma **View** que mostra um botão é chamado, claro, de **Button**
- Além dessas, há muitos outros tipos de Views no Android que poderão ser criados e mostrados, com forma de trabalho e criação praticamente iguais.

Trabalhando a Tela Principal: Views

- Com estas Views, poderemos trabalhar vários comandos diretamente na aplicação mostrando textos, imagens.
- Uma **SurfaceView** dedica todo buffer de superfície enquanto toda a **view** compartilha um buffer de superfície alocado pela ViewRoot. Em outra palavra, a **surfaceView** custa mais recursos.
- O surfaceView não pode ser acelerado por hardware (a partir do JB4.2), enquanto 95% das operações no modo de exibição normal são aceleradas por hardware usando o openGL ES.
- Na maioria dos jogos, uma SurfaceView renderá um bom processamento, pois com uma view normal, o Thread da interface do usuário estaria bloqueado.

Trabalhando a Tela Principal: Views

- Mais trabalho deve ser feito para criar uma surfaceView personalizada. Será necessário ouvir o evento surfaceCreated / Destroy, criar um thread de renderização, sincronizar o thread de renderização e o thread principal. No entanto, para personalizar uma View, tudo o que você precisa fazer é substituir o método onDraw.
- O tempo para atualizar é diferente. O mecanismo de atualização de visualização normal é restrito ou controlado pela estrutura .
- No Android, toda a atualização de uma View normal é sincronizada com o VSYNC para obter melhor suavidade. Agora, de volta ao surfaceView, você pode renderizá-lo a qualquer momento que desejar.

Trabalhando a Tela Principal: Views

- Para um teste inicial, será trabalhado uma view diretamente de um XML, para simplificação e entendimento.
- Desta forma, no arquivo XML (activity_main.xml), será necessário alterá-lo para conter apenas um LinearLayout e um TextView com a id aula02.

activity_main.xml | MainActivity.java

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent">
5      <TextView
6          android:id="@+id/aula02"
7          android:layout_width="wrap_content"
8          android:layout_height="wrap_content"
9          android:text="Iniciar" />
10 </LinearLayout>
```

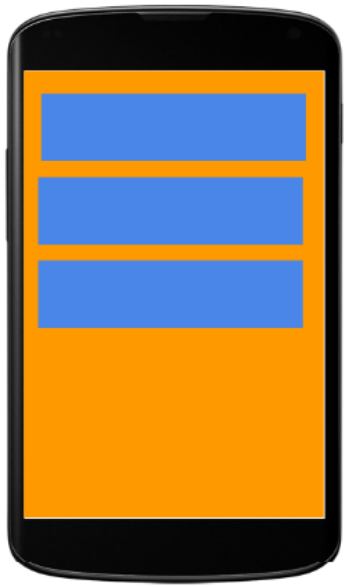
Trabalhando a Tela Principal: Views

- Para conseguir organizar as Views temos que usar os **ViewGroups**, que basicamente são uma estrutura que usamos para posicionar as Views da forma que precisarmos. A relação entre o ViewGroup e as Views é a mesma que de **Pai para Filhas**.
- Assim como as **Views**, os **ViewGroups** também podem ser customizados alterando cor, tamanho, posicionamento interno e espaços entre si.
- Cada **Android Layout** tem suas particularidades e funcionam de uma forma diferente.
- Cada **ViewGroup Pai** tem regras específicas sobre como ele irá posicionar as **Views Filhas** dentro dele, podendo posicionar as **Views** em uma única coluna **vertical** ou **horizontal**.

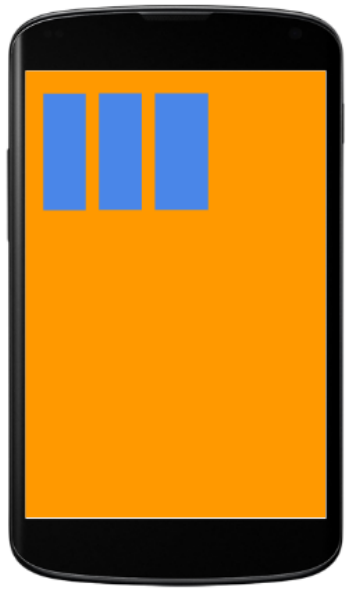
Trabalhando a Tela Principal: Views

- Para definirmos qual o tipo de orientação que queremos posicionar as nossas Views dentro do **LinearLayout**, nós usamos o atributo **orientation**. Esse atributo aceita dois valores, **vertical** e **horizontal**.

Layout Vertical



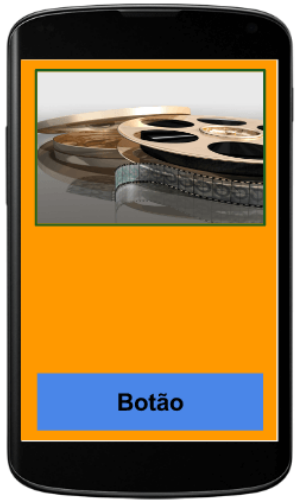
Layout Horizontal



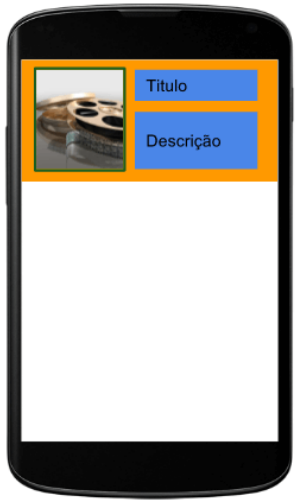
Trabalhando a Tela Principal: Views

- Podemos trabalhar também com o **RelativeLayout**.
- Com o **RelativeLayout**, você pode posicionar as **Views Filhas** em relação ao **Pai**, como por exemplo posicionar a **View** no topo ou no fim do layout. A outra opção, é posicionar as **Views** em relação a outras **Views** dentro do mesmo **RelativeLayout**.

Em relação ao Layout Pai



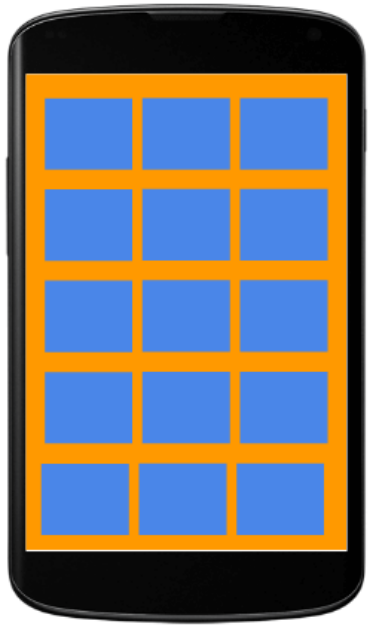
Em relação a outras Views



Trabalhando a Tela Principal: Views

- O **GridLayout** permite organizar as **Views** em um layout de grade onde um ou mais Views ficam dentro de **células de da grade**. Cada View pode ser configurada para ocupar várias células tanto horizontalmente e verticalmente. A posição e o comportamento da View dentro de uma célula é controlado através da definição das configurações de **gravity**.

Layout 03 Colunas



Trabalhando a Tela Principal: Activity

- Após a criação de uma ***view***, será necessário vinculá-la a uma ***Activity***.
  As ***Activities*** tem vários estados internos, elas são criadas, iniciadas, pausadas, reiniciadas e destruídas. Os eventos que se passam em uma ***Activity*** são conhecidos como **ciclo de vida**, e incluem esses estados.
- Será necessário alterar a ***MainActivity***: quando o assistente do Android Studio criou o projeto, fez com que a Activity fosse filha de ***AppCompatActivity***.
- A classe AppCompatActivity tem como objetivo disponibilizar nas versões mais antigas do Android algumas funcionalidades presentes apenas nas versões mais atuais do sistema. Podemos simplificá-la:

activity_main.xml | MainActivity.java

MainActivity

```
1  package com.example.arfso.aula02final;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5
6  public class MainActivity extends Activity {
7
8      @Override
9      protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_main);
12     }
13 }
14
```

Trabalhando a Tela Principal: Activity

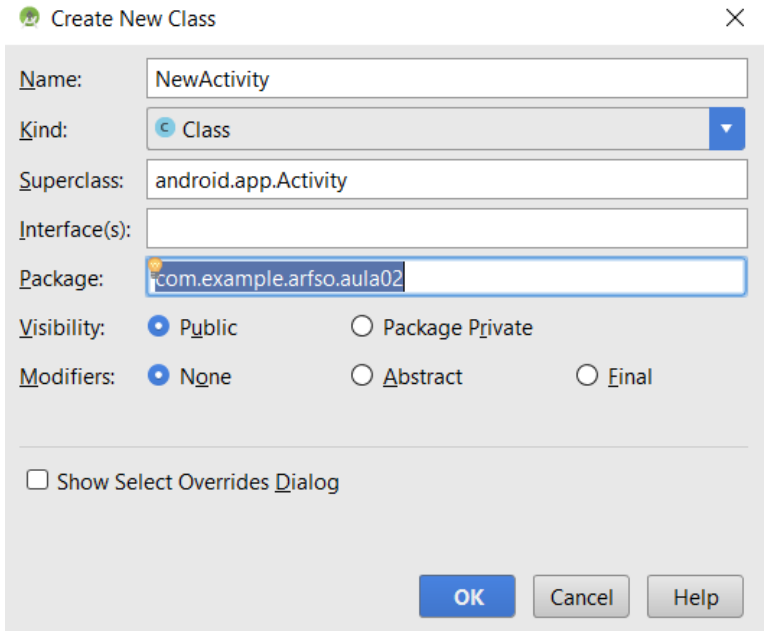
- Para criar uma transição de tela, podemos utilizar o texto criado pela nossa view através do textview aula02. A chamada deverá ser feita no onCreate da MainActivity.

```
package com.example.arfso.aula02final;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView alterar = (TextView) findViewById(R.id.aula02);

    }
}
```

Importação automática

Trabalhando a Tela Principal: Activity

- Devemos criar uma nova **activity** para desenvolver uma transição de tela. Desta forma, precisaremos de um novo XML para a view desta nova atividade e também modificar o arquivo **AndroidManifest.**
- Desta forma, será criado uma nova classe java (NewActivity), herdando a classe Activity no mesmo pacote da mainActivity: com.example.arfso.aula02



Trabalhando a Tela Principal: Activity

- Nesta nova Activity, podemos adicionar a classe onCreate da mesma forma da actvity principal.

activity_main.xml | MainActivity.java | NewActivity.java

```
1  package com.example.arfso.aula02final;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.support.annotation.Nullable;
6
7  /**
8   * Created by arfso on 21/08/2018.
9   */
10
11 public class NewActivity extends Activity {
12     @Override
13     protected void onCreate(@Nullable Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15     }
16 }
17
```

Importação automática

Trabalhando a Tela Principal

- Uma decisão importante quanto a criação de um aplicativo, diz respeito à orientação de tela: vertical (portrait) ou horizontal (landscape).
- No AndroidManifest.xml, será possível ver o registro de todas as activities de uma aplicação. Neste caso específico, como foi criado apenas a MainActivity, podemos vê-la registrada na tag <activity>.

Trabalhando a Tela Principal

- Para fixar uma orientação, podemos usar a propriedade android:screenOrientation na tag <activity> do XML:

activity_main.xml | MainActivity.java | AndroidManifest.xml | TelaPrincipal.java

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.arfso.aula02">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity"
                android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Trabalhando a Tela Principal

- Pode-se também limpar a tela principal da aplicação removendo a barra superior aplicação adicionando um tema fullscreen.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.example.arfso.aula02">
4
5      <application
6          android:icon="@mipmap/ic_launcher"
7          android:label="@string/app_name"
8          android:theme="@android:style/Theme.NoTitleBar.Fullscreen">
9          <activity android:name=".MainActivity"
10                 android:screenOrientation="portrait">
11             <intent-filter>
12                 <action android:name="android.intent.action.MAIN" />
13
14                 <category android:name="android.intent.category.LAUNCHER" />
15             </intent-filter>
16         </activity>
17     </application>
18 </manifest>
```

Trabalhando a Tela Principal

- Por último, será extremamente importante adicionar todas as **Activities** ao **AndroidManifest.**

```
<application
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.NoTitleBar.Fullscreen">
    <activity android:name=".MainActivity" android:screenOrientation="portrait">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".NewActivity"
        android:screenOrientation="portrait" />

</application>
```

Trabalhando a Tela Principal: Intent

- Com a nova atividade criada, podemos agora criar uma forma de ir da atividade principal para a nova. Isso pode ser feito de várias formas, uma delas seria ao clicar nas letras da TextView.
- Para sair de uma atividade para uma nova, precisaremos de uma intente.

Trabalhando a Tela Principal: Intent

- As **Intents** permitem que você interaja com componentes do mesmo aplicativo, bem como com componentes contribuídos por outras aplicações. Por exemplo, uma **Activity** pode iniciar uma **Activity** externa para tirar uma foto.
- As **Intents** são mensagens assíncronas que permitem que os componentes de um aplicativo solicitem a funcionalidade de outros componentes do Android.
- Uma **Intent** pode conter dados dentro dela através de um **Bundle**. Estes dados podem ser utilizados pelo componente que está recebendo a Intent.
- No Android, a **reutilização** de outros componentes de um aplicativo é um conceito conhecido como *tarefa*. Um aplicativo pode acessar outros componentes do Android para realizar uma *tarefa*.

Trabalhando a Tela Principal: Intent

- Na ***MainActivity***, podemos utilizar a função ***onClick*** para realizar a transição de tela com a ***intent***

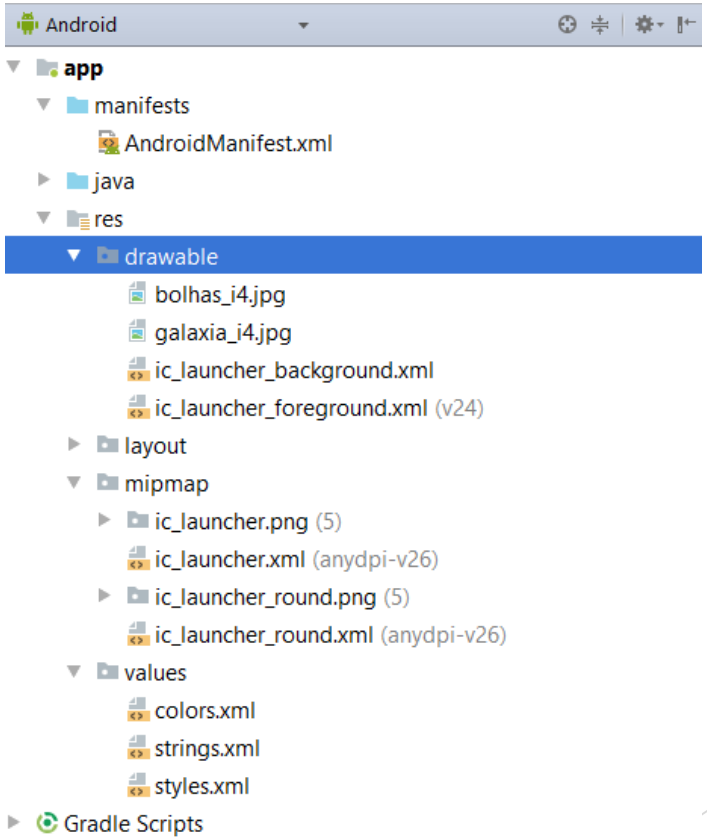
activity_main.xml | MainActivity.java | NewActivity.java | AndroidManifest.xml

MainActivity | onCreate() | new OnClickListener | onClick()

```
3  import android.app.Activity;
4  import android.content.Intent;
5  import android.os.Bundle;
6  import android.view.View;
7  import android.widget.TextView;
8
9  public class MainActivity extends Activity {
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_main);
15         TextView alterar = (TextView) findViewById(R.id.aula02);
16         alterar.setOnClickListener(new View.OnClickListener() {
17             @Override
18             public void onClick(View view) {
19                 Intent main = new Intent( packageContext: MainActivity.this,NewActivity.class);
20                 startActivity(main);
21             }
22         });
23
24     }
25 }
26
```

Transição de Tela

- Agora podemos trabalhar a **view** principal e criar uma **view** secundária de maneira que fique perceptível esta transição de tela.
- *Acrescentando um fundo (background) e um texto diferenciado, podemos finalizar este aplicativo.*
- *Caso queira colocar um arquivo de imagem e chamá-lo como background, a imagem deverá ser colocada na pasta app\src\main\res\drawable-nodpi do seu aplicativo.*



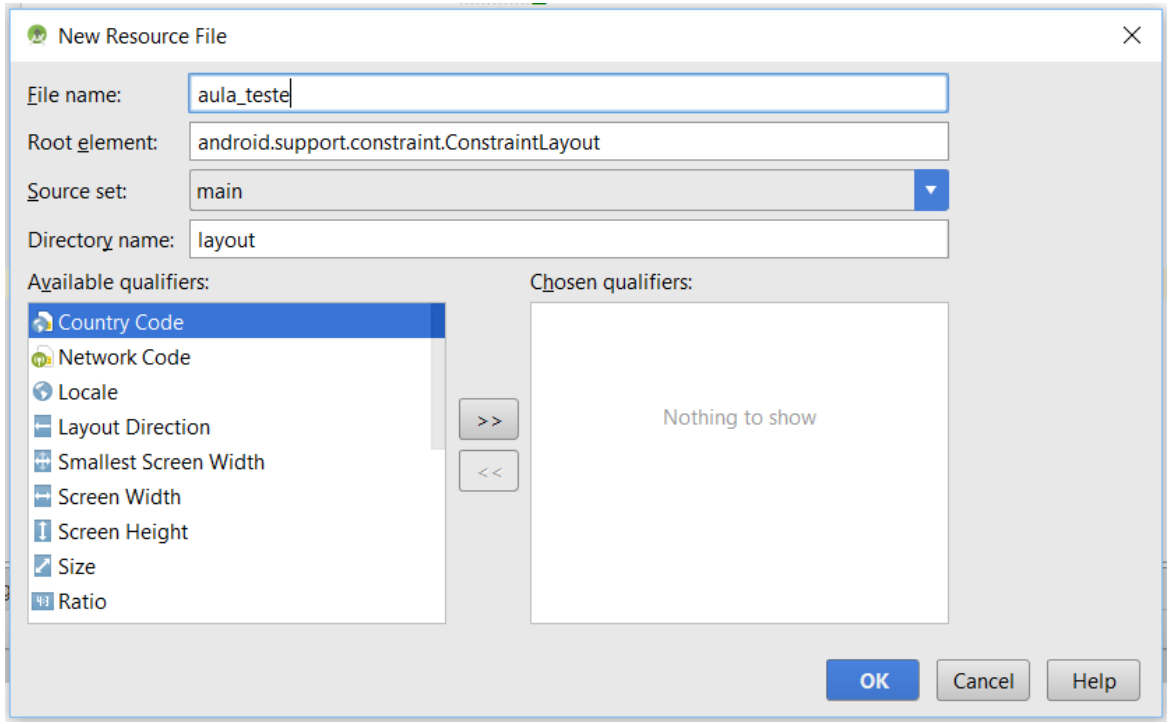
Transição de Tela

- Na View principal trabalhamos algumas modificações de design.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="center"
    android:background="@drawable/bolhas_i4"
    android:gravity="center"
    android:orientation="vertical">
    <TextView
        android:id="@+id/aula02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Iniciar"
        android:shadowColor="#000000"
        android:shadowRadius="2"
        android:textAlignment="center"
        android:textColor="#FF0000"
        android:textSize="70sp"
        android:textStyle="bold"/>
</LinearLayout>
```

Transição de Tela

- Da mesma forma, foi criada um novo arquivo XML em **layout** para trabalhar uma nova **view: aula_teste.xml**.



Transição de Tela

- Para a **view** secundária, algumas configurações diferentes.

activity_main.xml | aula_teste.xml | MainActivity.java | NewActivity.java | AndroidManifest.xml

android.support.constraint.ConstraintLayout > LinearLayout

```
4       android:layout_height="match_parent">
5
6       <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
7           android:layout_width="match_parent"
8           android:layout_height="match_parent"
9           android:layout_gravity="center"
10          android:background="@drawable/galaxia_i4"
11          android:gravity="center"
12          android:orientation="vertical">
13          <TextView
14              android:layout_width="wrap_content"
15              android:layout_height="wrap_content"
16              android:shadowColor="#000000"
17              android:shadowDx="3"
18              android:shadowDy="3"
19              android:shadowRadius="2"
20              android:text="Fim"
21              android:textAlignment="center"
22              android:textColor="#FF0000"
23              android:textSize="35sp"
24              android:textStyle="bold" />
25
26      </LinearLayout>
27
```

Transição de Tela

- Finalmente, precisaremos chamar este layout na **NewActivity**.

activity_main.xml | aula_teste.xml | MainActivity.java | NewActivity.java | AndroidManifest.xml

```
1  package com.example.arfso.aula02final;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5  import android.support.annotation.Nullable;
6
7  /**
8   * Created by arfso on 21/08/2018.
9   */
10
11 public class NewActivity extends Activity {
12     @Override
13     protected void onCreate(@Nullable Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.aula_teste);
16
17     }
18 }
19
```

Exercícios

- Acrescente ícones para o aplicativo
- *Crie a ação de voltar a tela anterior*
- *Tente trabalhar a transição de telas com o button*