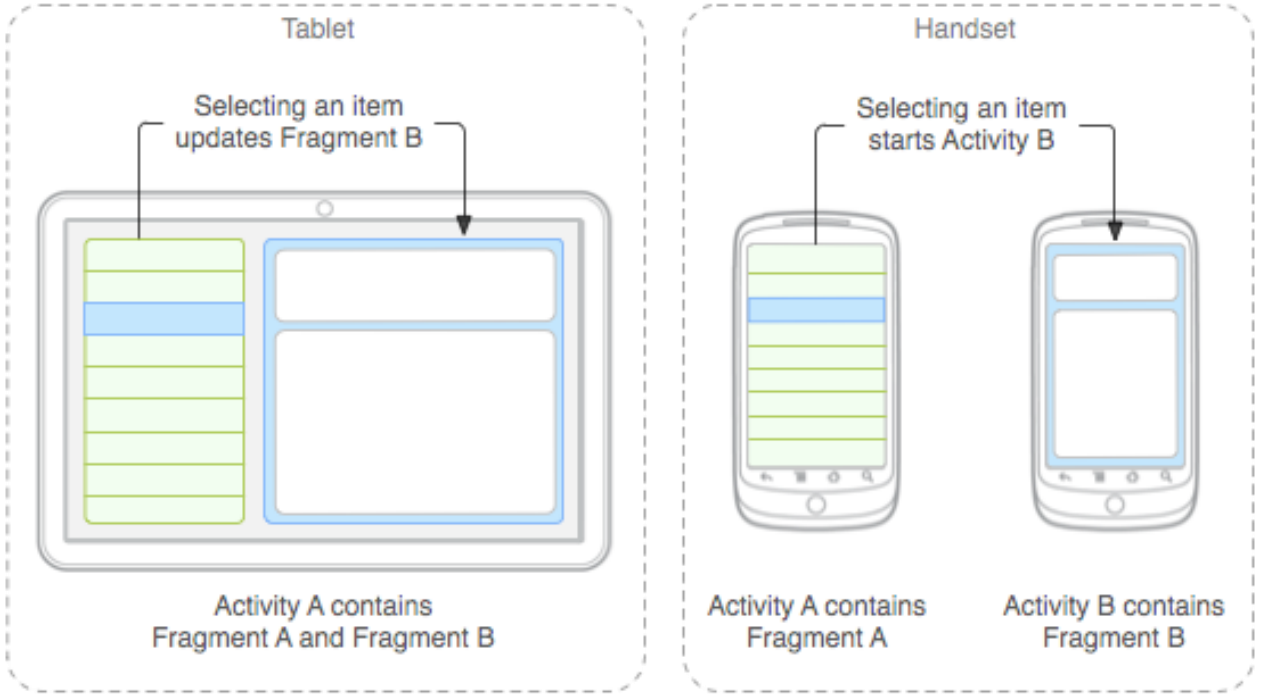
Desenvolvimento Móvel em Android

Fragmentos e NavigationView

Mário Viana, mario.viana@ipleiria.pt
Sónia Luz, sonia.luz@ipleiria.pt
David Safadinho, david.safadinho@ipleiria.pt
Departamento de Engenharia Informática
Escola Superior de Tecnologia e Gestão
Instituto Politécnico de Leiria
1º Semestre - 2021/2022

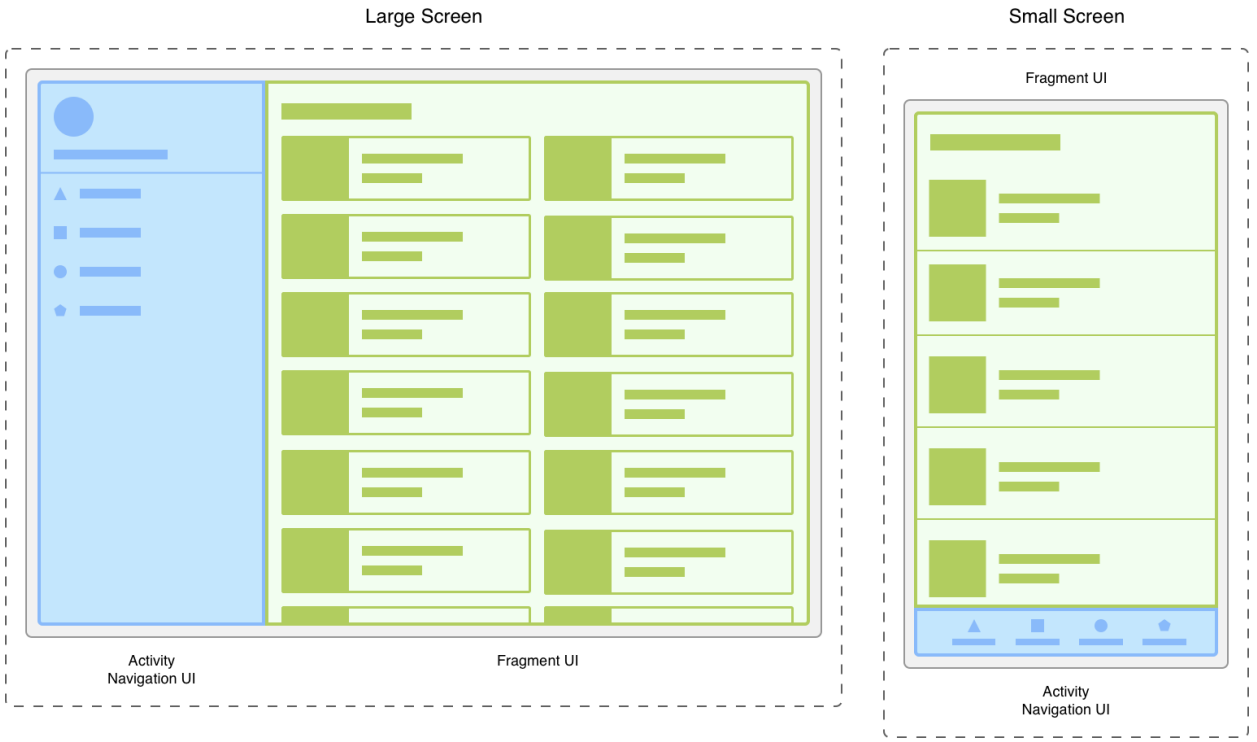
Fragmentos – Conceito

- Um fragmento é uma classe reutilizável que implementa uma porção de uma atividade
  - Tipicamente, define uma parte da interface;
- Os fragmentos devem ser incorporados em atividades;



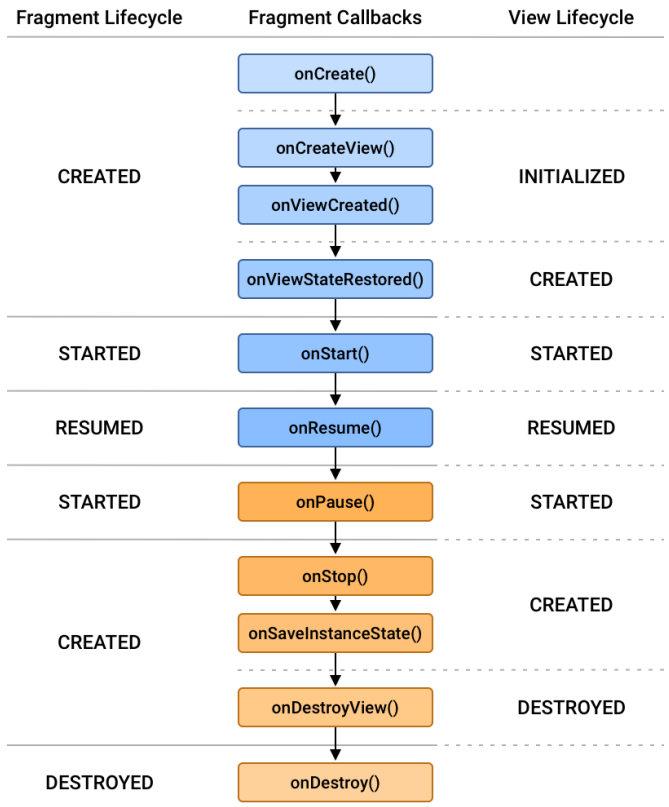
Fragmentos – Conceito

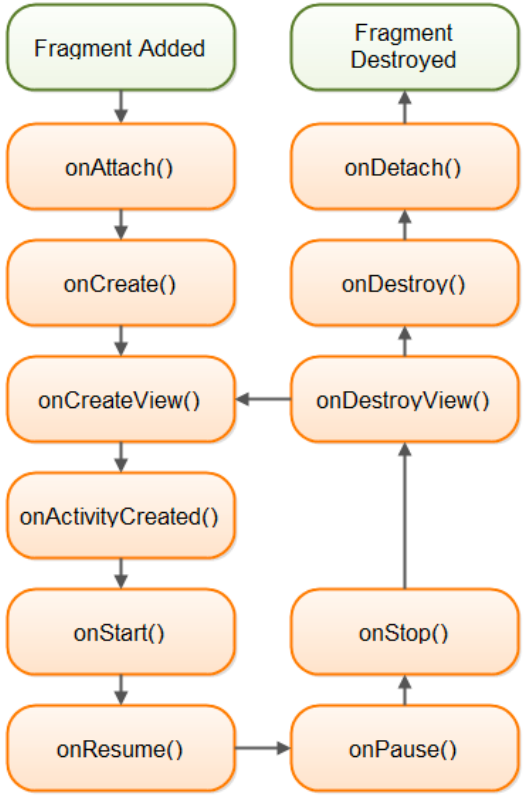
- Os fragmentos não podem ser executados independentemente;
- Contudo, um fragmento é um objeto independente de uma atividade
  - Podendo ser usado por múltiplas atividades;

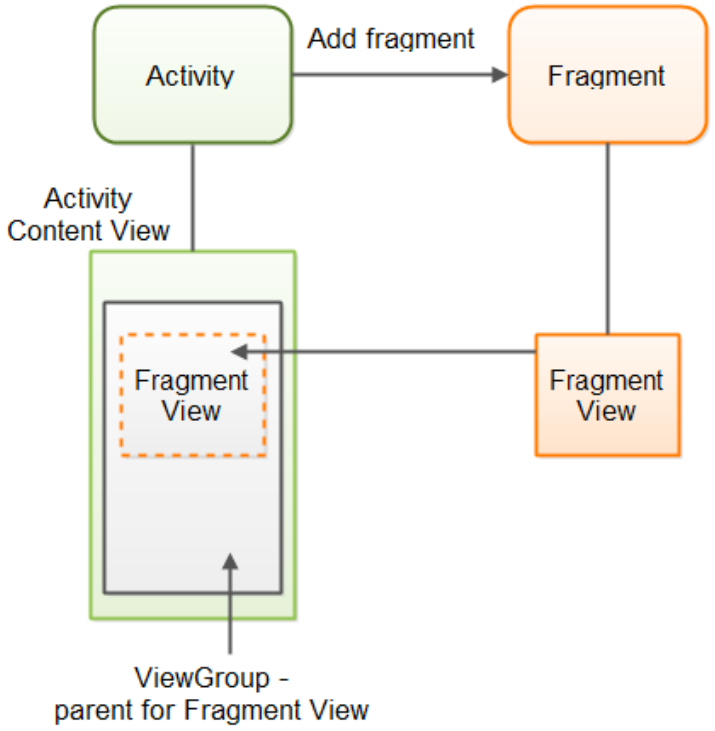


Fragmentos – Ciclo de Vida

- O fragmento possui o seu próprio ciclo de vida
  - mas este depende sempre do ciclo de vida da sua atividade







Fragmentos – Ciclo de Vida

- **Métodos envolvidos na <u>criação</u> de um fragmento:**
  - ***onAttach(Activity)*** – chamado após o fragmento se associar à atividade;
  - ***onCreate(Bundle)*** - chamado após a criação do fragmento (ideal para inicializar componentes);
  - ***onCreateView(LayoutInflater, ViewGroup, Bundle)*** – chamado quando a interface do fragmento é desenhada; devolve o objeto *View* ligado ao layout (*inflate*);
  - ***onViewCreated(View, Bundle)*** – chamado logo após o desenho da interface (ideal para restaurar qualquer estado adicional associado ao interface do fragmento);
  - ***onStart()*** – chamado quando o fragmento fica visível;
  - ***onResume()*** – chamado antes do utilizador poder interagir com o fragmento.

Fragmentos – Ciclo de Vida

- **Métodos envolvidos na destruição de um fragmento:**
  - ***onPause( )*** – chamado quando o utilizador já não pode interagir (altura para tratar da persistência de dados);
  - ***onStop( )*** – chamado quando o fragmento já não está visível;
  - ***onSaveInstanceState(Bundle)*** - chamado logo após ao *callback onStop* (ideal para guardar os valores do estado atual do fragmento);
  - ***onDestroyView()*** – chamado quando a *View* associada ao fragmento é destruída (altura adequada para libertar recursos);
  - ***onDestroy( )*** – Chamado quando o fragmento já não está em uso (altura adequada para libertar recursos utilizados pelo fragmento);
  - ***onDetach( )*** – chamado quando o fragmento deixa de estar associado à atividade;

Fragmentos – Incorporar numa Activity

- As *Activities* que acolhem fragmentos devem estender as classes ***FragmentActivity*** ou ***AppCompatActivity***

  (esta última dá suporte para fragmentos para todas as versões)

- A incorporação pode ser efetuada de forma:
  - **Estática**: declaração do fragmento dentro do layout da *Activity*;
  - **Dinâmica**: adicionar o fragmento em tempo de execução;

Fragmentos – Incorporar numa Activity

- Incorporação estática (através dos views ***Fragment*** ou ***FragmentContainerView***):

```
<fragment
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name=”com.fragmentosapp.MyFragment"
    android:id="@+id/list"
/>
```

```
<androidx.fragment.app.FragmentContainerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="com.com.fragmentoapp.MyFragment”
    android:id="@+id/list"
 />
```

Fragmentos – Incorporar numa Activity

- A incorporação dinâmica contempla:
  1. Incluir, no layout da *Activity*, um *ViewGroup* (por ex. *FrameLayout ou FragmentContainerLayout*) para acolher o fragmento:

```
<LinearLayout ...”>
    <FrameLayout
        android:id=“@+id/placeholder” ... />
</LinearLayout>
```

```
<LinearLayout ...”>
    <androidx.fragment.app.FragmentContainerView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/placeholder"/>
</LinearLayout>
```

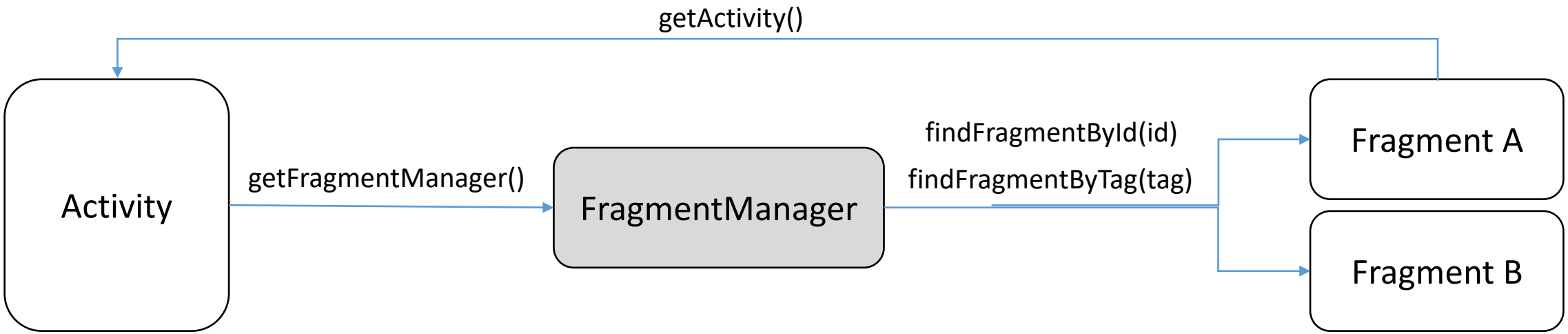
Fragmentos – Incorporar numa Activity

2. Implementar, na classe da *Activity*, o código para adicionar (*add*), substituir (*replace*) ou eliminar (*delete*) um fragmento ao *ViewGroup* definido no layout:

```
// Obter uma instância do FragmentManager através do método getSupportFragmentManager
FragmentManager fragmentManager = getSupportFragmentManager();

// instanciar um objeto FragmentTransation através de FragmentManager
FragmentTransacttion ft = fragmentManager.beginTrasaction();

// invocar métodos de incorporação de fragmentos (add, replace, remove)
ft.add(R.id.placeholder, new MyFragment());

// aplicar as alterações previamente definidas
ft.commit();
```

Fragmentos – Obter Instâncias

- Um fragmento pode aceder à instância da *Activity* e vice-versa



Fragmentos – Obter Instâncias

- O acesso à *Activity* através do fragmento é efetuado através do método ***getActivity( )***,
  - Podendo, assim, realizar as suas tarefas ou aceder aos seus objetos *View*:

```
View listview = getActivity().findViewById(R.id.list);
```

Fragmentos – Obter Instâncias

- Por seu lado, o acesso a uma instância de um fragmento pode ser realizado através:

1. Do método *findFragmentById()* se o fragmento foi definido **estaticamente** no layout:

```
DemoFragment fragmentDemo = (DemoFragment)
        getSupportFragmentManager().findFragmentById(R.id.fragmentDemo);
```

2. Do método *findFragmentByTag()* se o fragmento foi adicionado **dinamicamente**:

```
// adição dinâmica de um fragmento a um contentor
getSupportFragmentManager().beginTrasaction()
        .replace(R.id.container, new DemoFragment(), "MY_TAG").commit();
//obtenção da instância do fragmento por tag
DemoFragment fragmentDemo = (DemoFragment)
        getSupportFragmentManager().findFragmentByTag("MY_TAG");
```

Fragmentos – Comunicação

- Os fragmentos, como componentes modulares e reutilizáveis
  - Não comunicam uns com os outros diretamente, mas através da mediação da *Activity*;
- Os dois métodos de comunicação mais populares são efetuados através de:
  - ***Bundle*** – a atividade pode criar um fragmento e definir os seus argumentos;
  - ***Métodos*** – a atividade pode invocar métodos da instância do fragmento;

Fragmentos – Comunicação (**Bundle**)

- **Bundle** permite, na criação do fragmento, enviar argumentos através de um método estático (por ex. `newInstance`);

- Esta prática implica:

  1. A criação de um método estático que: 1) recebe dados em argumentos: 2) cria uma instância do fragmento; 3) cria uma instância da classe *Bundle*; 4) adiciona os argumentos ao objeto *Bundle*; 5) associa o *Bundle* à instância do fragmento e 6) devolve-o;

  2. A obtenção, no método **`onCreate()`** do fragmento, dos argumentos do fragmento através do método **`getArguments()`**;

  3. O carregamento do fragmento de forma dinâmica na atividade, recorrendo ao método estático criado.

Fragmentos – Comunicação (**Bundle**)

```
public class DemoFragment extends Fragment{
    @Override
    public void onCreate(Bundle b){
        ...
        //obter argumentos
        int myint = getArguments().getInt("someInt", 0);
        String myString = getArguments().getString("someString", "");
    }

    public static DemoFragment newInstance(int myInt, String myString){
        DemoFragment fragmentDemo = new DemoFragment();
        Bundle args = new Bundle();
        args.putInt("someInt", myInt);
        args.putString("someString", myString);
        fragmentDemo.setArguments(args);
        return fragmentDemo;
    }
}

//código na Activity
FragmentTransacttion ft = getSupportFragmentManager().beginTrasaction();
DemoFragment fragmentDemo = DemoFragment.newInstance(3, "Hello fragment!");
ft.add(R.id.placeholder, fragmentDemo);
ft.commit();
```

Fragmentos – Comunicação (**Bundle**)

- **Outra forma** da utilização de um **Bundle** é defini-lo como argumento dos métodos *add*, *replace* ou *delete,* no momento do *FragmentManager* iniciar uma transação:

```
public class ExampleActivity extends AppCompatActivity {
    public ExampleActivity() {
        super(R.layout.example_activity);
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState == null) {
            Bundle bundle = new Bundle();
            bundle.putInt("some_int", 0);

            getSupportFragmentManager().beginTransaction()
                .setReorderingAllowed(true)
                .add(R.id.fragment_container_view, ExampleFragment.class, bundle)
                .commit();
        }
    }
}
```

Fragmentos – Comunicação (**Bundle**)

- Do lado do fragmento é possível obter os dados enviados no callBack ***onViewCreated*** através do método ***requireArguments***:

```
class ExampleFragment extends Fragment {
    public ExampleFragment() {
        super(R.layout.example_fragment);
    }

    @Override
    public void onViewCreated(@NonNull View view,
                              Bundle savedInstanceState) {
        int someInt = requireArguments().getInt("some_int");
        ...
    }
}
```

Fragmentos – Comunicação (**Métodos**)

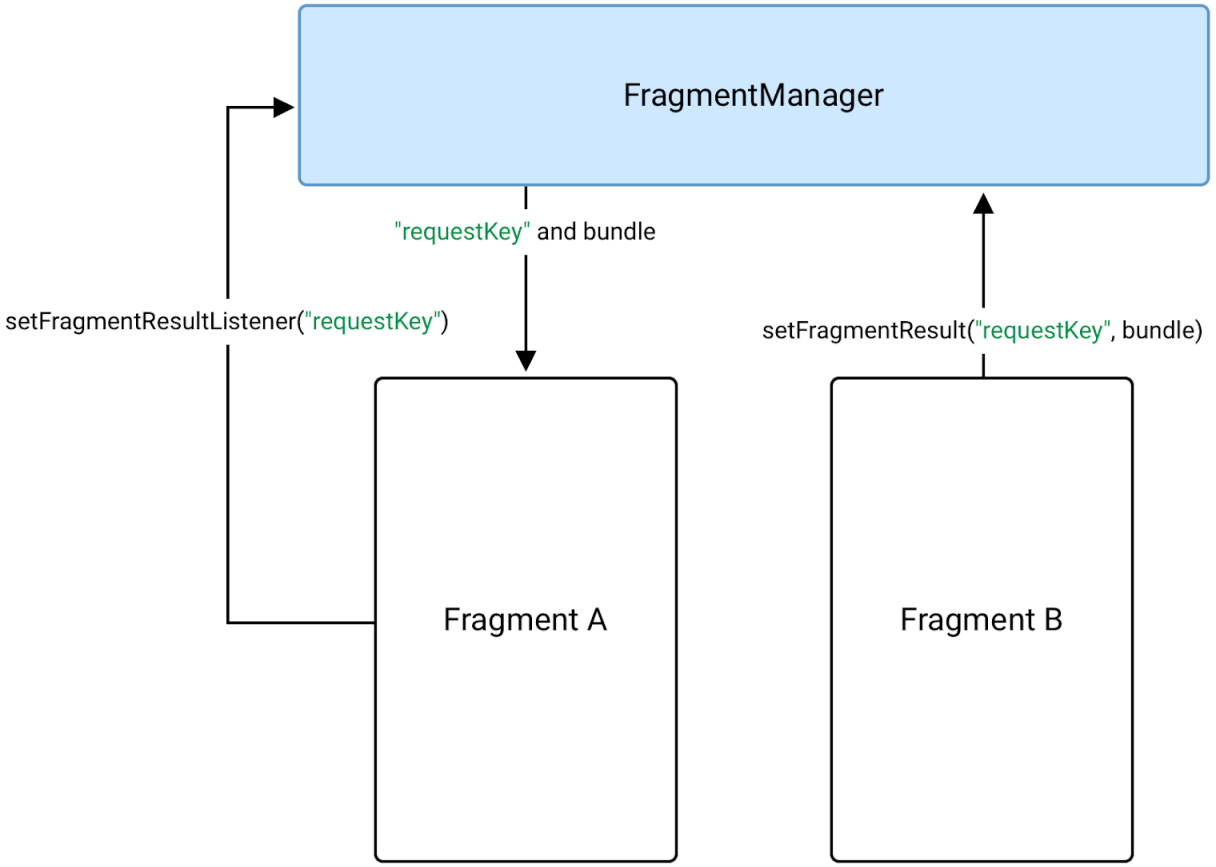
- Permite à *Activity* executar ações no fragmento
  - Invocando os seus métodos através da referência à instância do fragmento;
- Esta prática implica:
  1. A criação dos métodos na classe do fragmento;
  2. A obtenção, na *Activity*, da referência da instância do fragmento;
  3. A invocação dos métodos do fragmento através da referência obtida.

Fragmentos – Comunicação (**Métodos**)

```
public class DemoFragment extends Fragment{
    @Override
    public void doSomething(String param){
        // código
    }
}


//código na Activity
public class MainActivity extends AppCompatActivity{
    @Overrride
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        DemoFragment fragmentDemo = (DemoFragment)
                getSupportFragmentManager().findFragmentById(R.id.fragmentDemo);
        fragmentDemo.doSomething("Hello fragment!");
}
```

Fragmentos – Comunicação entre Fragmentos

- É possível enviar informação para outros fragmentos. Uma das formas é através do *FragmentManager* da atividade mãe
  - Nota: a comunicação entre fragmentos nunca é realizada diretamente. Recorre à atividade que a integra ou a recursos por ela disponibilizados



Fragmentos – Comunicação entre Fragmentos

- O fragmento que pretende enviar uma informação, deve instanciar um ***Bundle*** e enviá-lo através do método ***setFragmentResult*** disponibilizado pelo *FragmentManager* da atividade integradora, obtido através do método ***getParentFragmentManager*** *:*

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Bundle result = new Bundle();
        result.putString("bundleKey", "result");
        getParentFragmentManager().setFragmentResult("requestKey", result);
    }
});
```

Fragmentos – Comunicação entre Fragmentos

- Por sua vez, no fragmento recetor deve definir-se um *listener,* através do método ***setFragmentResultListener***, para obter a informação disponibilizada através do *FragmentManager* da atividade integradora:

```
@Override
public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getParentFragmentManager()
         .setFragmentResultListener("requestKey", this,
                                    new FragmentResultListener() {
        @Override
        public void onFragmentResult(@NonNull String requestKey,
                                     @NonNull Bundle bundle) {
            String result = bundle.getString("bundleKey");
            // Do something with the result
        }
    });
}
```

Fragmentos – Guardar o Estado

- Em diversas circunstâncias é importante salvar um conjunto de valores que definem o estado do fragmento e recuperá-los no momento da sua reativação.
  - Esta operação implica a reescrita do método *onSaveInstanceState*, recorrendo a um Bundle que o método fornece como argumento:

```
@Override
public void onSaveInstanceState(@NonNull Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putBoolean(IS_EDITING_KEY, isEditing);
    outState.putString(RANDOM_GOOD_DEED_KEY, randomGoodDeed);
}
```

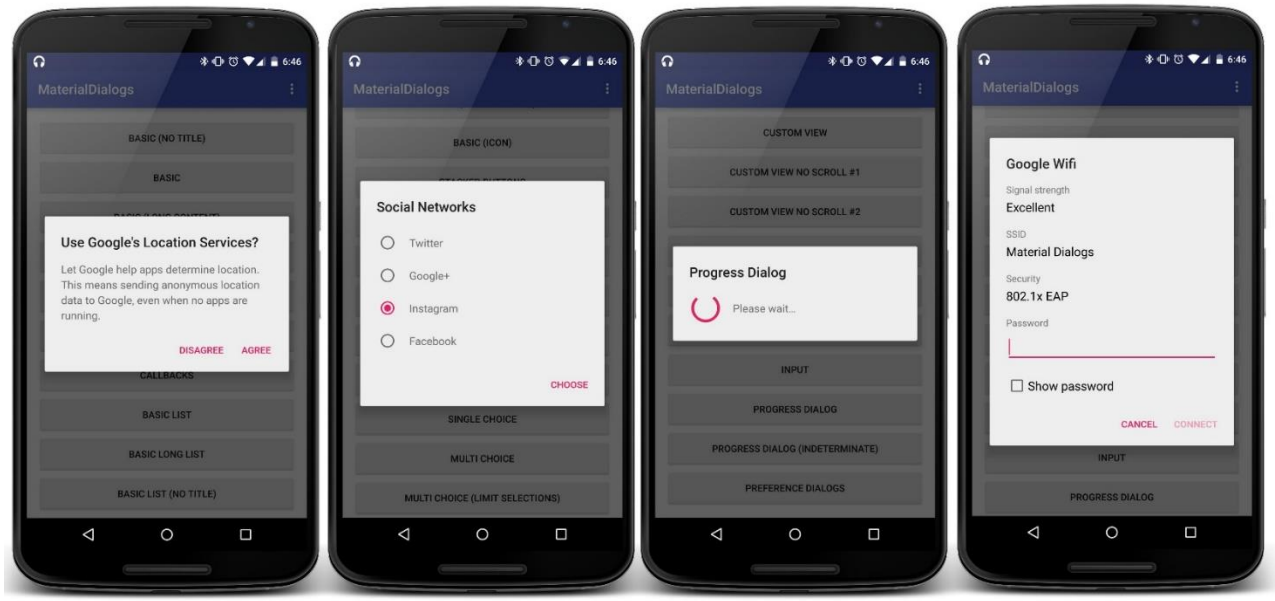
Fragmentos – Guardar o estado

- A recuperação dos valores guardados podem ser recolhidos no métodos *onCreate*, *onCreateView* e *onViewCreated* do fragmento:

```
@Override
public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (savedInstanceState != null) {
        isEditing = savedInstanceState.getBoolean(IS_EDITING_KEY, false);
        randomGoodDeed = savedInstanceState.getString(RANDOM_GOOD_DEED_KEY);
    } else {
        randomGoodDeed = viewModel.generateRandomGoodDeed();
    }
}
```

Fragmentos – DialogFragment

- Um *DialogFragment* é um fragmento específico que se sobrepõe à *Activity*
  - Funcionando como uma *modal window*;
- É utilizado para construir caixas de diálogo personalizadas;



Fragmentos – DialogFragment

- A criação de uma caixa de diálogo através de fragmentos
  - É realizada de forma idêntica à construção de fragmentos
  - Contudo, a classe implementada deve herdar a classe ***DialogFragment***;

Fragmentos – DialogFragment

- O processo de construção implica:
  - Criar o layout da caixa de diálogo;

my_dialog_fragment.xml

```
<LinearLayout ...
    android:orientation="vertical”>

    <TextView ...
        android:text=”Confirma?..."/>

    <LinearLayout ....
        android:orientation="horizontal">
        <Button ...
            android:text="Não"
            android:id="@+id/botao_nao" />
        <Button ..."
            android:text="Sim"
            android:id="@+id/botao_sim"/>
    </LinearLayout>
</LinearLayout>
```

Fragmentos – DialogFragment

- Implica, ainda:
  - Implementar uma classe que herda a classe *DialogFragment*;
  - Reescrever o método *onCreateView*() onde:
    1. se associe o layout (***inflate***);
    2. se obtenham as referências aos objetos do layout (método ***view.findViewById***);
    3. se defina e programe os ***listeners*** necessários das *Views*;
    4. se defina o tipo *modal/não modal* (método ***setCancelable***);
    5. se defina os pontos de encerramento da caixa de diálogo (método ***dismiss***)

MyDialogFragment.java

```
public class MyDialogFragment extends DialogFragment
                implements View.OnClickListener{
  @Override
  public View onCreateView(LayoutInflater li, ViewGroup vg, Bundle b){
    View view = li.inflate(R.layout.my_dialog_fragment, null);

    Button btnNao = (Button) view.findViewById(R.id.botao_nao);
    Button btnSim = (Button) view.findViewById(R.id.botao_sim);
    btnNao.setOnClickListener(this);
    btnSim.setOnClickListener(this);
     setCancelable(false);
     return view;
  }

  @Override
  public void onClick(View view) {
    if(view.getId() == R.id.botao_sim)
        Toast.makeText(getActivity(),
            "yes clicked",Toast.LENGTH_SHORT).show();
    if(view.getId() == R.id.botao_nao)
        Toast.makeText(getActivity(),
            "no clicked",Toast.LENGTH_SHORT).show();
    dismiss();
  }
}
```

Fragmentos – Chamar DialogFragment

- A apresentação da caixa de diálogo é executada através
  - De uma instância de *FragmentManager* e do seu método *show()*.

```
FragmentManager fm = getFragmentManager();
MyDialogFragment mdf = new MyDialogFragment ();
mdf.show(fm, "TAG");
```

DialogFragment (Comunicação)

- A comunicação entre a *DialogFragment* e a *activity*
  - Pode efetuar-se através de uma interface interna;
- A sua implementação compreende do lado da *DialogFragment*:
  - A criação de uma interface interna;
  - A declaração de um atributo da classe que guarde a referência à atividade que implementa a interface;
  - A reescrita do método *onCreate()* para associar a *Activity* à interface;

```
public class MyDialogFragment extends DialogFragment
                    implements View.OnClickListener {
    private Button btnYes, btnNo;
    //referencia para o interface
    private Communicator communicator;

    @Override
    public void onCreate(Bundle b) {
        super.onCreate(b);
        communicator = (Communicator) getActivity();
    }

    @Override
    public View onCreateView(LayoutInflater li,
                        ViewGroup vg, Bundle b) { …  }

    @Override
    public void onClick(View view) { … }

    interface Communicator{
        public void onDialogMessage(String message);
    }
}
```

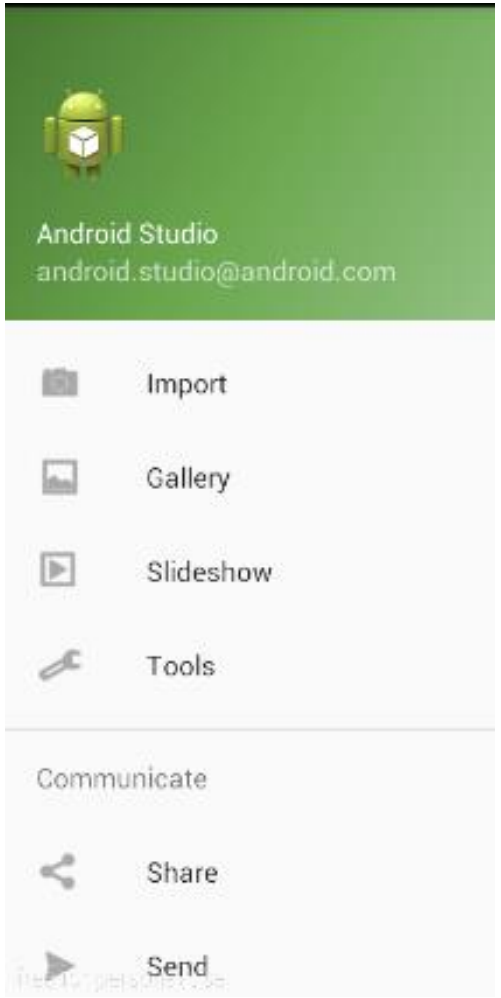
DialogFragment (Comunicação)

- A sua implementação compreende do lado da *Activity*:
  - Implementar a interface criada;
  - Reescrever o método referenciado na interface.

```
public class MainActivity extends AppCompatActivity
                    implements MyDialogFragment.Communicator {
    private TextView txtResponse;

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

    public void callDialog(View view) {
        FragmentManager fm = getFragmentManager();
        MyDialogFragment myDialogFragment=
                    new MyDialogFragment();
        myDialogFragment.show(fm,"Mydialog");
    }

    @Override
    public void onDialogMessage(String message) {
        txtResponse.setText(message);
    }
}
```

Navigation Drawer

- O *Navegation Drawer* é um padrão muito utilizado em aplicações modernas
  - Que a *Design Support Library* simplificou ao disponibilizar o componente ***NavigationView***;
- O componente *NavigationView*, a incluir num *DrawerLayout*
  - permite exibir itens a partir de um ficheiro menu;
- O Android Studio já disponibiliza um tipo de atividade que implementa um *Navigation Drawer*
  - a partir da qual é *simples* adaptar os ficheiros gerados;
- A *NavigationView* deve ser implementada conjuntamente com uma *Toolbar*;



Navigation Drawer

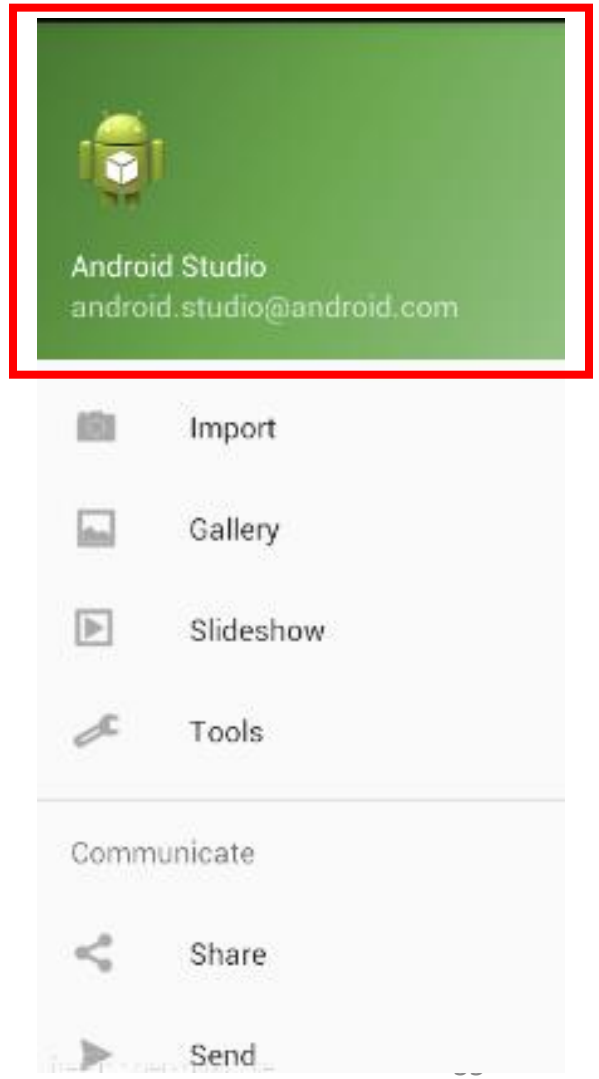
- O processo de implementação implica:
  - Incluir no **build.gradle** as dependências de compilação:
    `implementation 'com.google.android.material:material:1.4.0'`
  - Construir o **recurso de menu** (por ex., *navigation_menu.xml*);

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:showIn="navigation_view">

    <group
        android:checkableBehavior="single">
        <item
            android:id="@+id/nav_lista"
            android:icon="@drawable/ic_action_lista"
            android:title="@string/txtItemLista" />
        <item ... />
        <item ... />
    </group>
</menu>
```

Navigation Drawer

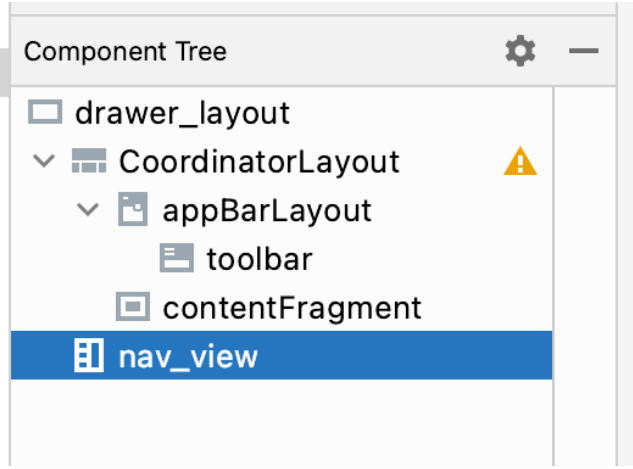
- Construir o **recurso de layout** para o cabeçalho do *NavigationView* (por ex., *navigation_header.xml*);

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#009688"
    android:orientation="vertical">

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="148dp"
        android:layout_height="141dp"
        app:srcCompat="@drawable/ic_logo_android" />

    <TextView
        android:id="@+id/textViewTitle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="”Android Studio"/>
    ...
</LinearLayout>
```



Navigation Drawer

- Alterar o layout da *Activity* para ***DrawerLayout***, dentro do qual se incluem:
  1. um ***CoordinatorLayout*** (contendo uma *Toolbar* e conteúdo da atividade);
  2. o componente *NavigationView*;



- A definição da *Toolbar* implica:
  - Incluir no manifesto da app, no elemento <application>, um dos temas NoActionBar do *appcompat*. Isto impede que a app use a classe ActionBar nativa:

```
<application
    ...
    android:theme="@style/AppTheme.NoActionBar"
/>
```

Navigation Drawer

- Implementar uma *Toolbar* dentro do *CoordinatorLayout*;

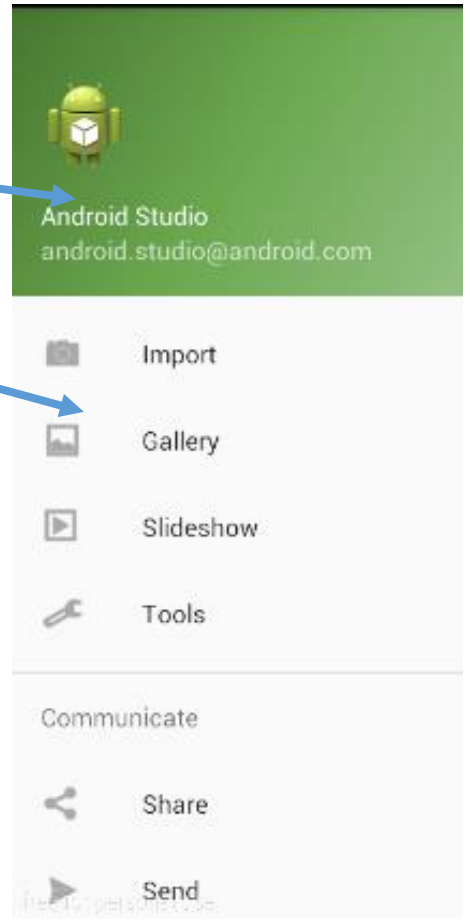
```
<com.google.android.material.appbar.AppBarLayout
    android:id="@+id/appBarLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:theme="@style/AppTheme.AppBarOverlay">

    <androidx.appcompat.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        app:popupTheme="@style/AppTheme.PopupOverlay"/>
</com.google.android.material.appbar.AppBarLayout>
```

- Definir/acrescentar os estilos a aplicar na *Toolbar* no ficheiro *themes.xml* ou *styles.xml*

```
<style name="AppTheme.NoActionBar">
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
</style>

<style name="AppTheme.AppBarOverlay"
        parent="ThemeOverlay.AppCompat.Dark.ActionBar"/>

<style name="AppTheme.PopupOverlay"
        parent="ThemeOverlay.AppCompat.Light"/>
```

Navigation Drawer

- Inserir no componente *NavigationView* a referência aos ficheiros de menu e de layout do cabeçalho, através dos atributos:
  app:headerLayout="@layout/navigation_header"
  app:menu:"@menu/navigation_menu"




```
<com.google.android.material.navigation.NavigationView
    android:id="@+id/nav_view"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    android:fitsSystemWindows="true"
    app:headerLayout="@layout/nav_header_main"
    app:menu="@menu/activity_main_drawer" />
```

Navigation Drawer

- Efetuar, ainda, as seguintes tarefas no método *onCreate()*, da *Activity*:
  - definir a *Toolbar* definida no layout *CoordinatorLayout* como barra da atividade

```
Toolbar toolbar = findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
```

  - instanciar um objeto com a referência ao *DrawerLayout* (*findViewById*)

```
DrawerLayout drawer = findViewById(R.id.drawer_layout);
```

NavigationView

- **Adicionar o hamburger item**. Implica:
  - Implementar uma *ActionBarDrawerToggle* e passar-lhe:
    - a atividade, o *drawerLayout*, a toolbar, o "open drawer" e o "close drawer":

```
DrawerLayout drawer = (DrawerLayout)findViewById(R.id.drawer_layout);
ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
            this, drawer, toolbar, R.string.open_drawer,
            R.string.close_drawer);
```

  - Criar um *Listener* para o *drawerLayout* e passar-lhe a *ActionBarDrawerToggle*:
    `drawer.addDrawerListener(toggle);`
  - Sincronizar o estado da *ActionBarDrawerToggle*:
    `toggle.syncState();`

NavigationView

- A **deteção dos clicks** nos itens do menu da *NavigationView*
  - Implica, na *Activity*:
    - Criar uma instância da *NavigationView*
    - Ativar o *listener OnNavigationItemSelectedListener*

```
public class MainActivity extends AppCompatActivity implements
                        NavigationView.OnNavigationItemSelectedListener {
...

@Override
protected void onCreate(Bundle savedInstanceState) {
        ...
    NavigationView navigationView = (NavigationView)
                                findViewById(R.id.navigation_view);
    navigationView.setNavigationItemSelectedListener(this);
}
```

NavigationView

… e ainda:

- Reescrever o método *OnNavigationItemSelected*
  - para detetar o *click* nos itens do menu e ativar, por exemplo, o fragmento associado a cada opção:

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {
  switch (item.getItemId()){
    case R.id.drawer_menu_camera_id:
      FragmentTransaction ft=getSupportFragmentManager().beginTransaction();
      ft.replace(R.id.frame_layout, new CameraFragment());
      ft.addToBackStack(null); // coloca na back stack
      ft.commit();
      break;
    case R.id.drawer_menu_gallery_id:
      ...
      break;
  }
  drawerLayout.closeDrawer(GravityCompat.START);
  return true;
}
```

Fontes e Mais Informação

- Fragmentos
  - https://developer.android.com/guide/fragments
- *DialogFragment*
  - https://developer.android.com/guide/fragments/dialogs
  - https://guides.codepath.com/android/using-dialogfragment
- *NavigationView*
  - https://developer.android.com/training/implementing-navigation
  - https://material.io/components/navigation-drawer/android#using-navigation-drawers
  - https://developer.android.com/reference/com/google/android/material/navigation/NavigationView

Próximo Tema:

Desenho de Interfaces Gráficas: ListView e GridView

- Arquitetura de Software MVC
  - https://pt.wikipedia.org/wiki/MVC
- *ViewGoups - ListView*
  - https://developer.android.com/guide/topics/ui/layout/listview.html
- *ViewGoups - GridView*
  - https://developer.android.com/guide/topics/ui/layout/gridview.html
- Adaptadores em Android
  - https://developer.android.com/reference/android/widget/Adapter.html
- Criação de Layouts com Adaptadores
  - https://developer.android.com/guide/topics/ui/declaring-layout.html#AdapterViews