soluções mobile

prof. Thyerri Mezzari

Android: Telas, Layouts, Componentes Visuais, Activity e Intents

Parte II

Importante

Vamos iniciar nossos slides com um "breve" *resumão* de conceitos:

Formato XML: Dialeto/formato padrão para composições de dicionários e componentes visuais na plataforma Android Nativa.

Activity: Camada lógica responsável por "montar" e controlar layouts e seus subcomponentes.

Fragment: Camada lógica em formato reutilizável não independente que serve para controlar trechos de layouts e seus subcomponentes.

Importante

Layout: Estrutura de elementos em formato XML que compõe telas e ou trechos de telas (fragments).

Componentes Visuais: Elementos padronizados (ou não) de componentes comuns a serem usados em um aplicativo. *Ex: Botões, Imagens, Campos de Texto e etc.*

Intent: "Script" lógico que indica ao aplicativo uma "intenção" de ação. *Ex: trocar uma tela.*

Parte II

Dando continuidade nos conteúdos iniciados na aula passada, em que introduzimos os conceitos de layouts, componentes e outros elementos importantes para montagem de telas, nesta aula iremos avançar na base de criação de um aplicativo ***Android Nativo*** caminhando agora para a camada "programática" *(código)* da parte mais fundamental de um app.

Activity

```
class MainActivity : AppCompatActivity()
```

"Atividades" do Android

Para cada tela que o usuário visualiza em seu app, para cada diálogo, botão, caixa de texto presente em uma ação há pelo menos uma ***Activity*** envolvida por trás.

Uma ***Activity*** nada mais é que uma classe programática que estrutura e "levanta" os componentes XML de seu aplicativo para a tela que o usuário interage. Sem pelo menos uma ***Activity*** registrada no arquivo *AndroidManifest.xml* seu aplicativo nem irá abrir.

"Atividades" do Android

A classe **Activity** é um componente crucial de um app para Android, e a maneira como as atividades são "lançadas" em tela é uma parte fundamental do modelo de aplicativo da plataforma nativa.

Diferentemente dos paradigmas de programação em que os apps são iniciados com um método ***main()*** *(Java-like)*, o sistema Android inicia o código em uma instância **Activity** invocando métodos de *callback* que correspondem a estágios específicos do **ciclo de vida** de uma aplicação.

Geralmente, uma **Activity** implementa uma tela em um app. Uma **Activity** fornece a janela na qual o app desenha a própria UI. Essa janela normalmente preenche a tela, mas pode ser menor do que a tela e flutuar sobre outras janelas.

"Atividades" do Android

Normalmente, uma **Activity** em um app é especificada como a atividade principal, que é a primeira tela a ser exibida quando o usuário inicia o app. Cada **Activity** pode iniciar outra para realizar ações diferentes caso seja necessário.

Por fim, para usar as atividades *(Activity)* no seu app, é necessário registrar as informações sobre elas no manifesto *(AndroidManifest.xml)* e você precisará gerenciar os **ciclos de vida** da atividade adequadamente para que tudo funcione como esperado.

Criando uma Activity

Por um bom tempo o mundo de desenvolvimento Android Nativo levava a máxima de que para a grande maioria das telas de um app, uma **Activity** seria necessária, ou seja, em um app com 4 telas, no mínimo 4 atividades.

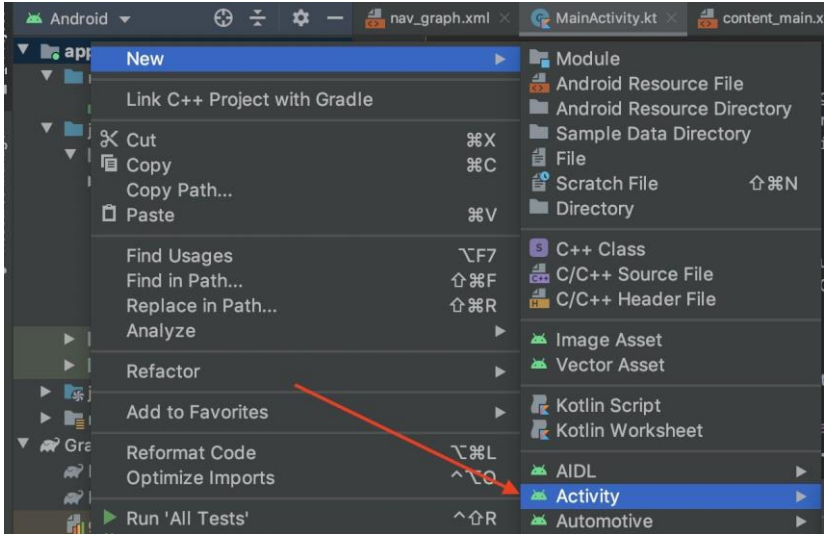
Uma vez que cada Activity têm seu ciclo de vida, por vezes o gerenciamento de memória e troca de dados entre tela poderia ser dificultoso e buscando uma melhora neste processo em 2018 o Google lançou um novo componente de navegação junto da biblioteca **Android Jetpack**.

Com esse novo componente de navegação a troca de telas ficou muito mais fácil e vários apps conseguiram reduzir quase todo o seu funcionamento a apenas uma **Activity** *(em conjunto com outras técnicas)*.

Criando uma Activity

Sempre que um novo App é criado usando o Android Studio, no mínimo uma **Activity principal** já será criada junto de seu projeto base. Essa Activity normalmente é chamada de **MainActivity** e poderá ter seu código escrito em *Kotlin* ou *Java*.

Caso a **MainActivity** não seja suficiente para sua necessidade o caminho mais fácil para criar uma segunda activity é utilizando o comando **File -> New -> Activity**.



Registrando uma Activity

Toda **Activity** criada em seu projeto deverá ser registrada no **AndroidManifest.xml** para que ela possa ser acessível e funcional ao sistema operacional do smartphone.

Para fazer isto basta editar o arquivo manifesto de seu app e adicionar a seguinte linha dentro do nó **<application>**:

```
<activity android:name=".CadastroActivity" android:label="@string/cadastre_se"></activity>
```

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

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Registrando uma Activity

Normalmente ao se criar um novo projeto através do *Android Studio* sua **Activity** principal será automaticamente inserida em seu arquivo de manifesto, junto com as diretrizes necessárias para classificá-la como atividade principal:

```
<activity
    android:name=".MainActivity"
    android:label="@string/app_name"
    android:theme="@style/AppTheme.NoActionBar">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Anatomia básica de uma Activity

Sendo uma Activity uma classe programática baseada em ***Kotlin*** ou ***Java***, sua estrutura mínima deverá sempre se parecer com isso:

Kotlin

```kotlin
package com.example.demoapa

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class CadastroActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_cadastro)
    }
}
```

Anatomia básica de uma Activity

Sendo uma Activity uma classe programática baseada em ***Kotlin*** ou ***Java***, sua estrutura mínima deverá sempre se parecer com isso:

Java

```java
package com.example.demoapa;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class DemoActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_demo);
    }
}
```

Anatomia básica de uma Activity

Seja em *Kotlin* ou *Java* toda Activity irá estender/herdar métodos de uma alguma classe baseada na estrutura base da classe Activity da Android SDK, sendo que normalmente estendemos da classe ***AppCompatActivity***.

Outro fator crucial é a declaração do método de ciclo de vida ***onCreate*** que será executado sempre que a tela desta atividade for "montada". É também dentro deste método *(onCreate)* que definimos qual o layout será relacionado a Activity utilizando o método ***setContentView***.

Ciclo de vida de uma Activity

Um conceito muito importante em relação a **Activity**, **Fragment** e outros componentes programáticos do Android é a utilização de métodos que fazem parte de um ciclo de vida em particular.

À medida que o usuário navega no aplicativo, sai dele e retorna a ele, as instâncias **Activity** no aplicativo transitam entre diferentes estados no ciclo de vida. A classe **Activity** fornece uma quantidade de callbacks que permite que a atividade saiba sobre a mudança do estado: informa a respeito da criação, interrupção ou retomada de uma atividade ou da destruição do processo em que ela reside por parte do sistema.

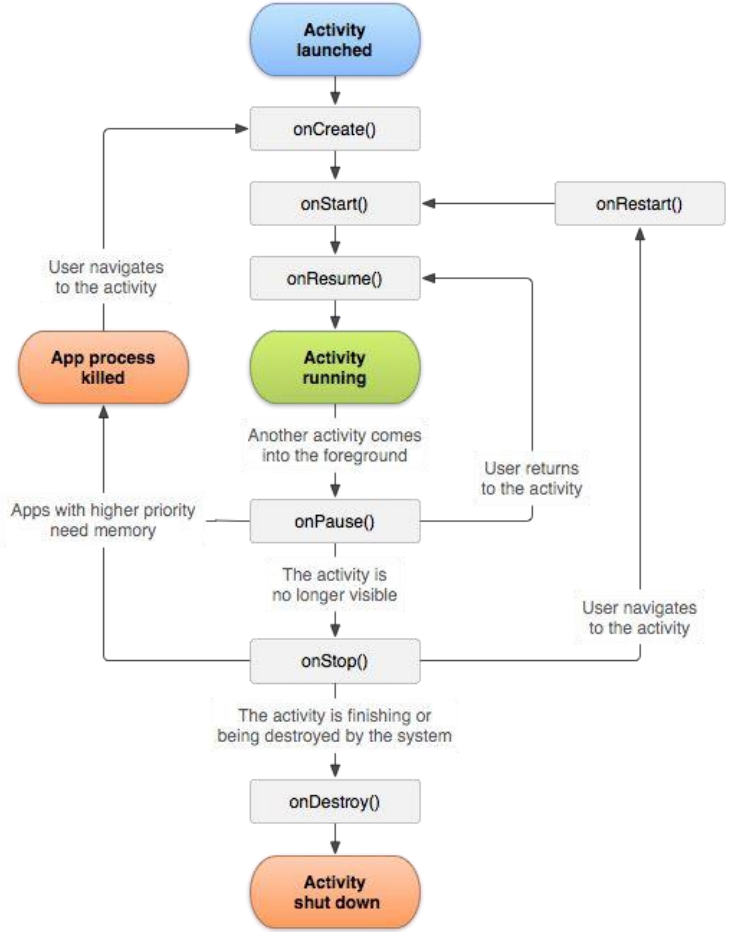
A ideia de manter um ciclo de vida coeso incluindo executar trechos de códigos em determinados momentos de ação, como por exemplo o método ***onCreate*** de uma **Activity** que é executado sempre que a mesma for criada e executada em memória do smartphone.

Ciclo de vida de uma Activity

Entender e aproveitar os ciclos de vida de uma Activity no Android é algo muito importante para um desenvolvimento sustentável, por isso recomendo a leitura completa do link oficial do assunto na documentação:

https://developer.android.com/guide/components/activities/activity-lifecycle?hl=pt_br

Ciclo de vida de uma Activity



Ciclo de vida de uma Activity

onCreate

Você precisa implementar esse callback, que é acionado quando o sistema cria sua atividade. A implementação inicializará os componentes essenciais da atividade. Por exemplo, aqui, o app criará os layouts e vinculará dados na tela.. Mais importante, é nesse local que você precisa chamar ***setContentView()*** para definir o layout da interface do usuário da atividade.

Quando **onCreate()** termina, o próximo callback sempre é **onStart()**.

Ciclo de vida de uma Activity

onStart

Quando *onCreate()* roda, a atividade entra no estado "Iniciado" e se torna visível para o usuário. Esse callback contém o que equivale aos preparativos finais da atividade para ir para o primeiro plano e se tornar interativa.

Ciclo de vida de uma Activity

onResume

O sistema invoca esse callback imediatamente antes de a atividade começar a interagir com o usuário. Neste ponto, a atividade fica na parte superior da pilha de atividades e captura toda a entrada do usuário. A maior parte da funcionalidade principal de um app é implementada no método **onResume()**.

Ciclo de vida de uma Activity

onPause

O sistema chama ***onPause()*** quando a atividade perde o foco e entra em um estado "Pausado". Esse estado ocorre quando, o usuário toca no botão "Voltar" ou "Recentes". Quando o sistema chama ***onPause()*** para sua atividade, isso significa, tecnicamente, que ela ainda está parcialmente visível. Porém, na maioria das vezes, é uma indicação de que o usuário está deixando a atividade e que logo ela entrará no estado "Interrompido" ou "Retomado".

Uma atividade no estado "Pausado" pode continuar atualizando a IU se o usuário estiver esperando por isso. Ex: a exibição da tela de um mapa de navegação ou de um player de mídia sendo reproduzido em bg. Mesmo que essas atividades percam o foco, o usuário espera que a IU continue sendo atualizada.

Ciclo de vida de uma Activity

onStop

O sistema chama ***onStop()*** quando a atividade não está mais visível para o usuário. Isso pode acontecer porque a atividade está sendo destruída, uma nova atividade está sendo iniciada ou uma atividade existente está entrando em um estado "Retomado" e está cobrindo a atividade interrompida. Em todos esses casos, a atividade interrompida não fica mais visível.

O próximo callback que o sistema chamará será **onRestart()**, se a atividade voltar a interagir com o usuário, ou **onDestroy()**, se essa atividade for completamente encerrada.

Ciclo de vida de uma Activity

onRestart

O sistema invoca esse callback quando uma atividade no estado "Interrompido" está prestes a ser reiniciada. ***onRestart()*** restaura o estado da atividade a partir do momento em que ela foi interrompida.

Esse callback é sempre seguido por ***onStart()***.

Ciclo de vida de uma Activity

onDestroy

O sistema invoca esse callback antes de uma atividade ser destruída.

Esse é o último callback que a atividade recebe. ***onDestroy()*** normalmente é implementado para garantir que todos os recursos de uma atividade sejam liberados quando ela (ou o processo que a contém) for destruída.

Fragment

```
class FirstFragment : Fragment()
```

"Fragmentos" do Android

O conceito primário de um **Fragment** resume-se a ser *"uma micro-Activity".* A ideia é ter um trecho de código isolável e independente que possui tanto camada visual (layout xml) quanto camada lógica (classe).

É possível combinar vários fragmentos em uma única Activity para criar uma UI de vários "pedaços" e reutilizar um fragmento em diversas atividades.

Podemos imaginar um fragmento como uma seção modular de uma atividade, que tem o próprio ciclo de vida, recebe os próprios eventos de entrada e que pode ser adicionada ou removida durante a execução da atividade.

"Fragmentos" do Android

Um **Fragment** deve sempre ser hospedado em uma atividade e o ciclo de vida dele é diretamente impactado pelo ciclo de vida da atividade do host. Por exemplo, quando a atividade é pausada, todos os fragmentos também são e, quando a atividade é destruída, todos os fragmentos também são.

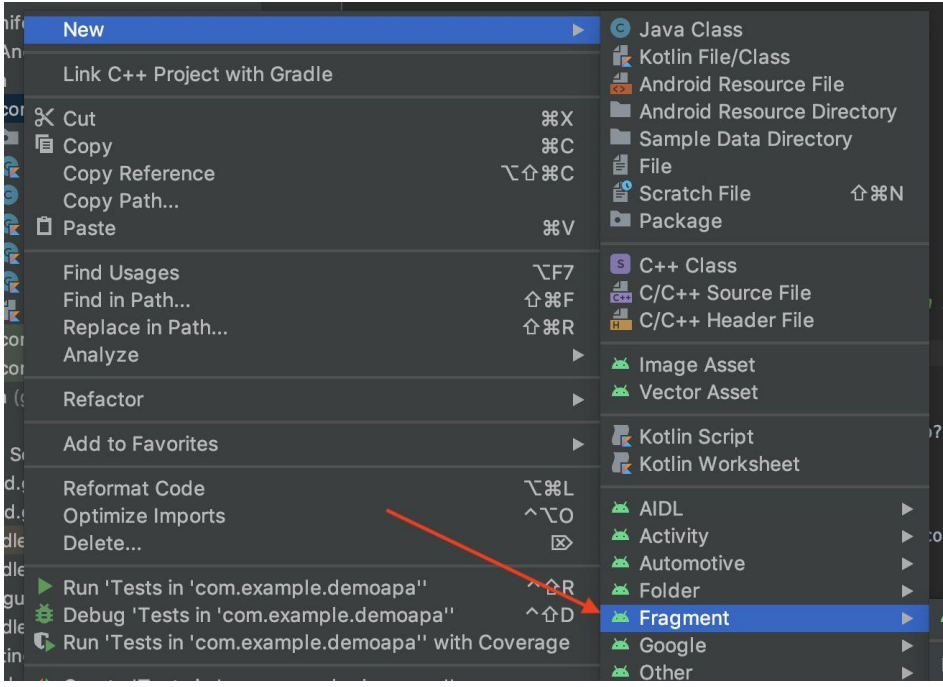
Por fim é possível inserir um fragmento diretamente no layout, declarando-o no arquivo de layout xml da atividade como um elemento/tag ***<fragment>***.

Criando um Fragment

Para criar um fragmento, é preciso criar uma subclasse que estende/herda algum **Fragment** *(ou usar uma subclasse existente dele)* da Android SDK.

Usando o Android Studio o jeito mais fácil de criar um novo Fragment é comando **File -> New -> Fragment**.

Diferente de uma Activity, um Fragment não precisa ser registrado no arquivo de manifesto, basta ser incluído em seu layout ou invocado via código.

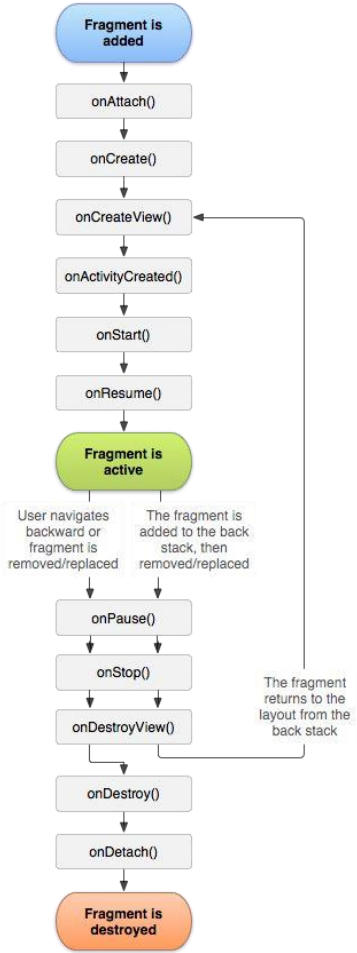


Ciclo de vida de um Fragment

Assim como uma *Activity*, um **Fragment** também possui um ciclo de vida bem definido, para facilitar o nosso controle de interação sobre este "componente reutilizável".

Uma vez que a classe **Fragment** tem um código que é muito parecido com o de uma Activity, um fragmento contém métodos de callback semelhantes aos de uma atividade, como ***onCreate()***, ***onStart()***, ***onPause()*** e ***onStop()***.

Ciclo de vida de um Fragment



Ciclo de vida de um Fragment

onCreate

O sistema o chama ao criar o fragmento. Dentro da implementação, deve-se inicializar os componentes essenciais do fragmento que se deseja reter quando o fragmento é pausado ou interrompido e, em seguida, retomado.

Ciclo de vida de um Fragment

onCreateView

O sistema chama isso quando é o momento de o fragmento desenhar a interface do usuário pela primeira vez. Para desenhar uma UI para o fragmento, você deve retornar uma ***View*** deste método, que é a raiz do layout do fragmento. É possível retornar como nulo *(null)* se o fragmento não tiver interface visual.

Anatomia básica de um Fragment

Sendo um Fragment uma classe programática baseada em ***Kotlin*** ou ***Java***, sua estrutura mínima deverá sempre se parecer com isso:

Kotlin

```kotlin
class ExampleFragment : Fragment() {

    override fun onCreateView(
            inflater: LayoutInflater,
            container: ViewGroup?,
            savedInstanceState: Bundle?
    ): View {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false)
    }

}
```

Anatomia básica de um Fragment

Sendo um Fragment uma classe programática baseada em ***Kotlin*** ou ***Java***, sua estrutura mínima deverá sempre se parecer com isso:

Java

```java
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false);
    }
}
```

Usando um Fragment

O método mais fácil de utilizarmos um **Fragment** em nosso projeto é inserindo o mesmo um "include" reutilizável em nossos layouts, podemos chamar um ***<fragment />*** de qualquer tela que necessitarmos do mesmo e também utilizá-lo de maneira múltipla como uma lista de itens que se repetem.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.DemoFragmentPainelEtc"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

Usando um Fragment

Para entender melhor o uso dos fragmentos recomendo a documentação oficial:

https://developer.android.com/guide/components/fragments?hl=pt_br

Acessando Componentes

findViewById(R.id.?)

Recursos de uma Aplicação

Uma aplicação Android Nativo possui diversos tipos de "recursos" envolvidos em sua criação, a grande maioria passa de alguma forma pela pasta ***resources***, desde imagens (drawlable), strings, colors, layouts, menus, mapa de navegação e etc.

Além destes recursos, temos também os componentes inseridos dentro de telas de layouts, fragments e em outros locais, todos estes comumente são identificados por seu nome ou através do atributo ***android:id***.

Sempre que um elemento ganha um id usando ***android:id="@+id/????"*** este mesmo passa a estar visual para a **Activity** e também para **Fragments**.

Recursos de uma Aplicação

Todos os códigos de recursos são definidos na classe ***R*** do projeto que a ferramenta *aapt* gera automaticamente.

Quando o aplicativo é compilado, o *aapt* gera a classe ***R***, que contém códigos de recursos para todos os recursos no diretório **res/**.

Para cada tipo de recurso, há uma subclasse R *(por exemplo, R.drawable para todos os recursos drawable)* e, para cada recurso daquele tipo, há uma referência estática (*por exemplo, R.drawable.icon)*. Essa referência é o ID do recurso que pode ser usado para recuperá-lo no código.

Recursos de uma Aplicação

Por exemplo, um botão adicionado ao LinearLayout cujo o android:id é "***botaoAzul***":

```
<LinearLayout
    android:background="@color/colorPrimaryDark"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <Button
        android:id="@+id/botaoAzul"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/previous"
        />
</LinearLayout>
```

Este mesmo botão ficará referenciado de forma programática através da seguinte variável/caminho:

```
R.id.botaoAzul
```

Acessando um Componente

Uma vez que saibamos o id/referência de um componente, a forma mais fácil de acessar este via programação para manipulá-lo dinamicamente e até monitorar eventos do mesmo, é usando a função ***findViewById***.

De uma **Activity**:

```
// Kotlin
val toolbar = findViewById<Toolbar>(R.id.toolbar)

// Java
Toolbar toolbar = findViewById(R.id.toolbar);
```

Acessando um Componente

Uma vez que saibamos o id/referência de um componente, a forma mais fácil de acessar este via programação para manipulá-lo dinamicamente e até monitorar eventos do mesmo, é usando a função ***findViewById***.

De um **Fragment**:

```
// Kotlin
val toolbar = getView().findViewById<Toolbar>(R.id.toolbar)

// Java
Toolbar toolbar = getView().findViewById(R.id.toolbar);
```

Manipulando um Componente

Após obtermos um componente e através da função findViewById e associá-lo a uma variável fica fácil de podermos manipular o mesmo via programação.

Por exemplo, depois de termos criado uma variável que "armazena" o controle da barra de topo de um aplicativo Android Nativo podemos mudar o título que aparece em uma tela assim:

```
// Kotlin
toolbar.title = "Cadastre-se"
```

Ou até mesmo esconder essa barra em uma determinada tela (caso precisamos de tela cheia):

```
// Kotlin
toolbar.visibility = View.GONE
```

Eventos de um Componente

Componentes interativos como caixas de texto, botões, seletores, checkboxes, radio buttons possuem uma série de eventos interativos, desde cliques, digitação de texto, opção selecionada e etc.

Estes eventos passam todos por um conceito de "listener" (ouvinte) muito semelhante ao JavaScript para web por exemplo.

Eventos de um Componente

O código abaixo adiciona um *listener* de **click** para um determinado botão em tela do android:

```kotlin
// Kotlin
val botaoAzul = findViewById<Button>(R.id.botaoAzul);
botaoAzul.setOnClickListener { view ->
    // faça algo quando clicar
}
```

```java
// Java
Button botaoAzul = findViewById(R.id.botaoAzul);
botaoAzul.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        // faça algo quando clicar
    }
});
```

Navegação entre telas

findNavController().navigate(R.id.?)

Navegar é preciso ¯_(ツ)_/¯

Dificilmente o seu app terá apenas uma tela, normalmente a funcionalidade de um aplicativo seja ele Android ou iOS irá passar por um número não fixo de telas, o usuário poderá ir e voltar entre passos, telas, etapas, menus e etc.

Logo as intenções de navegação serão sempre importantes e também algo que o desenvolvedor deva se preocupar e "desenhar" muito bem.

Até o ano de 2018, antes do lançamento do Android Jetpack e do novo componente navegação jeito mais comum era criar uma *Activity* para cada tela e utilizar o controle de ***Intents*** para navegar...

Intent

O Intent é um objeto de mensagem que pode ser usado para solicitar uma ação de outro componente do aplicativo (e até de fora dele).

Embora os intents facilitem a comunicação entre os componentes de diversas formas, há três casos fundamentais de uso:

1. Iniciar uma atividade
Uma Activity representa uma única tela em um aplicativo. É possível iniciar uma nova instância de uma Activity passando um Intent para startActivity().

2. Iniciar um serviço
O Service é um componente que realiza operações em segundo plano sem interface do usuário.

3. Fornecer uma transmissão
Transmissão é uma mensagem que qualquer aplicativo pode receber. Ex: compartilhar um texto via Facebook ou SMS.

Intent

Se você já tem uma Activity responsável por uma outra tela além da que está aberta no momento e deseja iniciar a mesma, use o comando abaixo:

```kotlin
// Kotlin
val intent = Intent(this, DemoActivity::class.java)
startActivity(intent)
```

```java
// Java
Intent intent = new Intent(this, DisplayMessageActivity.class);
startActivity(intent);
```

Intent

Caso você precise passar algum dado entre uma tela e outra, como por exemplo o código de um produto que o usuário deseja ver mais detalhes, utilize *putExtra*:

```kotlin
// Kotlin
val intent = Intent(this, DemoActivity::class.java).apply {
    putExtra("CODIGO", 3040)
}

startActivity(intent)
```

```java
// Java
Intent intent = new Intent(this, DisplayMessageActivity.class);
intent.putExtra("CODIGO", 3040);
startActivity(intent);
```

Intent

Atualmente o jeito mais comum de uso de um Intent é troca de mensagem entre aplicativos (share), por exemplo se você deseja compartilhar com qualquer aplicativo que aceite texto do Android uma mensagem compartilhável (whatsapp, facebook, twitter, SMS):

```
// Kotlin - Create the text message with a string
val sendIntent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT, "Olá Mundo!")
    type = "text/plain"
}

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(packageManager) != null) {
    startActivity(sendIntent)
}
```

Componente de Navegação

Conforme já comentado no início desta aula, atualmente o método recomendado para navegação entre telas é o componente de navegação do ***Android JetPack*** *(biblioteca mais moderna das últimas SDKs do Android)*.

Este componente trouxe uma maior integração com o ***Android Studio*** no que tange desenho de navegação entre telas e também facilitou o compartilhamento de fragmentos e outros componentes que possam estar presentes em mais de uma tela.

Componente de Navegação

O funcionamento do componente de navegação consiste de três partes principais, descritas abaixo:

Gráfico de navegação: é um recurso XML que contém todas as informações relacionadas à navegação em um local centralizado. Isso inclui todas as áreas de conteúdo individual no aplicativo, chamadas destinos, e todos os caminhos que podem ser percorridos pelo usuário no aplicativo. Este arquivo possui um modo de edição visual muito interativo no Android Studio. Normalmente é acessado e controlado de dentro de uma Activity ou Fragment.

Componente de Navegação

O funcionamento do componente de navegação consiste de três partes principais, descritas abaixo:

NavHost: é um contêiner vazio que mostra a telas do gráfico de navegação a medida que são solicitadas. O componente de navegação contém uma implementação *NavHost* padrão, *NavHostFragment*, que mostra os destinos do fragmento.

NavController: é um objeto que gerencia a navegação do aplicativo em um *NavHost*. O *NavController* organiza a troca do conteúdo de destino no *NavHost* conforme os usuários se movem pelo aplicativo. Normalmente é acessado e controlado de dentro de uma *Activity* ou *Fragment*.

Gráfico de Navegação

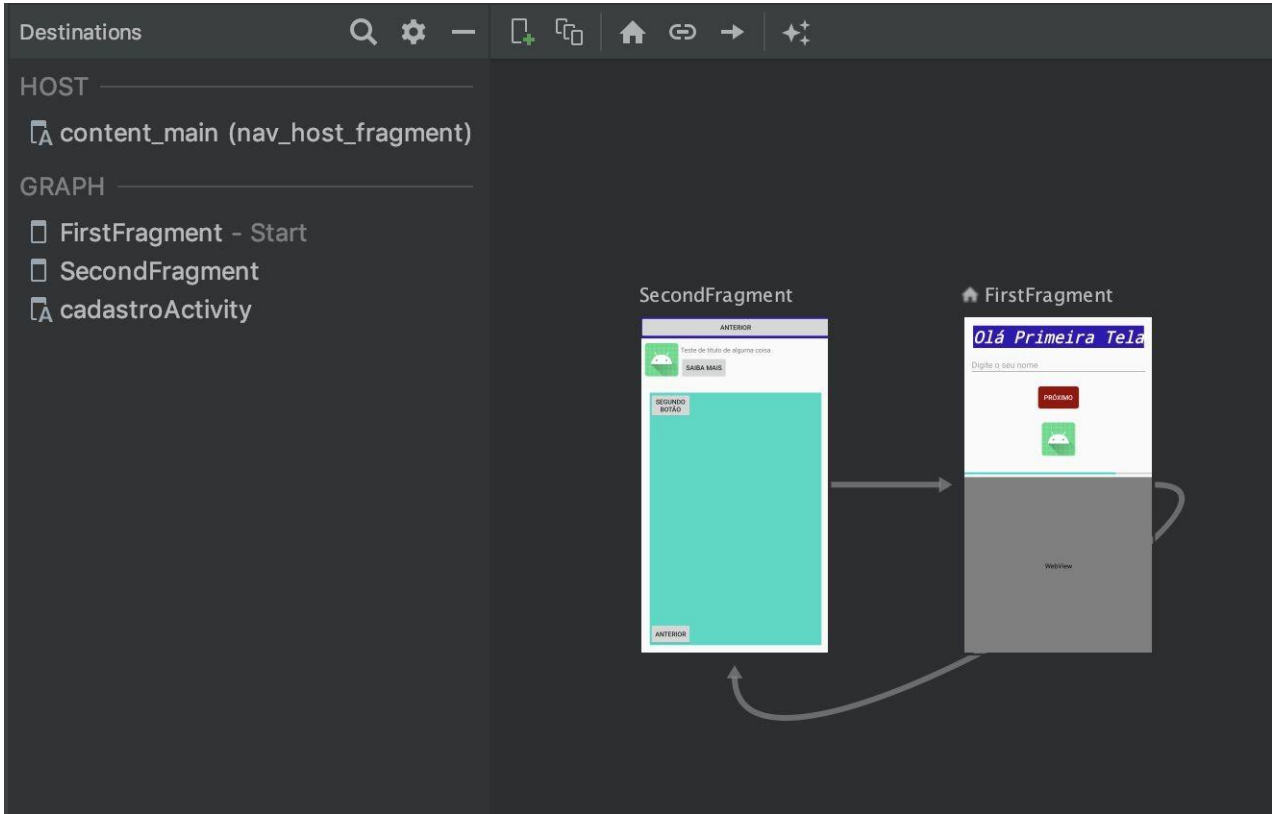
As estruturas de um gráfico de navegação estará sempre presente na pasta res/navigation. Normalmente um arquivo .xml contendo a estrutura de fluxo é o suficiente para um app de pequeno a médio porte.

Neste arquivo o desenvolvedor poderá declarar quais caminhos (rotas) será possível para o usuário navegar ao longo de seu app.

Gráfico de Navegação - Código

```
<?xml version="1.0" encoding="utf-8"?>
<navigation
    android:id="@+id/nav_graph"
    app:startDestination="@id/FirstFragment">
    <fragment
        android:id="@+id/FirstFragment"
        android:name="com.example.demoapa.FirstFragment"
        android:label="@string/first_fragment_label"
        tools:layout="@layout/fragment_first">
        <action
            android:id="@+id/action_FirstFragment_to_SecondFragment"
            app:destination="@id/SecondFragment"/>
    </fragment>
</navigation>
```

Gráfico de Navegação - Editor



NavHost

O componente **NavHost** é o elemento que você deve inserir em seu layout para receber as telas baseadas em **Fragment** que seu aplicativo irá mostrar ao usuário:

```
<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment
        android:id="@+id/nav_host_fragment"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:defaultNavHost="true"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:navGraph="@navigation/nav_graph" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

NavController

Por fim, depois de criar suas telas, definir as rotas de navegação em seu nav graph, incluir em seu app o nav host responsável por receber as mesmas, podemos enfim iniciar a mudança de telas através do **NavController**.

Exemplo de código que resulta em uma mudança de tela:

```
// Kotlin
findNavController().navigate(R.id.action_Primeira_to_Segunda_tela)
```

Monitorando a Navegação

Uma coisa que volta e meia será necessário e monitorar o estado da navegação, ou seja criar um evento programático que será acionado sempre que uma tela mudar. A utilidade disto pode ser relacionada troca de títulos, mudança de ações na barra de topo, mostrar ou esconder algum botão dependendo da tela e outras ideias.

O jeito mais fácil de fazer isso é criando um *addOnDestinationChangedListener*:

```
// Kotlin
navController.addOnDestinationChangedListener { _, destination, _ ->
    if(destination.id == R.id.full_screen_destination) {
        toolbar.visibility = View.GONE
    } else {
        toolbar.visibility = View.VISIBLE
    }
}
```

Componente de Navegação

Vale a pena dar uma olhada na documentação completa do componente de navegação do Android Jetpack:

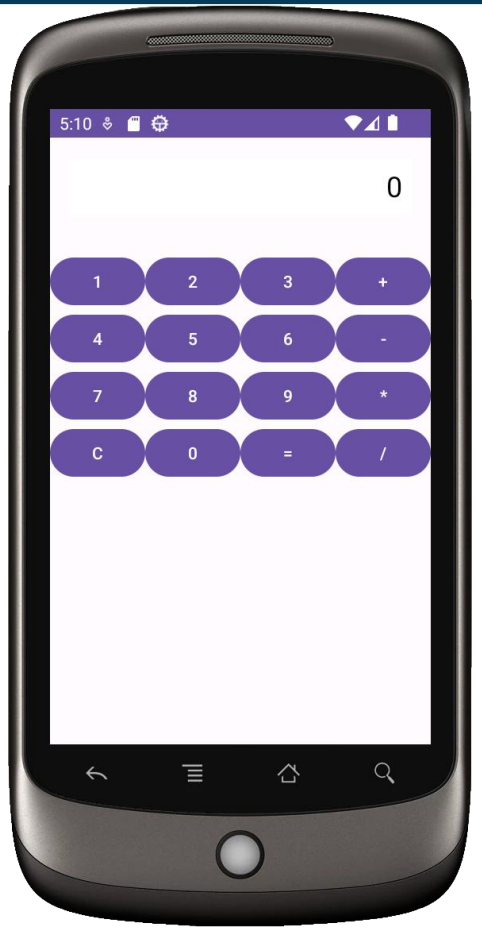
https://developer.android.com/guide/navigation

Exercício 1 (parte 2)

Continuar o desenvolvimento de nossa calculadora simples. Faça ela funcionar. Todas as operações (somar, subtrair, multiplicar, dividir, limpar)

Aplique os conceitos que aprendemos:

- Defina uma referência para os componentes (id)
  - *android:id="@+id/meubotao"*
- Acesse o componente em nossa MainActivity:
  - *findViewById*
- Prepare seu listener para ouvir eventos de click
  - setOnClickListener
- Declare variáveis conforme sua necessidade
- Android nativo com Java ou Kotlin



Entrega: 19/03/2024 19:00hrs PESO 0,5

obrigado 🚀