Universidade Federal do Piauí (UFPI)

Centro de Ciências da Natureza (CCN)

Departamento de Computação (DC)

Pedro Ivo Soares Barbosa

Engenharia de Software II *Professor: Armando Soares Sousa*

2016.2

Tutorial de Criação de Telas no Android

Índice

1. Criação da Activity

Uma activity é uma classe que deve herdar da classe *android.app.Activity* ou de alguma subclasse desta, a qual representa uma tela da aplicação e é responsável por tratar os eventos gerados nessa tela

Dentro do método sobrescrito *onCreate* , chamamos o método *setContentView,* cujo parâmetro é um arquivo XML responsável por moldar o visual da tela a ser apresentado ao usuário.

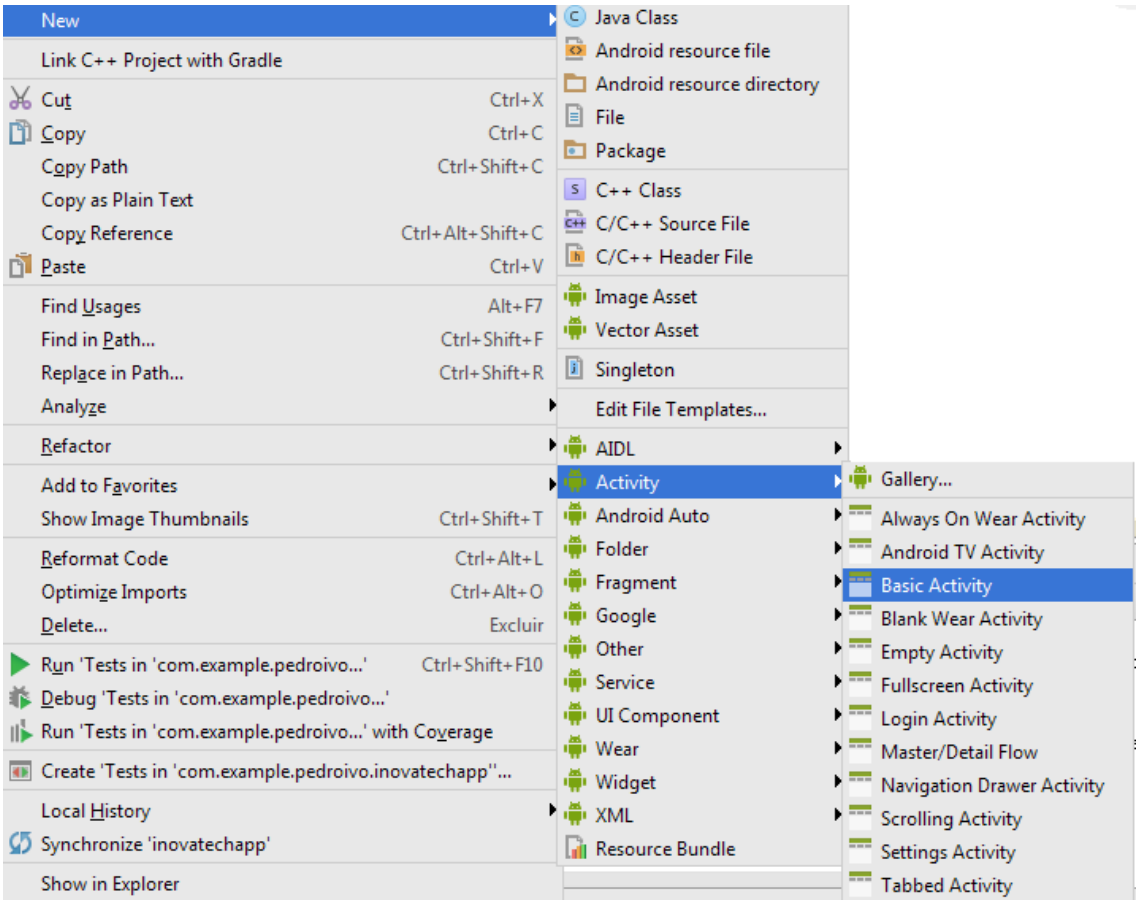
Além disso, no método *onCreate*, podemos declarar as Views (Que serão abordadas mais à frente), tratar os eventos das mesmas e etc.

Em palavras simples, para criar uma Activity, são necessários apenas dois passos :

1.Criar a Classe Java que estende de alguma subclasse de *android.app.Activity*

2. Criar o arquivo XML de Layout de Layout

3. Registrar a activity no arquivo AndroidManifest.xml



2. Acesso aos Elementos (Views)

Para que seja possível utilizar os elementos gráficos, chamados de Views, primeiramente é preciso declará-los dentro do documento XML responsável pelo Layout, como por exemplo, para declarar um elemento de texto, TextView, fazemos :

```
<TextView
        android:text=”Primeira TextView”
        android:layout_width=”wrap_content”
        android:layout_height=”wrap_content”/>
```

<TextView>, o nome da View englobada pelos símbolos ‘<’ e ‘>’ é a declaração da View no arquivo XML e “*android:text*” é a declaração do parâmetro, no caso, a cadeia de caracteres que será exibida dentro da View. Já a expressão do outro lado do sinal ‘=’ é o valor do parâmetro.

“*android:layout_width*” e “*android:layout_height*” , respectivamente, configuram , a largura e a altura do View dentro do layout no qual a mesma está inserida, podendo tais medidas serem definidas usando dp , px , ou expressões, como “*wrap_content*” ou “*match_parent*”, sendo que a primeira define que a medida em questão vai se adaptar à mesma medida do conteúdo da View e a outra define que tal medida vai se adaptar ao máximo do elemento que a engloba, podendo ser outra View, ou um Layout.

Um atributo de suma importância que precisamos fazer uso é o android:id=”@+id/algumaIdParaAView“, com o qual podemos definir, após a barra, um identificador único para cada View e, através do qual podemos acessar a View dentro das classes Java.

Na segunda parte, na Classe Java referente ao arquivo XML no qual inserimos e definimos os atributos das Views, precisamos declarar objetos referentes a tais Views.

Supondo que no XML esteja declara uma View tal qual está descrito a seguir :

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:id="@+id/Texto"/>
```

Então para acessar tal View, deveríamos fazer algo do tipo na Classe herdeira de Activity correspondente ao XML :

```java
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    TextView texto;

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

        texto = (TextView) findViewById(R.id.Texto);
    }
}
```

Primeiro, declaramos um objeto com o mesmo tipo da View que queremos acessar, não importa se declaramos o objeto fora ou dentro do método.

Logo após inicializamos o objeto utilizando o método findViewById, cujo parâmetro recebe justamente a Id. Esse método retorna um Objeto View, então usamos um cast para usarmos o mesmo para inicializar nosso objeto TextView.

Após isso, podemos manipular à vontade o objeto TextView, inclusive podendo até alterar os mesmos parâmetros que definimos no XML dentro de métodos Java no decorrer da execução da aplicação.

3. Tipos de Views

3.1. TextView

A mais básica de todas as Views, representa um texto fixo que o usuário não pode alterar.

3.2. EditText

Um campo de texto que pode ser alterado pelo usuário. Pode ser usado para entrada de texto normal, ou para aceitar apenas números ou campos de senha.

3.3. AutoCompleteTextView

É um campo de texto que complete automaticamente o texto que o usuário está digitando. Nesta view é necessário definir dois parâmetros:

android:completionThreshold, que é o número de letras que o usuário precisa digitar para iniciar o autopreenchimento de texto.

android:completionHint, que é um texto utilizado para exibir uma dica sobre o preenchimento do texto. O texto é exibido na parte inferior do popup com as opções quando este é aberto.

3.4. Button e ImageButton

São utilizadas para criar um botão na tela. A diferença é que a classe ImageButton permite usar uma imagem para desenhar o botão.

No Android, para tratar os eventos de um botão é utilizado o método *setOnClickListener(listener).* Esse método recebe como argumento uma instância da interface *android.view.View.OnClickListener*. A interface *View.OnClickListener* define o método *OnClick(view),* o qual é chamado quando o evento ocorrer, passando como argumento o objeto da view que gerou o evento, neste caso, o botão.

3.5. CheckBox e ToggleButton

CheckBox é uma espécie de botão que apresenta dois estados : Marcado ou Não Marcado.

Após declarado, pode-se facilmente verificar se ele está marcado ou não, utilizando o método isChecked(), o qual retorna um boolean.

O ToggleButton funciona da mesma maneira e também possui o método isChecked(), porém com esse elemento é possível também alterar o texto do para cada um dos dois estados utilizando os atributos *android:textOn* e *android:textOff.*

3.6. RadioButton

Esse componente permite selecionar uma única opção de uma lista. Nesse caso cada RadioButton definido no XML deve estar contido dentro de outro elemento, chamado RadioGroup, sendo que cada RadioButton deve ter seu próprio Id.

Para determinar se um elemento RadioButton foi selecionado, devemos obter a Id do RadioButton selecionado através de um método da classe RadioGroup, chamado *getCheckedRadioButtonId()* e comparar seu retorno com alguma Id conhecida, retornando *true* ou *false.*

3.7. Spinner

Utilizado para criar um combo com opções na tela. Para definir a lista que deve ser exibida no combo, é usada uma implementação de *android.widget.SpinnerAdapter* que herda de *android.widget.Adapter*.

Podemos inclusive dizer se a lista surgirá acima ou abaixo do botão do Spinner usando os atributos *android:drawSelectorOnTop* e *android:drawSelectorOnBottom*.

3.8. Toast

Utilizada para mostrar alertas para o usuário. Cada alerta pode ser visualizado na tela por um tempo curto, especificado pela constante *Toast.LENGTH_SHORT,* ou por um tempo longo se utilizar a constante *Toast.LENGTH_LONG.*

A forma mais simples de criar um alerta é com o método *Toast.makeText(contexo, mensagem, tempo***),** como no exemplo :

3.9. AlertDialog

Diferente da Classe *Toast,* que não permite interação com usuário, a Classe AlertDialog permite que o usuário pressione **OK** ou responda perguntas com opções **SIM** ou **NÃO.**

4. Tipos de Layout

4.1. FrameLayout

O mais simples de todos os gerenciadores de Layout e é utilizado para empilhar uma view sobre a outra. É possível adicionar vários componentes dentro do FrameLayout, e sempre os últimos componentes ficarão sobre os anteriores, seguindo o conceito de pilha.

4.2. LinearLayout

Com este gerenciador de layout é possível organizar os componentes na horizontal (padrão) ou na vertical. Tal orientação é configurada pelo atributo *android:orientation.*

Para controlar o posicionamento de um elemento no LinearLayout podemos usar o atributo *android:layout_gravity.* Os valores válidos para esse atributo são:

top (Topo)
bottom (Baixo)
left (Esquerda)
right(Direita)
center_vertical (Centralizar na Vertical)
center_horizontal (Centralizar na Horizontal)
*fill_horizontal (*Preencher na Horizontal)
*center (*Alinhar no centro)
fill (Preencher)*.*

Outra forma de controlar a organização dos elementos na tela é atribuir um peso (uma porcentagem) para cada um. O componente com o maior peso ocupará uma maior porcentagem do espaço na tela. Para configurar o peso de cada elemento usamos o atributo *android:layout_weight.*

4.3. TableLayout

É um dos layouts mais úteis para construir algumas telas, como por exemplo formulários. Funciona como uma tabela onde cada linha é formada por uma view TableRow.

Podemos manipular as colunas (views nas linhas) utilizando dois atributos, o *android:stretchColumns* e o *android:shrinkColumns* :

android:stretchColumns : Faz com que as colunas ocupem o espaço disponível na tela, expandindo-as

android:shrinkColumns : Faz com que as colunas especificadas sejam sempre exibidas. Caso o valor do texto seja muito grande e fique para fora da tela, a linha é quebrada e o texto é exibido em várias linhas da mesma coluna.

Ambos recebem o número das colunas que devem ser alteradas.

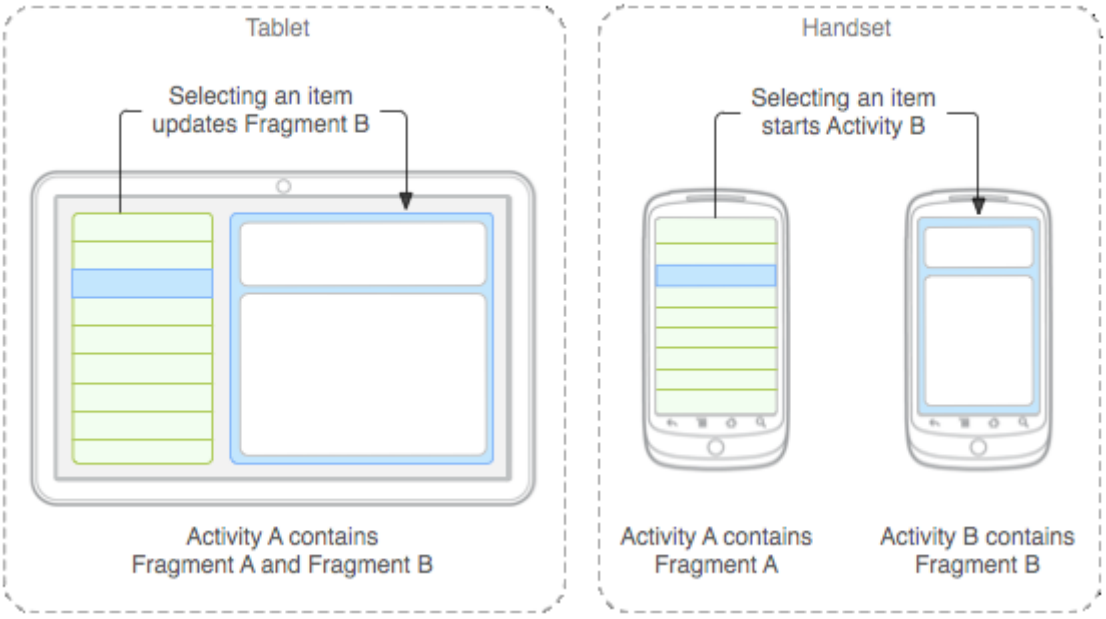
4.4. RelativeLayout

Neste layout é possível posicionar os componentes ao lado, abaixo ou acima de outro componente já existente. Para isso é necessário definir um id para cada componente da tela, pois o posicionamento de um componente depende de outro. Os seguintes atributos podem ser utilizados para informar a posição do novo componente inserido relativo ao componente já existente.

android:layout_below Posiciona baixo do componente indicado
android:layout_above Posiciona acima do componente indicado
android:layout_toRightOf Posiciona à direita do componente indicado
android:layout_toLeftOf Posiciona à esquerda do componente indicado
android:layout_alignParentTop Alinha no topo do layout-pai
android:layout_alignParentBottom Alinha abaixo do layout-pai
android:layout_alignParentRight Alinha à direita do layout-pai
android:layout_alignParentLeft Alinha à esquerda do layout-pai

5. Fragments

Um **Fragment** representa o comportamento ou uma parte da interface do usuário em um Activity. É possível combinar vários fragmentos em uma única atividade para compilar uma IU de painéis múltiplos e reutilizar um fragmento em diversas atividades. Um fragmento é como uma seção modular de uma atividade, que tem o próprio ciclo de vida, recebe os próprios eventos de entrada e que pode ser adicionado ou removido com a atividade em execução (uma espécie de "sub-atividade" que pode ser reutilizada em diferentes atividades). Podemos utilizar Fragments para manipular a interface de usuário de acordo com o espaço disponível do dispositivo:



Como criar um Fragment ?

Para criar um fragment, é preciso criar uma subclasse de Fragment (ou uma respectiva subclasse existente). A classe Fragment tem um código que é muito parecido com o de uma Activity. Ele contém métodos de retorno de chamada semelhantes aos de uma atividade, como onCreate(), onStart(), onPause() e onStop().

Geralmente, deve-se implementar pelo menos os seguintes métodos de ciclo de vida:

onCreate()

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

onCreateView()

O sistema chama isto quando é o momento de o fragment desenhar a interface do usuário pela primeira vez. Para desenhar uma IU para o fragment, você deve retornar uma View deste método, que é a raiz do layout do fragment. É possível retornar como nulo se o fragment não fornecer uma IU.

onPause()

O sistema chama esse método como o primeiro indício de que o usuário está saindo do fragment (embora não seja sempre uma indicação de que o fragment está sendo destruído). É quando geralmente deve-se confirmar qualquer alteração que deva se manter além da sessão atual do usuário (porque o usuário pode não retornar).

Adição de uma interface do usuário

Um fragment é geralmente usado como parte de uma interface do usuário da atividade e contribui para a atividade com seu próprio layout.

Para fornecer um layout para um fragmento, você deve implementar o método de retorno de chamada onCreateView(), que o sistema Android chama no momento em que o fragmento deve desenhar o layout. A implementação deste método deve retornar uma View, que é a raiz do layout do fragmento.

Observação: se o fragmento for uma subclasse de ListFragment, a implementação padrão retornará uma ListView de onCreateView(), então não será necessário implementá-lo.

Para retornar um layout de onCreateView(), é possível inflá-lo a partir de um recurso de layout definido no XML. Para ajudar a fazer isto, o onCreateView() fornece um objeto LayoutInflater.

Por exemplo, a seguir há uma subclasse de Fragment que carrega um layout do arquivo example_fragment.xml:

```
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);
    }
}
```

O parâmetro container passado para onCreateView() é o pai de ViewGroup (do layout da atividade) em que o layout do fragment será inserido. O parâmetro savedInstanceState é um Bundle que fornece dados sobre a instância anterior do fragmento, se o fragment estiver sendo O método inflate() usa três argumentos:

- O ID de recurso do layout que você quer inflar.
- O ViewGroup que será pai do layout inflado. Passar o container é importante para que o sistema aplique os parâmetros de layout à vista raiz do layout inflado, especificado pela view pai em que está ocorrendo.
- Um booleano que indica se o layout inflado deve ser anexado a ViewGroup (o segundo parâmetro) durante a inflação (neste caso, isto é falso, pois o sistema já está inserindo o layout inflado no container — retornar como verdadeiro criaria um grupo de views redundante no layout final).

Este é o modo de criar um fragment que fornece um layout. A seguir, é preciso adicionar o fragment à atividade.

Adição de um fragment a uma atividade

Geralmente, um fragment contribui com a atividade do host com uma parte da IU, que é embutida como parte da hierarquia de views geral da atividade. Há duas formas de adicionar um fragment ao layout da atividade:

- **Declarar o fragmento dentro do arquivo de layout da atividade.**

  Neste caso, é possível especificar as propriedades do layout para o fragment como se fosse uma view. Por exemplo, a seguir há o arquivo de layout para uma atividade com dois fragments:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/and
roid"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment
android:name="com.example.news.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment
android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

  O atributo android:name em <fragment> especifica a classe Fragment a instanciar no layout.

  Quando o sistema cria este layout de atividade, ele instancia cada fragmento especificado no layout e chama o método onCreateView() para cada um para recuperar o layout de cada fragmento. O sistema insere a View retornada pelo fragment diretamente no lugar do elemento <fragment>.

Observação: cada fragmento requer um identificador único que o sistema possa usar para restaurá-lo se a atividade for reiniciada (e que possa ser usado para capturar o fragment para realizar operações, como a remoção). Há três maneiras de fornecer um ID para um fragmento:

- Fornecer o atributo android:id com um ID único.
- Fornecer o atributo android:tag com uma string única.
- Caso não forneça nenhuma das alternativas anteriores, o sistema usará o ID da vista do recipiente.

- **Ou adicionar programaticamente o fragmento a um ViewGroup existente.**

A qualquer momento, enquanto a atividade está em execução, é possível adicionar fragments ao layout da atividade. Você precisa apenas especificar um ViewGroup para posicionar o fragment.

Para realizar operações de fragmentos na atividade (como adicionar, remover ou substituir um fragment), você deve usar APIs de FragmentTransaction. É possível adquirir uma instância de FragmentTransaction da Activity desta maneira:

```
FragmentManager fragmentManager =
getFragmentManager()
FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();
```

É possível adicionar um fragment usando o método add(), especificando o fragment que será adicionado e a vista em que será inserido. Por exemplo:

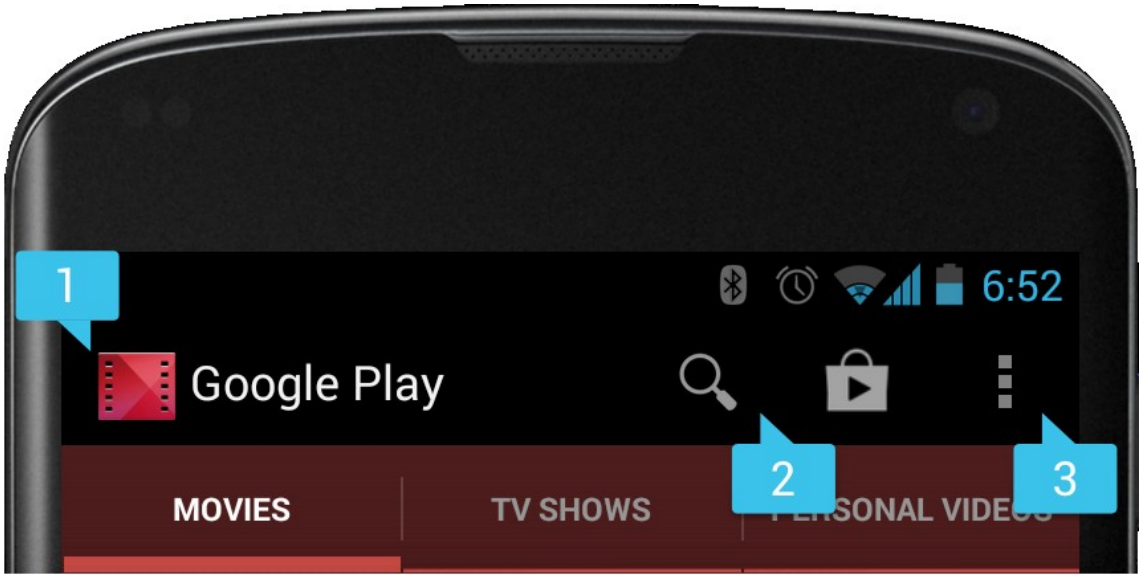
```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container,
fragment);
fragmentTransaction.commit();
```

O primeiro argumento passado para add() é ViewGroup, onde o fragment deve ser colocado, especificado pelo ID de recurso e o segundo parâmetro é o fragment a ser adicionado.

Ao realizar as alterações com FragmentTransaction, deve-se chamarcommit() para que as alterações entrem em vigor.

6. ActionBar

A ActionBar mostra de forma consistente para o usuário as possíveis ações que ele pode fazer no aplicativo. A grande vantagem é que os usuários de Android já estão acostumados a ela pois todos os aplicativos nativos são feitos assim.



1. App Icon: Por padrão o ícone da action bar mostra o ícone do projeto e pode ser customizado conforme sua necessidade. Nesse espaço também é mostrado o “up navigation”, que é a seta para esquerda que indica que o usuário pode navegar para cima na hierarquia de telas.

2. Action Buttons: Espaço dedicado para os botões que representam as ações mais comuns do seu aplicativo caso a quantidade de ícones não se encaixe no espaço reservado, automaticamente eles serão inseridos no menu do action overflow.

3. Action Overflow: Menu flutuante que mostra as ações que não são tão frequentes no aplicativo.

Como manipular a Action Bar

1° Criar um Arquivo XML no qual definimos os itens que estarão presentes na Action Bar

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

    <!-- "Mark Favorite", should appear as action
button if possible -->
    <item
        android:id="@+id/action_favorite"
        android:icon="@drawable/ic_favorite_black_48dp"
        android:title="@string/action_favorite"
        app:showAsAction="ifRoom"/>

    <!-- Settings, should always be in the overflow -->
    <item android:id="@+id/action_settings"
          android:title="@string/action_settings"
          app:showAsAction="never"/>

</menu>
```

app:showAsAction=”ifRoom” determina que tal opção só aparecerá caso haja espaço

app:showAsAction=”never” determina que tal opção nunca vai aparecer na app bar, mas sim no action overflow

2º Para utilizar esse arquivo XML de menu, a activity deve implementar o método *onCreateOptionsMenu(menu)* para inflar o menu XML.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {

getMenuInflater().inflate(R.menu.main_activity_actions,
menu);
```

```
    return super.onCreateOptionsMenu(menu);
}
```

3º Por fim, uma vez que o menu com botões da action bar estão configurados, basta implementar o método *onOptionsItemSelected(MenuItem)* para tratar os eventos gerados pelos botões. Por isso no arquivo XML foi definido um id para cada item.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_settings:
            // User chose the "Settings" item, show the
app settings UI...
            return true;

        case R.id.action_favorite:
            // User chose the "Favorite" action, mark
the current item
            // as a favorite...
            return true;

        default:
            // If we got here, the user's action was
not recognized.
            // Invoke the superclass to handle it.
            return super.onOptionsItemSelected(item);

    }
}
```

Opções de visualização dos action buttons

A manipulação da visualização dos actions buttons é determinada pelo atributo *app:showAsAction=”xxx”,* através do uso das seguintes constantes :

always : indica que o botão sempre vai ficar visível como action Button. Recomendo para as ações mais comuns do app

ifRoom : mostra o botão na action bar se existir espaço, ou move ele automaticamente para o menu action overflow caso não tenha

withText : mostra o título do botão ao lado do ícone, caso tenha espaço disponível na action bar

never : Nunca mostra o botão na action bar, deixando a ação no menu action overflow

7. Listagem de Elementos com ListView

A classe *android.widget.ListView* é um dos componentes visuais mais utilizados e representa uma lista na tela. Para utilizar ListView é preciso :

1º Declarar uma View do tipo ListView no arquivo XML de Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.example.pedroivo.teste.MainActivity">

    <ListView
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:id="@+id/Lista"/>

</LinearLayout>
```

2º Utilizamos um Adapter para popular o ListView.

Quem fornece os dados para preencher o ListView é esse Adapter, que é uma classe que implementa a interface *android.widget.ListAdapter.* Opcionalmente podemos estender a classe *android.widget.BaseAdapter* que já implementa essa interface e deixa poucos métodos abstratos para terminarmos a implementação.

```
public class SimplesAdapter extends BaseAdapter {

    private String[] filmes = new String[]{
            "Interestelar",
            "A Origem",
            "Star Trek",
            "Star Wars",
            "As Crônicas de Nárnia"
    };
    private Context c;

    public SimplesAdapter(Context c){
        super();
        this.c = c;// O context é necessário para criar a view
    }

    @Override
    public int getCount() {
        return filmes.length;//Retorna a quantidade de itens do adapter
    }

    @Override
    public Object getItem(int position) {
        return filmes[position];//Retorna o objeto para essa position
    }
    @Override
    public long getItemId(int position) {
        return position;//Retorna a id do objeto para esta position
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

        String filme = filmes[position];
        TextView t = new TextView(c);
        t.setText(filme);

        return t;
    }
}
```

3º Para finalizar, precisamos obter o objeto correspondente à ListView que declaramos no XML de layout, na classe Activity correspondente a este, então chamamos o método *setAdapter(adapter)* do *ListView* informando o adapter. A lista terá a quantidade de linhas que o adapter retornar no método *getCount().*

```
public class MainActivity extends AppCompatActivity {

     private ListView l;

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

        l = (ListView) findViewById(R.id.Lista);
        l.setAdapter(new SimplesAdapter(this));
    }
}
```

8. AlertDialog

Diferente da classe Toast que mostra uma mensagem com a qual o usuário que não pode interagir, a classe AlertDialog permite interação com usuário, permitindo que ele aperte **OK** ou responda **Sim** ou **Não** para uma pergunta.

```
public class MainActivity extends AppCompatActivity {

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

        AlertDialog.Builder b = new AlertDialog.Builder(this);
        b.setTitle("AlertDialog Example");
        b.setMessage("Message");

        b.setPositiveButton("Sim", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                Toast.makeText(getBaseContext(),"Clicou em Sim", Toast.LENGTH_SHORT).show();
                return;
            }
        });

        b.setNegativeButton("Não", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                Toast.makeText(getBaseContext(),"Clicou em Não", Toast.LENGTH_SHORT).show();
                return;
            }
        });

        AlertDialog aD = b.create();
        aD.show();
    }
}
```

Diferente do Toast, que aparece por um intervalo de tempo e some, esse obriga o usuário a responder algo.

9. Notifications

Uma Notification é uma mensagem especial que aparece na barra de status do Android para chamar atenção do usuário.

Como exemplo, criemos botões para que, ao clicados, gerem Notifications de diferentes tipos.

```
<Button
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:text="Notificação Simples"
    android:onClick="onClickNotificacaoSimples"/>

<Button
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:text="Notificação Heads Up"
    android:onClick="onClickNotificacaoHeadsUp"/>

<Button
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:text="Notificação Big"
    android:onClick="onClickNotificacaoBig"/>
```

Logo após, para encapsular a criação das notificações, criamos a classe NotificationUtil, que gere toda a criação e manipulação das notificações.

Primeiro método da classe

```
//Cria a PendingIntent para abrir a activity da Intent
private static PendingIntent getPendingIntent(Context context, Intent intent, int id){
    TaskStackBuilder stackBuilder = TaskStackBuilder.create(context);

    //Essa linha mantém a activity pai na pilha de activities
    stackBuilder.addParentStack(intent.getComponent());

    //Configura a Intent que vai abrir ao clicar na notificação
    stackBuilder.addNextIntent(intent);

    //Cria a PendingIntent e atualiza caso exista uma com mesmo id
    PendingIntent p = stackBuilder.getPendingIntent(id, PendingIntent.FLAG_UPDATE_CURRENT);

    return p;
}
```

Criação da Notificação Simples

```
public static void createNotificacaoSimples(Context context, Intent intent,
                                             String contentTitle,String contentText, int id){

    //Cria a PendingIntent( Contém a Intent Original )
    PendingIntent p = getPendingIntent(context, intent, id);

    //Cria a Notificação
    NotificationCompat.Builder b = new NotificationCompat.Builder(context);

    b.setDefaults(Notification.DEFAULT_ALL); //Ativa configurações padrão
    b.setContentTitle(contentTitle);//Título
    b.setContentText(contentText);//Mensagem
    b.setSmallIcon(R.drawable.ic_accessibility_black_24dp);//Ícone
    b.setContentIntent(p);//Intent que será chamada ao clicar na notificação
    b.setAutoCancel(true);//Autocancela a Notificação ao clicar nela
    NotificationManagerCompat nm = NotificationManagerCompat.from(context);
    //Neste caso a notificação será cancelada quando o usuário clicar nela
    //Mas o id serve para cancelá-la manualmente se necessário
    nm.notify(id, b.build());

}
```

Chamando o Método de Criação da Notificação Simples
(Ao declarar a Intent, informamos para qual Activity a aplicação irá ao interagir com a Notificação)

```
public void onClickNotificacaoSimples(View view){
    int id = 1;
    String contentTitle = "Nova Mensagem";
    String contextText = "Você possui 69 novas mensagens";
    Intent i = new Intent(this, MensagemActivity.class);
    getIntent().putExtra("msg", "Olá, como vai ?");
    NotificationUtil.createNotificacaoSimples(this, i , contentTitle, contextText, id);
}
```

Criação da Notificação Heads Up

```
public static void createNotificacaoHeadsUp(Context context, Intent intent,
                                             String contentTitle,String contentText, int id){

    //Cria a PendingIntent( Contém a Intent Original )
    PendingIntent p = getPendingIntent(context, intent, id);

    //Cria a Notificação
    NotificationCompat.Builder b = new NotificationCompat.Builder(context);

    b.setColor(Color.BLUE);
    b.setFullScreenIntent(p,false);

    b.setDefaults(Notification.DEFAULT_ALL); //Ativa configurações padrão
    b.setContentTitle(contentTitle);//Título
    b.setContentText(contentText);//Mensagem
    b.setSmallIcon(R.drawable.ic_accessibility_black_24dp);//Ícone
    b.setContentIntent(p);//Intent que será chamada ao clicar na notificação
    b.setAutoCancel(true);//Autocancela a Notificação ao clicar nela
    NotificationManagerCompat nm = NotificationManagerCompat.from(context);
    //Neste caso a notificação será cancelada quando o usuário clicar nela
    //Mas o id serve para cancelá-la manualmente se necessário
    nm.notify(id, b.build());

}
```

Chamando o Método de Criação da Notificação Heads Up

```
public void onClickNotificacaoHeadsUp(View view){
    int id = 1;
    String contentTitle = "Nova Mensagem";
    String contextText = "Você possui 69 novas mensagens";
    Intent i = new Intent(this, MensagemActivity.class);
    getIntent().putExtra("msg", "Olá, como vai ?");
    NotificationUtil.createNotificacaoHeadsUp(this, i , contentTitle, contextText, id);
}
```

Criação da Notificação Big

```
public static void createNotificacaoBig(Context context, Intent intent,
                                        String contentTitle, String contentText,
                                        List<String> lines, int id){

    //Cria a PendingIntent( Contém a Intent Original )
    PendingIntent p = getPendingIntent(context, intent, id);

    //Configura o estilo inbox
    int size = lines.size();
    NotificationCompat.InboxStyle inboxStyle = new NotificationCompat.InboxStyle();
    inboxStyle.setBigContentTitle(contentTitle);
    for(String s: lines){
        inboxStyle.addLine(s);
    }
    inboxStyle.setSummaryText(contentText);

    //Cria a Notificação
    NotificationCompat.Builder b = new NotificationCompat.Builder(context);

    b.setColor(Color.BLUE);
    b.setFullScreenIntent(p,false);

    b.setDefaults(Notification.DEFAULT_ALL); //Ativa configurações padrão
    b.setContentTitle(contentTitle);//Título
    b.setContentText(contentText);//Mensagem
    b.setSmallIcon(R.drawable.ic_accessibility_black_24dp);//Ícone
    b.setContentIntent(p);//Intent que será chamada ao clicar na notificação
    b.setAutoCancel(true);//Autocancela a Notificação ao clicar nela

    b.setNumber(size);
    b.setStyle(inboxStyle);

    NotificationManagerCompat nm = NotificationManagerCompat.from(context);
```

Chamando o Método de Criação da Notificação Big

```java
public void onClickNotificacaoBig(View view){
    int id= 2;
    String contentTitle = "Nova Mensagem";
    String contextText = "Você possui 69 novas mensagens";
    Intent i = new Intent(this, MensagemActivity.class);
    getIntent().putExtra("msg", "Olá, como vai ?");

    List<String> lines = new ArrayList<>();
    lines.add("Mensagem 1");
    lines.add("Mensagem 2");
    lines.add("Mensagem 3");
    NotificationUtil.createNotificacaoBig(this, i, contentTitle, contextText , lines, id );
}
```

Métodos de Cancelamento

```java
public static void cancell(Context context, int id){
    NotificationManagerCompat nm = NotificationManagerCompat.from(context);
    nm.cancel(id);
}

public static void cancellAll(Context context){
    NotificationManagerCompat nm = NotificationManagerCompat.from(context);
    nm.cancelAll();
}
```