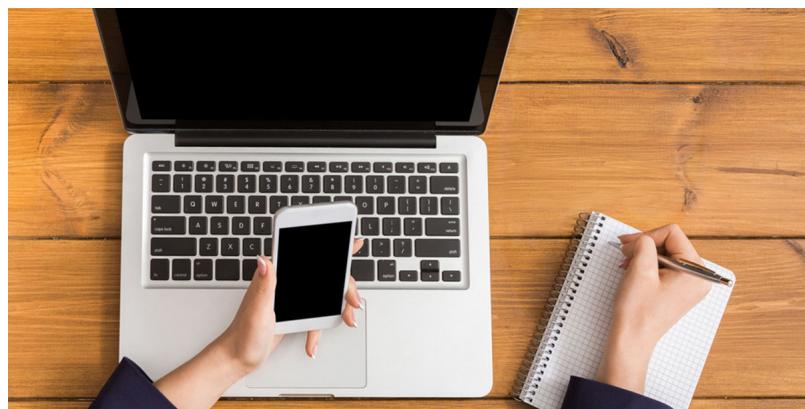


Programação para dispositivos móveis

Aula 1 - Introdução à computação móvel, a plataformas de desenvolvimento e à programação móvel Android

INTRODUÇÃO



O Android é um sistema operacional para dispositivos móveis open source baseado em Linux, criado em 2003 para o mercado de mobile. Desde então, a demanda por profissionais desta área tem exigido, cada vez mais, uma melhor qualificação profissional.

Esta aula tem como tema central apresentar os conceitos de computação móvel, dispositivos móveis, bem como o Android e seus componentes essenciais a um aplicativo.

OBJETIVOS



Descrever os principais conceitos referentes à tecnologia mobile;

Identificar os principais conceitos referentes ao sistema operacional Android;

Descrever os componentes básicos do sistema Android para a construção de um aplicativo.

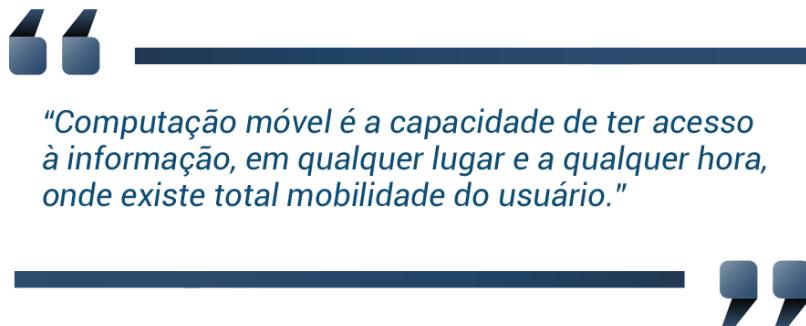
INTRODUÇÃO AO ANDROID

Você já deve ter notado que o crescimento das tecnologias da informação (TI) tem impactado profundamente nossas vidas, na medida em que tem modificado a forma como nos comunicamos, aprendemos e, porque não dizer, percebemos o mundo.

Precisamos cada vez mais de um maior volume, mais qualidade e velocidade no acesso às informações.

Neste "Admirável Mundo Novo" é fundamental o acesso às informações e serviços, independente de onde estejamos localizados e a qualquer momento, dando oportunidade ao aparecimento da computação móvel como um novo paradigma computacional.

Veja sua definição:

 "Computação móvel é a capacidade de ter acesso à informação, em qualquer lugar e a qualquer hora, onde existe total mobilidade do usuário."

DISPOSITIVO MÓVEL

Um dispositivo móvel usado em larga escala, a partir de 1986, foi o Teletrim, cuja finalidade era a recepção de mensagens de texto (de poucas linhas) aos usuários do serviço. Era equivalente ao serviço de SMS (Short Message Service) provido pelas operadoras de telefonia nos dias de hoje.



Fonte da Imagem:

Um dispositivo móvel (*handheld*) é um computador de bolso, normalmente equipado com uma pequena tela (*output*) e um teclado em miniatura (*input*). Em alguns dispositivos móveis, o teclado está incorporado à tela, no que chamamos de dispositivo touchscreen, tal como nos tablets.

Existem diversas categorias de dispositivos de computação móvel. Entre os mais comuns estão:

- *Smartphone*;
- *Tablet*;
- *PDA (Personal Digital Assistant)*;
- Celular;
- Console portátil;
- Coletor de dados;
- *GPS (Global Positioning System)*.

CONCEITO DE MOBILIDADE

O conceito base que impulsionou o desenvolvimento dos dispositivos móveis foi a mobilidade, que pode ser definida como:



"A capacidade de poder ser movido fisicamente e/ou poder ser utilizado enquanto está em movimento".



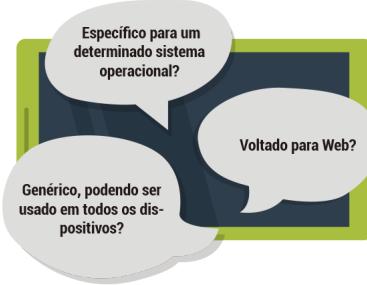
Para isso, os dispositivos móveis possuem determinadas características essenciais:



APLICATIVOS

Com o crescimento do uso de dispositivos móveis, surgiram alguns sistemas operacionais e também uma crescente demanda por aplicativos.

Mas que tipo de aplicativo deve-se construir?



Atenção

, Antes de iniciarmos o desenvolvimento de uma aplicação para um dispositivo móvel é necessário definir o tipo de aplicação desejada.

Os possíveis tipos são:

NATIVO

- Desenvolvidas especificamente para uma determinada plataforma móvel;
- Faz uso da linguagem de programação suportada pela plataforma e seu respectivo SDK (Software Development Kit);
- Normalmente, são instaladas através de uma loja de aplicativos, como, por exemplo, App Store e Google Play.

WEBMOBILE

- Diferente das aplicações nativas, consiste em um site com um layout otimizado para plataforma móvel;
- Faz uso de linguagens web (Html, Css, Javascript);
- Pode ser usado por qualquer plataforma móvel.

HÍBRIDA

- Consiste na combinação dos tipos nativo e WebMobile;
- Em geral, possui um navegador de internet customizado para o site do aplicativo;
- É desenvolvido para uma plataforma móvel específica;
- Tem se destacado nos últimos tempos.

MULTIPLATAFORMA

- Faz uso de Framework para geração de aplicações móveis.

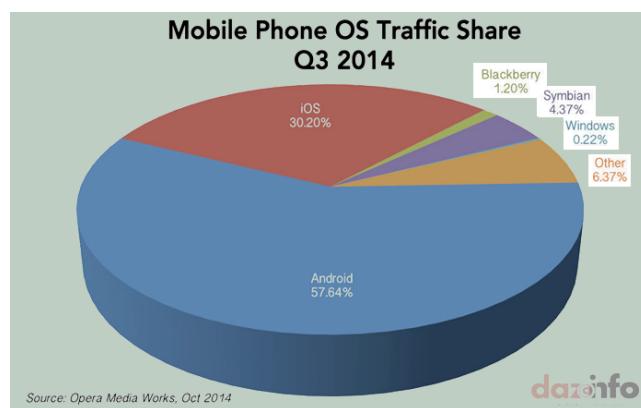
PRINCIPAIS SISTEMAS OPERACIONAIS DE DISPOSITIVOS MÓVEIS

Existem vários sistemas operacionais para dispositivos móveis no mercado, merecendo destaque:



É público e notório que o Android tornou-se o sistema operacional mais usado em dispositivos móveis.

Veja a ilustração de um levantamento realizado no 3º bimestre de 2014:



Sistema Operacional Android

Um pouco de história...

Android é um software open-source para dispositivos móveis, baseado no sistema operacional Linux.

Foi desenvolvido por uma empresa chamada Android Inc., situada na Califórnia-EUA, e fundada por Andy Rubin, Rick Miner, Nick Sears e Chris White, inicialmente para o desenvolvimento de aplicativos para celulares.

Em 2005, foi adquirida pela Google, que manteve Andy Rubin como colaborador no desenvolvimento da plataforma Android.

Em novembro de 2007, foi criada a Open Handset Alliance(OHA), formada por várias empresas do setor e capitaneada pela Google. A OHA fortaleceu o crescimento da plataforma Android.

Finalmente, em 2008, foi lançado o primeiro dispositivo móvel, usando o sistema operacional Android, batizado de HTC T-Mobile, cuja imagem é apresentada a seguir.

PRINCIPAIS CARACTERÍSTICAS DO ANDROID

Handset Layout	<ul style="list-style-type: none"> • Oferece a possibilidade de desenvolvermos layouts de qualidade para dispositivos mais simples, assim como para smartphones e tablets. Suporta inclusive 2D ou 3D.
Conectividade	<ul style="list-style-type: none"> • Disponibiliza suporte às tecnologias de GSM / EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC e WiMAX.
Armazenamento	<ul style="list-style-type: none"> • É utilizado o SQLite para o armazenamento de dados.
Supoorte de mídia	<ul style="list-style-type: none"> • Oferece suporte às tecnologias H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF e BMP.
Mensagens	<ul style="list-style-type: none"> • Oferece formas de envio de mensagens SMS (texto simples) e MMS (texto, imagem, áudio e vídeo).
Multi-touch	<ul style="list-style-type: none"> • Tem suporte nativo para multi-touch.
Multi-tasking	<ul style="list-style-type: none"> • Vários aplicativos podem ser executados simultaneamente.
Supoorte Java	<ul style="list-style-type: none"> • As aplicações desenvolvidas nesta plataforma são implementadas na linguagem Java.
Market	<ul style="list-style-type: none"> • É uma loja de aplicativos (gratuitos e pagos) que permite instalar diretamente no dispositivo sem a necessidade de computador.

MÁQUINA VIRTUAL ANDROID

Assim como na linguagem Java, a plataforma Android possui sua máquina virtual própria, denominada Dalvik Virtual Machine (DVM).

Otimizada para consumir menos memória, ela difere da Java Virtual Machine (JVM) porque os arquivos .class são convertidos para o formato .dex (Dalvik Executable), que corresponde à aplicação Android compilada, e compactados em um arquivo com extensão .apk (Android Package File) que representa a aplicação final.

A partir do Android 4.4, a DVM foi substituída pela Android Runtime (ART) que apresenta um desempenho muito superior em relação à DVM. Uma das principais diferenças entre a DVM e a ART é a forma de compilação.



Na DVM, o processo de compilação é baseado em JIT (Just in time). Como próprio nome diz, somente a parcela do código necessária para execução é compilada naquele momento. É importante lembrar que, devido somente uma parte do código ser compilada, esse processo consome menos memória e menos espaço físico no dispositivo.



Já na ART, ele é totalmente compilado na instalação do aplicativo e isso ocorre somente uma vez. Com isso, o código é muito mais rápido em sua execução porque não precisa ser compilado muitas vezes. Além disso, por demandar menor recursos de CPU, consome menos bateria.

PLATAFORMA DO ANDROID

 —  

"No Android, uma versão do sistema operacional é conhecida como plataforma" (LECHETA, 2015, p.43).

Cada versão possui um código identificador (número inteiro) denominado API Level, que corresponde à versão da plataforma Android.

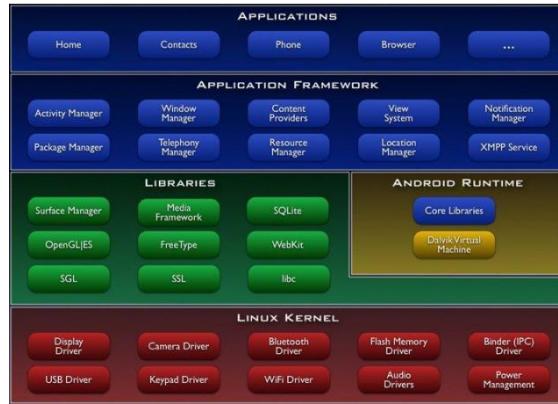
É importante ressaltar que o valor da API Level será usado quando forem definidos os dispositivos alvos para seu aplicativo.

Clique [aqui \(glossário\)](#) e baixe a tabela que apresenta a relação de Api Level do Android, até o presente momento.

ARQUITETURA ANDROID

Segundo a ilustração a seguir, observamos que a arquitetura do Android é agrupada em 3 níveis e é composta de 5 módulos:

- 1) Linux Kernel;
- 2) Libraries;
- 3) Android Runtime;
- 4) Application Framework;
- 5) Applications.



Veja maiores detalhes de cada módulo:

Applications



É nesse módulo que se encontram os aplicativos instalados, como, por exemplo:
 Cliente email;
 Navegador;
 Contatos;
 Mapas;
 Programas para SMS.

Application Framework



Fornece vários serviços para aplicações na forma de classes Java.

Os desenvolvedores de aplicativos têm permissão para fazer uso desses serviços em suas aplicações.

Os principais serviços são:

- Activity Manager - Controla todos os aspectos do ciclo de vida da aplicação e da pilha de atividade;
- Content Provider - Permite que os aplicativos publiquem e partilhem dados com outras aplicações;
- Resource Manager - Fornece acesso a recursos, como Strings, configurações de cores e

layouts de interface do usuário;

- Notifications Manager - Permite que os aplicativos exibam alertas e notificações para o usuário;
- View System - Conjunto de views destinado a criar interfaces de usuário.

Libraries



São as bibliotecas destinadas ao desenvolvimento Android.

Dentre elas, merecem destaque:

- Android.app - Fornece acesso à aplicação;
- Android.content - Facilita o acesso ao conteúdo, publicação e mensagens entre aplicativos e componentes dos aplicativos;
- Android.database - Usado para acessar os dados publicados por provedores de conteúdo e inclui classes de gerenciamento de banco de dados SQLite;
- Android.opengl - Uma interface Java para os gráficos OpenGL;
- Android.os - Fornece acesso aos serviços do sistema operacional padrão, incluindo mensagens, serviços do sistema e comunicação entre processos;
- Android.text - Usado para processar e manipular texto em uma tela do dispositivo;
- Android.view - Construção das interfaces com o usuário do aplicativo;
- Android.widget - Uma rica coleção de componentes de interface do usuário pré-construídos, tais como botões, etiquetas, exibições de lista, gerenciadores de layout, botões de rádio etc.

Android Runtime



Módulo que fornece a Dalvik Virtual Machine, que é uma espécie de Java Virtual Machine, especialmente projetado e otimizado para o Android.

Linux kernel



É o nível de abstração do hardware do dispositivo que proporciona o gerenciamento de memória, de energia, de processos, de segurança, dentre outros;

Possui drivers essenciais do hardware.

COMPONENTES DE UM APlicATIVO ANDROID

Componentes são blocos de construção essenciais a um aplicativo Android.

O arquivo `AndroidManifest.xml`, que é único em cada aplicação, descreve cada componente da aplicação e como eles interagem.

Nele constam todas as configurações necessárias para executar a aplicação, como, por exemplo, o nome do pacote utilizado, o nome das classes de cada activity, as permissões que o aplicativo possui, qual a versão mínima da API Android, dentre outras configurações.



Os principais componentes que podem ser utilizadas dentro de uma aplicação Android são:

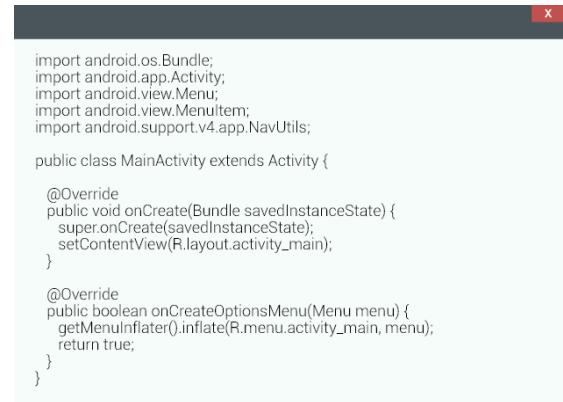
Activity

É considerado o componente base de uma aplicação Android. Por isso, é o mais utilizado.

Consiste em uma classe gerenciadora de UI (Interface do usuário). Representa uma única tela do usuário.

Tanto o fluxo da aplicação como eventos de tela são de sua responsabilidade. Isso não significa que todo aplicativo precisa ter uma interface do usuário, mas, se tiver, precisará de, pelo menos, uma Activity.

Exemplo:



```
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.MenuItem;
import android.support.v4.app.NavUtils;

public class MainActivity extends Activity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
}
```

Intent

Uma Intent (intenção) é uma descrição abstrata de uma operação a ser executada. Ela pode ser utilizada para iniciar uma Activity, “ativar” um broadcast, enviar uma mensagem para uma aplicação que roda em outro processo etc. Uma Intent faz parte da arquitetura do Android e é um conceito básico que deve ser dominado por todos que desejam programar para Android.

Sendo assim, em linguagem mais humana, uma Intent representaria uma "Mensagem", um pedido que é encaminhado ao sistema operacional. O sistema pegará a mensagem, verificará qual é a “Intenção da mensagem” e tomará uma decisão que pode ser desde abrir uma página na Web (um browser abrirá e a página será exibida), fazer uma chamada telefônica ou iniciar uma Activity.

Service

Quando o aplicativo possuir um ciclo de vida mais longo, normalmente colocamos em um Service.

Conhecido por ser um componente de background, responsável pelo processamento em segundo plano associado a uma aplicação.

Geralmente é utilizado para realizar tarefas de sincronização, como, por exemplo, sincronização de dados em segundo plano.

Exemplo:

```

import android.app.Service;
import android.os.IBinder;
import android.content.Intent;
import android.os.Bundle;

public class HelloService extends Service {
    int mStartMode;
    IBinder mIBinder;
    boolean mAllowRebind;
    @Override
    public void onCreate() {
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
    }
    @Override
    public IBinder onBind(Intent intent) {
        return mIBinder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        return mAllowRebind;
    }
    @Override
    public void onRebind(Intent intent) {
    }
    @Override
    public void onDestroy() {
    }
}

```

BroadCast Receiver

Lidam com a comunicação entre sistema operacional e aplicativos Android. São responsáveis por tratar eventos do sistema ou de outras aplicações.

Se um aplicativo precisar receber e responder a qualquer evento global, precisa registrar-se como um BroadCast Receiver.

Exemplo:

```

public class MyReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Intent Detected.", Toast.LENGTH_LONG).show();
    }
}

```

Content Provider

Responsável pela gestão de dados e banco de dados.

Além de permitir o compartilhamento de dados entre aplicações, centraliza as rotinas de armazenamento e recuperação em um único local, provendo às aplicações dados.

Em outras palavras, podemos afirmar que corresponde a uma abstração de dados para seus clientes.

É importante lembrar que o *Content Provider* pode acessar qualquer forma de armazenamento de dados existente na plataforma Android, entre eles arquivos, banco de dados *SQLite* ou até mesmo mapas *hash* de memória, caso não precise persistir dados.

Exemplo:



COMPONENTES ADICIONAIS:

FRAGMENTS

- Permite a modularização da interface do usuário;
- Representam uma parte da UI em uma atividade;
- Assim como o Activity, possuem seu próprio ciclo de vida e podem ser definidos como fragmento de uma tela em um aplicativo Android;
- São uma espécie de “subatividade” que pode ser reutilizada em diferentes atividades.

VIEWS

- Elementos de interface do usuário que são desenhados na tela, como botões, quadros de texto e até mesmo gerenciadores de layouts;
- São a classe-pai de todos os componentes visuais.

LAYOUTS

- Controlam o formato de tela e a aparência da interface do usuário, sendo possível serem criados com declaração de elementos XML ou através de programação Java.

INTENTS

- Representam uma ação que a aplicação deseja executar. Isso se dá através de mensagem (Broadcast) enviada ao sistema operacional, que decidirá a realização ou não da ação, a partir do conteúdo dessa mensagem;
- É importante lembrar que se constituem uma intenção, como próprio nome diz. Não há certeza de que a aplicação será executada.

MANIFEST

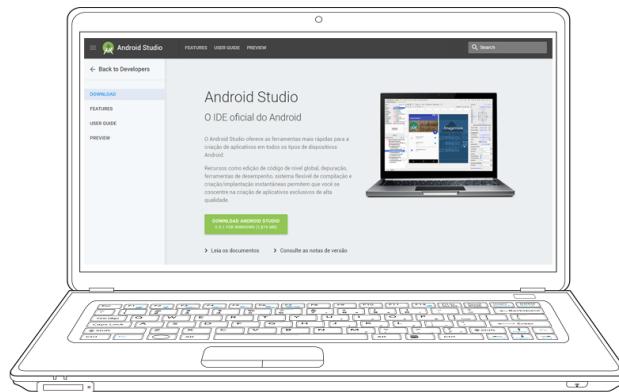
- Arquivo que possui informações essenciais referentes à configuração para execução da aplicação, como, por exemplo, nome do pacote utilizado, nome das classes de cada activity e outros;
- Deve estar na pasta raiz do projeto.

PRINCIPAIS IDES:

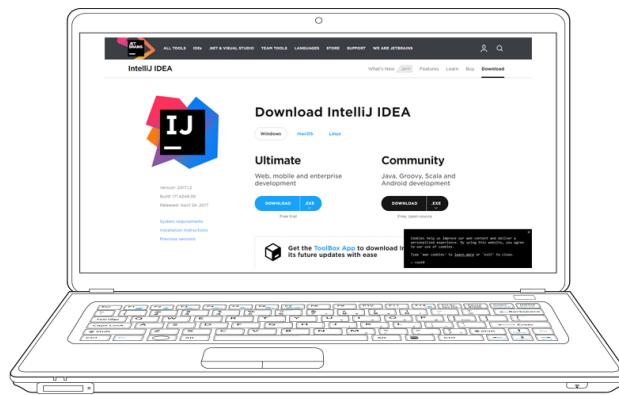
A IDE é um ambiente integrado, que acelera o desenvolvimento de aplicativos durante a fase de programação.

Abaixo os 3 IDEs mais populares do mercado, para a plataforma Android:

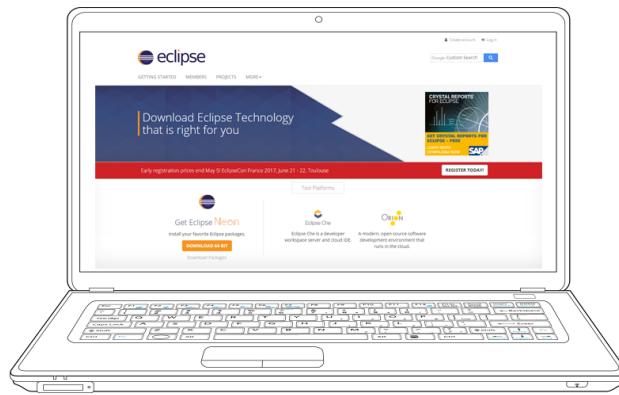
Android Studio:
<https://developer.android.com/sdk/index.html>



IntelliJ:
<https://www.jetbrains.com/idea/download/>



Eclipse:
<https://eclipse.org/downloads/>



Atenção

, Em nossas aulas usaremos a IDE Android Studio.

ATIVIDADE

A figura abaixo ilustra os principais componentes da plataforma de desenvolvimento Android. Explique cada um dos componentes demonstrados:



Resposta Correta

Glossário

Programação para dispositivos móveis

Aula 2 - Activity e Intent

INTRODUÇÃO



Esta aula visa apresentar a classe Activity, a pilha de execução de uma aplicação Android, os estados de uma Activity e seu ciclo de vida. Também apresenta a Classe Intent e Intent Filters, bem como desenvolvimento de um pequeno aplicativo.

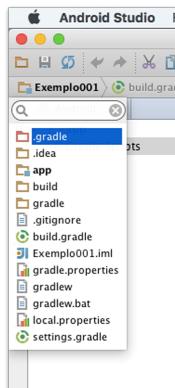
OBJETIVOS



Identificar os principais conceitos referentes à Activity Android, bem como aplicá-la no desenvolvimento de uma aplicação simples;

Identificar os principais conceitos referentes à Intent e Intent Filters e seu uso no desenvolvimento de uma aplicação simples.

ESTRUTURA DE UM PROJETO ANDROID



Fonte da Imagem:

O Android Studio pode abrir um projeto de cada vez.

Conforme é demonstrado na figura, cada projeto pode possuir um ou mais módulos.

Observe, na tabela, os arquivos do diretório raiz do projeto:

Pasta	Descrição
app	Módulo app do projeto. Este módulo é padrão.
build.gradle	Arquivo de configuração do Gradle. Vale para todo o projeto, incluindo todos os módulos.
gradle.properties	Arquivo de propriedades para customizar o build do Gradle.
gradlew.bat	Script que executa o build do Gradle para compilar o projeto.
local.properties	Arquivo com as configurações locais do projeto.
settings.gradle	Arquivo de configuração do gradle que indica quais módulos devem ser compilados.

Veja, também, os arquivos do módulo app:

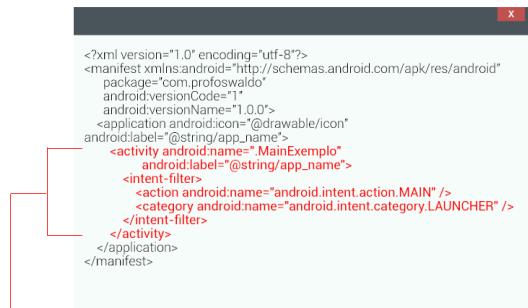
Pasta	Descrição
build	Local onde ficam os arquivos compilados do módulo.
R.java	A classe R que é gerada automaticamente ao se compilar o projeto. Esta permite que a aplicação acesse qualquer recurso.
libs	Onde devem ser inseridos os arquivos .jars a serem compilados com o projeto.
src/main/java	Pasta que contém os arquivos Java.
src/main/res	Pasta que contém os recursos da aplicação.
res/drawable	Pasta que contém as imagens da aplicação.
res/mipmap	Pasta com o ícone da aplicação.
res/layout	Pasta que contém os arquivos XML de layouts para construir as telas da aplicação.
res/menu	Pasta que contém os arquivos XML que criam os menus da aplicação, que são os botões com ações na action bar.
res/values	Pasta que contém os arquivos XML utilizados para a internacionalização, configuração de temas e outras configurações.

Agora, vamos conhecer detalhes sobre alguns tipos de arquivo:

Arquivo AndroidManifest.xml

Considerado um dos principais arquivos de sua aplicação. É nele que são descritas informações essenciais à execução de seu projeto como, por exemplo:

- Nome do pacote utilizado;
- Nome das Activities;
- Permissões que o aplicativo possui;
- Versão mínima da API Android.



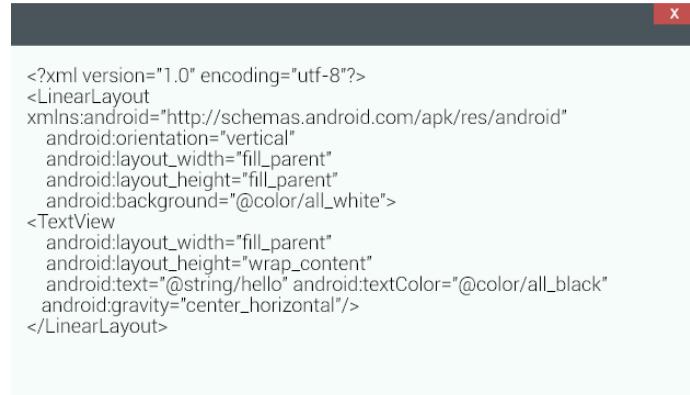
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.profosaldo"
    android:versionCode="1"
    android:versionName="1.0.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".MainExemplo"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Precisamos declarar obrigatoriamente todas as Activities implementadas no projeto nesse arquivo. Isso é realizado através da tag `<activity></activity>`, como podemos observar no exemplo.

Arquivo activity_main.xml

Esse arquivo, por default, possui este nome. Porém, podemos escolher um nome mais adequado.

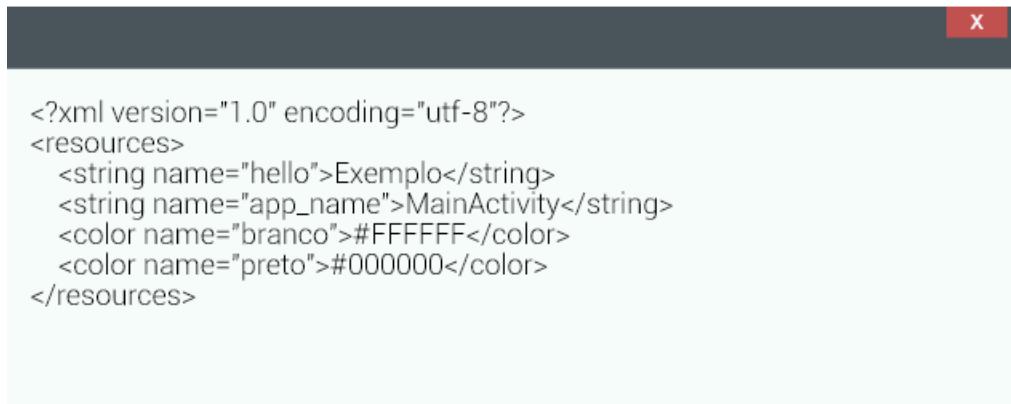
Nesse arquivo são definidas as configurações para criação do layout da tela.



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@color/all_white">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" android:textColor="@color/all_black"
        android:gravity="center_horizontal"/>
</LinearLayout>
```

Arquivo strings.xml

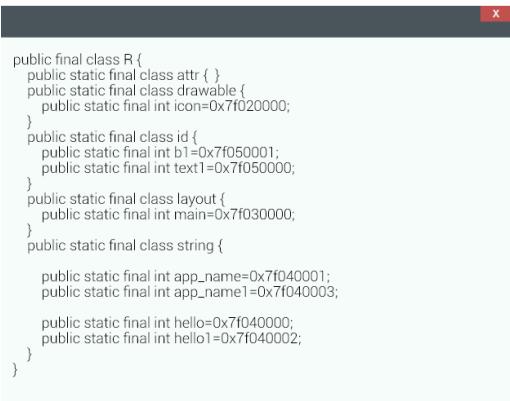
Esse arquivo centraliza as mensagens de seu aplicativo. Facilita muito, inclusive, a internacionalização do aplicativo.



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Exemplo</string>
    <string name="app_name">MainActivity</string>
    <color name="branco">#FFFFFF</color>
    <color name="preto">#000000</color>
</resources>
```

Classe R

Esse arquivo possui as referências para acessar os recursos de seu projeto.
É gerada automaticamente pelo compilador. É recomendável que essa classe não seja alterada manualmente.



```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int b1=0x7f050001;
        public static final int text1=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int app_name1=0x7f040003;
        public static final int hello=0x7f040000;
        public static final int hello1=0x7f040002;
    }
}
```

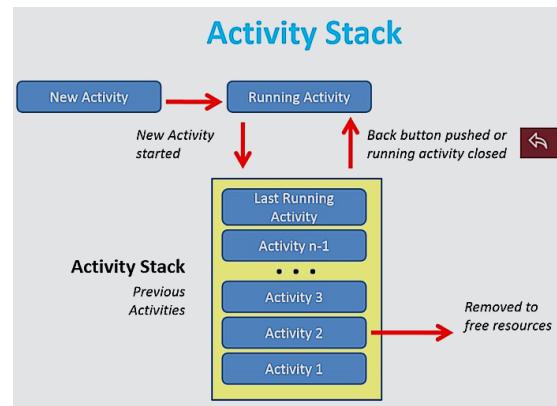
ACTIVITY

Muito similar ao JFrame do J2SE, a Activity é responsável por construir uma tela em Android, bem como tratar os eventos gerados por ela. Toda aplicação Android deve implementar ao menos um Activity, podendo chamar outras Activities.

O Android é responsável por gerenciar o ciclo de vida dos Activities. Para tanto, faz uso do conceito de pilha, chamada de "Activity Stacks" (pilha de atividades). Toda Activity ao ser executada é inserida no topo dessa pilha. A Activity anterior é parada e move-se para baixo da pilha.

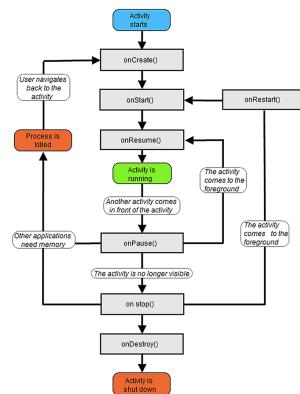
O Android pode até mesmo encerrar Activities, se precisar de recursos. Neste caso, ele verifica a pilha de atividade para determinar a prioridade das atividades e quais podem ser fechadas.

Veja na imagem o exemplo de uma pilha de atividades:



Fonte: <http://4.bp.blogspot.com/-HSwGfpS91g/UZx5IRzamSI/AAAAAAAEEs/y6wK1CnLUw0/s1600/ACTIVITYSTA.png>

CICLO DE VIDA ACTIVITY



Observamos, na imagem, os principais métodos do ciclo de vida Activity. São eles:

onCreate()
X

onCreate()

É a primeira função executada quando a Activity é criada. Tem por responsabilidade carregar os layouts XML, inicializar os objetos, variáveis e outras operações de inicialização. É importante lembrar que é executada somente uma vez.

onStart()

É executado antes da Activity ficar visível na tela do dispositivo, podendo ser chamado após os métodos **onCreate()** ou **onRestart()**.

onResume()

Representa o estado que a Activity está executando. É chamado logo após o evento onStart.

onRestart()

É chamado imediatamente após ao método onStart(), quando uma Activity que estava parada volta ao foco.

onPause()

É chamado sempre que a tela da Activity fechar. Isto ocorre quando uma outra Activity (da sua aplicação ou não) ganhar o foco.

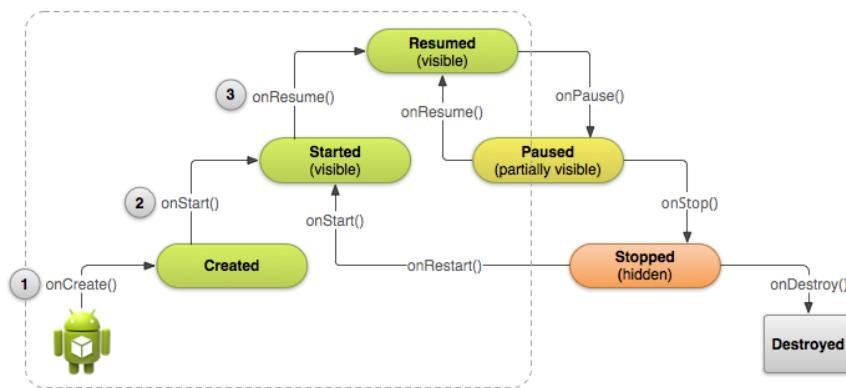
onStop()

É chamado após o método **onPause()**, quando a Activity não está mais visível e está sendo encerrada.

onDestroy()

É chamado antes da Activity ser destruída e logo após será liberada a memória.

Você pode observar no diagrama referente ao ciclo de vida da Activity os 3 níveis:



Entire lifetime	Visible lifetime	Foreground lifetime
Tempo de vida completo Ocorre desde a primeira chamada ao método onCreate() até a chamada do método onDestroy() , os quais são executados apenas uma única vez durante o ciclo de vida da Activity.	Tempo de vida visível Ocorre entre uma chamada do método onStart() e a chamada correspondente do método onStop() . A Activity pode estar no topo da pilha (visível para o usuário) ou em segundo plano.	Tempo de vida no topo da pilha Ocorre entre uma chamada do método onResume() e a chamada correspondente do método onPause() . A Activity está no topo da pilha e interagindo com o usuário.

Exemplo

, Antes de seguir em frente, clique aqui ([galeria/aula2/docs/a02_t05.pdf](#)) e veja um exemplo.

APLICANDO INTENT NA PRÁTICA

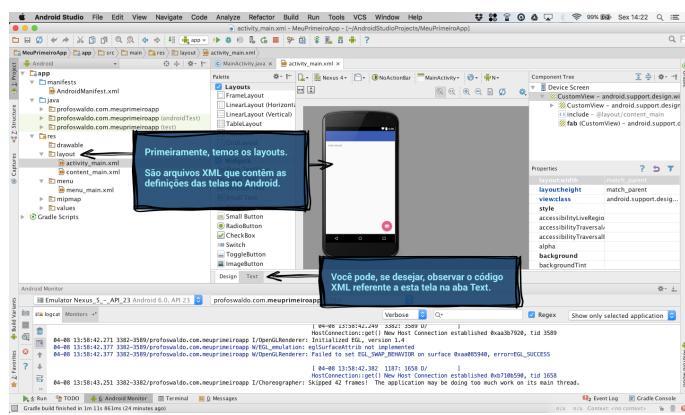
Embora você ainda não conheça todos os componentes e a estrutura do Android Studio, vamos implementar um pequeno exemplo demonstrando que é muito fácil desenvolver uma aplicação com o Android Studio.

Execute o Android Studio e assista ao vídeo:



Antes vamos fazer uma análise mais detalhada de nosso app, lembrando que ainda discutiremos cada componente com mais profundidade nas próximas aulas.

Veja a tela abaixo:



Vinculado a essa tela, temos uma Activity, no nosso caso MainActivity.

Trata-se de um arquivo Java. Nele, entre outras atividades, são definidos os tratamentos dos eventos gerados em nosso dispositivo.

Veja a seguir:

Viu como foi fácil?

CLASSE INTENT

Como estudamos, cada Activity corresponde a uma tela de nossa aplicação no Android. Muito embora seja muito mais do que simplesmente uma tela, não é mesmo?

E se precisássemos trabalhar com mais de uma tela?

Devido à limitação de tamanho de nossos dispositivos, é muito comum distribuirmos componentes por várias telas visando facilitar o uso da aplicação.

O que fazer então?

Entre os vários componentes fundamentais, na programação de aplicativos Android (como Activities, Services e BroadCast Receivers), a **Intent** possibilita realizar a ligação, em tempo de execução, de componentes separados (por exemplo, chamar Activities diferentes).

Em outras palavras, podemos dizer que uma Intent nada mais é do que a intenção da aplicação em realizar uma determinada tarefa.

Trata-se de uma intenção, ou seja, não necessariamente será executada, pois depende da permissão do dispositivo.

Mas isso como é possível?

A Intent envia ao sistema operacional o equivalente a uma mensagem (broadcast). Este receberá a chamada e, dependendo do conteúdo, tomará as **providências necessárias (glossário)**.

Uma Intent é basicamente um conjunto de dados que possui informações de interesse para os componentes que a recebem e também para o Android.

Deve conter:

Nome do componente

Representa o nome do componente que tratará a Intent. Deve ser o nome completo da classe alvo do componente. Este nome deve ser uma combinação do pacote do componente e o pacote definido no manifesto.

Ação

É o que você deseja executar propriamente dito. Consiste em uma String com o nome da ação a ser executada. No caso de um broadcast, a ação que aconteceu e que está sendo reportada.

Dados

URI e MIME type dos dados adicionados à Intent. Diferentes ações são paradas com diferentes tipos de dados.
Exemplo: Ao enviar foto para que outro aplicativo trate, precisamos definir a URI e o MIME type.

Categoria

String com informações adicionais sobre o tipo do componente que deve tratar a Intent. Qualquer número de categorias pode ser adicionado em uma Intent.

Extras

Pares chave-valor para informações adicionais que devem ser entregues para o componente que tratará a Intent.

Flags

Funcionam como instruções para o sistema Android, de como lançar uma Activity ou como tratá-la depois de lançada. Todas as falas são definidas na classe Intent.

Podemos definir ações específicas de nossa aplicação, biblioteca ou ações pré-definidas conforme a tabela abaixo:

Constante	Componente alvo	Ação
ACTION_CALL	activity	Inicia uma chamada de telefone.
ACTION_EDIT	activity	Mostra dados para o usuário editar.
ACTION_MAIN	activity	Inicia a atividade que está marcada como inicial em modo vazio e com nenhum retorno.
ACTION_SYNC	activity	Sincroniza dados no servidor a partir do dispositivo móvel.
ACTION_BATTERY_LOW	broadcast receiver	Mostra um aviso que a bateria está baixa.
ACTION_HEADSET_PLUG	broadcast receiver	Mostra que um fone de ouvido foi plugado no dispositivo ou desplugado.
ACTION_SCREEN_ON	broadcast receiver	A tela foi ligada.
ACTION_TIMEZONE_CHANGED	broadcast receiver	As configurações da zona de tempo foram mudadas.

Segue abaixo as principais constantes pré-definidas:

Constante	Significado
CATEGORY_BROWSABLE	A atividade alvo pode ser chamada de maneira segura pelo navegador para mostrar dados referenciados por um link (imagem ou e-mail etc.).
CATEGORY_GADGET	A atividade pode ser embarcada dentro de outra atividade que hospede gadgets.
CATEGORY_HOME	A atividade mostra a tela home. A primeira tela que o usuário vê quando o dispositivo é ligado ou quando a tela HOME é pressionada.
CATEGORY_LAUNCHER	A atividade pode ser a atividade inicial ou uma tarefa e é listada no tipo da aplicação launcher.
CATEGORY_PREFERENCE	A atividade alvo é o painel de preferências.

TIPOS DE INTENTS

Como estudamos, cada Activity corresponde a uma tela de nossa aplicação no Android. Muito embora seja muito mais do que simplesmente uma tela, não é mesmo?



INTENT FILTER

Basicamente, um Intent Filter informa ao sistema quais Intents um certo componente pode tratar. Um componente pode ter uma ou mais Intent Filters. Veja o exemplo a seguir:



```

<activity android:name=".MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:name=".ShareActivity">
    <!-- The activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <action android:name="android.intent.action.SEND_MULTIPLE" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="application/vnd.google.panorama360-jpg" />
        <data android:mimeType="image/*" />
        <data android:mimeType="video/*" />
    </intent-filter>
</activity>

```

Atenção

, Com relação ao tipo implícito, as Intents serão entregues se um dos filtros atender aos critérios da Intent.

Já no caso da explícita, será entregue diretamente ao componente designado, não importando o filtro, pois nem chega a consultá-lo.

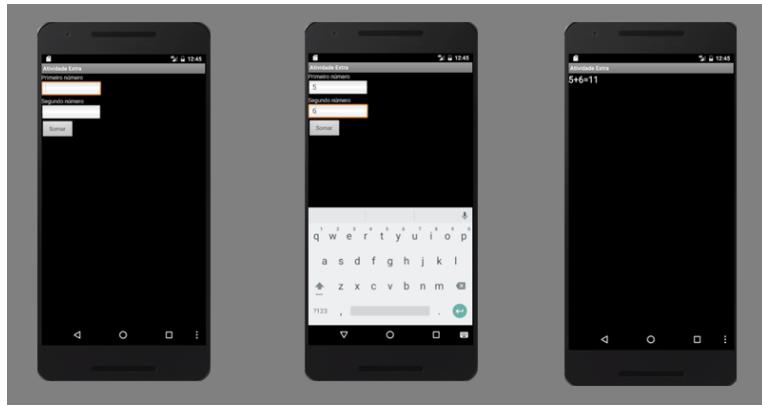
APLICANDO INTENT NA PRÁTICA

Vamos criar um projeto demonstrando a Intent em ação? Aperte o play e vamos lá!

ATIVIDADE

Para fixarmos o conteúdo é muito importante praticarmos. Vamos lá!

Observe as telas abaixo:



Desenvolva um aplicativo que, conforme demonstrado nas telas acima, efetue a soma de 2 valores.

Este deverá ser composto de duas Activities. Sendo uma principal, que efetuará a leitura dos dados e a soma, e uma resultado que deverá exibir o resultado desta soma.

Depois de terminar a atividade, veja o gabarito e compare.

PDF

, Clique aqui ([galeria/aula2/docs/a02_t10.pdf](#)) para ver o gabarito.

Glossário

PROVIDÊNCIAS NECESSÁRIAS

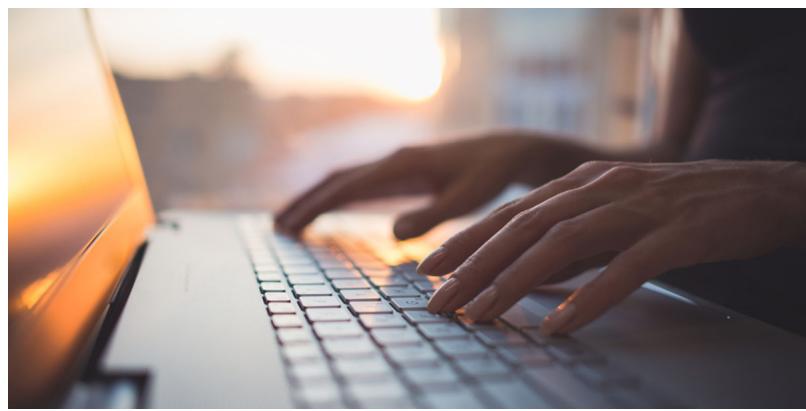
Como exemplo, podemos citar:

- Iniciar uma nova Activity;
- Iniciar Bluetooth do aparelho;
- Ligar o GPS (Global Positioning System);
- Efetuar uma ligação telefônica;
- Abrir o programa de envio de SMS (Short Message Service);
- Chamar o navegador Web;
- Enviar mensagens para o Sistema Operacional.

Programação para dispositivos móveis

Aula 3 - Interface Gráfica do Usuário

INTRODUÇÃO



A criação de uma interface gráfica mobile é uma das grandes dificuldades no desenvolvimento voltado para telas de pequeno tamanho. Para tanto, o Android oferece um conjunto de objetos que facilitam este tipo de desenvolvimento.

Esta aula visa apresentar a classe View, ViewGroup, bem como os principais Widgets e gerenciadores de layout Android.

OBJETIVOS



Descrever a classe View, View Group e seus tipos;

Explicar os principais gerenciadores de layouts;

Explicar a implementação de componentes gráficos e gerenciadores de layouts em aplicativo Android.

COMPONENTES GRÁFICOS



Fonte da Imagem:

A interface com o usuário é um ponto fundamental para o desenvolvimento de aplicativos Android, pois é através desta que o usuário interage.

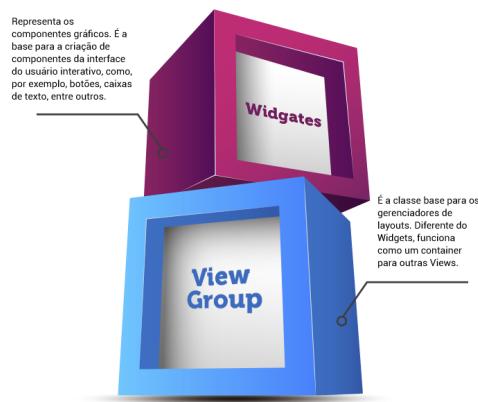
A construção desse canal de comunicação demanda bastante tempo para desenvolvimento, principalmente no cenário mobile. Nele existe uma variedade muito grande de tipos dispositivos móveis, que, muitas vezes, possuem uma capacidade de tela superior até mesmo às televisões de última geração.

Para tanto, o Android oferece suporte nativo para o desenvolvimento de interfaces gráficas sofisticadas.

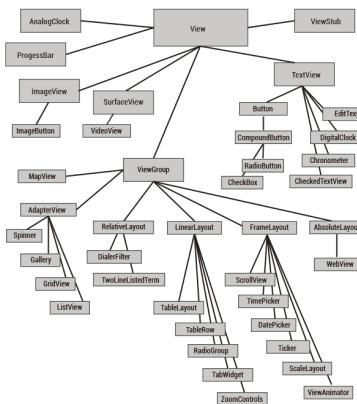
Podemos destacar a View, que é a classe base de todo e qualquer componente gráfico. Responsável por desenhar e controlar os eventos.

TIPOS DE COMPONENTES

O Android fornece um conjunto muito grande de componentes, mas podemos agrupá-los em dois tipos:



A imagem abaixo ilustra a hierarquia da classe View, demonstrando que o Android oferece um sofisticado e poderoso modelo baseado em Widgets e Layouts.



TIPOS DE COMPONENTES GRÁFICOS

Existem vários tipos de componentes. Entre eles, podemos destacar:

TEXTVIEW

Considerado um dos componentes mais usados em Android. Define um texto que será exibido na tela.

Por default, não pode ser editado, embora possamos alterar essa configuração.

< TextView

```

        android:id="@+id/resultado"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:scrollHorizontally="true"
        android:textSize="40dp" />
  
```

BUTTON

Nada mais é do que um simples botão.

Após clicarmos nele, podemos executar uma ação.

< Button

```

        android:id="@+id/button1"
        android:layout_width="80dp"
        android:layout_height="70dp"
        android:text="7"
        android:textSize="40dp" />
  
```

IMAGEVIEW

Responsável por inserir imagens.

< ImageView

```

        android:id="@+id/imageView1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:src="@drawable/ic_launcher" />
  
```

EDITTEXT

Uma caixa onde é digitada alguma informação, que poderá ser usada para interagir com a aplicação Android.

< EditText

```

        android:id="@+id/editText1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:ems="10" />
  
```

CHECKBOX

Um botão do tipo “caixa de seleção”, onde existe dois estados: verdadeiro ou falso.

```
< CheckBox
    android:id="@+id/cbbTeste"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Exemplo" />
```

RADIOBUTTON

Similar ao CheckBox, porém não deve permitir a seleção de mais de uma opção.

RADIOGROUP

Tem por objetivo agrupar estes radiobuttons, permitindo a seleção de apenas um RadioButton dentro deste Radiogroup.

```
< RadioGroup
    android:id="@+id/radioSex"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >

    < RadioButton
        android:id="@+id/radioMale"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/radio_male"
        android:checked="true" />

    < RadioButton
        android:id="@+id/radioFemale"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/radio_female" />
```

IMPLEMENTAÇÃO DE UIS

Podemos implementar as Uis do usuário de forma declarativa e programática. Na declarativa, fazemos uso do XML para declarar a aparência da interface.

Já na programática, implementamos através da linguagem Java.

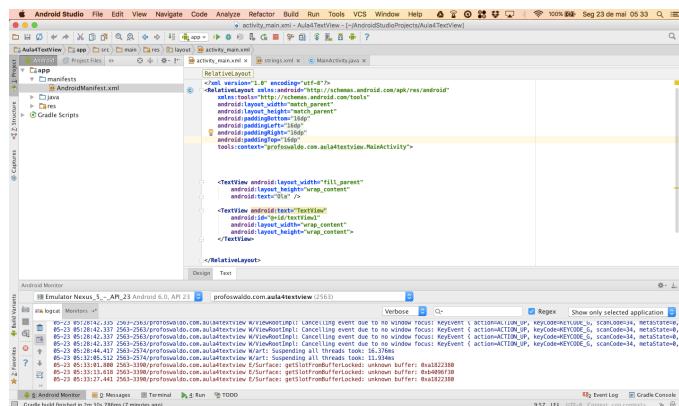
Podemos fazer uma analogia de declarativa com HTML e programática com a Swing e AWT do Java.

Devemos dar preferência sempre a forma declarativa, pois tende a ser mais fácil de entender, desenvolver e até mesmo manter.

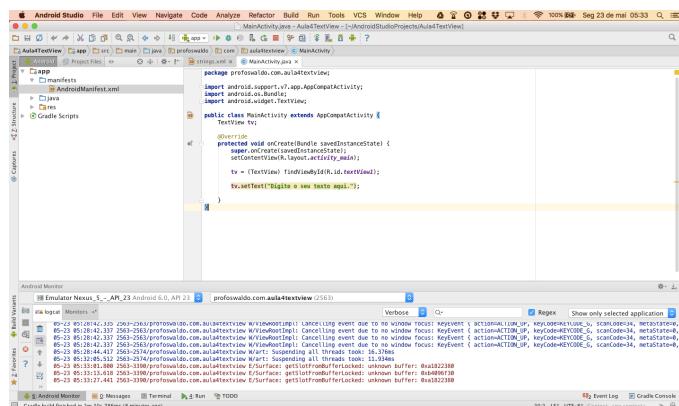


Exemplo

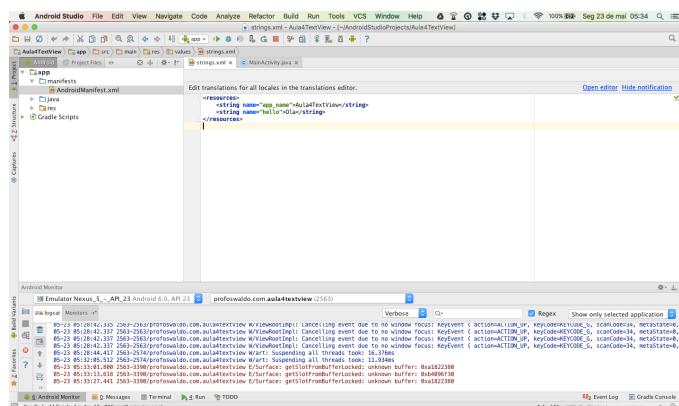
, Crie um projeto com o nome que desejar e altere o arquivo activity_main.xml, conforme mostrado na tela:, ,



, , Agora vamos definir nosso arquivo **MainActivity**, conforme tela abaixo:, ,



, , Não podemos esquecer do arquivo **string.xml**, ,



Gerenciadores de Layout

Aprendemos como desenvolver telas gráficas no Android, a partir de elementos visuais conhecidos por Widgets.

Porém, precisamos melhorar a qualidade de nossas telas, tarefa que nem sempre é tão fácil, no que tange à implementação.

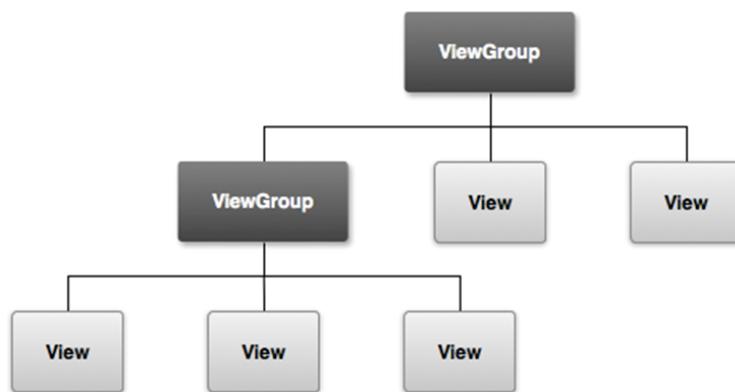
Nossas aplicações precisam ser executadas em vários tipos de dispositivos diferentes, mas nem todos possuem as mesmas características, como, por exemplo, tamanho da tela.

Já imaginou o trabalho que seria para organizar os componentes em cada tipo de tela?

Nesse ponto, o Android nos ajuda muito através de um conjunto de classes, pré-definidas, denominadas Gerenciadores de Layout, que tem por finalidade organizar os componentes de nossas telas.



Observe a figura abaixo:



Como apresentado anteriormente, a classe View é a base para todo e qualquer componente visual no Android. Já a classe ViewGroup tem como responsabilidade organizar os componentes de uma tela.

São conhecidos como Gerenciadores de Layouts. Funcionam como container para outros tipos de Views, podendo ser Widgets ou até mesmo outros ViewGroup's.

Principais layouts

- **AbsoluteLayout (android.widget.AbsoluteLayout):**

Os componentes são posicionados na tela em função das coordenadas X e Y fornecidas.

- **FrameLayout (android.widget.FrameLayout):**

O componente ocupa a tela inteira. Funciona como uma pilha sendo que uma view fica por cima da outra.

- **LinearLayout (android.widget.LinearLayout):**

Os componentes são organizados na vertical ou horizontal.

- **TableLayout (android.widget.TableLayout):**

Assim como em uma tabela, os componentes são organizados em colunas e linhas.

- **RelativeLayout (android.widget.RelativeLayout):**

A organização do componente é implementada relativa a outro componente.

AbsoluteLayout(deprecated)

Nesse tipo de gerenciador de layout, todos os componentes são posicionados em função de abcissa e ordenada, tendo como base o canto superior esquerdo.

É bastante oportuno lembrar que esse é menos flexível e mais difícil de manter que os demais gerenciadores. É importante também saber que esse gerenciador está obsoleto (Deprecated).



Exemplo:

FrameLayout

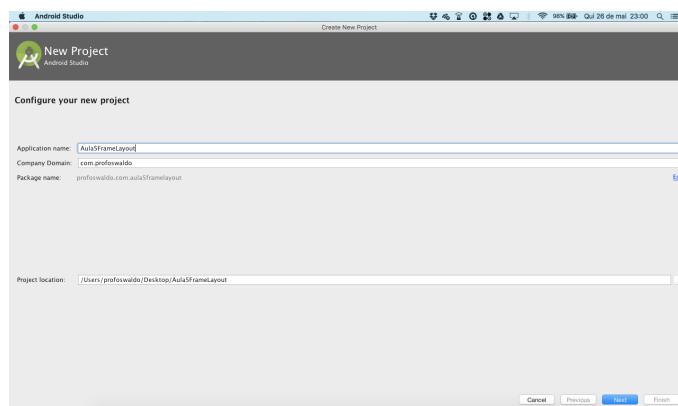
Serve para reservar um espaço na tela que será utilizado por um ou mais componentes.

Quando mais de um componente for inserido dentro de um único FrameLayout, haverá uma sobreposição dos mesmos. A menos, é claro, que você divida a tela e coloque cada um na sua posição.

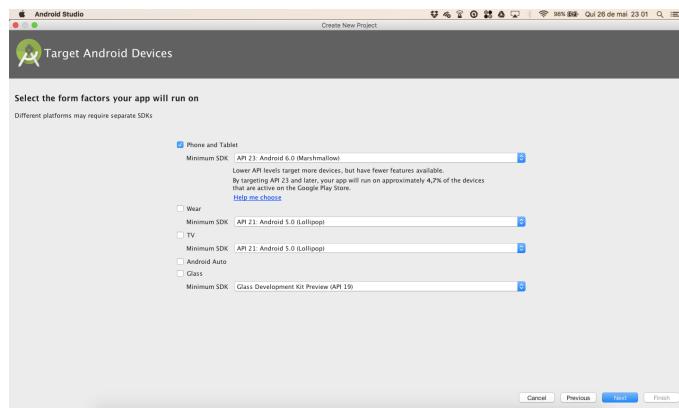


Exemplo:

Como no exemplo anterior, criaremos o nosso projeto com o nome **Aula5FrameLayout**.

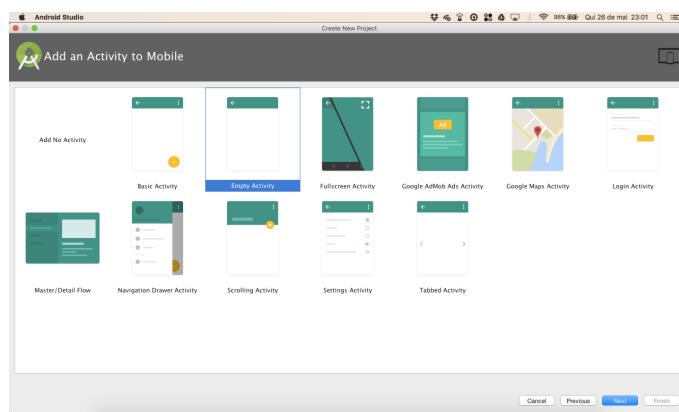


Assim como em todos os nossos exemplos nesta aula, usaremos a API 23, conforme podemos verificar na tela:



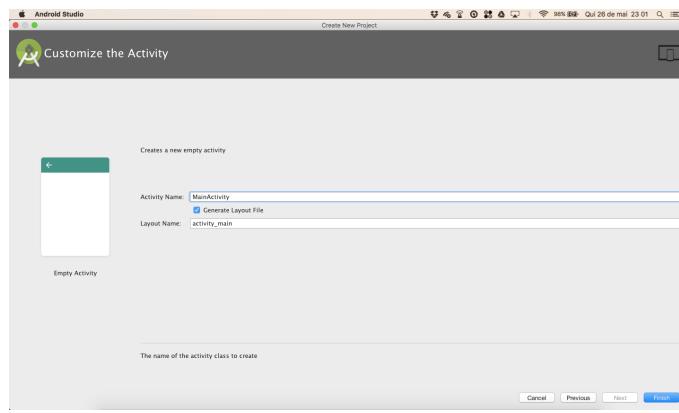
Para fins pedagógicos, selecionamos uma Empty Activity.

Precisamos dominar esse conceito melhor, não é mesmo? Por isso, depois trabalharemos com os demais tipos.



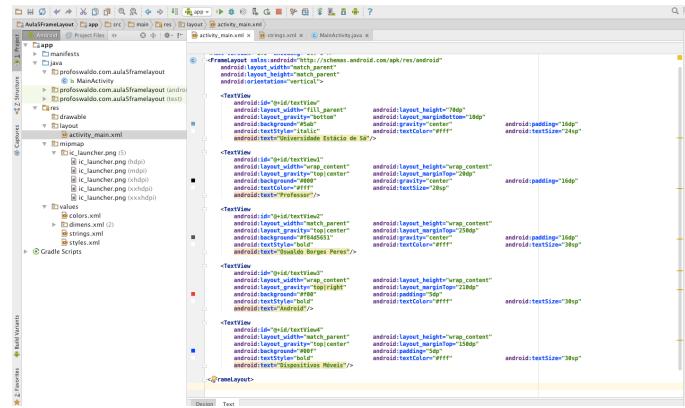
Para facilitar, manteremos o nome sugerido pelo Android Studio para atividade e layout.

Veja a tela abaixo:



Este é o coração de nosso layout. No caso, FrameLayout.

Altere o conteúdo do arquivo, conforme tela abaixo:



Alguns parâmetros precisam ser discutidos para entendermos melhor esse tipo de layout. Criamos, então, os componentes textView, textView1, textView2, textView3, textView4.

Já verificamos, no exemplo referente à AbsoluteLayout, os parâmetros android: layout_height e android: layout_width.

Contudo, existem novos. Veja:

- **android:layout_marginTop:**

Propriedade que define o tamanho da margem superior;

Embora não tenhamos implementado em nosso exemplo, existem suas correlatas, como:

- **android:layout_marginBottom:**

Margem inferior;

- **android:layout_marginLeft:**

Margem esquerda;

- **android:layout_marginRight:**

Margem direita;

- **android:layout_margin:**

As quatro margens.

Exemplo:

`android:layout_margin="10dp"`

- **android:layout_gravity:**

Define a posição dos componentes em relação ao gerenciador de layout que os contém.

- **android:gravity:**

Define o posicionamento do conteúdo do componente;

- **android:padding:**

Define a distância entre o conteúdo interno do componente e sua borda. Nesse caso, todos os 4 espaçamentos serão iguais.

Poderíamos também escolher o espaçamento que desejássemos usando:

- **android:paddingTop:**

Espaçamento superior;

- **android:paddingBottom:**

Espaçamento inferior;

- **android:paddingRight:**

Espaçamento lado direito;

- **android:paddingLeft:**

Espaçamento lado esquerdo;

- **android:background:**

Define uma cor (em hexadecimal) ou imagem referente ao background de nosso layout;

- **android:textColor:**

Define a cor de nossa fonte;

- **android:textSize:**

Define o tamanho de nossa fonte;

- **android:textStyle:**

Define o estilo do texto (bold, italic, normal).



LinearLayout

Esse tipo de gerenciador de layout tem por característica a capacidade de alinhar verticalmente ou horizontalmente os componentes. Isso depende, obviamente, da orientação escolhida.

Veja os exemplos abaixo:

LinearLayout Horizontal:

Como já criamos vários projetos passo a passo anteriormente, vamos direto ao ponto:

Na tela abaixo, estão as alterações que efetuamos no arquivo **activity_main.xml** para demonstrar o **LinearLayout Horizontal**.

```

<?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="match_parent"
    android:orientation="horizontal"
    android:padding="10dp">
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="#FF0000"/>
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="#000000"/>
    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="#008000"/>
    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="#800000"/>


```

Ao executar, você poderá constatar, conforme exibido na tela abaixo, que os widgets estão alinhados na horizontal.

Até o momento, não usamos nenhum atributo desconhecido, exceto, é claro, o LinearLayout.

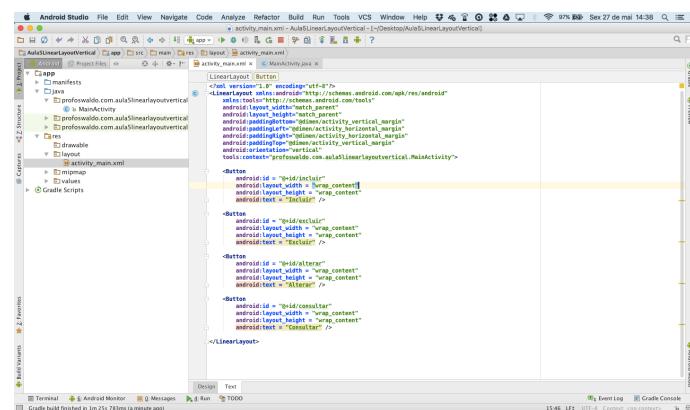


LinearLayout Vertical:

Agora vamos transformar o exemplo anterior em vertical.

Para tanto, só precisamos alterar o atributo **android:orientation="vertical"**, conforme demonstrado na tela.

Repare que esse atributo não foi modificado no exemplo anterior, pois o default é horizontal.



Pronto!
É necessário
somente isso para
demonstrar os
nossos
componentes
sendo alinhados
na vertical.

Fácil, não
é mesmo?



RelativeLayout

O RelativeLayout organiza os componentes do layout de uma tela em relação uns aos outros.

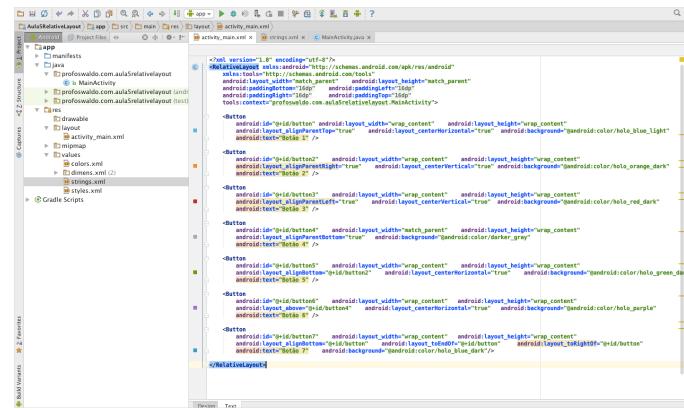
Isso é uma grande vantagem deste gerenciador de layout, no que tange à flexibilidade de posicionamento dos

componentes.

Na teoria, é muito fácil de entender, mas como seria em um código fonte?

Vamos ver o exemplo abaixo:

Altere o arquivo **activity_main.xml**, conforme a tela:



Principais atributos:

- **android:layout_below:**

Posiciona abaixo do componente indicado;

- **android:layout_above:**

Posiciona acima do componente indicado;

- **android:layout_toRightOf:**

Posiciona à direita do componente indicado;

- **android:layout_toLeftOf:**

Posiciona à esquerda do componente indicado;

- **android:layout_alignParentTop:**

Alinha no topo do layout-pai;

- **android:layout_alignParentBottom:**

Alinha abaixo do layout-pai;

- **android:layout_alignParentRight:**

Alinha à direita do layout-pai;

- **android:layout_alignParentLeft:**

Alinha à esquerda do layout-pai;

- **android:layout_alignTop:**

Alinha no topo do componente indicado;

- **android:layout_alignBottom:**

Alinha abaixo do componente indicado;

- **android:layout_alignRight:**

Alinha à direita do componente indicado;

- **android:layout_alignLeft:**

Alinha à esquerda do componente indicado;

- **android:layout_marginTop:**

Define a margem superior do componente;

- **android:layout_marginBottom:**

Define a margem inferior do componente;

- **android:layout_marginRight:**

Define a margem direita do componente;

- **android:layout_marginLeft:**

Define a margem esquerda do componente;

- **android:layout_centerHorizontal:**

Centraliza o componente horizontalmente;

- **android:layout_centerVertical:**

Centraliza o componente verticalmente.



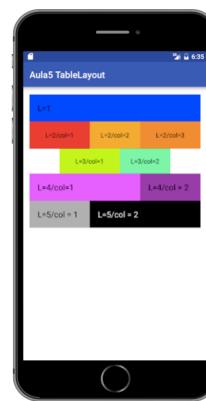
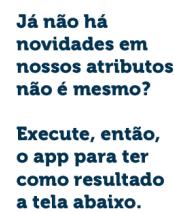
TableLayout

Esse gerenciador de layout organiza seus componentes em colunas e linhas, ou seja, como uma tabela.

Cada `<tableRow></tableRow>` corresponde a uma linha de nossa tabela. As colunas são criadas à medida que incluímos os componentes em cada linha.

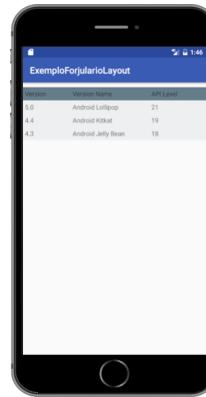
Veja o exemplo:

Altere o arquivo `activity_main.xml`, conforme tela:



ATIVIDADE

Observe a tela abaixo:



Usando XML, implemente o layout demonstrado na tela acima.

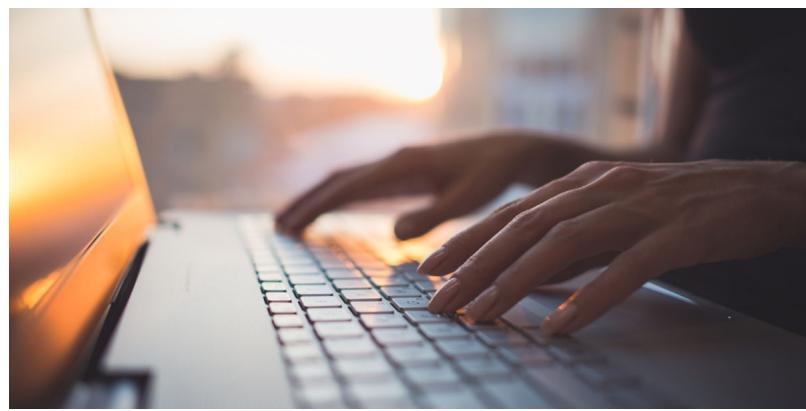
Resposta Correta

Glossário

Programação para dispositivos móveis

Aula 4 - Interface Gráfica Avançada

INTRODUÇÃO



Esta aula visa apresentar o desenvolvimento de telas gráficas implementadas pelas classes ScrollView e HorizontalScrollView, pelo aninhamento de gerenciadores de layouts e compostas por listas de itens.

OBJETIVOS



Demonstrar o emprego das classes ScrollView e HorizontalScrollView;

Ilustrar o aninhamento de gerenciadores de layouts Android;

Ilustrar o desenvolvimento de telas gráficas compostas por lista de itens.

SCROLLVIEW



Fonte da Imagem:

É muito comum que telas de aplicativos tenham muitos componentes, dificultando a exibição de todos em uma única tela. Por isso, o componente ScrollView permite que barras de rolagens sejam apresentadas automaticamente, caso sejam necessárias para exibição de todos os componentes na mesma tela.

Embora o ScrollView possua vários métodos, implementá-los é bastante simples. Apenas precisamos nos lembrar que esta classe somente pode possuir um componente-filho e que as barras de rolagens serão inseridas automaticamente quando a View ultrapassar o tamanho da tela física.

Deve-se inserir dentro do ScrollView um ViewGroup que será responsável por conter todos os demais componentes.

O emprego do ScrollView é demonstrado abaixo:

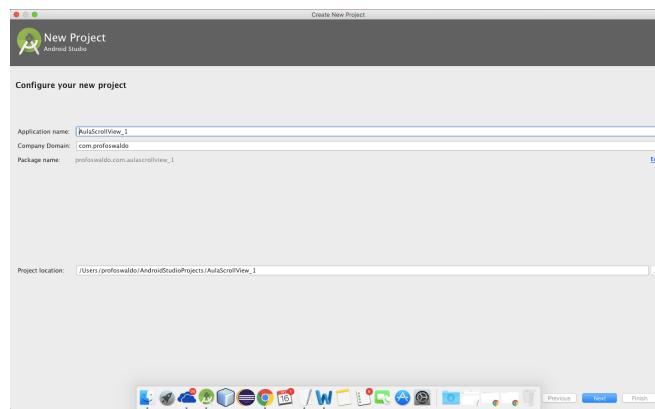
```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="wrap_content" >
    <LinearLayout
        android:layout_width="fill_parent" android:layout_height="fill_parent"
        android:orientation="vertical" >
        <RadioButton
            android:id="@+id/radioButton2" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="RadioButton 1" />
        <RadioButton
            android:id="@+id/radioButton1" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="RadioButton 2" />
        <Button
            android:id="@+id/button4" android:layout_width="match_parent"
            android:layout_height="wrap_content" android:text="Botão 8" />
        <CheckBox
            android:id="@+id/checkBox1" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="CheckBox" />
        <!--Foram omitidos neste ponto vários outros componentes -->
    </LinearLayout>
</ScrollView>
```

Exemplo:

ScrollView Vertical:

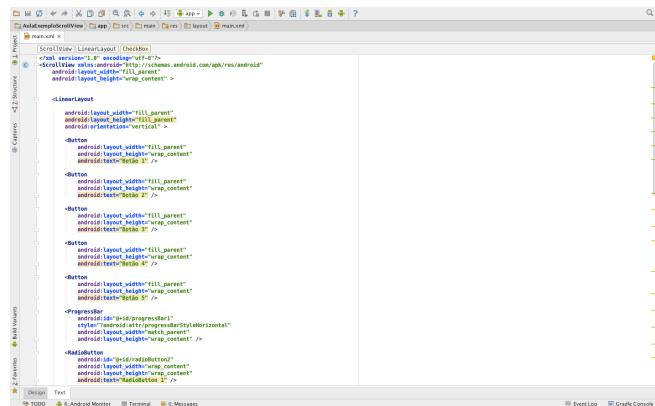
Agora, vamos desenvolver um exemplo na prática:

Crie um projeto Android. O nosso aqui se chamará **AulaScrollView_1**.

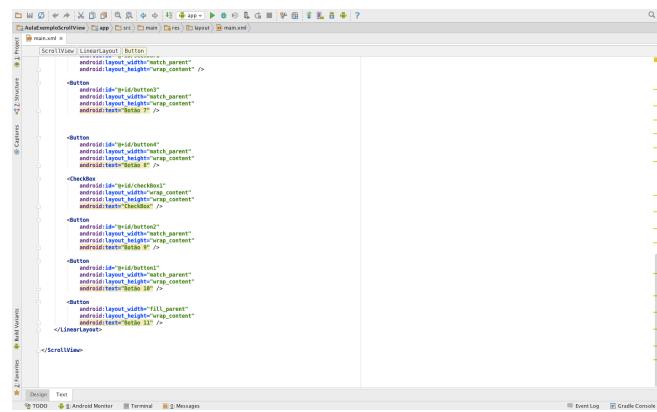


Os demais passos para construção de um projeto estão sendo omitidos porque seguem o mesmo padrão dos exemplos anteriormente desenvolvidos.

Agora crie o **arquivo de layout (main.xml)**, conforme a tela:

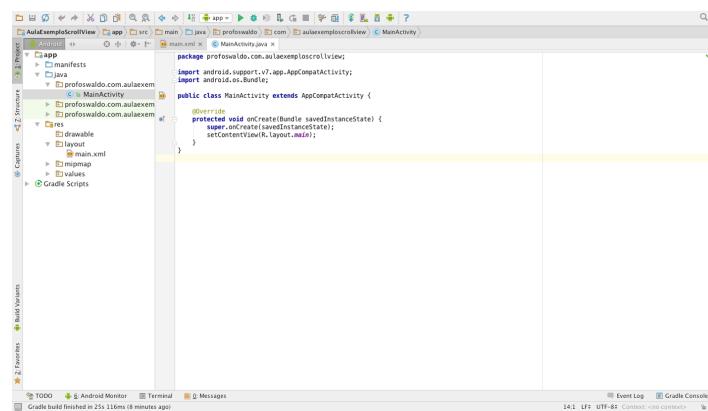


O código referente ao **main.xml** está sendo complementado na tela por ser muito grande.

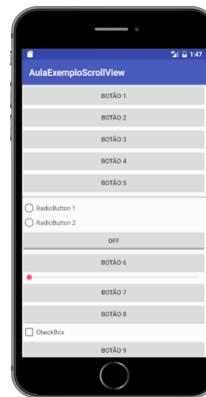


Nossa atividade principal (**MainActivity.java**) é bastante simples, pois nosso foco está no emprego do **ScrollView**.

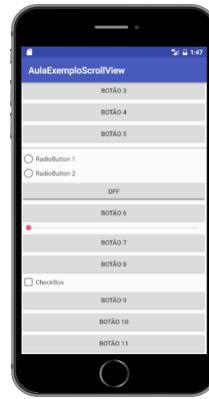
Veja e tela:



Agora é só executar e será exibido no AVD a tela abaixo:



Para ver o efeito do ScrollView, apenas arraste a tela para cima.



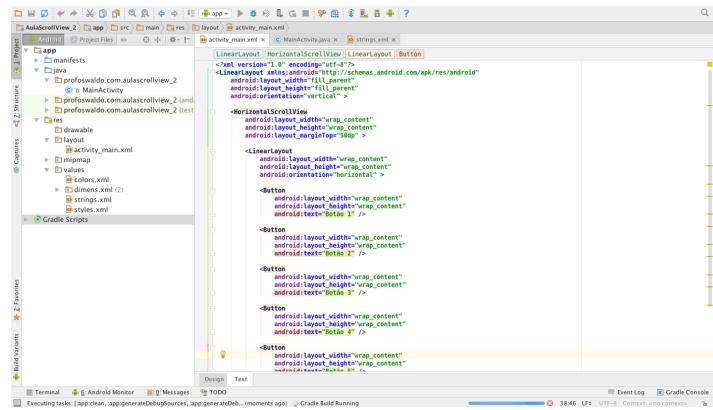
Para rolamento horizontal, devemos usar a classe `HorizontalScrollView`, pois a `ScrollView` suporta somente rolamento vertical

HorizontalScrollView:

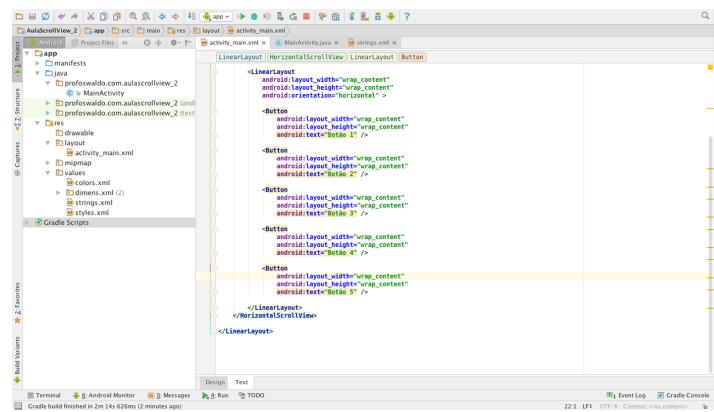
Vamos criar juntos um exemplo demonstrando o uso da `HorizontalScrollView`?

O nome do nosso projeto é **AulaScrollView_2**, criado exatamente como o exemplo anterior.

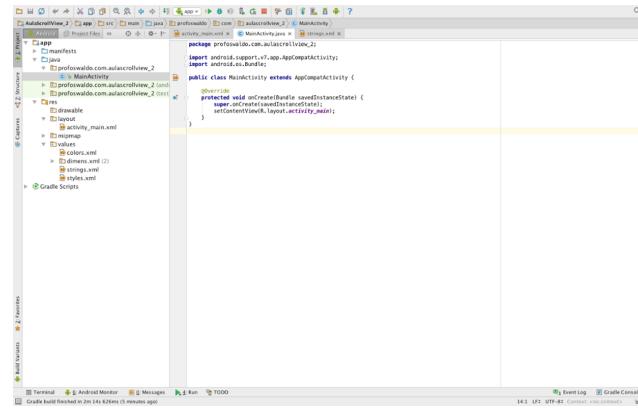
Agora implemente o arquivo de layout (**activity_main.xml**) demonstrado na tela:



Como esse arquivo também é maior do que a tela pode exibir, o complemento está na tela:



Nosso último passo será criar a atividade principal. Veja:



Basta executar para ver a tela a seguir.

Assim como o exemplo anterior, basta arrastar os botões para o lado para ver o **HorizontalScrollView** em ação.



GERENCIADORES DE LAYOUTS ANINHADOS

O Android permite a implementação de layouts aninhados, pois os Gerenciadores de Layouts são classes Views e, como estudado anteriormente, podem conter tanto Views simples (Widgets) como layouts (ViewGroups).

Assim, é possível implementar telas muito mais elaboradas apenas combinando as características de cada tipo de gerenciador de layouts.



Exemplo:

Poderíamos implementar uma tela de formulário onde cada campo apareça na vertical, ou seja, um abaixo do outro. Contudo, poderíamos exibir na última linha os botões “Enviar” e “Cancelar” com alinhamento horizontal, ou seja, lado a lado.

Implementar layouts aninhados não é difícil.

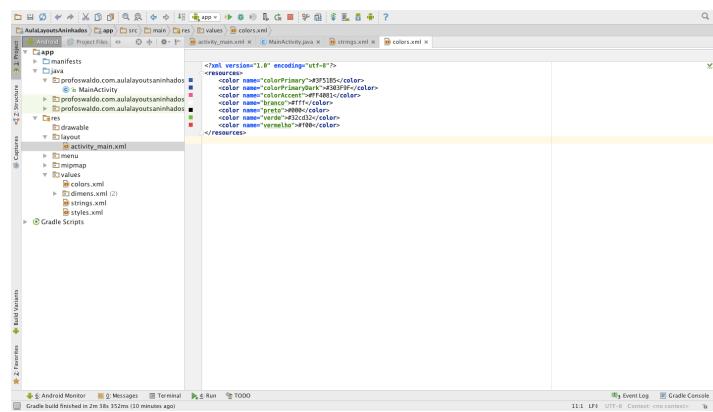
Vamos desenvolver um pequeno exemplo juntos.

Carregue o Android Studio e crie um projeto chamado **AulaLayoutsAninhados** nos mesmos moldes dos dois últimos exemplos.

Repare que dentro do diretório values existe um arquivo chamado **colors.xml**. Nele podemos definir as cores que serão usadas por toda nossa aplicação.

Já existem algumas cores definidas, não é mesmo?

Então, é só acrescentar as cores, conforme demonstrado abaixo:

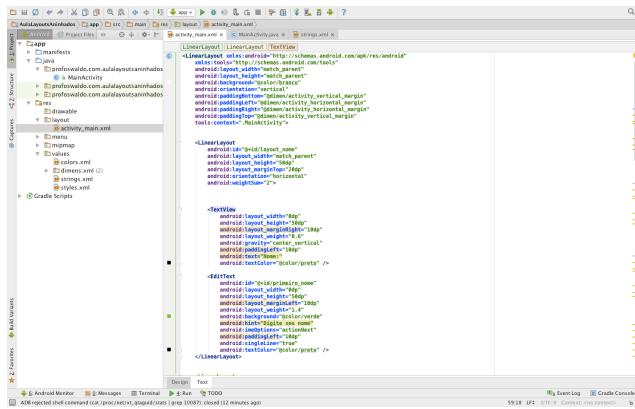


Atenção

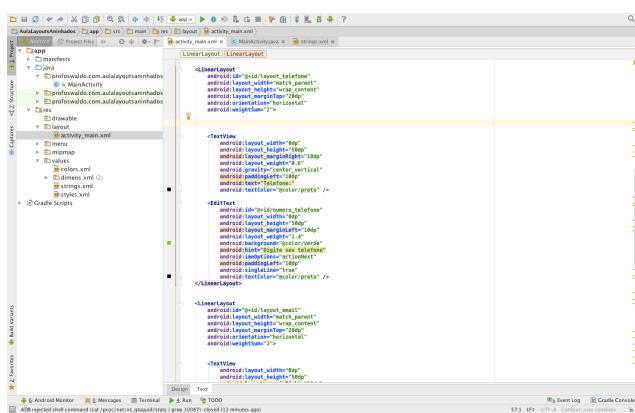
, Estudaremos sobre esse arquivo mais tarde em nossa disciplina.

Chegou a vez de desenvolvemos o layout de nossa tela.

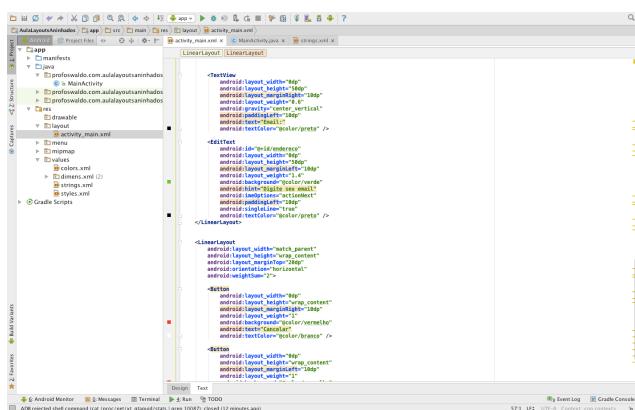
Edite o arquivo **activity_main.xml** a partir da tela:



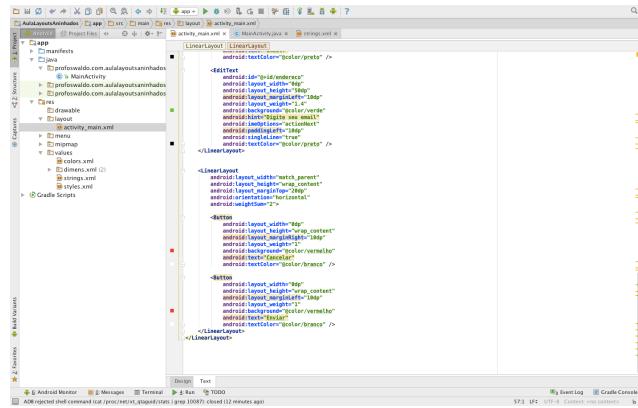
Segue o complemento do arquivo **activity_main.xml**:



Ainda temos mais alguns componentes faltando na tela anterior:

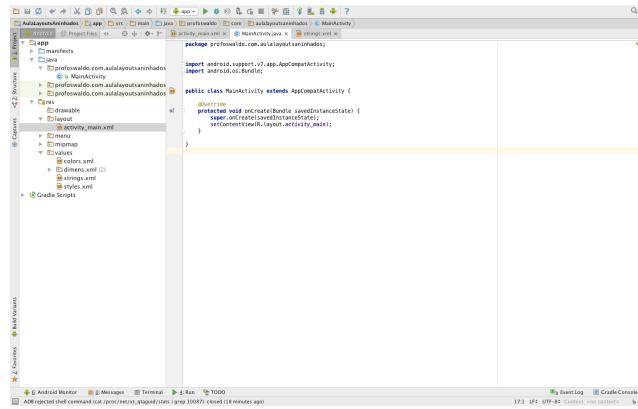


Complemente ainda com o código abaixo para finalizar esse arquivo de layout.



Como você pode observar em nosso exemplo, para aninhar os layout basta declará-los dentro de outro.

Nosso último passo é desenvolver uma atividade simples para executar este layout. Veja:



Pronto.

Agora execute seu código para ver a tela.

LISTAS

Sem sombra de dúvidas, um dos componentes mais implementados no Android é o ListView.

Normalmente, é utilizado quando precisamos exibir uma grande quantidade de dados na forma de lista, que pode possuir rolagem (scroll).

Podemos otimizá-lo com textos e imagens, tratar eventos de clique e de seleção.

Criar um ListView é muito fácil. Vamos ver na prática?

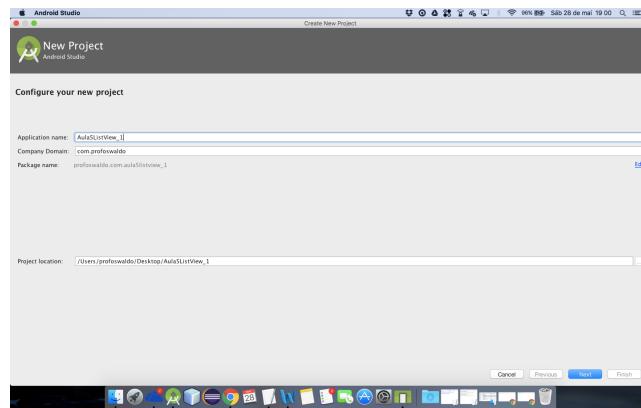


Exemplo:

Primeiramente, devemos criar um projeto novo.

No caso do menu exemplo, o nome é **Aula5ListView_1**.

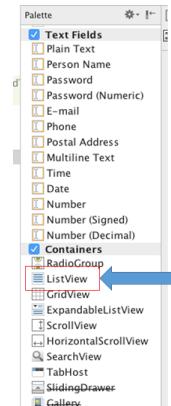
Assim como os exemplos anteriores, manteremos o tipo do layout, o nome da Activity e do layout.



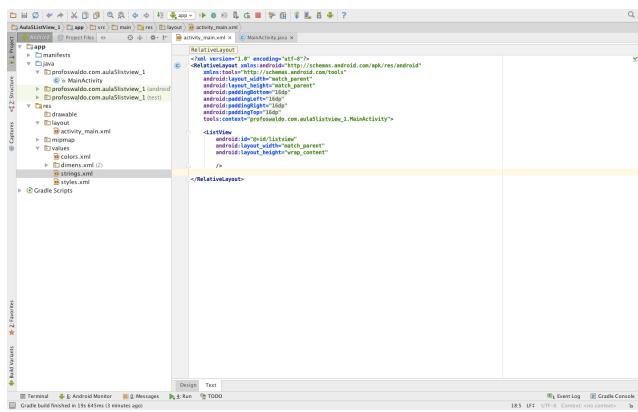
Logo após este primeiro passo, precisamos criar a nossa **ListView**.

Poderíamos criar graficamente, arrastando a **ListView** contida na **Pallete** para dentro do layout, assim como fizemos no primeiro exemplo de nossa aula.

Porém, para realizarmos a verificação de forma mais detalhada, optamos por trabalhar no arquivo xml do layout.



Para criarmos o ListView "na unha", precisamos alterar o arquivo **activity_main.xml** conforme definido na tela:



Atenção

, Até o momento, a única novidade é o ListView.



Fonte da Imagem:

Quanto aos seus atributos, foram todos discutidos nos exemplos anteriores.

Feito isso, precisamos desenvolver a Activity Principal, que possuirá os dados que comporão esta lista.

Para inserir os dados nesta ListView, usaremos a classe adapter ArrayAdapter.

Não é a única forma de se implementar uma ListView, mas essa facilita muito o trabalho.

Assim como os demais tipos de Adapter, ela representa a ligação entre a View e alguma fonte de dados.

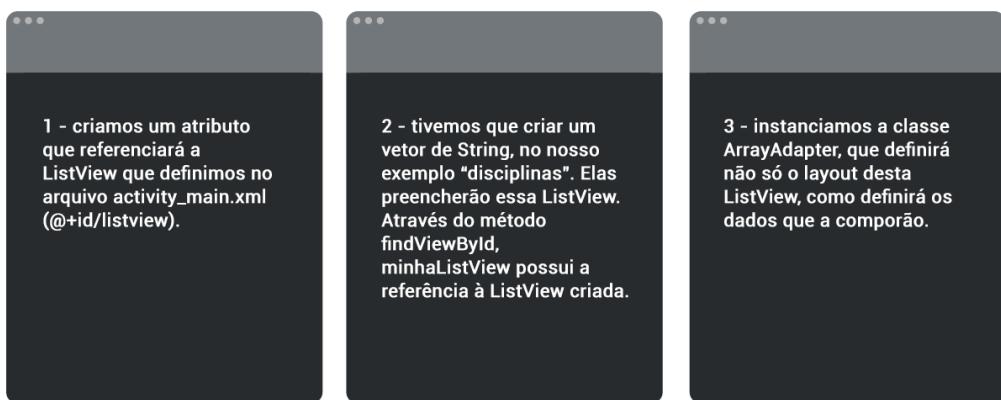
Altere, então, sua **MainActivity**, conforme imagem:

```

package profowaldo.com.aulaListView_1;
import android.support.v1.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

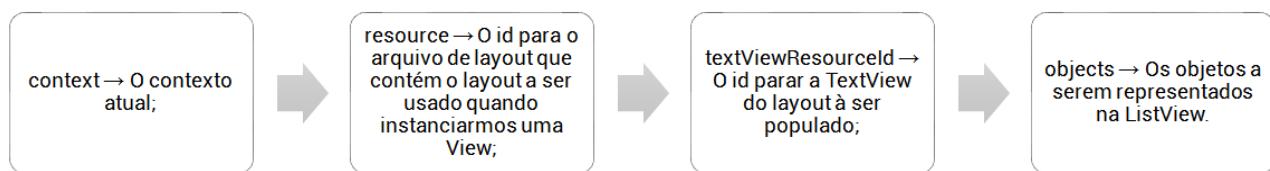
public class MainActivity extends AppCompatActivity {
    private ListView minhaListView;
    private ArrayAdapter<String> arrayAdapter;
    String[] disciplinas = {"Matemática", "Português", "Física", "Biologia", "Inglês", "Geografia", "História", "Informatica", "Música"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        minhaListView = (ListView) findViewById(R.id.listView);
        arrayAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, disciplinas);
        minhaListView.setAdapter(arrayAdapter);
    }
}

```



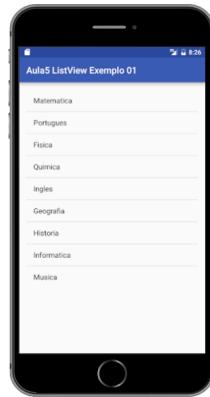
Observe que o construtor da ArrayAdapter possui quatro parâmetros, sendo:

public ArrayAdapter (Context context, int resource, int textViewResourceId, T[] objects)



Por último, apenas configure a classe adapter para **minhalistView** através do método **setAdapter**.

Agora só precisamos executar para ver a tela:



Outro ponto importante a ser observado é que, se a lista for muito grande, automaticamente será exibida a barra de rolagem.

Podemos também implementar o ListView, estendendo à classe ListActivity.

Veja como implementar:

Exemplo:

Abaixo se encontram as opções do nome de nossa aplicação e o domínio de nossa companhia.



Agora altere sua **MainActivity**, conforme tela:

```

package profosolbo.lista.ListView_2;
import android.app.ListActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
public class MainActivity extends ListActivity {
    private TextView textView;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textView = (TextView) findViewById(R.id.textView);
        String[] diasSemana = new String[]{"Domingo", "Segunda-feira", "Terça-feira", "Quarta-feira", "Quinta-feira", "Sexta-feira", "Sábado"};
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, diasSemana);
        setListAdapter(adapter);
    }
}

```

Observe que não efetuamos a vinculação da nossa ListView através do método `findViewById`.

Na verdade, isso geraria um erro, pois, diferente da Activity, a ListActivity já implementa, internamente, a ListView.

Ao executar, você contemplará uma tela semelhante a esta:



Apesar da lista estar visível em nosso aplicativo, ela ainda não reconhece as nossas seleções.

Como poderíamos implementar esse tratamento de eventos?

Altere seu código conforme tela:

```

package profosolbo.lista.ListView_2;
import android.app.ListActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
public class MainActivity extends ListActivity {
    private TextView textView;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textView = (TextView) findViewById(R.id.textView);
        String[] diasSemana = new String[]{"Domingo", "Segunda-feira", "Terça-feira", "Quarta-feira", "Quinta-feira", "Sexta-feira", "Sábado"};
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_2, diasSemana);
        setListAdapter(adapter);
    }
    protected void onListItemClick(ListView listView, View view, int posicaoSelecionada, long id) {
        super.onListItemClick(listView, view, posicaoSelecionada, id);
        int i = posicaoSelecionada;
        String dia = (String) listView.getAdapter().getItem(posicaoSelecionada);
        textView.setText("Dia na semana (" + i + ") " + dia);
    }
}

```

A ListActivity, dentre muitos métodos, possui o chamado `onListItemClick`, que tem por objetivo identificar qual item da lista foi selecionado. Com isso, podemos programar o que desejarmos em resposta a esse evento.

Sua sintaxe é:

```
protected void onListItemClick (ListView l, View v, int position, long id) {  
    super.onListItemClick(l, v, position, id);  
}
```

Seus parâmetros são:

l - o ListView onde o clique aconteceu;

v - o View que foi clicado dentro do ListView;

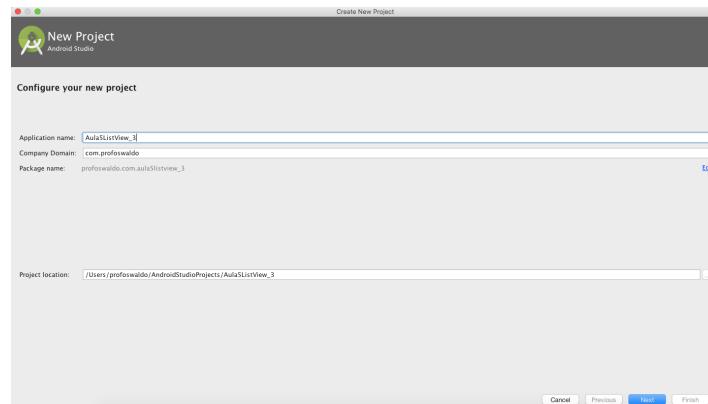
position - a posição da View na lista;

id - o ID da linha do item que foi clicado.

Agora ficou fácil entender o nosso código, não é mesmo? Execute seu aplicativo e verifique a seleção das opções em ação.

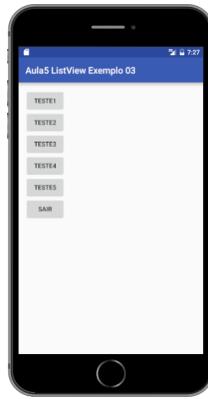
Para consolidar o que aprendemos nesta aula e nas aulas passadas, que tal desenvolvemos mais um código fonte juntos?

Crie um novo projeto conforme tela:



Nosso aplicativo será composto por 6 atividades.

A principal(MainActivity) será responsável por chamar as demais 5 atividades (Teste1Atividade, Teste2Atividade, Teste3Atividade, Teste4Atividade, Teste5Atividade) e seu layout será similar ao exibido na tela:



Embora você possa compor a tela através do recurso "arrasta-soltar" que o Android Studio disponibiliza, preferimos, para fins pedagógicos, desenvolver diretamente o arquivo xm.

Altere, então, o **activity_main.xml** conforme tela:

```
activity_main.xml
<RelativeLayout>
    <!-- Main content area -->
    <FrameLayout android:id="@+id/content_frame" android:layout_width="match_parent" android:layout_height="match_parent" android:paddingBottom="@dimen/activity_vertical_margin" android:paddingLeft="@dimen/activity_horizontal_margin" android:paddingRight="@dimen/activity_horizontal_margin" android:paddingTop="@dimen/activity_vertical_margin" android:background="#e0e0e0" />
    <include android:id="@+id/toolbar" android:layout_width="match_parent" android:layout_height="wrap_content" android:layout_alignParentTop="true" android:layout_alignParentLeft="true" android:layout_alignParentStart="true" android:background="#3399FF" />
    <include android:id="@+id/progress_bar" android:layout_width="wrap_content" android:layout_height="wrap_content" android:layout_alignParentTop="true" android:layout_centerHorizontal="true" android:layout_centerVertical="true" android:background="#3399FF" />
    <include android:id="@+id/footer" android:layout_width="wrap_content" android:layout_height="wrap_content" android:layout_alignParentBottom="true" android:layout_centerHorizontal="true" android:background="#3399FF" />
</RelativeLayout>
```

Já estudamos todas as tags e atributos implementados nesse arquivo. Falta apenas destacar, para o nosso exemplo, o atributo android:onClick.

Registraremos, para cada botão, um método que será executado quando o selecionarmos.

Dando mais um passo, criamos a **MainActivity**.

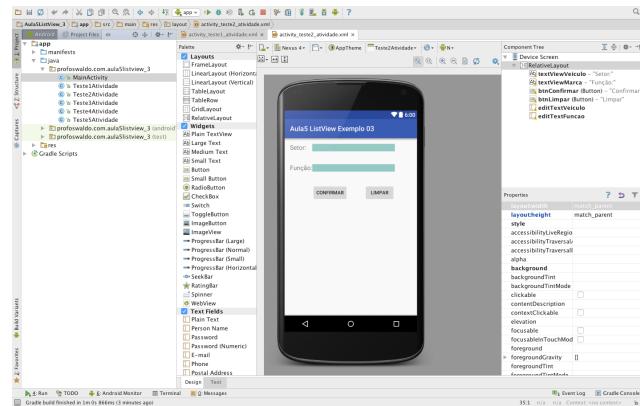
Altere, conforme a tela:

Apenas implementamos os métodos correspondentes a cada um dos botões definidos no layout da tela principal.

O método `executarFim` fechará a aplicação.

Para os demais, apenas codificamos a Intent, visando chamar a atividade correspondente a este botão.

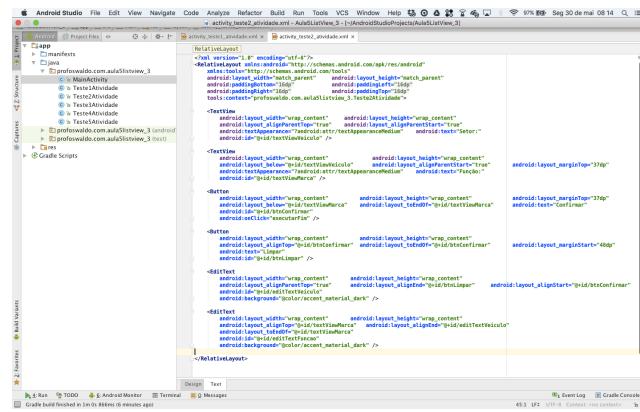
Agora, precisamos criar as outras 5 atividades, conforme você pode observar na tela:



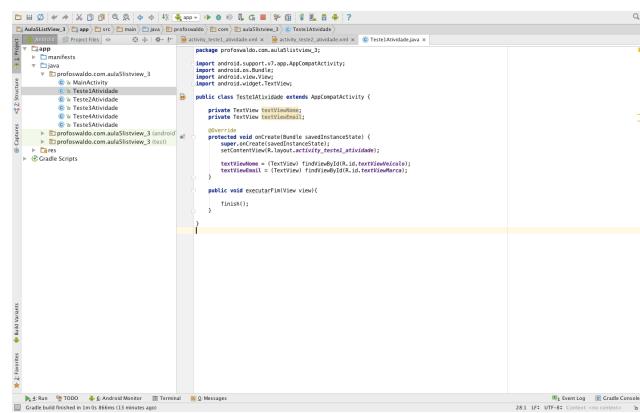
Para tornar mais simples o nosso exemplo, todas as 5 atividades têm a mesma estrutura.

Apenas alteramos os nomes dos componentes e o seu texto.

Seguindo o definido na tela:



Não podemos nos esquecer de desenvolver cada atividade adicional conforme tela:



Não se esqueça que nas demais apenas alteramos o nome da atividade e de seus componentes, conforme definido nos respectivos layouts.

Pronto. Agora é só executar. Fácil mesmo!

ATIVIDADE

Observe a tela abaixo:



Implemente a ListView demonstrada na tela.

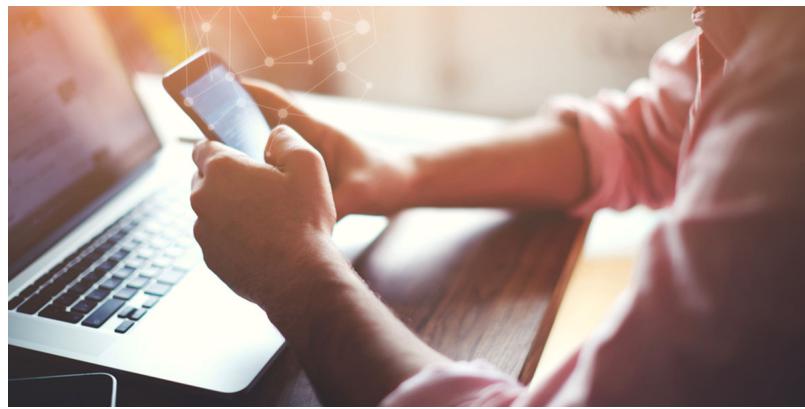
Resposta Correta

Glossário

Programação para dispositivos móveis

Aula 5: Menus e Fragmentos

INTRODUÇÃO



Esta aula apresentará as classes referentes à implementação de menus e fragmentos em Android.

OBJETIVOS



Ilustrar o desenvolvimento de telas gráficas compostas por menus.

Explicar o emprego dos principais conceitos referentes a fragmentos.

MENUS



Fonte da Imagem:

Em aplicativos Android, um recurso extremamente utilizado é o de menu. Basta você baixar alguns aplicativos na Google Play Store que os encontrará facilmente.

Embora sejam de fácil implementação, devemos reforçar os cuidados em relação à usabilidade e aparência dos mesmos. É muito comum encontrarmos aplicativos sem nenhum menu, menus sem ícones ou até mesmo com títulos inapropriados.

Por isso, é importante lembrarmos de que um bom apelo visual e/ou textual facilitará muito a interação do usuário com o aplicativo.

Podemos trabalhar com três tipos de menus em Android:

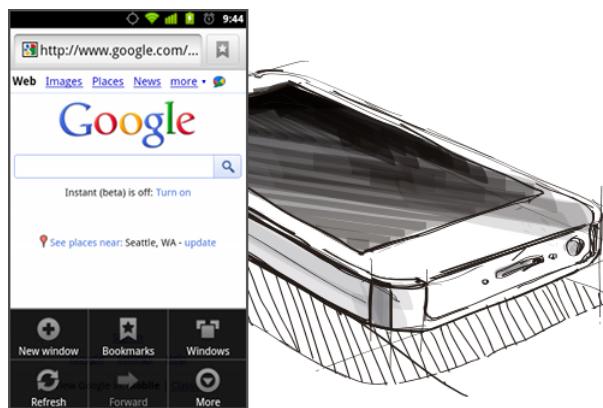
Menu de opção e barra de opção

- É o menu default das aplicações Android;
- Normalmente encontramos nesse menu as principais opções.

Existem dois modelos:

I. Ícone (icon menu):

- Disponível parte inferior da tela;
- Suporta até seis itens de menu;
- Suporta ícones;
- Não suporta caixa de seleção;
- Não suporta botões de rádio;
- Android 2.3.x – API Level 10 ou Inferior.



Fonte: https://developer.android.com/images/options_menu.png?hl=pt-br

II. Expandido (expanded menu):

- Suporta mais de seis itens de menu;
- Apresentado automaticamente na opção Mais (More) quando possuir mais de seis itens de menu;
- Android 3.0 – API Level 11 ou superior.



Fonte: https://dab1nmslvntp.cloudfront.net/wp-content/uploads/2014/04/1396465832ab_with_action_buttons-181x300.jpg

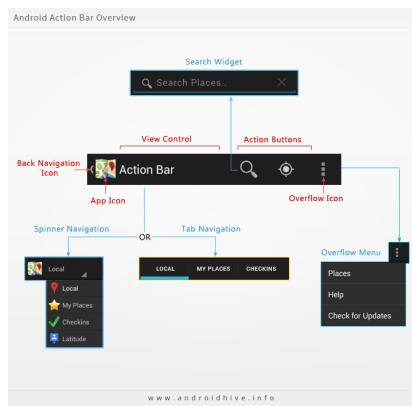
Menu de contexto

-
- É exibido quando o usuário clica e segura, por mais de 2 segundos, um componente visual;
 - Não suporta atalhos, ícones ou até mesmo submenus;
 - Pode ser compartilhado entre diferentes Views.

Menu Pop-up

-
- Abre quando tocamos no item de menu Options (Opções) ou em menu contextual;
 - Não suporta ícones;
 - Não suporta submenus aninhados;

ACTION BAR



App Icon

- Exibe o ícone do projeto ou logo customizado;
- *Back Navigation Icon* - permite a navegação para cima na hierarquia de telas.

View Control

- Exibe o título do aplicativo ou a tela em que o usuário se encontra;
- Exibe o controle de Navegação (*Drop-down ou Tabs*).

Action Buttons

- Exibem as ações mais comuns em seu aplicativo;
- Os ícones que não couberem nesse espaço serão inseridos automaticamente no *Action Overflow*.

Action OverFlow

- Exibe as ações não utilizadas frequentemente de seu aplicativo.

Podemos desenvolver menus Android através de:



Nossa sugestão é que você defina em um arquivo XML, pois isso facilita a visualização da estrutura do menu, bem como nos ajuda a implementar para diferentes versões do Android, tamanhos de tela etc.

Gerenciar as interações efetuadas pelos usuários ficou ainda mais fácil. Para isso, implementamos os métodos `onOptionsItemSelected()` e `onContextItemSelected()`.

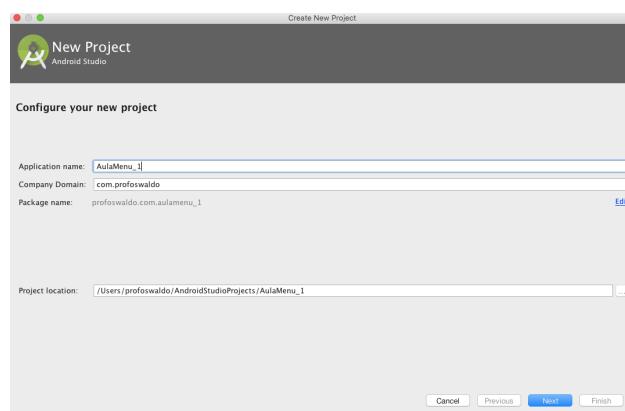
Vamos desenvolver exemplos para visualizarmos a implementação de menus na prática?

Exemplo:

Crie um novo projeto Android.

Em nosso exemplo, o chamaremos de AulaMenu_1.

Na sua tela, deverá ser similar ao exibido:

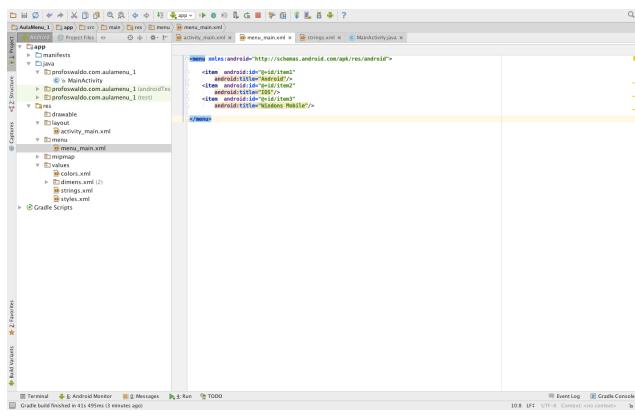


Provavelmente, não existe a pasta menu em sua árvore de recursos. Por isso, clique com o botão direito na pasta res e crie o diretório menu.

Agora crie o arquivo xml, chamado menu_main.

Nesse arquivo, serão definidos os itens que comporão nosso menu.

Veja a tela:



Se você olhar mais atentamente o código, perceberá que cada **tag item** é referente a uma opção do menu.

No nosso exemplo, os parâmetros são:

- android:id - Esse id é exclusivo para cada item. É através dele que podemos identificar o item de menu;
- android:title - É o texto título de nosso item de menu;
- android:icon - Embora não tenha sido usado nesse exemplo, poderia definir um ícone para o nosso menu. O valor aqui é a referência a um drawable;
- android:showAsAction - Define a forma de exibição do componente.

As constantes que devemos empregar são:

always

- O componente sempre fica visível;
- Recomendado para ações mais comuns do aplicativo.

ifRoom

- O componente é exibido na action bar, se existir espaço;
- Adequado para manter compatibilidade com diversos tipos de dispositivos e também com telas na vertical ou horizontal.

withText

- O componente exibe o seu título ao lado do ícone, caso tenha espaço disponível.

never

- Não exibe o componente na action bar.

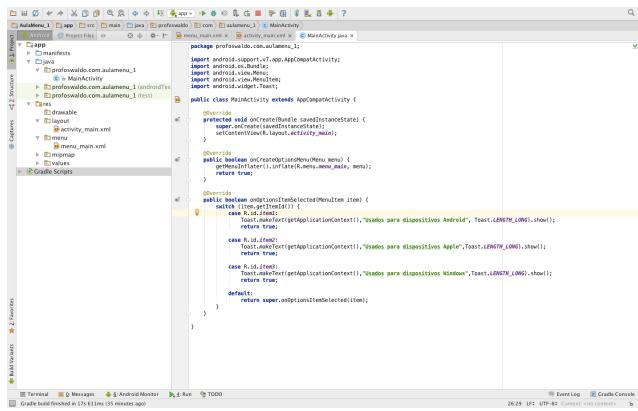
collapseActionView

- Quando a view é grande, deve ser contraída para exibir apenas um botão.

É oportuno saber que também podemos combinar as constantes com separadores, como, por exemplo, `ifRoom|withText`.

Faça um teste em nossos exemplos. Você vai se surpreender com essas combinações.

Nesta próxima tela, temos algumas novidades.



Quando o botão físico do menu for acionado, o método `onCreateOptionsMenu (Menu menu)` de nossa atividade é invocado e, em nosso exemplo, faz uso do `MenuItemInflator` para criar o menu definido no arquivo `menu_main.xml`.

Isso pode ser verificado no método abaixo:

```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}

```

Ao clicar em uma das opções do menu, o `onOptionsItemSelected(MenuItem item)` entra em ação.

Mas não para por aí, a classe `Toast(import android.widget.Toast)` é outra grande novidade. Ela é bastante similar ao nosso velho conhecido JOptionPane. Seu objetivo é exibir, por alguns segundos, uma pequena e breve mensagem de alerta para nosso usuário, sobre a tela vigente de nossa aplicação.

Podemos definir sua posição e até mesmo personalizar seu layout, mas esta nunca receberá o foco.

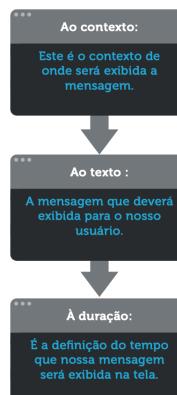
Sua sintaxe é bastante simples de entender. Veja o exemplo abaixo:

```

Toast toast = Toast.makeText(contexto, texto, duracao);
toast.show();

```

Os parâmetros dessa classe correspondem:



Podemos configurar a partir das seguintes constantes:

- `Toast.LENGTH_LONG` - 4 segundos;
- `Toast.LENGTH_SHORT` - 2 segundos.

Após configurarmos nossa mensagem, é necessário executar o método `.show()` para que possa ser exibida.

Pronto. Execute sua aplicação para ver a tela abaixo.



Fonte da Imagem:

Selecione uma das opções de nosso menu.

Observe que a mensagem foi exibida na parte inferior da tela.

Podemos alterar isso. Basta definirmos o **método `.setGravity()`**.

Sua sintaxe é `toast.setGravity(constante, valor_x, valor_y)`, onde:

- Constante:

Constante Gravity (glossário). Existem várias constantes que você pode empregar e até mesmo combinar.

-
- Valor_x:

Deslocamento x da posição;

-
- Valor_y:

Deslocamento y da posição.

Exemplo 1: `toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);`

Vamos subir um pouco mais o nível de nosso menu?

Então, crie um novo projeto Android, veja abaixo.

FRAGMENTO

Bastante similar a uma Activity, um fragment (*fragment*) consiste em uma pequena porção de Activity, que permite um projeto mais modular.

Não seria errado afirmarmos, assim, que um fragment é uma espécie de *subactivity*.

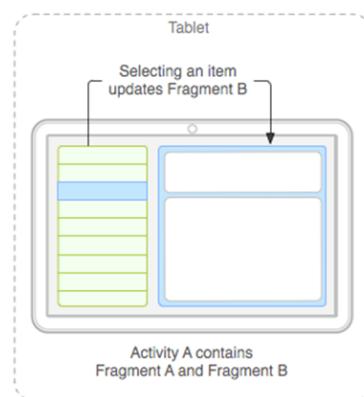
Esse conceito surgiu com o Android 3.0 (*Honeycomb*) devido à necessidade de customizá-lo para as interfaces dos aplicativos, em função da pluralidade de tipos e tamanhos de dispositivos, em especial os *tablets*.

Veremos agora que um *fragment* é muito mais do que dividir a tela em duas ou mais.

Vamos analisar o exemplo da tela abaixo para compreender o conceito:



Podemos perceber que na do *SmartPhone*, ao selecionarmos uma opção da lista, é necessário apresentar outra tela, devido ao tamanho do dispositivo.

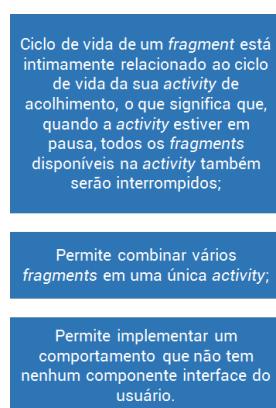
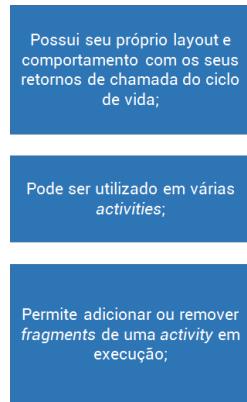


Já na figura do tablet a naveabilidade se dá na mesma tela.

É uma boa prática de desenvolvimento em *Android* sempre encapsular o código de uma *activity* em um *fragment*. Isso permite a reutilização deste código em outro contexto.

CARACTERÍSTICAS

Veja algumas características importantes desse:



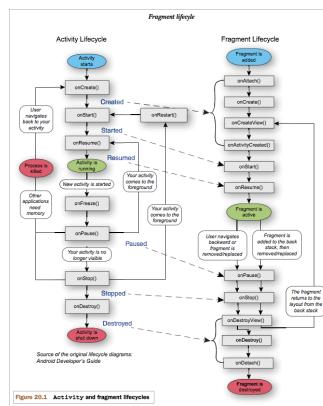
CARACTERÍSTICAS

Ciclo de vida de Fragment:

Embora o *Fragment* possua seu próprio ciclo de vida, que é bastante similar ao de uma *activity*, este não é uma entidade independente.

Na verdade, é parte de uma *activity* hospedeira, que orienta o seu ciclo de vida.

Isso é demonstrado na figura abaixo:



<https://docs.google.com/file/d/0BxbayAAcS8liRld4WTVoZE5Bd0U/edit>

MÉTODOS DO CICLO DE VIDA

Esta lista mostra os principais métodos do ciclo de vida:

Metodo	Descrição
public void onAttach(Activity activity)	Chamado quando a Activity está no estado Created depois que o fragment é associado a sua Activity
public void onCreate(Bundle savedInstanceState)	Chamado quando a Activity está no estado Created para fazer a criação inicial do fragmente
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)	Chamado quando a Activity está no estado Created para criar e retornar a hierarquia de views associada ao fragment
public void onActivityCreated(Bundle savedInstanceState)	Chamado quando a Activity está no estado Created para informar ao fragment de que sua Activity concluiu sua própria Activity onCreate()
public void onStart()	Chamado quando a Activity está no estado Started, indicando que o fragment está visível para o usuário
public void onResume()	Chamado quando a Activity está no estado Resumed, indicando que o fragment agora está interagindo com o usuário
public void onPause()	Chamado quando a Activity está no estado Paused, indicando que o fragment não está mais interagindo com o usuário porque sua Activity está sendo pausada ou uma operação de fragmento está modificando na Activity
public void onStop()	Chamado quando a Activity está no estado Stopped, indicando que o fragment não está mais visível para o usuário porque sua Activity está sendo interrompida
public void onDestroyView()	Chamado quando a Activity está no estado Destroyed para permitir que o fragment limpe os recursos associados a sua view
public void onDestroy()	Chamado quando a Activity está no estado Destroyed para permitir que o fragment faça a limpeza final do estado do fragment
public void onDetach()	Chamado quando a Activity está no estado Destroyed imediatamente anterior ao fragment deixar de estar associado a sua Activity

FRAGMENTS E SUAS SUBCLASSES

Para implementar um *fragment* é preciso estender uma das classes abaixo:

Fragments - Comportamento específico dentro de uma *Activity* (Ex: parte de interface ou operação);

DialogFragment - Exibir uma janela por cima da janela de sua *Activity*;

ListFragment - Exibe uma lista de itens de uma fonte de dados;

PreferenceFragment - Armazena e acessa dados de configuração de uma aplicação;

WebViewFragment - Exibe *WebView*.

API DE FRAGMENTS:

As principais classes da API Fragments são:

Fragment(android.app.Fragment)

- Classe que o fragment deve estender;
- É necessário sobrescrever o método `onCreate` (`inflater, container, bundle`) para criar a view.

Fragment(android.app.Fragment)

Classe que gerencia os fragments pela API;

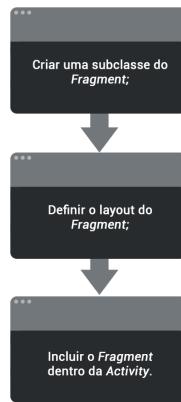
- Possui métodos `findFragmentById(id)` e `findFragmentByTag(tag)` utilizados para encontrar os fragments no layout, de forma similar ao método `findViewById(id)` que uma activity utiliza para buscar uma view.

Fragment(android.app.FragmentTransaction)

- Classe utilizada para adicionar, remover ou substituir os fragments dinamicamente no layout.

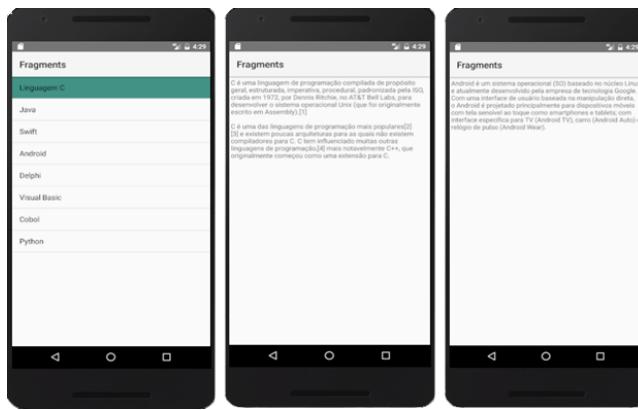
CRIANDO FRAGMENTS

Para implementar um Fragment, basta seguir os três passos básicos:



ATIVIDADE

Observe as imagens:



Desenvolva um pequeno aplicativo que, como demonstrado na tela abaixo, ao selecionarmos uma linguagem de programação, exiba uma breve descrição da mesma.

Resposta Correta

Glossário