Unidade 4

Aula 1

# Navegação

É muito difícil manter um aplicativo que contenha apenas uma tela (Activity). Quando aplicativos ganham um mínimo de complexidade, a necessidade de criar múltiplas telas para dividir as atividades aumenta, e para transitar entre estas telas podemos utilizar diversos recursos para navegação. Nesta aula vamos falar um pouco sobre os Intents, a Toolbar, os TabLayouts e a navegação por menu Drawer.

## Intents

Para transitar de uma tela para outra precisamos utilizar um intermediador. No Android uma tela não tem conhecimento sobre as outras, e elas operam totalmente independentes entre si. Para que haja comunicação entre duas telas, é necessário usar Intent.

Uma Intent (intenção) é uma descrição abstrata de uma operação a ser executada. Ela pode ser utilizada para iniciar uma Activity, ou seja, “ativar” um broadcast (transmissão em larga escala sem necessariamente um receptor), 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” ou um pedido que é encaminhado ao sistema operacional. O sistema receberá 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.

Para exemplificar vamos analisar nosso App **MailList**.

Modificamos o projeto especialmente para esta aula. Então descompacte o arquivo **Unidade\_4\_-\_Aula\_1\_-\_Exemplo\_1.zip** e abra o projeto contido nele no seu Android Studio.

O objetivo desta prática será abrir uma tela a partir da tela Login. Por enquanto esta nova tela não terá funcionalidade nenhuma, mas em aulas futuras a utilizaremos como a tela de **lista de e-mails.**

1. Com o projeto aberto entre no menu **File > New > Activity >Empty Activity.**

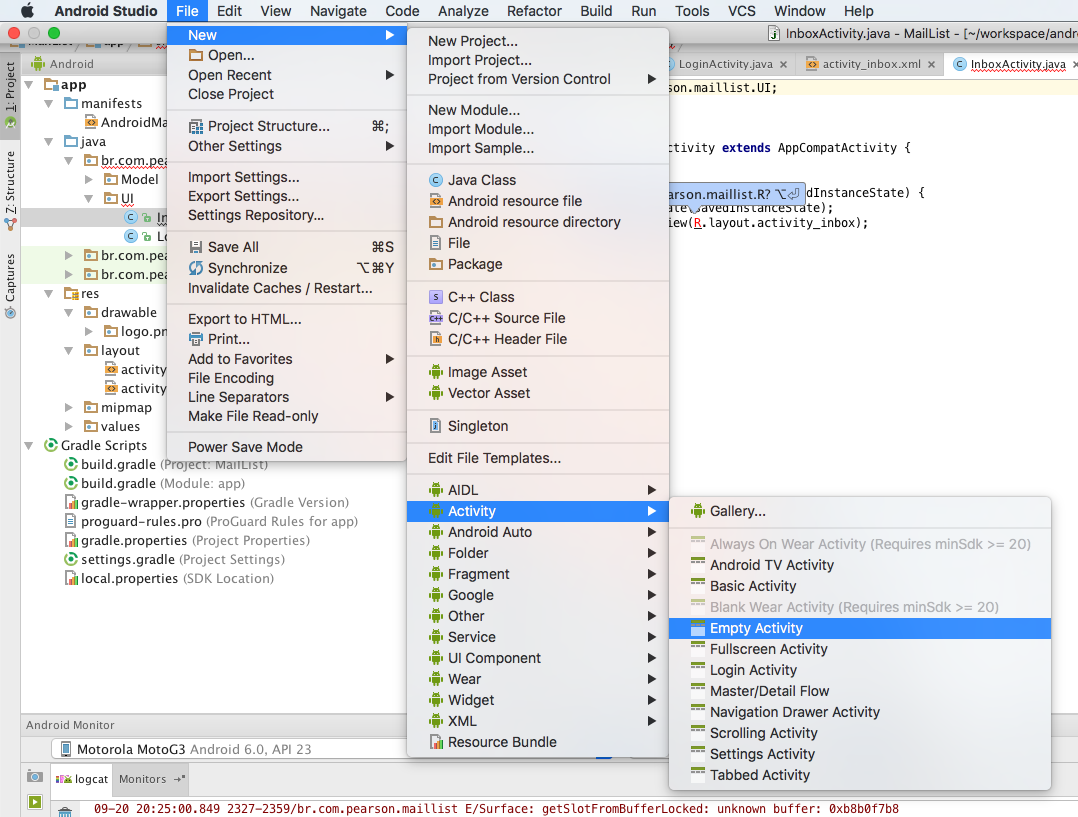


Figura 1 - Criando uma nova Activity

1. A seguinte tela se abrirá:

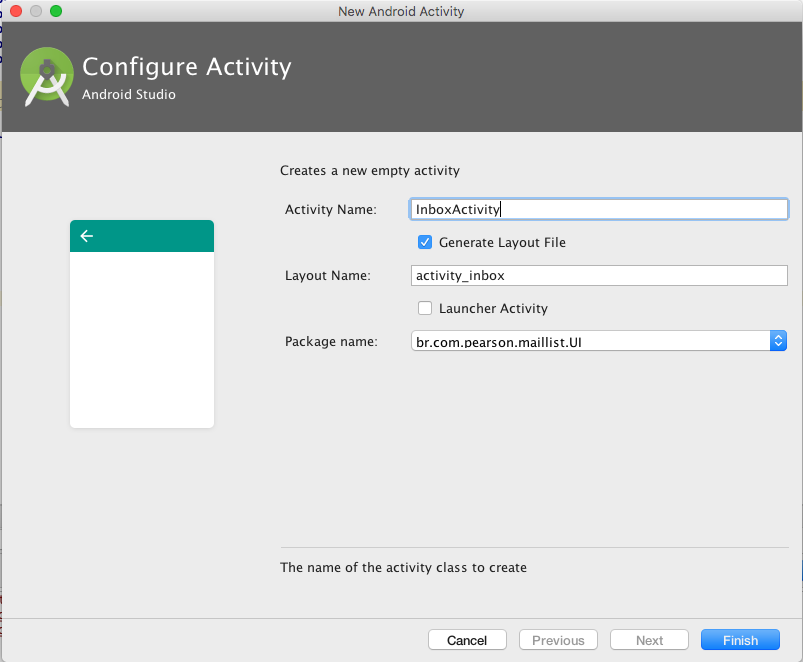


Figura 2 - Configurações iniciais de uma nova Activity

1. No campo “Activity Name” escreva **InboxActivity.** No campo “Layout Name” escreva **activity\_inbox.** Clique em **Finish.**

Vamos colocar uma TextView no nosso novo layout também. Este passo serve apenas para identificarmos nossa nova tela quando realizarmos a transição via Intent.

Agora que temos nossas telas criadas vamos finalmente fazer a transição entre elas utilizando um Intent. Você verá que é mais simples do que se imagina.

1. Entre na classe **LoginActivity.java** e localize o método **startNewActivity().** Substitua este método para o seguinte:

**public** **void** **startNextActivity(**View view**)** **{**

Intent intent **=** **new** Intent**(this,** InboxActivity**.**class**);**

startActivity**(**intent**);**

**}**

Como a assinatura do método mudou (não existe mais o parâmetro **view**) você terá que fazer as devidas correções nos demais lugares que este método é invocado.

1. Execute o app e veja o resultado.

Vamos explicar as novas linhas de código. Começando por:

Intent intent **=** **new** Intent**(this,** InboxActivity**.**class**);**

Nesta linha estamos inicializando o Intent em si. No primeiro parâmetro devemos informar o contexto em que estamos (classe atual - **this**), e no segundo parâmetro informamos a nossa classe destino.

startActivity**(**intent**);**

Esta linha é simples. Como o próprio nome diz, ela inicia uma Activity. Activity esta, informada no Intent criado.

Existem inúmeras possibilidades usando o Intents além de navegações entre telas. Que tal verificar a documentação oficial do Android e se aprimorar ainda mais?

Não perca tempo, conhecimento é sempre bom!

Agora que sabemos como transitar entre telas, vamos aprender mais uma técnica para melhorar a usabilidade do seu app. Vamos inserir a ação de voltar na nossa AppBar.

## ToolBar e Menu Overflow

A AppBar é a barra superior onde se encontra o título da Activity atual. Nela podemos inserir alguns botões que realizam certas ações. Estes botões podem ser colocados tanto ao lado direito quanto ao lado esquerdo do app. Uma AppBar com estes botões se torna uma ToolBar.

Um ponto a se atentar é que no lado esquerdo devemos colocar somente um botão. Geralmente este botão é o botão “voltar” ou o chamado “menu sanduíche”. Já no lado direito podemos inserir quantos botões forem necessários.

É claro que há uma limitação de espaço na nossa ToolBar, então se os botões não couberem nela, acontece o que chamamos de “menu overflow”, onde os botões com menos prioridade se agrupam em um menu representado pelo símbolo . Agora mãos na massa.

Veja como nossa tela ficará:

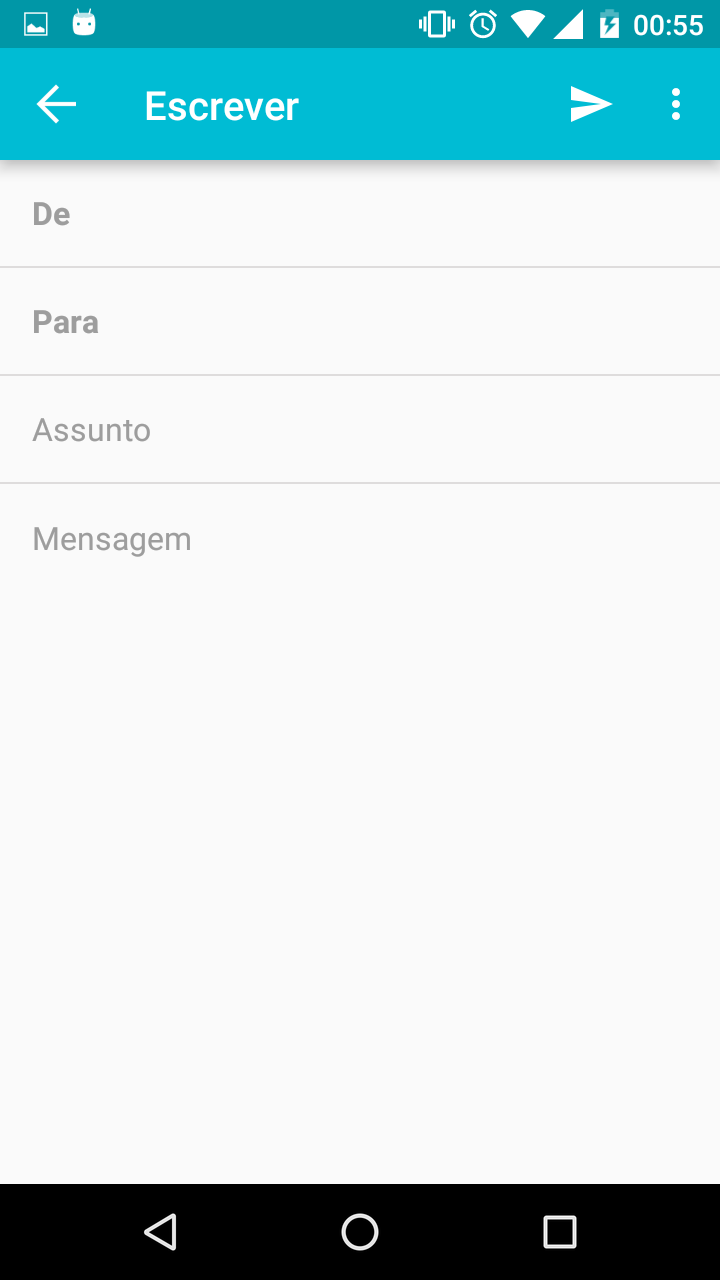


Figura 3 - Tela **Escrever** com itens na barra superior

Novamente deixamos preparado um projeto como ponto de partida para você iniciar este exemplo. Abra o arquivo **Unidade\_4\_-\_Aula\_1\_-\_Exemplo2.zip.** Neste projeto estão inseridas as mudanças feitas na aula anterior. Adicionamos novamente a tela ComposeActivity. É nela que iremos trabalhar. Neste exemplo iremos inserir o botão **Enviar** e o botão **Voltar.** Começaremos pelo botão **Voltar.**

1. Na classe **ComposeActivity** insira o seguinte método:

**public** **void** **setHomeButton()** **{**

**if** **(**getSupportActionBar**()** **!=** **null)** **{**

getSupportActionBar**().**setDisplayHomeAsUpEnabled**(true);**

getSupportActionBar**().**setHomeButtonEnabled**(true);**

getSupportActionBar**().**setHomeAsUpIndicator**(**R**.**drawable**.**back**);**

**}**

**}**

**Entendendo o código:** Este é o trecho que utilizamos para adicionar o botão back na nossa Activity. Os métodos **getSupportActionBar()** são métodos da AppCompatActivity(que é a mãe da nossa ComposeActivity)e ele nos provê a barra de ações. Na sequência estamos deixando o botão home (nome dado ao botão posicionado à esquerda na barra de ações) visível, deixamos então ele habilitado, ou seja, ele receberá interação do usuário e por fim configuramos a imagem que irá aparecer.

1. Agora chame o método criado ao final do método **onCreate().**  Ainda precisamos inserir mais um método.
2. Insira o método que é responsável por dar ação ao botão **back.**

@Override

**public** **boolean** **onOptionsItemSelected(**MenuItem item**)** **{**

**switch** **(**item**.**getItemId**())** **{**

**case** android**.**R**.**id**.**home**:**

finish**();**

**return** **true;**

**}**

**return** **super.**onOptionsItemSelected**(**item**);**

**}**

1. Execute o projeto e veja o que acontece.

Quando você tocar no botão **back** o app irá fechar. Ele executa a ação de finalizar a Activity. Se esta tela tivesse sido aberta a partir de outra Activity, o botão **back** faria com que voltássemos para esta Activity. Como nenhuma Actvity além da ComposeActivity foi aberta ainda, o app irá fechar.

Para testar a ação de voltar para outra Activity, vamos executar os seguintes passos.

Para mudarmos a Activity inicial, precisaremos mudar algumas configurações no AndroidManifest.xml. Entre no **AndroidManifest.xml** e mova o <intent-filter> da ComposeActivity para a InboxActivity. Seu manifesto ficará da seguinte maneira:

**<?xml version="1.0" encoding="utf-8"?>**

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="br.com.pearson.maillist">

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=".UI.LoginActivity">

</activity>

<activity

android:name=".UI.InboxActivity"

android:label="Entrada">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

<activity

android:name=".UI.ComposeActivity"

android:label="Escrever">

</activity>

</application>

</manifest>

1. Agora entre no arquivo **activity\_inbox.xml** e insira um botão nele como na imagem abaixo:

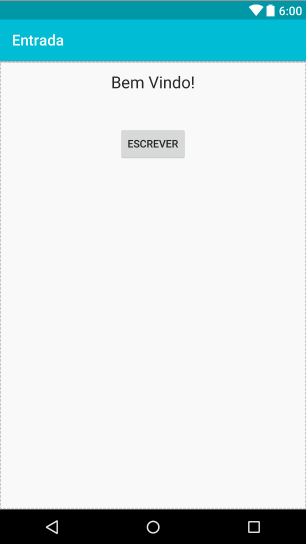


Figura 4 - Tela com o botão de acesso à tela Compose

Não se esqueça de dar um id para ele (demos o id "@+id/compose").

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Escrever"

android:id="@+id/compose"

android:layout\_below="@+id/textView"

android:layout\_centerHorizontal="true"

android:layout\_marginTop="45dp" />

1. Agora entre no InboxActivity.java e inicialize o ButterKnife no método **onCreate()**:

@Override

**protected** **void** **onCreate(**Bundle savedInstanceState**)** **{**

**super.**onCreate**(**savedInstanceState**);**

setContentView**(**R**.**layout**.**activity\_inbox**);**

ButterKnife**.**bind**(this);**

**}**

1. Adicione o método onClick para o botão **compose.**

@OnClick**(**R**.**id**.**compose**)**

**public** **void** **composeButtonClicked()** **{**

**}**

Por fim, inicie um Intent com destino a classe ComposeActivity.

@OnClick**(**R**.**id**.**compose**)**

**public** **void** **composeButtonClicked()** **{**

Intent intent **=** **new** Intent**(this,** ComposeActivity**.**class**);**

startActivity**(**intent**);**

**}**

1. Execute o app e veja o resultado. Você poderá entrar e sair da tela Compose.

Agora quando tocamos no botão **back** da ComposeActivity, o app retorna para a InboxActivity.

### Menu Overflow

Aprendemos a adicionar o chamado botão **home** que demos a funcionalidade de voltar para a tela anterior. Agora iremos adicionar botões do lado direito da ToolBar e testar o menu overflow. Lembrando que a Toolbar é a barra superior do nosso app. Quando ela está em seu estado básico (somente com um título) é chamada de AppBar, mas quando possui botões ela passa a ser uma Toolbar (traduzindo do inglês, uma barra de ferramentas). Neste exemplo iremos inserir o botão **send** e a opção **Sobre**, que ficará dentro do menu. Siga os seguintes passos.

1. A primeira coisa que devemos fazer é criar um recurso para o menu. Clique no menu **File > New > Android Resource File.** A seguinte tela irá aparecer:

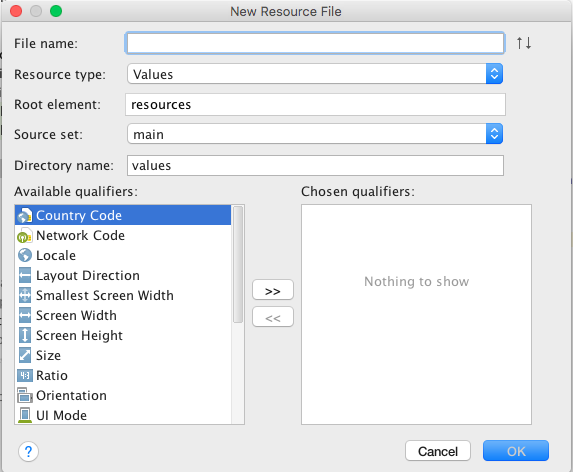


Figura 5 - Tela "New Resource File"

1. No campo “File Name” insira o nome **compose\_menu**. No campo “Resource Type”, escolha a opção **Menu.** Clique em **OK.**

Você terá o seguinte código:

**<?xml version="1.0" encoding="utf-8"?>**

<menu xmlns:android="http://schemas.android.com/apk/res/android">

</menu>

1. Este é um arquivo que utilizamos para criar um menu. Adicione a seguinte propriedade:

**<?xml version="1.0" encoding="utf-8"?>**

<menu xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto">

</menu>

1. Agora adicione estes dois itens dentro do menu:

<item android:id="@+id/send"

android:title="Send"

android:icon="@drawable/send"

android:orderInCategory="99"

app:showAsAction="always"/>

<item android:id="@+id/about"

android:title="Sobre"

android:orderInCategory="100"

app:showAsAction="never"/>

**Entendendo o código:** Estes itens mostrarão algumas opções no menu que estamos criando. Algumas propriedades são intuitivas, como id (identificador do item), title (texto que será exibido no item) e icon (ícone do item), porém existem dois atributos peculiares aqui. O primeiro é o orderInCategoryque determina a ordem que os itens aparecerão, sendo que quanto maior o número, mais a direita o item irá aparecer. O seguinte atributo é o que determina quando o item irá aparecer. Neste caso determinamos que o botão **send** aparecerá sempre e o botão **about** não aparecerá nunca, ou seja, ele transbordará e ficará dentro do menu overflow.

Para finalizar. Entre na classe ComposeActivity e insira o seguinte método:

@Override

**public** **boolean** **onCreateOptionsMenu(**Menu menu**)** **{**

**super.**onCreateOptionsMenu**(**menu**);**

getMenuInflater**().**inflate**(**R**.**menu**.**compose\_menu**,** menu**);**

**return** **true;**

**}**

**Entendendo o código:** Este é um método que infla o menu na nossa Activity. O termo inflar é utilizado com o significado de injetar, ou instanciar dentro. Utilizamos o utilitário MenuInflater da nossa Activity e inserimos o nosso **compose\_menu** nela.

1. Execute o código e veja o resultado.

Parece fácil não é? Existem muitas outras possibilidades de menus que podemos implementar. Uma delas é o NavigationDrawer, um clássico menu lateral que iremos aprender a seguir.

## Navigation Drawer

O NavigationDrawer é um menu lateral, bastatne popular nos aplicativos. O nome Drawer vem de gaveta, pois podemos utilizar um movimento para puxá-lo como uma gaveta.

Vamos inserir um menu como este no nosso aplicativo MailList. Como estamos nos baseando no app Gmail. Vamos tentar criar um Drawer próximo deste:

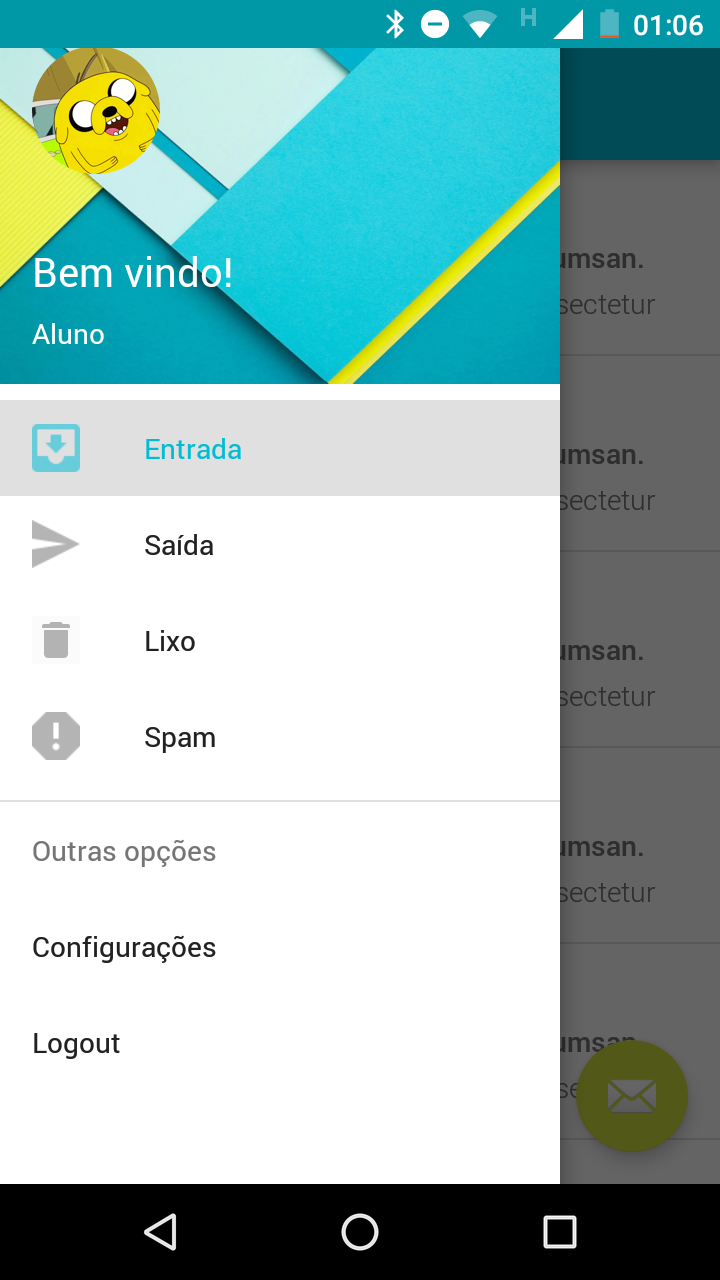


Figura 6 - Menu proposto

1. Vamos criar alguns arquivos. Primeiro crie o arquivo **menu/drawer\_view.xml** (estamos resumindo o procedimento de criação de arquivos pois você já aprendeu em lições anteriores).
2. Substitua o código gerado deste novo arquivo pelo seguinte:

<menu xmlns:android="http://schemas.android.com/apk/res/android">

<group android:checkableBehavior="single">

<item

android:id="@+id/nav\_inbox\_fragment"

android:icon="@drawable/ic\_inbox"

android:title="Entrada" />

<item

android:id="@+id/nav\_outbox\_fragment"

android:icon="@drawable/ic\_outbox"

android:title="Saída" />

<item

android:id="@+id/nav\_third\_fragment"

android:icon="@drawable/ic\_trash"

android:title="Lixo" />

<item

android:id="@+id/nav\_fourth\_fragment"

android:icon="@drawable/ic\_spam"

android:title="Spam" />

</group>

<item android:title="Outras opções">

<menu>

<group android:checkableBehavior="single">

<item

android:title="Configurações" />

<item

android:title="Ajuda" />

</group>

</menu>

</item>

</menu>

**DICA**: Note que você pode deixar um destes itens como selecionado por padrão. Basta adicionar o atributo android:checked="true" no item que deseja deixar pré-selecionado.

1. Ao passo que nosso Drawer será deslizado por cima da nossa ActionBar, precisamos criar nossa própria ActionBar. A ToolBar pode ser embarcada dentro dos nossos layouts então crie o arquivo **res/layout/toolbar.xml.**
2. Substitua o código do **toolbar.xml**  pelo seguinte código:

<android.support.v7.widget.Toolbar

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

android:id="@+id/toolbar"

android:layout\_height="wrap\_content"

android:layout\_width="match\_parent"

android:fitsSystemWindows="true"

android:minHeight="?attr/actionBarSize"

app:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"

android:background="?attr/colorPrimary"

android:elevation="8dp">

</android.support.v7.widget.Toolbar>

Neste trecho de layout estamos criando uma Toolbar básica para nossa aplicação. Note que temos o atributo fitsSystemWindows=true, sem ele a status bar (a barra que mostra as notificações dos apps, bateria, hora, etc...) ficaria por cima da Toolbar, dando um efeito translúcido. Este atributo aumenta o padding do topo da Toolbar para que não haja esta sobreposição.

**DICA:** Padding e margin são termos utilizados para definir espaçamento entre views. A diferença entre eles é que o padding aumenta o espaçamento entre frame (área da View) da View e o próprio conteúdo interno da mesma. A margin aumenta o espaçamento entre views.

1. Antes de criar mais arquivos, precisamos mudar o tema da nossa aplicação, para que ele não insira uma ActionBar. Assim podemos utilizar a nossa Toolbar tranquilamente. Para isto entre no arquivo **res/values/styles.xml**  e substitua o valor do atributo parent da tag style para :

**"Theme.AppCompat.Light.NoActionBar"**

**Entendendo o código:** Neste arquivo **styles** podemos trocar o tema do aplicativo. Um tema vai desde uma combinação de cores até adição ou remoção de alguns componentes gráficos. O tema selecionado faz com que o app não tenha uma ActionBar (ou Toolbar), para que possamos criar e usar nossa própria Toolbar.

1. Agora que definimos trocamos o tema padrão da nossa ActionBar (com o passo anterior), que controlarão tudo isto. Primeiro crie o arquivo **res/layout/activity\_main.xml**.
2. Neste novo arquivo remova todo o código e insira o seguinte código.

*<!-- Este DrawerLayout tem duas views filhas em sua raiz -->*

<android.support.v4.widget.DrawerLayout

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

android:id="@+id/drawer\_layout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

*<!-- Este LinearLayout representa o conteúdo da tela -->*

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical">

*<!-- Esta ActionBar/Toolbar é mostrada no topo da tela -->*

<include

layout="@layout/toolbar"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" />

*<!-- Este frame é onde nossos Fragments serão colocados -->*

<FrameLayout

android:id="@+id/flContent"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

</LinearLayout>

</android.support.v4.widget.DrawerLayout>

**Entendendo o código:** Note que nossa Toolbar foi injetada através do elemento **include.** Como não tínhamos mais Toolbar devido ao tema selecionado no passo 5 e criamos nossa própria Toolbar, precisamos de certa forma inserí-la no layout. O elemento include é responsável por injetar arquivos de layout em um outro layout.

1. Agora é hora de criar nossa Activity. No pacote UI, crie uma nova Activity chamada MainActivity.
2. Substitua todo o conteúdo da nova activity pelo seguinte código:

public class MainActivity extends AppCompatActivity {

@BindView(R.id.***drawer\_layout***) DrawerLayout **drawerLayout**;

@BindView(R.id.***toolbar***) Toolbar **toolbar**;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

ButterKnife.bind(this);

setSupportActionBar(toolbar);

setHomeButton();

}

public void setHomeButton() {

if (getSupportActionBar() != null) {

getSupportActionBar().setDisplayHomeAsUpEnabled(true);

getSupportActionBar().setHomeButtonEnabled(true);

getSupportActionBar().setHomeAsUpIndicator(R.drawable.*menu*);

}

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

*// A ação home/up action irá abrir ou fechar o drawer.*

switch (item.getItemId()) {

case android.R.id.*home*:

**drawerLayout**.openDrawer(GravityCompat.*START*);

return true;

}

return super.onOptionsItemSelected(item);

}

*// `onPostCreate` é chamado depois `onStart()` quando a inicialização da Activity está completa*

@Override

protected void onPostCreate(Bundle savedInstanceState) {

super.onPostCreate(savedInstanceState);

}

}

**Entendendo o código:** Vamos entender o código que acabamos de inserir. Primeiro veja as instruções no método **onCreate():**

toolbar = (Toolbar) findViewById(R.id.*toolbar*);

setSupportActionBar(toolbar);

Através deste comando podemos adicionar nossa toolbar na nossa Activity, funcionalmente. Agora veja o método **setHomeButton()** e perceba que ele é igual ao implementado na nossa ComposeActivity, com a diferença de possuir um ícone diferente. Desta vez o ícone é o **menu**, pois não poderíamos manter o botão **back** porque não faria sentido a real ação do home button que agora irá abrir um menu, e não voltar pra tela anterior.

Veja que temos também o método **onOptionsItemSelected()** com a diferença que a ação do botão home agora é abrir o menu drawer.

Você ainda pode inserir um cabeçalho ao seu menu. Para isso siga os seguintes passos:

1. Deixamos pronto para você o arquivo de header. Entre no arquivo **activity\_main.xml**  adicione o seguinte atributo para o elemento NavigationView.

<android.support.design.widget.NavigationView

android:id="@+id/nvView"

android:layout\_width="wrap\_content"

android:layout\_height="match\_parent"

android:layout\_gravity="start"

android:background="@android:color/white"

app:menu="@menu/drawer\_view"

app:headerLayout="@layout/nav\_header"/>

1. Agora estamos com o visual do nosso menu pronto. Ele ficará como na figura abaixo:

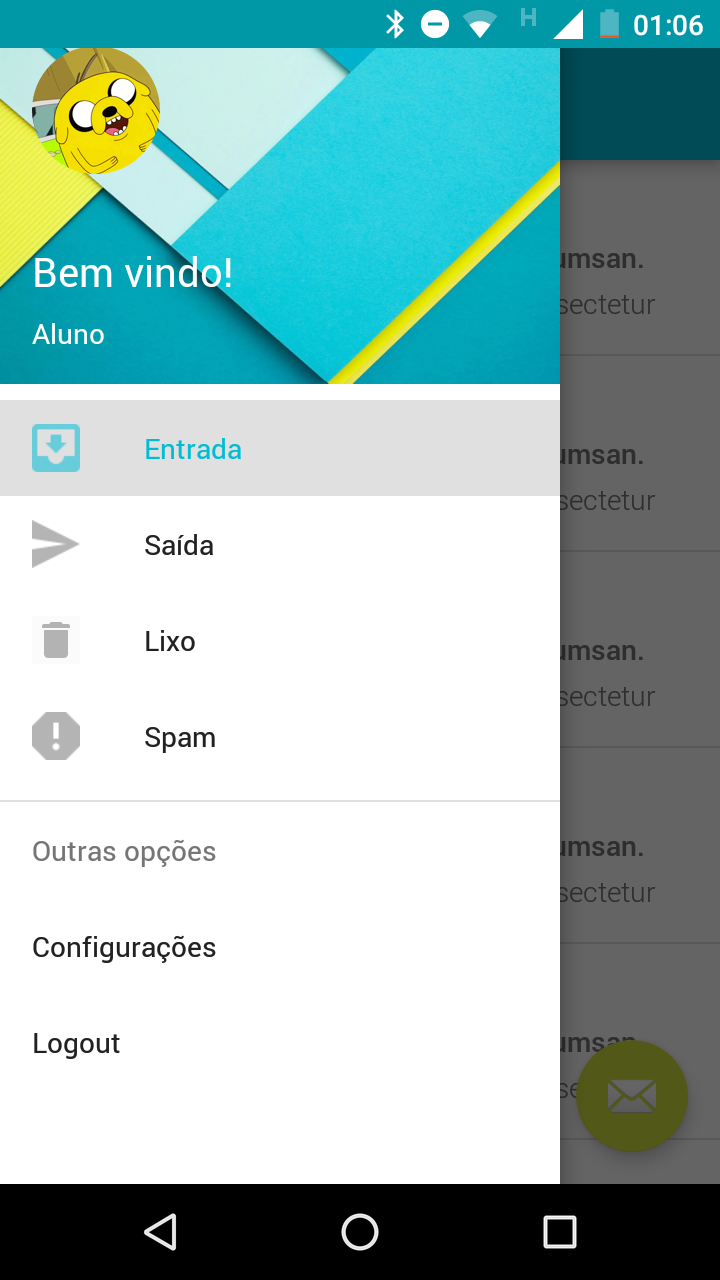


Figura 7 - Menu pronto

**Obs**.: Esta imagem é ilustrativa. Se a sua tentativa não estiver idêntica a esta imagem não se preocupe. A nossa intenção é apresentar a funcionalidade.

Estamos com o visual do nosso menu pronto. Você conseguirá visualizá-lo clicando no ícone de menu no canto superior direito da tela do seu app. Confira a próxima aula, que falaremos sobre Activities e Fragments, e iremos terminar a navegação via Drawer.

## Resumo

Aprendemos algumas técnicas de navegação entre telas e entre elas a navegação via Intents e via Navigation Drawer. Além disso aprendemos a customizar nossa Toolbar e otimizar nossas Activities para reúso, criando uma Acitivity base.

## Exercícios

## TDP

Aula 2

# Activities e Fragments

Depois de tanto usarmos nas aulas anteriores, finalmente você vai entender o que é uma Activity. Esta é uma das classes principais e que teremos mais contato no desenvolvimento Android. Ao lado de Activities temos os Fragments. Nesta aula você irá entender o que é uma Activity e um Fragment, vai saber diferenciá-los e como usá-los. Vamos lá.

## Activities

A Activity é popularmente conhecida como uma tela do seu aplicativo. Em partes isto é correto, na arquitetura do Android, utilizamos uma Activity para uma tela. Na verdade ela é a controladora da tela e fornece ações (atividades) no qual o usuário pode interagir e fazer algo, como mandar uma mensagem, escrever um e-mail, aplicar um filtro em uma foto, e etc. Geralmente a janela produzida por esta Activity preenche a tela do seu smartphone, mas podemos fazer telas menores flutuantes.

As Activities são pouco vinculadas entre si, e isto evita o acoplamento, com exceção de casos muito específicos, onde existe uma complexidade visual muito grande da interface do usuário. A maneira que as Activities se comunicam entre si é por meio de intenções (Intents), que falaremos mais sobre futuramente.

Lembra que definimos uma Activity no início dos projetos? Este tipo de Activity é especificada como “principal”, e será apresentada ao usuário ao iniciar o aplicativo. Cada Activity pode então iniciar outras Activities para executar outras ações diferentes.

Quando iniciamos uma nova Activity, ela é colocada no que chamamos de “pilha de retorno”. Esta pilha é uma estrutura de dados LIFO (Last in, First out - ou **Último a entrar é o primeiro a sair**) que guarda a sequência e estado das Activities instanciadas, dando então a possibilidade de usar a ação “Voltar” para desempilhar estas telas, ou melhor, voltar para a tela anterior. Após iniciar uma nova Activity, a atividade da anterior é suspensa (falaremos mais sobre o ciclo de vida das Activities).

Quando uma atividade é interrompida devido ao início de uma nova atividade, ela é notificada acerca dessa alteração de estado por meio de métodos de retorno de chamada do ciclo de vida da atividade. Há diversos métodos de retorno de chamada que uma atividade pode receber devido a uma alteração em seu estado (quando o sistema a está criando, interrompendo, retomando ou destruindo) e cada retorno de chamada oferece uma oportunidade de executar trabalhos específicos adequados a essa alteração de estado. Por exemplo: quando interrompida, a atividade deve liberar todos os objetos grandes, como conexões com a rede ou com um banco de dados. Quando a atividade for retomada, será possível readquirir os recursos necessários e retomar as ações interrompidas. Essas transições de estado são parte do ciclo de vida da atividade.

### Criação de uma atividade

Para criarmos uma atividade é preciso criar uma subclasse de Activity ou de uma respectiva subclasse existente.

**DICA:** Existem várias subclasses de Activcity, cada uma com um conjunto de funcionalidades diferentes - pesquise na documentação oficial do Android.

Ao criarmos uma subclasse, precisamos implementar o que chamamos de métodos de retorno de chamada que o sistema chama quando ocorre a transição entre os diversos estados de seu ciclo de vida. Eis aí mais um Padrão de Projeto para entender: o **Template Method.** Para que você possa entender com mais propriedade o funcionamento destes métodos de retorno de chamada, na Unidade 5 preparamos uma explicação sobre o Template Method, que é o padrão de projeto utilizado pela API do Android para este tipo de atividade.

Os dois métodos mais importantes de retorno de chamada são:

onCreate()

É preciso implementar esse método. O sistema o chama ao criar a atividade. Na implementação, é preciso inicializar os componentes essenciais da Activity. E é necessariamente neste método que devemos invocar o setContentView() para relacionar o layout da interface com a Activity que o controlará.

onPause()

O sistema chama esse método como o primeiro indício de que o usuário está saindo da Activity (embora não seja sempre uma indicação de que a atividade será destruída). É quando geralmente se deve confirmar qualquer alteração que deva persistir além da sessão do usuário atual (porque o usuário pode não retornar).

Há outros métodos de retorno de chamada do ciclo de vida que se pode usar para oferecer uma experiência fluida ao usuário entre atividades e manipular interrupções inesperadas que venham a parar ou até a destruir a atividade.

### Gerenciamento do ciclo de vida da atividade

O ciclo de vida de uma atividade é diretamente afetado por sua associação a outras Activities, sua tarefa e sua pilha de retorno. Uma Activity existe essencialmente em três estados:

**Retomada**

A Activity está em primeiro plano na tela e tem o foco do usuário (em geral, chama-se esse estado de "em execução”).

**Pausada**

A Activity ainda está aberta, mas outra Activity está em primeiro plano e tem o foco. Ou seja, outra Activity está visível por cima desta e está parcialmente transparente ou não cobre inteiramente a tela.

**Interrompida**

A atividade está totalmente suplantada por outra (a atividade passa para "segundo plano"). Uma atividade interrompida ainda está ativa (o objeto Activity está retido na memória, mantém todas as informações de estado e do membro, mas *não* está anexado ao gerenciador de janelas). No entanto, ela não fica mais visível para o usuário e pode ser eliminada pelo sistema se a memória for necessária em outro processo.

Se uma atividade estiver pausada ou interrompida, o sistema poderá descartá-la da memória solicitando a finalização do processo (chamando seu método finish()) ou simplesmente eliminando-o. Quando a Activity for reaberta (depois de finalizada ou eliminada), ela deverá ser totalmente recriada.

#### Implementação de retornos de chamada do ciclo de vida

Como, afinal, implementamos estes métodos de retorno de chamada? Uma Activity transita entre os diferentes estados citados acima de acordo com o uso do aplicativo, quando o estado é alterado, seu objeto Activity é notificado através de alguns métodos. Estes métodos são chamados de métodos de retorno de chamada, pois em algum lugar, em uma camada mais baixa da API eles são chamados, mas não possuem ação alguma, pois em sua definição se comportam como abstratos. Eles são executados quando acontece uma mudança de estado na Activity, e podemos interceptar estas mudanças apenas implementando estes métodos nas nossas subclasses de Activity.

Esta operação de métodos Abstratos é parecida com o **exemplo dos empregados** no livro de POO, onde criamos uma classe abstrata **Employee** que possuía um método abstrato **getBonus().** O bônus era calculado somente nas subclasses de **Employee.**  Quando calculamos a calculadora de bônus, ela simplesmente recebia um funcionário, independente da especialização dele e calculava seu bônus. É mais ou menos assim que as Activities funcionam. Na aula de padrões de projeto sobre o **Template Method** você irá entender com mais fluidez, mas por enquanto vamos assimilar com o exemplo dos empregados.

O esqueleto de atividade a seguir contém cada um dos métodos do ciclo de vida fundamentais:

public class ExampleActivity extends Activity {

@Override

public void [onCreate](https://developer.android.com/reference/android/app/Activity.html#onCreate(android.os.Bundle))(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

// A Activity está sendo criada

}

@Override

protected void [onStart()](https://developer.android.com/reference/android/app/Activity.html#onStart()) {

super.onStart();

// A Activity está prestes a ficar visível

}

@Override

protected void [onResume()](https://developer.android.com/reference/android/app/Activity.html#onResume()) {

super.onResume();

// A Activity está agora visível e foi retomada

}

@Override

protected void [onPause()](https://developer.android.com/reference/android/app/Activity.html#onPause()) {

super.onPause();

// Outra Activity está visível e em foco, mas esta ainda não foi destruída, apenas está pausada.

}

@Override

protected void [onStop()](https://developer.android.com/reference/android/app/Activity.html#onStop()) {

super.onStop();

// A Activity não está mais visível (ela foi parada)

}

@Override

protected void [onDestroy()](https://developer.android.com/reference/android/app/Activity.html#onDestroy()) {

super.onDestroy();

// A Activity está prestes a ser destruída

}

}

**DICA:** A implementação desses métodos do ciclo de vida deve sempre chamar a implementação da superclasse (**super**) antes de realizar qualquer trabalho, conforme ilustrado no exemplo acima.

Juntos, esses métodos definem todo o ciclo de vida da atividade. Ao implementá-los, é possível monitorar três momentos importantes no ciclo de vida da atividade:

* **Todo o tempo de vida** de uma **Activity** acontece entre a chamada de **onCreate**() e a chamada de **onDestroy**(). A **Activity** deve executar todo pré-processamento, inicialização de recursos, definir o layout e outras coisas “globais” ao funcionamento desta **Activity** no método **onCreate()** e deve liberar memória, desalocando objetos ou salvando um dado ou estado no cache no método **onDestroy()**.
* **O tempo de vida visível** de uma **atividade** acontece entre a chamada de **onStart**() e a chamada de **onStop**(). Durante este tempo podemos ver a **Activity** na tela e consequentemente podemos interagir com ela durante este tempo.
* **O** **tempo de vida em primeiro plano** de uma atividade ocorre entre a chamada de **onResume**() e a chamada de **onPause**(). Durante esse tempo, a **Activity** está na frente de todas as outras **Activities** na tela e tem o foco de interação do usuário. Frequentemente, uma atividade pode transitar entre o primeiro e o segundo plano, por exemplo, **onPause**() é chamado quando o dispositivo está em suspensão ou quando uma caixa de diálogo é exibida. Como esse estado pode transitar frequentemente, o código nesses dois métodos deve ser bem leve para evitar transições lentas que façam o usuário esperar.

A figura abaixo é uma ilustração clássica do ciclo de vida das Activities e que está disponível na documentação oficial do Android. Os retângulos representam os métodos de retorno de chamada que podem ser implementados para executar operações quando a atividade transita entre estados.



Figura 7 - Ciclo de Vida de uma Activity

A figura acima demonstra o fluxo que o aplicativo opera e podemos identificar algumas regras de transição entre os métodos de chamada de retorno, por exemplo, não é permitido transitar partindo do método **onDestroy()** para o **onStop()** ou **onPause()** por exemplo, pois este caminho não faz parte do ciclo.

Daqui para frente você pode notar que teremos a necessidade de repetir muitos códigos entre as Activities. Por exemplo, na aula passada criamos o botão **back.** Existirá a necessidade de inserir este botão em diversas telas daqui para frente, então utilizaremos uma boa prática que é criar um **BaseActivity** que servirá como Activity base para evitarmos repetição de código que é são reutilizados entre as Activities.

1. No pacote UI, crie uma nova Activity chamada **BaseActivity**.
2. Faça com que esta classe estenda de **AppCompatActivity**.

**public class** BaseActivity **extends** AppCompatActivity { //… }

1. Lembra que criamos uma Toolbar na aula anterior? Vamos reaproveitá-la aqui também, então adicione-a nesta classe:

@BindView(R.id.***toolbar***) Toolbar **toolbar**;

1. Implemente o método **setContentView()** para iniciar o bind do ButterKnife:

@Override

**public void** setContentView(**int** layoutResID) {

**super**.setContentView(layoutResID);

ButterKnife.*bind*(**this**);

}

Aqui temos o nosso primeiro código que gerará repetição entre as Activities, a inicialização do **BetterKnife**.

1. Neste mesmo método vamos configurar nossa Toolbar, chamando o seguinte método:

@Override

**public void** setContentView(**int** layoutResID) {

**super**.setContentView(layoutResID);

ButterKnife.*bind*(**this**);

setSupportActionBar(**toolbar**);

}

1. Agora vamos inserir o código que gera o botão **back.** Insira o seguinte método na classe:

**public void** setHomeButton(@DrawableRes **int** drawableId) {

**if** (getSupportActionBar() != **null**) {

getSupportActionBar().setDisplayHomeAsUpEnabled(**true**);

getSupportActionBar().setHomeButtonEnabled(**true**);

getSupportActionBar().setHomeAsUpIndicator(drawableId);

}

}

Neste caso, estamos criando um botão com imagem dinâmica, ou seja poderemos especificar o drawable desejado ao chamar o método.

1. Após inserir um botão back, vamos dar ação a ele. Insira o seguinte método:

@Override

**public boolean** onOptionsItemSelected(MenuItem item) {

*// The action bar home/up action should open or close the drawer.*

**switch** (item.getItemId()) {

**case** android.R.id.***home***:

finish();

**return true**;

}

**return super**.onOptionsItemSelected(item);

}

1. Ainda não é necessário chamar o método **setHomeButton()** no nosso **onCreate()** pois nem todas as Acitivities terão este botão.

Agora temos nossa Activity base. Nos exemplos seguintes iremos refatorar aos poucos nossas Activities e você verá o efeito desta nova classe.

## Fragments

Agora que conhecemos as Activities, apresentamos-lhe o **Fragment.**

O **Fragment** é uma estrutura poderosa, inserida na versão 3.0 (HoneyComb) do Android, justamente para auxiliar na criação de interfaces flexíveis e fluidas em tablets.

Como o próprio nome sugere, o **Fragment** é um fragmento de uma tela, do seu app. Podemos dizer que é uma espécie de “mini Activity”, pois sua estrutura é muito similar à da Activity e possui seu próprio ciclo de vida.

Por ser um “pedaço” de tela, podemos utilizá-lo em diversos lugares e aproveitar o poder da reusabilidade que o **Fragment** nos proporciona.

Mas qual, de fato, é a utilidade dos **Fragments?**

Um problema bem comum no desenvolvimento de aplicativos Android é manter um app com uma experiência de uso de interface gráfica consistente para todos os dispositivos Android, independente do tamanho da tela. O Fragment poderia resolver este problema para você.

Como uma Activity, o Fragment também possui seu próprio ciclo de vida, como ilustra a imagem abaixo:



Figura 8 - Ciclo de vida de um Fragment

Seus métodos são bem similares aos de uma Activity, então não entraremos em detalhes.

Vamos entender com um exemplo prático.

Imagine que queiramos colocar um menu lateral (chamado de Drawer) no nosso app MailList. Olhando o comportamento do Gmail (aplicativo que inspira o nosso MailList) vemos que este menu fica oculto na lateral esquerda e é mostrado quando tocamos no respectivo ícone da AppBar.

Se executarmos o Gmail em um tablet, este menu lateral fica visível em tempo integral, pois existe espaço hábil para isto.

Veja o esboço do layout dessas duas situações:



Figura 9 - Esboço de aplicação de Fragments em múltiplas telas

### Formas de resolver este problema

Sem os Fragments, você teria que preparar um layout para cada esquema de tela (Tablet e Smartphone), até ai tudo bem.

Você também deveria ter uma Activity, que teria que realizar várias comparações para saber se o app está sendo executado em um tablet ou em um smartphone, para assim saber como montar a tela ou saber como proceder quando um item do menu for clicado.

No final das contas teríamos uma Activity de código grande e de difícil manutenção. Usando um Fragments você teria um Fragment (menu) que listaria as opções e outros Fragments que seriam o conteúdo das respectivas opções do menu.

Como resultado sua Activity ficará muito mais leve, uma vez que as responsabilidades de exibir lista e mostrar detalhes foram delegadas para dois Fragments, e não ficará apenas em uma Activity.

### Exemplo prático

Vamos retomar nosso aplicativo MailList. Abra o arquivo **Unidade\_4\_-\_Aula\_2\_-\_Exemplo\_1.zip.**

O objetivo agora é fazer a troca das telas, para isto precisamos deixar de usar as Activities em algumas situações e trocar pelos Fragments. Vamos agora fazer a troca de telas através do Navigation Drawer.

1. Entre na nossa **MainActivity.java** e a primeira coisa que devemos fazer é fazer com que nossa Activity Extenda a **BaseActivity.**
2. Vamos otimizar nosso código utilizando a BaseActivity. Remova a chamada **ButterKnife.bind(this);** do método **onCreate().**
3. Remova o método setHomeButton()
4. No método onCreate(), localize a chamada do método setHomeButton() e substitua por:
5. Otimizações feitas, vamos inserir as funcionalizadas para manipulação dos fragments. Primeiro vamos substituir algumas Activities por Fragments:
   1. Crie uma classe Java estendendo um Fragment e dê o nome **InboxFragment.**
   2. Agora renomeie o arquivo **activity\_inbox** para **fragment\_inbox**.
   3. Na classe **InboxFragment** insira o seguinte código:

@Override

**public** View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.***fragment\_inbox***, container, **false**);

ButterKnife.*bind*(**this**, view);

**return** view;

}

**Entendendo o código:** O código acima representa o que dizemos ser relativo ao método **onCreate()**  de uma Activity, pois o ciclo de vida de um Fragment é bem similar ao de uma Activity.

1. Agora que temos nosso Fragment para a tela Inbox, crie outro Fragment qualquer para mostrarmos. Neste exemplo criamos um Fragment para a tela “Saída” com um texto simples apenas para nível de demonstração, mas certifique-se em ter pelo menos dois Fragments criados.
2. Volte para a **MainActivity**; nós iremos inserir as funcionalidades faltantes para conseguirmos transitar entre telas. Primeiro vamos criar um método que faz a troca dos fragments utilizando um FragmentManger. Insira o seguinte método:

**public void** changeToFragment(Fragment fragment) {

FragmentManager fragmentManager = getSupportFragmentManager();

fragmentManager.beginTransaction().replace(R.id.***flContent***, fragment).commit();

}

**Entendendo o código:**  este procedimento é parecido com o procedimento de troca de Activity via Intents visto na aula anterior. Para os Fragments temos o FragmentManager, que é responsável pelo gerenciamento dos Fragments.

1. Agora vamos resgatar nossos Fragments através dos respectivos métodos:

**public** InboxFragment getInboxFragment() {

InboxFragment inboxFragment = **new** InboxFragment();

setTitle(**"Entrada"**);

**return** inboxFragment;

}

**public** OutboxFragment getOutboxFragment() {

OutboxFragment outboxFragment = **new** OutboxFragment();

setTitle(**"Saída"**);

**return** outboxFragment;

}

**Entendendo o código:** Nestes métodos temos uma simples instanciação dos métodos, com a diferença que trocamos o título da nossa MainActivity de acordo com o fragment instanciado.

1. Precisamos então criar um método que decide qual Fragment instanciar, de acordo com o item do menu selecionado. Para isto utilizaremos o seguinte método:

**public void** selectDrawerItem(MenuItem menuItem) {

**switch**(menuItem.getItemId()) {

**case** R.id.***nav\_inbox\_fragment***:

changeToFragment(getInboxFragment());

**break**;

**default**:

changeToFragment(getOutboxFragment());

}

*// Destaca o item do menu selecionado*

menuItem.setChecked(**true**);

*// Troca o título na Toolbar*

setTitle(menuItem.getTitle());

*// Fecha o drawer*

**drawerLayout**.closeDrawers();

}

**Entendendo o código:** Este método será chamado sempre que um item do menu Drawer for selecionado, então por parâmetro recebemos um item do menu, e de acordo com este instanciamos o Fragment correto. Em seguida utilizamos o método setChecked() para manter o item selecionado com uma coloração de destaque. Depois modificamos o título da tela, e por fim fecha-se o Drawer

1. Agora vamos utilizar todos estes métodos criados. Os itens do menu também possuem listeners que são disparados quando clicamos em um deles, mas infelizmente a biblioteca ButterKnife não da suporte para este listener. Crie o seguinte método:

**private void** setupDrawerContent(NavigationView navigationView) {

changeToFragment(getInboxFragment());

navigationView.setNavigationItemSelectedListener(

**new** NavigationView.OnNavigationItemSelectedListener() {

@Override

**public boolean** onNavigationItemSelected(MenuItem menuItem) {

selectDrawerItem(menuItem);

**return true**;

}

});

}

**Entendendo o código:** Primeiramente utilizamos o **changeToFragment()** para deixar como tela inicial, a nossa tela **Inbox**. Em seguida implementamos o listener de seleção de um item do Navigation Drawer.

1. Agora que já criamos toda a lógica de navegação e configuração do nosso Drawer, chame o método **setupDrawerContent()** ao final do método **onCreate().**

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

setHomeButton(R.drawable.*menu*);

setupDrawerContent(navigationView);

}

1. Execute o código e teste o resultado.

## Resumo

Aprendemos nesta aula um pouco sobre Activities e Fragments e vimos suas diferenças. O Fragment é como um subview da Activity que possuem funcionalidades mais dotadas do que uma view comum, por exemplo, seu próprio ciclo de vida.

## Exercícios

## TDP