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 essas telas podem ser utilizados diversos recursos para navegação. Nesta aula você verá um pouco sobre os *Intents*, a *Toolbar*, os *TabLayouts* e a navegação por menu *Drawer*.

## *Intents*

Para transitar de uma tela para outra é necessário 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 um *Intent*.

Um *Intent* (intenção) é uma descrição abstrata de uma operação a ser executada. Ele pode ser utilizado 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. Um *Intent* faz parte da arquitetura do Android e trata-se de um conceito básico que deve ser dominado por todos que desejam programar para Android.

Sendo assim, em linguagem mais humana, um *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 será analisado o *app* **MailList**.

O projeto foi modificado 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 essa nova tela não terá funcionalidade nenhuma, mas em aulas futuras ela será utilizada como a tela de **lista de *e-mails*.**

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

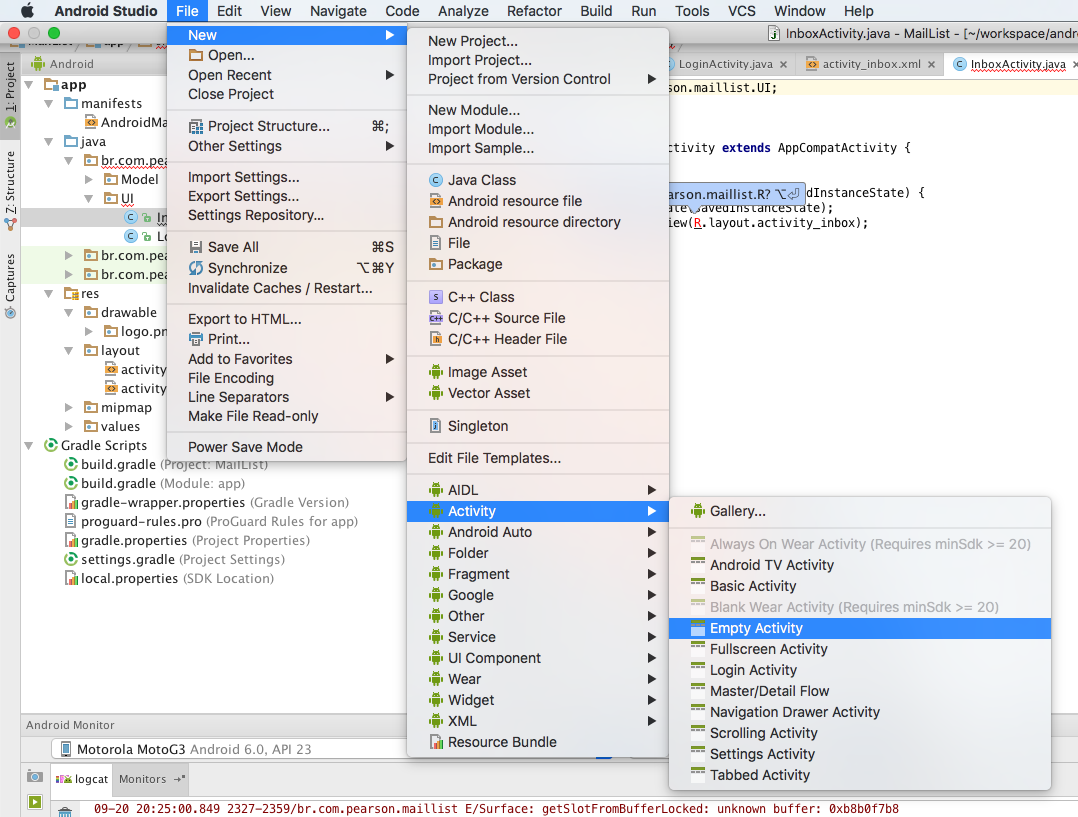


Figura - Criando uma nova Activity

1. A seguinte tela se abrirá:

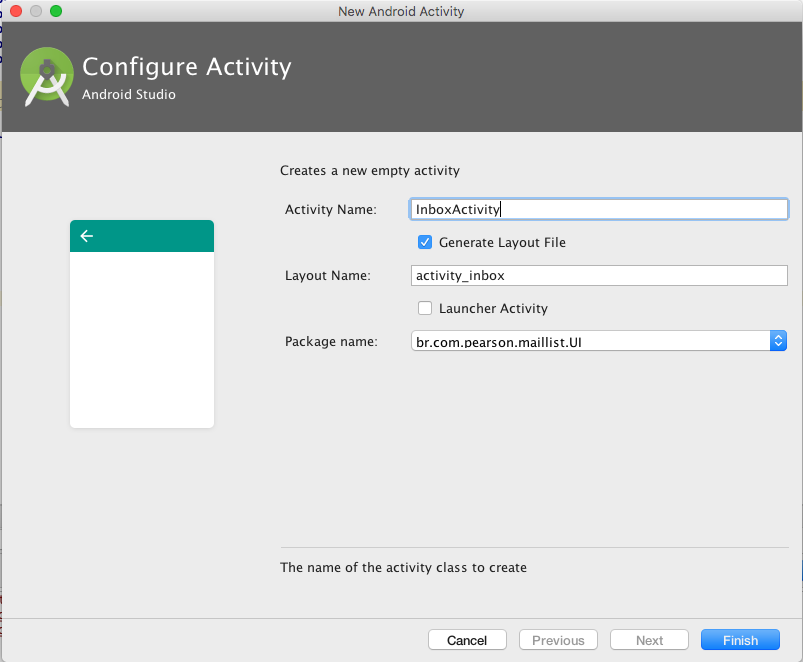


Figura - Configurações iniciais de uma nova Activity

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

Será colocada uma *TextView* no seu novo *layout* também. Esse passo serve apenas para identificar sua nova tela quando for realizada a transição via *Intent*.

Agora que suas telas foram criadas, você vai 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 esse método pelo 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 esse método é invocado.

1. Execute o *app* e veja o resultado.

Agora serão explicadas as novas linhas de código. Começando por:

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

Nessa linha está sendo inicializado o Intent em si. No primeiro parâmetro deve-se informar o contexto em que se está (classe atual - **this**) e, no segundo parâmetro, informar a sua classe destino.

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

Essa linha é simples. Como o próprio nome diz, ela inicia uma *Activity*, a qual está informada no *Intent* criado.

Existem inúmeras possibilidades usando os *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 você sabe como transitar entre telas, será ensinada mais uma técnica para melhorar a usabilidade do seu *app*. Será inserida a ação de voltar na sua *AppBar*.

## *ToolBar* e Menu *Overflow*

A *AppBar* é a barra superior na qual se encontra o título da *Activity* atual. Nela é possível inserir alguns botões que realizam certas ações. Esses botões podem ser colocados tanto no lado direito quanto no lado esquerdo do *app*. Uma *AppBar* com esses botões torna-se uma *ToolBar*.

Um ponto a se atentar é que no lado esquerdo deve-se colocar somente um botão. Geralmente esse botão é o botão “voltar” ou o chamado “menu sanduíche”. Já no lado direito, é possível inserir quantos botões forem necessários.

É claro que há uma limitação de espaço na sua *ToolBar*, então, se os botões não couberem nela, acontece o que é chamado de “menu *overflow*”, em que os botões com menos prioridade se agrupam em um menu representado pelo símbolo . Agora mãos à obra.

Veja como sua tela ficará:

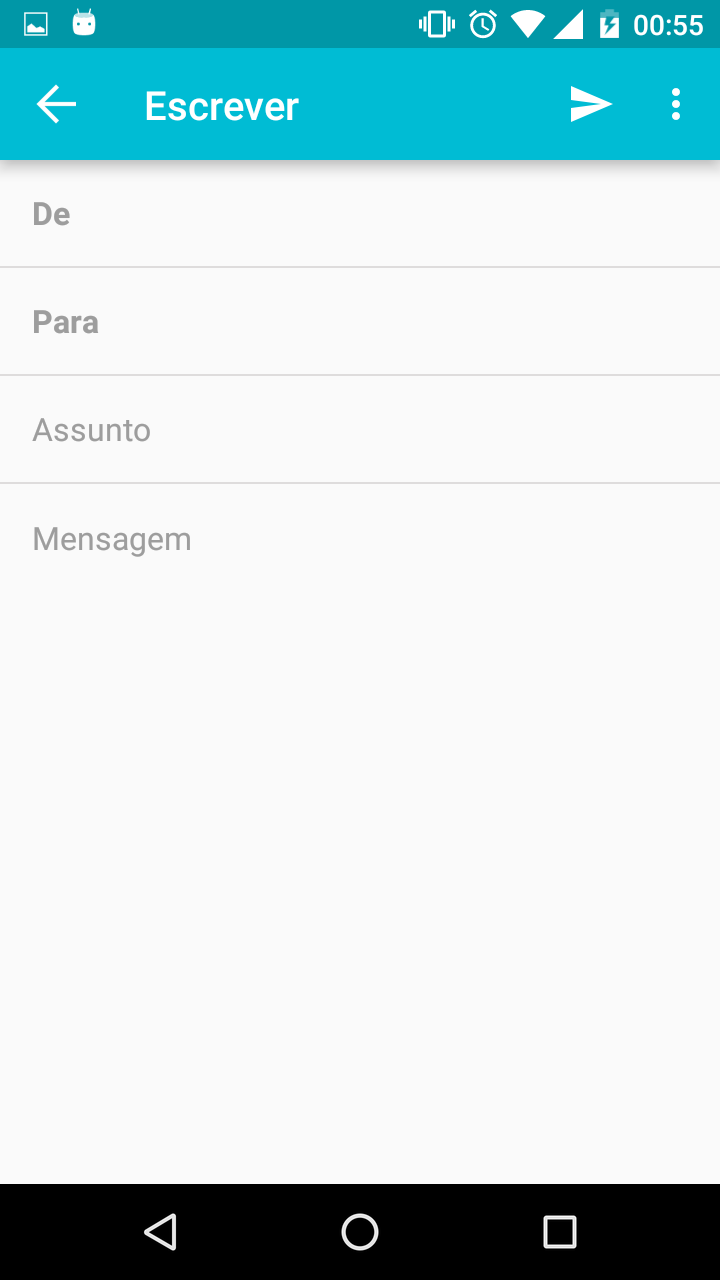


Figura - Tela **Escrever** com itens na barra superior

Novamente foi deixado preparado um projeto como ponto de partida para você iniciar esse exemplo. Abra o arquivo **Unidade\_4\_-\_Aula\_1\_-\_Exemplo2.zip.** Nesse projeto estão inseridas as mudanças feitas na aula anterior. Foi adicionada novamente a tela ComposeActivity. É nela que você trabalhará. Nesse exemplo será inserido o botão **Enviar** e o botão **Voltar.** Comece 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:** esse é o trecho que será utilizado para adicionar o botão *back* na sua *Activity*. Os métodos **getSupportActionBar()** são métodos da AppCompatActivity(que é a mãe da sua ComposeActivity)e eles nos proveem a barra de ações. Na sequência foi deixado o botão *home* (nome dado ao botão posicionado à esquerda na barra de ações) visível e habilitado, ou seja, ele receberá interação do usuário, e por fim foi configurada a imagem que aparecerá.

1. Agora chame o método criado ao final do método **onCreate().**  Ainda é preciso 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* fechará. Ele executa a ação de finalizar a *Activity*. Se essa tela tivesse sido aberta a partir de outra *Activity*, o botão ***back*** faria com que se voltasse para esta mesma *Activity*. Como nenhuma *Actvity* além da ComposeActivity foi aberta ainda, o *app* fechará.

Para testar a ação de voltar para outra *Activity*, execute os seguintes passos.

Para mudar a *Activity* inicial, é preciso 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 como na imagem abaixo:

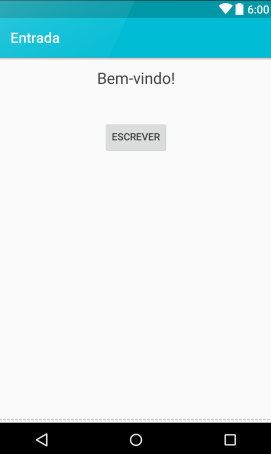


Figura - Tela com o botão de acesso à tela Compose

Não se esqueça de dar um id para ele (foi dado 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 à 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 você toca no botão ***back*** da ComposeActivity, o *app* retorna para a InboxActivity.

### Menu *Overflow*

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

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

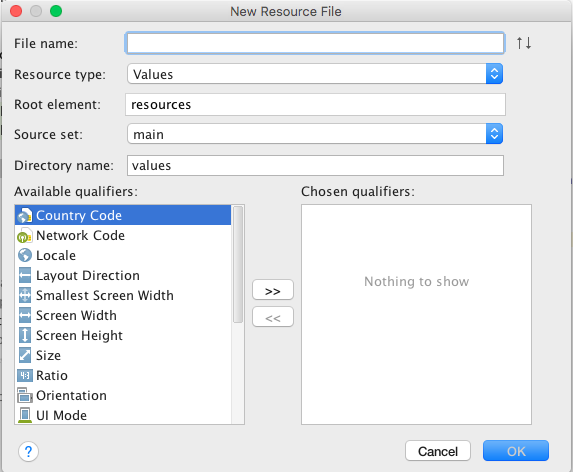


Figura 5 - Tela "New Resource File"

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. Esse é um arquivo que será utilizado 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:** esses itens mostrarão algumas opções no menu que está sendo criado. 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 orderInCategory,que determina a ordem que os itens aparecerão, sendo que quanto maior o número, mais à direita o item aparecerá. O seguinte atributo é o que determina quando o item aparecerá. Nesse caso foi determinado que o botão ***send*** sempreaparecerá 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:** esse é um método que infla o menu na sua *Activity*. O termo inflar é utilizado com o significado de injetar ou instanciar dentro. Utilizou-se o utilitário MenuInflater da sua *Activity* e nela foi inserido o **compose\_menu**.

1. Execute o código e veja o resultado.

Parece fácil, não é? Existem muitas outras possibilidades de menus que se pode implementar. Uma delas é o NavigationDrawer, um clássico menu lateral que você aprenderá a seguir.

## *Navigation Drawer*

O *NavigationDrawer* é um menu lateral bastante popular nos aplicativos. O nome *Drawer* vem de gaveta, pois pode-se utilizar um movimento para puxá-lo.

Será inserido um menu como esse no seu aplicativo MailList. Como ele está sendo baseado no *app* Gmail, você vai tentar criar um *Drawer* próximo deste:

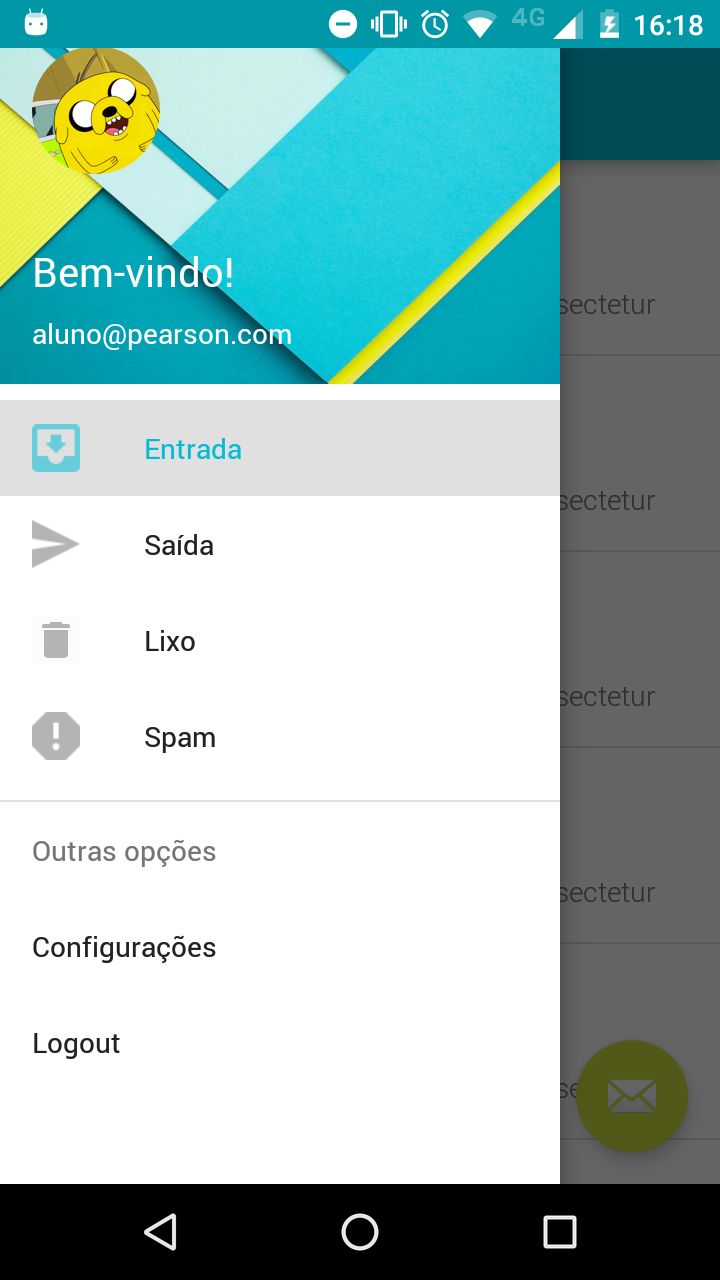


Figura - Menu proposto

1. Você criará alguns arquivos. Primeiro crie o arquivo **menu/drawer\_view.xml.** O procedimento de criação de arquivos está sendo resumido porque você já aprendeu isso em lições anteriores.
2. Substitua o código gerado desse 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 desses itens como selecionados por padrão. Basta adicionar o atributo android:checked="true" no item que deseja deixar pré-selecionado.

1. Uma vez que seu *Drawer* será deslizado por cima da sua *ActionBar*, é preciso criar sua própria *ActionBar*. A *ToolBar* pode ser embarcada dentro dos seus *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>

Nesse trecho de *layout* está sendo criada uma *Toolbar* básica para sua aplicação. Note que há o atributo fitsSystemWindows=true, sem ele a *status bar* (barra que mostra as notificações dos *apps*, bateria, hora etc.) ficaria por cima da *Toolbar*, dando um efeito translúcido. Esse atributo aumenta o *padding* do topo da *Toolbar* para que não haja essa 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 o *frame* da *view* (área da *view*) e o próprio conteúdo interno dela. A *margin* aumenta o espaçamento entre *views*.

1. Antes de criar mais arquivos, é preciso mudar o tema da sua aplicação, para que ele não insira uma *ActionBar*. Assim você pode utilizar a sua *Toolbar* tranquilamente. Para isso, 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:** nesse arquivo ***styles*** pode-se trocar o tema do aplicativo. Um tema vai desde uma combinação de cores até a adição ou remoção de alguns componentes gráficos. O tema selecionado faz com que o *app* não tenha uma *ActionBar* (ou *Toolbar*) para que você possa criar e usar sua própria *Toolbar*.

1. Agora que alteramos o tema do aplicativo, crie o arquivo **res/layout/activity\_main.xml**.
2. Nesse novo arquivo remova todo o código e insira o seguinte:

*<!-- 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" />

*<!-- É neste frame que seus 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 sua *Toolbar* foi injetada por meio do elemento ***include*.** Como não havia mais *Toolbar* devido ao tema selecionado no passo 5 e foi criada a sua própria *Toolbar*, é necessário, de certa forma, inseri-la no *layout*. O elemento *include* é responsável por injetar arquivos de *layout* em outro *layout*.

1. Agora é hora de criar sua *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 abrirá ou fechará 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:** agora é hora de entender o código que você acabou de inserir. Primeiro veja as instruções no método **onCreate():**

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

setSupportActionBar(toolbar);

Por meio desse comando é possível adicionar sua *toolbar* na sua *Activity*, funcionalmente. Agora veja o método **setHomeButton()** e perceba que ele é igual ao implementado na sua ComposeActivity, com a diferença de possuir um ícone diferente. Desta vez o ícone é o **menu**, pois não seria possível manter o botão ***back*** porque não faria sentido a real ação do *home button* que agora abrirá um menu e não voltará para a tela anterior.

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

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

1. Já está pronto para você o arquivo de *header*. Entre no arquivo **activity\_main.xml**  e 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 você está com o visual do seu menu pronto. Ele ficará como na figura abaixo:

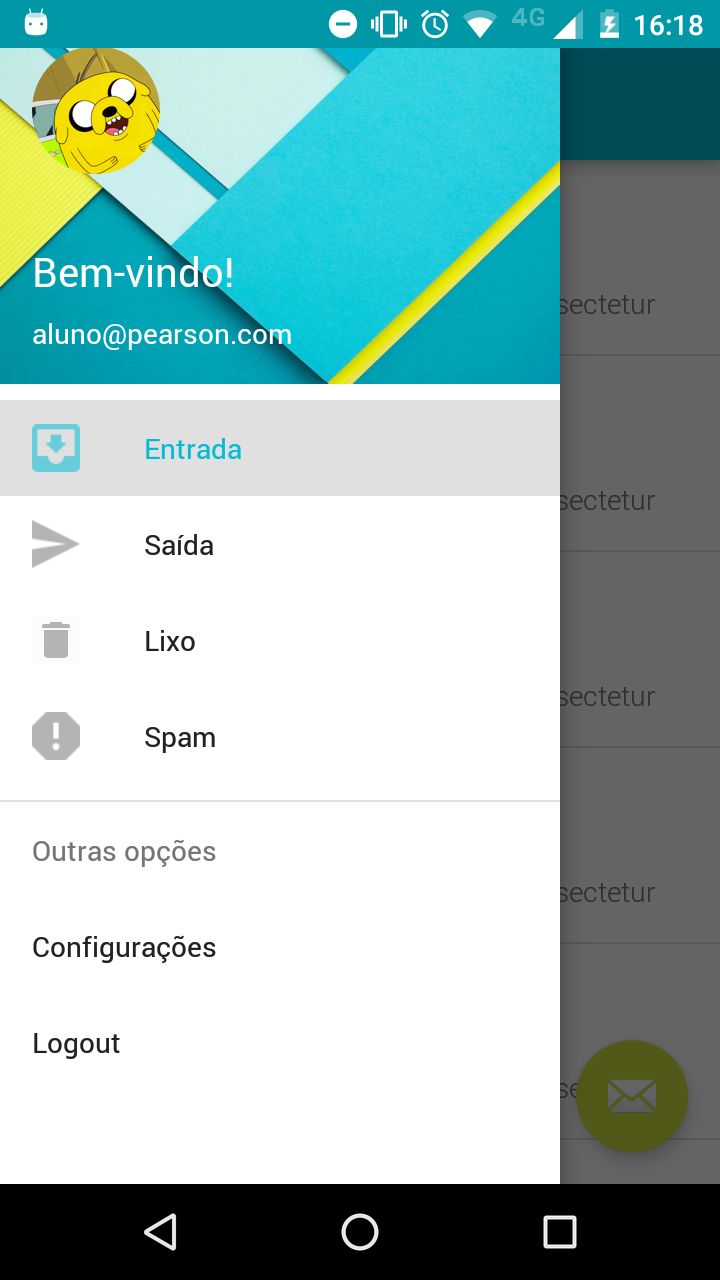


Figura - Menu pronto

**Obs**.: Essa imagem é ilustrativa. Se a sua tentativa não estiver idêntica a essa, não se preocupe. A intenção é apresentar a funcionalidade.

O visual do seu menu está pronto. Você conseguirá visualizá-lo clicando no ícone de menu no canto superior direito da tela do seu *app*. Confira a próxima aula, que falará sobre *Activities* e *Fragments* e na qual será finalizada a navegação via *Drawer*.

## Resumo

Você aprendeu algumas técnicas de navegação entre telas e entre elas a navegação via *Intents* e *Navigation Drawer*. Além disso, você viu como customizar sua *Toolbar* e otimizar as *Activities* para reuso, criando uma *Acitivity* base.

## Exercícios

## TDP

Aula 2

# *Activities* e *Fragments*

Depois de tanto usar nas aulas anteriores, finalmente você vai entender o que é uma *Activity*. Esta é uma das classes principais e que você terá mais contato no desenvolvimento Android. Ao lado de *Activities,* há os *Fragments*. Nesta aula você entenderá o que é uma *Activity* e um *Fragment* e vai saber diferenciá-los e como usá-los.

## *Activities*

A *Activity* é popularmente conhecida como uma tela do seu aplicativo. Em partes isso é correto, na arquitetura do Android, é utilizada uma *Activity* para uma tela. Na verdade, ela é a controladora da tela e fornece ações (atividades) nas quais o usuário pode interagir e fazer algo como mandar uma mensagem, escrever um *e-mail*, aplicar um filtro em uma foto etc. Geralmente a janela produzida por essa *Activity* preenche a tela do seu *smartphone*, mas é possível fazer telas menores flutuantes.

As *Activities* são pouco vinculadas entre si e isso evita o acoplamento, com exceção de casos muito específicos, nos quais 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*), o que será mais bem abordado futuramente.

Lembra-se de que foi definida uma *Activity* no início dos projetos? Esse tipo de *Activity* é especificado como “principal” e será apresentado ao usuário ao iniciar o aplicativo. Cada *Activity* pode então iniciar outras *Activities* para executar outras ações diferentes.

Quando se inicia uma nova *Activity*, ela é colocada no que é chamado de “pilha de retorno”. Essa pilha é uma estrutura de dados LIFO (*Last in*, *First out* ou **O último a entrar é o primeiro a sair**) que guarda a sequência e o estado das *Activities* instanciadas, dando então a possibilidade de usar a ação “Voltar” para desempilhar essas telas, ou melhor, voltar para a tela anterior. Após iniciar uma nova *Activity*, a atividade da anterior é suspensa. Será mais bem abordado sobre o ciclo de vida das *Activities* futuramente.

Quando uma atividade é interrompida devido ao início de uma nova, 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 em razão de 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 criar 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 criar uma subclasse, é preciso implementar o que é chamado de métodos de retorno de chamada que o sistema requere 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 desses métodos de retorno de chamada, na Unidade 5 foi preparada uma explicação sobre o *Template Method*, que é o padrão de projeto utilizado pela API do Android para esse 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 nesse método que se deve 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 podem ser usados 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 anexo 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-a. 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, implementar esses métodos de retorno de chamada? Uma *Activity* transita entre os diferentes estados citados de acordo com o uso do aplicativo. Quando o estado é alterado, seu objeto *Activity* é notificado por meio de alguns métodos. Esses 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 é possível interceptar essas mudanças apenas implementando esses métodos nas subclasses de *Activity*.

Essa operação de métodos abstratos é parecida com o **exemplo dos empregados** no livro de POO, no qual foi criada uma classe abstrata ***Employee*** que possuía um método abstrato **getBonus().** O bônus era calculado somente nas subclasses de ***Employee*.**  Quando era calculada 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 vai entender com mais fluidez, mas por enquanto você vai 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 anterior.

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 o pré-processamento, a inicialização de recursos, definir o *layout* e outras coisas “globais” do funcionamento dessa ***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 esse tempo pode-se ver a ***Activity*** na tela e consequentemente é possível interagir com ela durante esse período.
* **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 a seguir 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 - Ciclo de Vida de uma Activity

A figura acima demonstra o fluxo que o aplicativo opera e é possível 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()**, pois esse caminho não faz parte do ciclo.

Daqui em diante, você pode notar que haverá a necessidade de repetir muitos códigos entre as *Activities*. Por exemplo, na aula passada, foi criado o botão ***back*.** Existirá a necessidade de inserir esse botão em diversas telas daqui para a frente, então será utilizada uma boa prática que é criar uma **BaseActivity** que servirá como *Activity* base para evitar repetição de código, que é reutilizado entre as *Activities*.

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

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

1. Lembra-se de que foi criada uma *Toolbar* na aula anterior? Você vai 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 está o seu primeiro código que gerará repetição entre as *Activities*, a inicialização do **BetterKnife**.

1. Nesse mesmo método você vai configurar sua *Toolbar*, chamando o seguinte método:

@Override

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

**super**.setContentView(layoutResID);

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

setSupportActionBar(**toolbar**);

}

1. Agora você vai 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);

}

}

Nesse caso, está sendo criado um botão com imagem dinâmica, ou seja, pode-se especificar o *drawable* desejado ao chamar o método.

1. Após inserir um botão *back*, dê 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 seu **onCreate()**, pois nem todas as *Acitivities* terão esse botão.

Agora você tem a sua *Activity* base. Nos exemplos seguintes você vai refatorar aos poucos suas *Activities* e verá o efeito dessa nova classe.

## *Fragments*

Agora que você conhece as *Activities*, será apresentado 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*. Pode-se 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, pode-se utilizá-lo em diversos lugares e aproveitar o poder da reusabilidade que o ***Fragment*** 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, independentemente do tamanho da tela. O *Fragment* poderia resolver esse problema para você.

Como uma *Activity*, o *Fragment* também possui seu próprio ciclo de vida, como ilustrado a seguir:



Figura - Ciclo de vida de um Fragment

Seus métodos são bem similares aos de uma *Activity*, então detalhes não serão abordados.

Agora você verá um exemplo prático.

Imagine que você queira colocar um menu lateral (chamado de *Drawer*) no seu *app* MailList. Olhando o comportamento do Gmail (aplicativo que inspira o seu MailList), vê-se que esse menu fica oculto na lateral esquerda e é mostrado quando se toca no respectivo ícone da AppBar.

Se executado o Gmail em um *tablet*, esse menu lateral fica visível em tempo integral, pois existe espaço hábil para isso.

Veja o esboço do *layout* dessas duas situações:



Figura - Esboço de aplicação de Fragments em múltiplas telas

### Formas de resolver esse problema

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

Você também deveria ter uma *Activity* que realizasse 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 como proceder quando um item do menu for clicado.

No final das contas, você teria uma *Activity* de código grande e de difícil manutenção. Usando *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

Você vai retomar seu aplicativo MailList. Abra o arquivo **Unidade\_4\_-\_Aula\_2\_-\_Exemplo\_1.zip.**

O objetivo agora é fazer a troca das telas, para isso é preciso deixar de usar as *Activities* em algumas situações e trocá-las pelos *Fragments*. Agora você fará a troca de telas por meio do *Navigation Drawer*.

1. Entre na sua **MainActivity.java** e a primeira coisa que deve ser feita é fazer com que sua *Activity* estenda a **BaseActivity.**
2. Otimize seu 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:

setHomeButton(R.drawable.***menu***);

1. Otimizações feitas, é hora de inserir as funcionalidades para manipulação dos *fragments*. Primeiro substitua 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 é dito como 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 você tem seu *Fragment* para a tela *Inbox*, crie outro *Fragment* qualquer para mostrar. Nesse exemplo foi criado um *Fragment* para a tela “Saída” com um texto simples apenas para nível de demonstração, mas certifique-se de ter pelo menos dois *Fragments* criados.
2. Volte para a **MainActivity**. Você vai inserir as funcionalidades faltantes para conseguir transitar entre telas. Primeiro crie 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:** esse procedimento é parecido com o procedimento de troca de *Activity* via *Intents* visto na aula anterior. Para os *Fragments,* há o FragmentManager, que é responsável pelo gerenciamento dos *Fragments*.

1. Agora você vai resgatar seus *Fragments* por meio 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:** nesses métodos há uma simples instanciação dos métodos, com a diferença de que é trocado o título da sua MainActivity de acordo com o *fragment* instanciado.

1. É necessário então criar um método que decida qual *Fragment* instanciar, de acordo com o item do menu selecionado. Para isso será utilizado 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:** esse método será chamado sempre que um item do menu *Drawer* for selecionado, então por parâmetro é recebido um item do menu, e de acordo com ele é instanciado o *Fragment* correto. Em seguida, é utilizado o método setChecked() para manter o item selecionado com uma coloração de destaque. Depois modifica-se o título da tela e, por fim, fecha-se o *Drawer.*

1. Agora você vai utilizar todos esses métodos criados. Os itens do menu também possuem *listeners* que são disparados quando se clica em um deles, mas infelizmente a biblioteca ButterKnife não dá suporte para esse *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 foi utilizado o **changeToFragment()** para deixar como tela inicial a sua tela ***Inbox***. Em seguida, implementou-se o *listener* de seleção de um item do *Navigation* *Drawer*.

1. Agora que você já criou toda a lógica de navegação e a configuração do seu *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

Nesta aula você aprendeu um pouco sobre *Activities* e *Fragments* e viu suas diferenças. O *Fragment* é como uma *subview* da *Activity* que possue funcionalidades mais dotadas do que uma *view* comum, por exemplo, seu próprio ciclo de vida.

## Exercícios

## TDP